CAPÍTULO 5

Estudo de Caso no Varejo: Um Guia Prático para Transformar Dados em Decisões

https://doi.org/10.22533/at.ed.526172513115

Lorena Gonçalves Fernandes

Elisa Norberto Ferreira Santos

RESUMO: Com o avanço das tecnologias da informação e comunicação, a geração de grandes volumes de dados tornou-se uma constante, tornando indispensável a automação das etapas de coleta, processamento e análise para subsidiar processos de tomada de decisão. Nesse contexto, a Inteligência Artificial (IA) consolida-se como uma aliada estratégica, capaz de transformar grandes quantidades de dados em informações relevantes para o planejamento e a gestão organizacional. No setor varejista, a evolução tecnológica é particularmente significativa: o registro manual de transações deu lugar a sistemas digitalizados e orientados por dados, impulsionados por ferramentas analíticas e soluções baseadas em IA. Diante desse cenário, este trabalho apresenta um tutorial aplicado de análise de dados voltado ao varejo, demonstrando como o uso de bibliotecas Python — como Pandas, Matplotlib, Seaborn e Scikit-Learn — pode aprimorar a eficiência gerencial e gerar insights sobre o comportamento de compra dos consumidores. O estudo tem como propósito facilitar o acesso a técnicas de análise de dados por meio de um guia prático, configurando-se como uma ferramenta acessível e didática que contribui para a democratização do conhecimento técnico e o fortalecimento da cultura de decisões baseadas em dados.

PALAVRAS-CHAVE: Análise de dados. Varejo. Python. RFM. K-Means. Árvore de decisão.

INTRODUÇÃO

Com o avanço contínuo da tecnologia da informação e comunicação, a geração de grandes volumes de dados tornou-se uma constante. Para extrair informações úteis desses dados, é essencial automatizar diversas etapas de coleta, processamento e análise, com o objetivo de apoiar a tomada de decisões. Devido ao vasto volume de dados disponíveis, essas tarefas não podem ser realizadas manualmente de forma eficiente. É nesse cenário que a Inteligência Artificial (AI) se destaca, buscando replicar o comportamento do cérebro humano por meio de máquinas. (Kalinowski,2023)

As aplicações computadorizadas evoluíram de simples atividades de processamento e monitoramento de transações para funções mais complexas, como análise e solução de problemas. Ferramentas de análise de dados, incluindo armazenamento de dados, mineração de dados, processamento analítico online (OLAP), *dashboards* e sistemas baseados na nuvem para suporte à tomada de decisões, tornaram-se pilares fundamentais da gestão moderna (Sharda, 2019).

O varejo pode ser descrito como o processo de adquirir produtos em grandes quantidades de atacadistas ou outros fornecedores e, em seguida, vendê-los em quantidades menores ao consumidor final. Também pode ser definido como o conjunto de atividades envolvidas na venda de produtos e serviços para atender às necessidades pessoais do consumidor final.

A tecnologia no varejo passou de simples métodos de troca e registro para um cenário altamente digital e automatizado. Desde as primeiras caixas registradoras até as avançadas soluções de IA e *blockchain*, ela tem desempenhado um papel fundamental na transformação do setor. À medida que novas inovações continuam a surgir, o varejo precisa se adaptar constantemente para atender às expectativas dos consumidores e capitalizar as oportunidades oferecidas pelas tecnologias emergentes. O futuro do varejo será definido pela capacidade das empresas de integrar e utilizar essas tecnologias de forma estratégica e eficiente. (Guardelli, 2024).

Uma das tecnologias que pode ser utilizada nesse cenário é o Google Colaboratory, também conhecido como Colab, que é uma plataforma digital de acesso livre oferecida pela Google, voltada para a aplicação de conhecimentos em programação utilizando a linguagem Python.

O objetivo geral deste trabalho foi facilitar a análise de dados para o usuário, oferecendo ferramentas e metodologias que simplifiquem o processo de tomada de decisão com base em informações relevantes. Para atingir esse objetivo, os objetivos específicos foram desenvolvidos por meio da criação de um capítulo dedicado, onde será apresentado um passo a passo prático que ensinará como realizar uma análise de dados aplicada ao setor de varejo usando Python.

ACESSANDO O GOOGLE COLAB

O Google Colab, também chamado de "Colaboratório", é um ambiente digital gratuito, hospedado na nuvem pelo Google. Essa ferramenta tem como finalidade oferecer recursos para o desenvolvimento em Python. O ambiente funciona como uma máquina virtual, chamada de notebook, onde o usuário pode realizar práticas de programação em Python. (Da Silva, 2020).

Primeiramente deve-se acessar o site do Google Colab. Para isso, abra seu navegador e entre no endereço "https://colab.research.google.com". Se ainda não estiver logado, o Google vai pedir seu login. Pode-se usar sua conta do Gmail (Figura 1 e 2).

Figura 1 - Ambiente de Desenvolvimento Google Colab.

Fonte: O autor (2025).

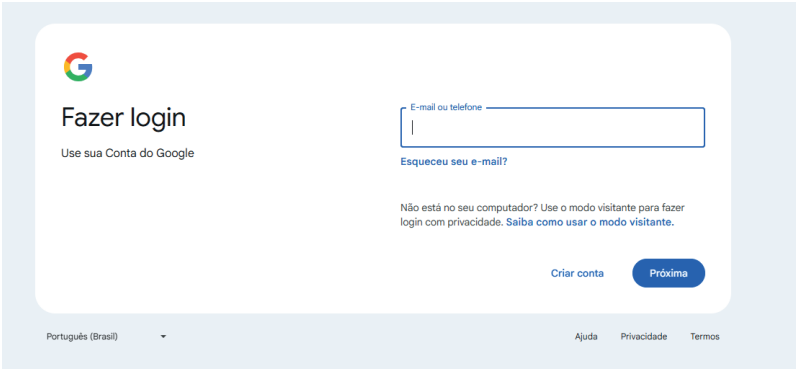

Figura 2 - Interface de Autenticação de Conta Google.

Fonte: O autor (2025).

Jupyter é a plataforma mais utilizada para programação literária interativa. (Shen,2014)

O Jupyter foi criado como uma evolução do IPython, um sistema REPL (laço de leitura-avaliação-impressão) para a linguagem Python. Embora ofereça suporte a diversas linguagens de programação, a linguagem mais frequentemente utilizada para códigos executáveis em Notebooks (documentos do Jupyter) ainda é o Python.

O Jupyter armazena os Notebooks como arquivos no formato JSON com extensão ".ipynb". Esses arquivos são formados por células, que podem ser de três tipos: células de código, que contêm código executável e geram resultados; células de Markdown, que contêm texto formatado; e células brutas, que apresentam texto que não é nem executável nem em Markdown. Geralmente, as células brutas são utilizadas por ferramentas externas para converter os notebooks em outros formatos. (Pimentel,2021)

Criar um novo notebook

Na janela do Google Colab, clique em "Novo notebook" (ou "New Notebook") como mostrado na Figura 3.

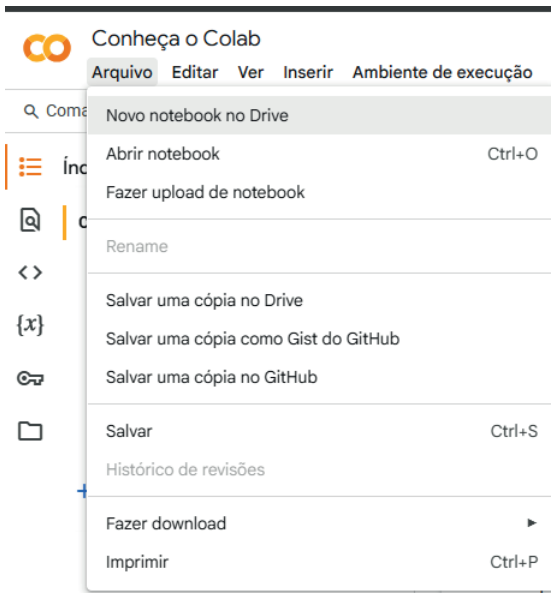

Figura 3 - Menu "Arquivo" no Google Colab.

Fonte: O autor (2025)

No lado esquerdo da tela, clique no ícone de pasta ( ) para abrir o painel de arquivos, conforme mostrado na Figura 4.

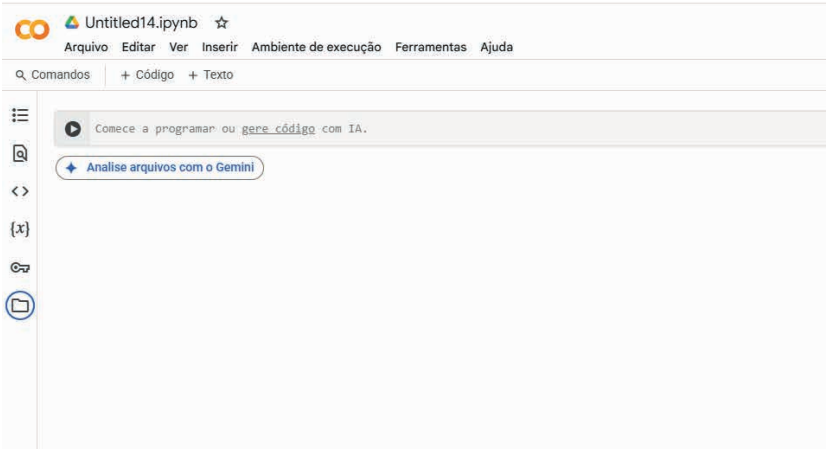

Figura 4 - Interface de um Notebook Vazio no Google Colab.

Fonte: O autor (2025)

Clique em **"Upload".**

O termo *upload* refere-se ao processo de envio de dados de um dispositivo local para um sistema remoto, geralmente do computador do usuário (cliente) para um servidor (KUROSE, 2014).

Após abrir o painel de arquivos, clique no ícone de upload e selecione o arquivo **.csv** armazenado em seu computador.

A sigla **CSV** significa *Comma Separated Values*, ou "valores separados por vírgula", em português. De forma simplificada, trata-se de um tipo de arquivo estruturado em que os dados são organizados em linhas e colunas, separados por vírgulas, obedecendo a certas regras de formatação (OLIVEIRA, 2015).

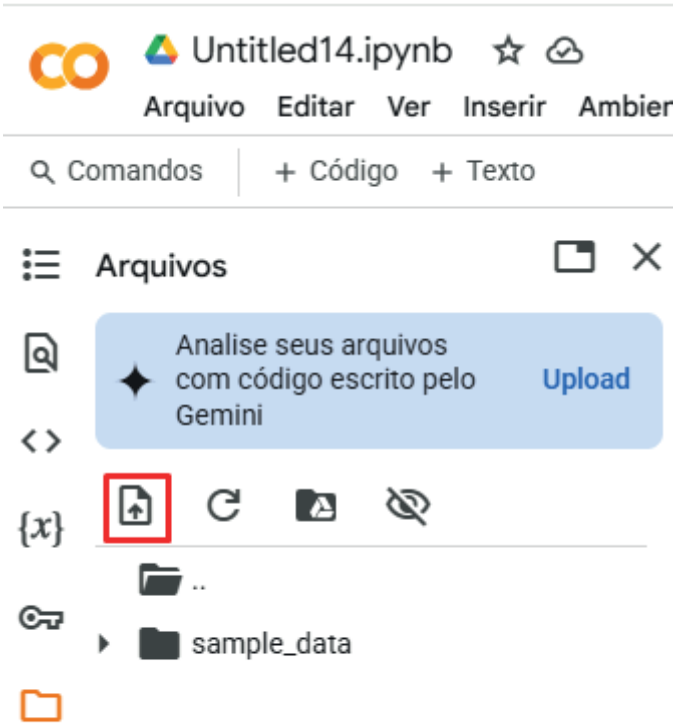

Figura 5 - Painel de Arquivos no Google Colab

Fonte: O autor (2025)

GUIA PRÁTICO PARA TRANSFORMAR DADOS EM DECISÕES

Importando as Bibliotecas

As bibliotecas são coleções de funções prontas que podem ser utilizadas para diferentes finalidades.

Nesta fase inicial, vamos importar todas as bibliotecas que serão utilizadas ao longo do projeto. Cada uma delas tem um papel específico na manipulação de dados, visualizações, modelagem e tratamento de warnings (Figura 6).

```
[ ]  import pandas as pd
     import kagglehub
     import os
     import datetime as dt
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.cluster import KMeans
     from warnings import simplefilter
     from sklearn.preprocessing import StandardScaler
     import plotly.express as px
     import numpy as np
```

Figura 6 - Importação de Bibliotecas Python.

Fonte: O autor (2025)

Lendo um arquivo CSV.

Nesta etapa, vamos importar um conjunto de dados de vendas online a partir de um arquivo no formato .csv, utilizando a biblioteca pandas. Essa ação é fundamental para iniciar a análise dos dados, pois permite que o conteúdo do arquivo seja transformado em uma estrutura tabular (DataFrame) que facilita a manipulação e visualização das informações.

Pandas é uma biblioteca open source da linguagem Python, desenvolvida para facilitar a manipulação e a análise de dados. Ela disponibiliza estruturas de dados eficientes, versáteis e intuitivas — como o DataFrame e a Series — que são especialmente úteis no tratamento de dados organizados e rotulados. McKinney (2018)

DataFrame é um objeto rápido e eficiente para manipulação de dados com indexação embutida. Trata-se de uma estrutura em forma de tabela, composta por linhas e colunas. As linhas possuem um índice próprio para acesso, que pode ser representado por um nome ou um valor. As colunas, conhecidas como series, são um tipo especial de estrutura de dados que corresponde a uma lista de valores, cada um com seu respectivo índice. Assim, o DataFrame pode ser comparado a uma planilha, porém com muito mais flexibilidade.(Pimentel, 2021)

```
[ ]  retail_online_df = pd.read_csv('/content/OnlineRetail.csv', encoding='ISO-8859-1')
```

Figura 7 - Código Python para Importação da Base de Dados de Varejo

Fonte: O autor (2025)

Visualizando as Primeiras Linhas do DataFrame.

A função .head() será utilizada para exibir as primeiras linhas do DataFrame no pandas. Por padrão, ela retorna as cinco primeiras linhas, permitindo uma visualização inicial da estrutura dos dados, como os nomes das colunas, os tipos de valores e possíveis inconsistências no conteúdo (Figura 8).

Utilizamos um conjunto de dados disponível publicamente na plataforma Kaggle, que reúne informações de transações do varejo online. Esse dataset serve como base para todas as análises e modelagens realizadas. Ele pode ser acessado no link: https://www.kaggle.com/datasets/ishanshrivastava28/tata-online-retail-dataset

```
[ ] retail_online_df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	01-12-2010 08:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 08:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01-12-2010 08:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01-12-2010 08:26	3.39	17850.0	United Kingdom

Figura 8 - Carregamento de Dados CSV.

Fonte: O autor (2025)

Verificando as Dimensões do DataFrame

Agora vamos conferir o tamanho do nosso DataFrame. Para isso, usamos a propriedade .shape

Quando executamos esse comando, ele nos retorna uma tupla no formato (n_linhas, n_colunas).

O primeiro valor indica o número de linhas, ou seja, quantos registros temos.

O segundo valor mostra o número de colunas, que corresponde às variáveis ou campos disponíveis (Figura 9).

```
[ ] retail_online_df.shape

(541909, 8)
```

Figura 9 - Dimensões do Conjunto de Dados.

Fonte: O autor (2025)

Explorando as Informações Gerais do DataFrame

Agora vamos usar o comando .info() para dar uma olhada mais completa no nosso DataFrame: Esse comando exibe um resumo geral da estrutura dos dados. Quando executamos, ele mostra:

O número total de entradas (linhas).

O nome de cada coluna.

A quantidade de valores não nulos em cada coluna (ótimo para identificar dados ausentes).

O tipo de dado de cada coluna (como int, float, object etc.).

É uma etapa super útil pra gente entender rapidamente o que temos no dataset e o que talvez precise ser ajustado antes de seguir com a análise (Figura 10).

Um dataset é um conjunto de dados que pode reunir diversas variáveis relacionadas a um ou mais objetos, sendo utilizado como base para análises estatísticas, algoritmos de aprendizado de máquina e demais métodos computacionais. IBM (2020)

```
[ ]  retail_online_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

Figura 10 - Resumo Detalhado dos Dados.

Fonte: O autor (2025)

O conjunto de dados possui 541.909 registros de vendas (Figura 10), conforme indicado no RangeIndex. Podemos observar que algumas colunas apresentam valores diferentes, sendo a coluna Description e a coluna CustomerID.

Verificando Valores Ausentes

Agora que já conhecemos a estrutura dos dados, vamos verificar se temos valores ausentes (nulos) em alguma coluna.

```
[ ]  retail_online_df.isnull().sum()
```

	0
InvoiceNo	0
StockCode	0
Description	1454
Quantity	0
InvoiceDate	0
UnitPrice	0
CustomerID	135080
Country	0

dtype: int64

Figura 11 - Verificação de Dados Ausentes.

Fonte: O autor (2025)

Podemos observar pela Figura 11 que algumas colunas apresentam valores faltantes, sendo a coluna Description com uma quantidade menor de ausências e a coluna CustomerID com um número consideravelmente maior de dados em falta.

Removendo Dados Duplicados

Agora vamos realizar uma etapa de limpeza dos dados, removendo possíveis linhas duplicadas. Essa ação é importante para garantir que cada linha represente uma transação única, evitando que informações repetidas distorçam os resultados da nossa análise (Figura 12).

```
[ ]  retail_online_df.drop_duplicates(inplace = True)
```

Figura 12 - Tratamento de Registros Duplicados.

Fonte: O autor (2025)

Removendo Registros sem ID de Cliente

Agora vamos limpar o DataFrame removendo as linhas que não possuem o ID do cliente, ou seja, onde o valor da coluna CustomerID está ausente (Figura 13). Esta limpeza é essencial porque, sem o ID do cliente, não conseguimos associar a compra ao cliente.

```
[ ]  retail_online_df.dropna(subset = ['CustomerID'], inplace = True)
```

Figura 13 - Remoção de Valores Ausentes na Coluna CustomerID.

Fonte: O autor

Verificando Valores Ausentes

Agora que já removemos os registros duplicados e sem ID de cliente, vamos verificar novamente se ainda existem valores ausentes no nosso DataFrame.

```
[ ]  retail_online_df.isnull().sum()
```

	0
InvoiceNo	0
StockCode	0
Description	0
Quantity	0
InvoiceDate	0
UnitPrice	0
CustomerID	0
Country	0

dtype: int64

Figura 14 - Confirmação de Ausência de Nulos. (2025)

Fonte: O autor (2025)

A Figura 14 confirma que o dataset não possui valores ausentes em nenhuma de suas colunas, uma vez que o número de células com dados faltantes é igual a zero em todas elas.

Criando a Coluna de Valor Total da Compra

Agora vamos adicionar uma nova coluna chamada Total (Figura 15), que representa o valor total de cada transação (quantidade × preço unitário).

```
[ ]  retail_online_df['Total'] = retail_online_df['Quantity'] * retail_online_df['UnitPrice']
```

Figura 15 - Criação da Coluna 'Total'

Fonte: O autor (2025)

Verificando a Data da Última Compra

Esse comando retorna a data mais recente registrada na coluna InvoiceDate, que representa o momento em que a fatura foi emitida — ou seja, a data em que a compra foi realizada (Figura 16).

```
[ ]  retail_online_df["InvoiceDate"].max()

'31-10-2011 17:13'
```

Figura 16 - Identificação da Última Data de Fatura no Conjunto de Dados.

Fonte: O autor (2025)

Convertendo a Coluna de Datas para o Formato Correto

Agora vamos garantir que a coluna InvoiceDate esteja no formato de data e hora (datetime), para que possamos fazer filtros, agrupamentos por mês, análises por período e etc. (Figura 17).

```
[ ]  retail_online_df['InvoiceDate'] = pd.to_datetime(retail_online_df['InvoiceDate'], dayfirst=True)
```

Figura 17 - Conversão da Coluna 'InvoiceDate' para Formato de Data e Hora. Fonte: O autor (2025)

Criando a Tabela RFM (Recency, Frequency, Monetary Value)

Agora vamos construir uma análise RFM (Figura 18) para entender melhor o comportamento dos clientes com base em três métricas:

Recency (Recência): Quanto tempo faz desde a última compra.

Frequency (Frequência): Quantas compras o cliente realizou.

Monetary Value (Valor Monetário): Quanto ele gastou no total.

Estamos agrupando os dados por cliente (CustomerID) e calculando:

Recency: diferença entre a última data do dataset e a data da última compra do cliente.

Frequency: total de compras (contagem de faturas).

MonetaryValue: soma do valor total gasto pelo cliente.

```
[ ]  latest_date = retail_online_df['InvoiceDate'].max() + dt.timedelta(days = 1)

     rfm = retail_online_df.groupby('CustomerID').agg({
         'InvoiceDate': lambda x: (latest_date - x.max()).days,
         'InvoiceNo': 'count',
         'Total': 'sum'
     }).reset_index()

     rfm.rename(columns = {
         'InvoiceDate': 'Recency',
         'InvoiceNo': 'Frequency',
         'Total': 'MonetaryValue'
     }, inplace = True)
```

Figura 18 - Geração das Métricas de Recência, Frequência e Valor Monetário. Fonte: O autor (2025)

Criando os Scores de Recência e Frequência

Agora vamos atribuir notas de 1 a 5 para as métricas de Recência e Frequência. Isso nos ajuda a identificar os clientes mais valiosos e engajados.(Figura 19)

Score de Recência

pd.qcut(...) divide os valores em quintis (5 faixas com a mesma quantidade de clientes).

A ideia é que quanto menor a recência, melhor — ou seja, clientes que compraram mais recentemente recebem nota 5.

Por isso as notas são invertidas: quanto menor o número de dias, maior o score.

Score de Frequência

Usamos o .rank() para garantir que cada cliente tenha uma posição única antes da divisão em faixas.

Neste caso, quanto mais compras, melhor, então os clientes mais frequentes recebem nota 5.

```
[ ]  rfm["Recency_score"] = pd.qcut(rfm['Recency'], 5, labels = [5, 4, 3, 2, 1])

     rfm["Frequency_score"] = pd.qcut(rfm['Frequency'].rank(method = "first"), 5, labels = [1, 2, 3, 4, 5])
```

Figura 19 - Atribuição de Scores de Recência e Frequência.

Fonte: O autor (2025)

Criando o Segmento RFM Combinado

Com os scores de Recência e Frequência já criados, agora vamos juntá-los em uma única string para formar o segmento RFM (Figura 20).

Transformamos os valores de Recency_score e Frequency_score em strings com .astype(str).

Depois, concatenamos os dois para formar um código como "55", "43", "21" etc.

Cada combinação representa um perfil de cliente diferente.

```
[ ]  rfm["rfm_segment"] = rfm['Recency_score'].astype(str) + rfm['Frequency_score'].astype(str)
```

Figura 20 - Combinação dos Scores RFM

Fonte: O autor (2025)

Estamos quase finalizando a preparação da análise RFM e, agora, vamos visualizar alguns registros da tabela rfm, conforme mostrado na Figura 21.

```
[ ]  rfm
```

	CustomerID	Recency	Frequency	MonetaryValue	Recency_score	Frequency_score	rfm_segment
0	12346.0	326	2	0.00	1	1	11
1	12347.0	2	182	4310.00	5	5	55
2	12348.0	75	31	1797.24	2	3	23
3	12349.0	19	73	1757.55	4	4	44
4	12350.0	310	17	334.40	1	2	12
...	...	...	...	...	...	...	...
4367	18280.0	278	10	180.60	1	1	11
4368	18281.0	181	7	80.82	1	1	11
4369	18282.0	8	13	176.60	5	1	51
4370	18283.0	4	721	2045.53	5	5	55
4371	18287.0	43	70	1837.28	3	4	34

Figura 21 - Estrutura do Conjunto de Dados RFM

Fonte: O autor (2025)

Visualizando a Distribuição da Recência

Vamos criar um histograma mostrando como os clientes estão distribuídos de acordo com a Recência. Este tipo de visualização ajuda a identificar quantos clientes ainda estão ativos (Figura 22).

```
simplefilter(action='ignore', category=FutureWarning)
sns.histplot(data = rfm, x = 'Recency', bins = 30)
plt.title('Recency Distribution')
plt.show()
```

Figura 22 - Código para Histograma de Recência

Fonte: O autor (2025)

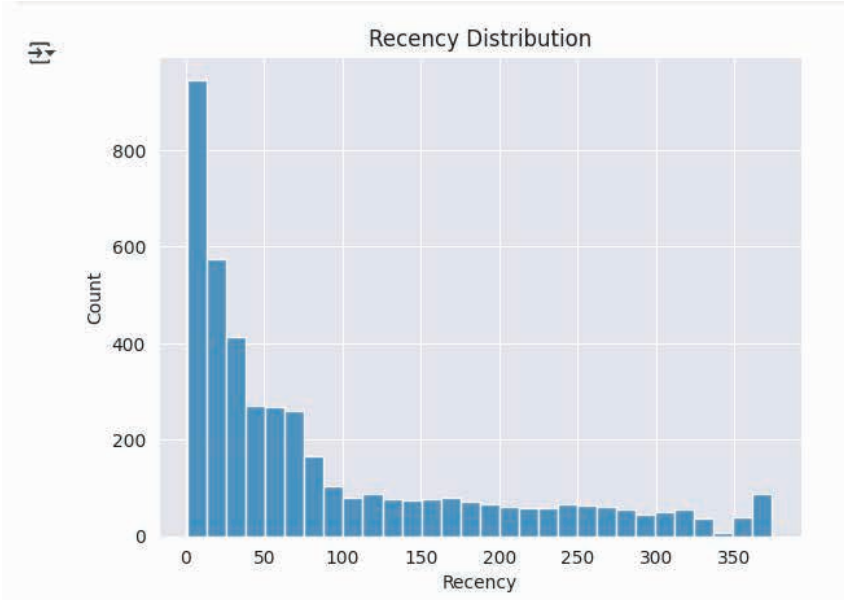

Figura 23 - Visualização da Distribuição da Recência

Fonte: O autor (2025).

Na Figura 23, observamos o histograma da variável Recência, que mostra como os clientes estão distribuídos de acordo com o tempo desde a última compra. É possível perceber um grande pico à esquerda, indicando que muitos clientes realizaram compras recentemente. Essa é uma característica comum em bases com clientes ativos, já que a maioria mantém um relacionamento recente com a empresa, enquanto apenas uma pequena parcela está há mais tempo sem comprar.

Visualizando a Distribuição da Frequência

Nesta etapa vamos criar um histograma que mostra quantas vezes cada cliente comprou.

É comum que a maior parte dos clientes estejam concentrada nas faixas mais baixas do gráfico, pois a maioria tende a realizar poucas compras. Apenas uma pequena parcela dos clientes costuma comprar com frequência mais alta (Figura 24).

```
sns.histplot(data = rfm, x = 'Frequency', bins = 30)
plt.title('Frequency Distribution')
plt.show()
```

Figura 24 - Código para Histograma da Frequência.

Fonte: O autor (2025).

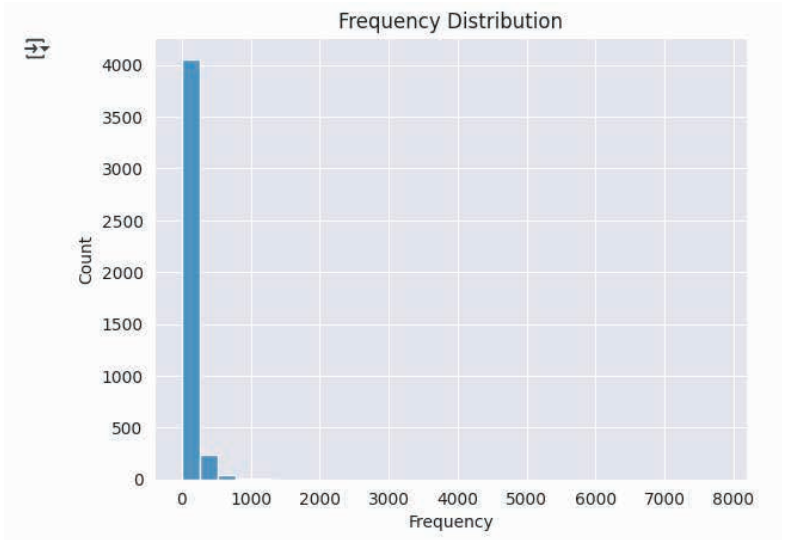

Figura 25 - Histograma da Distribuição de Frequência.

Fonte: O autor (2025).

Ao observarmos a Figura 25, que apresenta o histograma da variável Frequência, é possível perceber um grande pico à esquerda, indicando que a maioria dos clientes realizou poucas compras. Essa é uma característica comum em bases de clientes, pois, geralmente, apenas uma pequena parcela apresenta maior frequência de compras.

Visualizando a Distribuição do Valor Monetário

Com o código abaixo, vamos criar um histograma mostrando quanto os clientes gastaram no total (Figura 26).

É comum observar que a maioria dos clientes gasta pouco, e poucos clientes concentram os maiores valores.

```
sns.histplot(data = rfm, x = 'MonetaryValue', bins = 30)
plt.title('Monetary Value Distribution')
plt.show()
```

Figura 26 - Código para Histograma do Valor Monetário.

Fonte: O autor (2025).

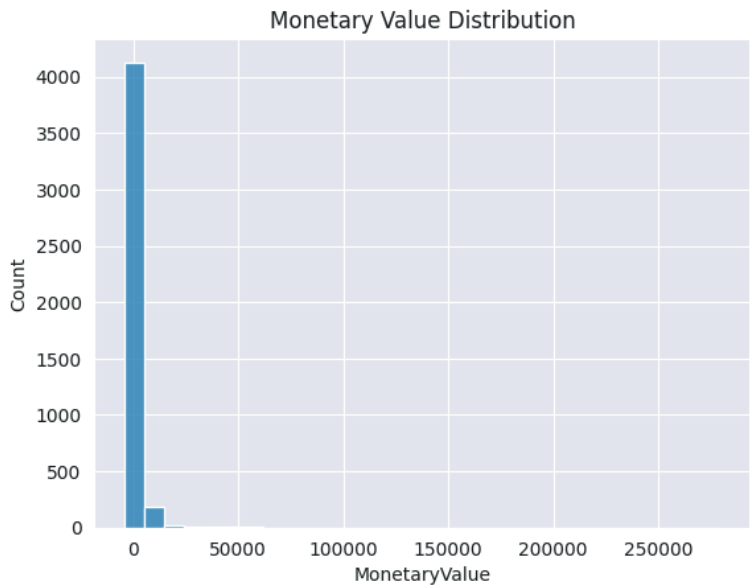

Figura 27 - Histograma da Distribuição do Valor Monetário.

Fonte: O autor (2025).

Ao analisarmos a Figura 27, que apresenta o histograma da variável MonetaryValue, observamos uma concentração extremamente elevada de clientes na faixa de valores mais baixos. Esse grande pico à esquerda indica que a maior parte

dos consumidores realizou compras de baixo valor total no período analisado. À medida que avançamos para valores monetários mais altos, a quantidade de clientes diminui rapidamente, revelando uma cauda longa típica de distribuições financeiras.

Definindo Rótulos

Na Figura 28, estamos definindo rótulos estratégicos para os diferentes perfis de clientes, com base na combinação dos scores de Recência e Frequência (o rfm_segment que criamos anteriormente).

```
segment_map = {
    r'[1-2][1-2]' : 'Clientes Inativos',
    r'[1-2][3-4]' : 'Clientes em Risco',
    r'[1-2]5'     : 'Ex-Clientes Valiosos',
    r'3[1-2]'     : 'Quase Perdidos',
    r'33'         : 'Clientes Indecisos',
    r'[3-4][4-5]' : 'Clientes Frequentes',
    r'41'         : 'Clientes Promissores',
    r'51'         : 'Novo Cliente',
    r'[4-5][2-3]' : 'Potencial Cliente Fiel',
    r'5[4-5]'     : 'Clientes Campeões',
}
```

Figura 28 – Definição dos Rótulos

Fonte: O autor (2025).

Ao analisarmos a Figura 28, observa-se que combinações de scores mais baixos são associadas a perfis como Clientes Inativos ou Clientes em Risco, indicando indivíduos que não compram há muito tempo ou que reduziram drasticamente sua interação com a empresa. Por outro lado, combinações que refletem alta recência, frequência ou valor monetário recebem rótulos como Clientes Frequentes, Potencial Cliente Fiel e Clientes Campeões, representando consumidores altamente engajados e valiosos para o negócio.

Aplicando os Segmentos RFM

Agora que já definimos o dicionário com os segmentos, chegou a hora de traduzir os códigos de score (como '55', '21', etc.) para nomes de segmentos mais descritivos, que facilitam a leitura e a análise (Figura 29).

```
[ ]  rfm['rfm_segment'] = rfm['rfm_segment'].replace(segment_map, regex = True)
```

Figura 29 - Aplicação do Mapeamento de Segmentos.

Fonte: O autor (2025).

Visualizando os Registros do DataFrame RFM

Depois de aplicarmos os segmentos, vamos verificar nosso DataFrame rfm pra verificar se os dados estão como esperado.

```
rfm
```

	CustomerID	Recency	Frequency	MonetaryValue	Recency_score	Frequency_score	rfm_segment
0	12346.0	326	2	0.00	1	1	Clientes Inativos
1	12347.0	2	182	4310.00	5	5	Clientes Campeões
2	12348.0	75	31	1797.24	2	3	Clientes em Risco
3	12349.0	19	73	1757.55	4	4	Clientes Frequentes
4	12350.0	310	17	334.40	1	2	Clientes Inativos
...	...	...	...	...	...	...	...
4367	18280.0	278	10	180.60	1	1	Clientes Inativos
4368	18281.0	181	7	80.82	1	1	Clientes Inativos
4369	18282.0	8	13	176.60	5	1	Novo Cliente
4370	18283.0	4	721	2045.53	5	5	Clientes Campeões
4371	18287.0	43	70	1837.28	3	4	Clientes Frequentes

4372 rows × 7 columns

Figura 30 - Visualizando os Registros do DataFrame RFM

Fonte: O autor (2025).

Ao analisarmos a Figura 30, que apresenta uma amostra do DataFrame RFM após o cálculo dos scores e a atribuição dos segmentos, é possível observar claramente como cada cliente foi classificado com base em seu comportamento recente de compra. As colunas de Recency, Frequency e MonetaryValue revelam os dados brutos, enquanto Recency_score, Frequency_score e o rótulo final rfm_segment traduzem essas informações em uma categorização estratégica.

Padronizando os Dados com StandardScaler

Antes de aplicarmos algoritmos de agrupamento (como K-Means), é importante padronizar os dados — ou seja, colocar todas as variáveis (Recency, Frequency e MonetaryValue) na mesma escala.

Pra isso, usamos o StandardScaler do Scikit-Learn (Figura 31).

```
[ ]  scaler = StandardScaler()
     rfm_scaled = scaler.fit_transform(rfm[['Recency', 'Frequency', 'MonetaryValue']])

     rfm_scaled
```

```
array([[ 2.32202285, -0.39205597, -0.23041952],
       [-0.89373323,  0.39329343,  0.29405454],
       [-0.1691956 , -0.26552745, -0.01171748],
       ...,
       [-0.83418219, -0.34406239, -0.20892947],
       [-0.87388289,  2.74497857,  0.01849636],
       [-0.48680114, -0.09536842, -0.00684511]])
```

Figura 31 - Padronização dos Dados Numéricos.

Fonte: O autor (2025).

Aplicando o K-Means para Agrupar os Clientes

Com os dados já padronizados, chegou a hora de aplicar o algoritmo K-Means, para explorar agrupamentos baseados em características similares, visualizando padrões no comportamento dos clientes (Figura 32).

O **K-means** é um algoritmo que procura organizar um conjunto de dados em grupos (clusters), de forma que os pontos dentro de um mesmo grupo sejam mais parecidos entre si do que com os de outros grupos.

Ele funciona assim:

1. **Início**: os pontos de dados são divididos aleatoriamente em um número kkk de grupos.
2. **Cálculo dos centros**: para cada grupo, calcula-se um ponto central (chamado centróide), que é a média de todos os pontos que pertencem a esse grupo.
3. **Atribuição de pontos**: cada ponto do conjunto de dados é comparado com os centróides e passa a pertencer ao grupo cujo centro está mais próximo dele.
4. **Atualização dos centróides**: depois da nova atribuição, os centróides de cada grupo são recalculados.
5. **Repetição**: os passos 3 e 4 se repetem até que os centróides praticamente não mudem mais ou até atingir um número máximo de repetições.

```
[ ]  kmeans = KMeans(n_clusters = 4, random_state = 42)
     kmeans.fit(rfm_scaled)
     rfm['Cluster'] = kmeans.labels_
```

Figura 32 - Aplicação do Algoritmo K-Means.

Fonte: O autor (2025).

```
[ ] rfm
```

	CustomerID	Recency	Frequency	MonetaryValue	Recency_score	Frequency_score	rfm_segment	Cluster
0	12346.0	326	2	0.00	1	1	Clientes Inativos	1
1	12347.0	2	182	4310.00	5	5	Clientes Campeões	0
2	12348.0	75	31	1797.24	2	3	Clientes em Risco	0
3	12349.0	19	73	1757.55	4	4	Clientes Frequentes	0
4	12350.0	310	17	334.40	1	2	Clientes Inativos	1
...	...	...	...	...	...	...	...	...
4367	18280.0	278	10	180.60	1	1	Clientes Inativos	1
4368	18281.0	181	7	80.82	1	1	Clientes Inativos	1
4369	18282.0	8	13	176.60	5	1	Novo Cliente	0
4370	18283.0	4	721	2045.53	5	5	Clientes Campeões	0
4371	18287.0	43	70	1837.28	3	4	Clientes Frequentes	0

4372 rows × 8 columns

Figura 33 - Resultados do Agrupamento K-Means no DataFrame RFM.

Fonte: O autor (2025).

Na Figura 33, é possível visualizar o resultado desse processo, com cada cliente recebendo um rótulo numérico correspondente ao seu grupo.

Os clusters encontrados pelo K-Means ajudam a evidenciar que há grupos bem distintos dentro da base, facilitando a personalização de estratégias de relacionamento com cada segmento.

Criando um Novo DataFrame com os Principais Indicadores RFM

Agora vamos criar um novo DataFrame chamado new_rfm, contendo apenas as colunas mais relevantes para análise: Recency, Frequency, MonetaryValue e o segmento RFM que atribuímos anteriormente (Figura 34).

```
[ ]  new_rfm = rfm[["Recency", "Frequency", "MonetaryValue", "rfm_segment"]]
```

Figura 34 - Criação de Novo DataFrame com Colunas RFM.

Fonte: O autor (2025).

Ajustando o Índice do DataFrame

Antes de visualizar as primeiras linhas, vamos garantir que o índice do new_rfm esteja no formato correto.

```
new_rfm.index = new_rfm.index.astype(int)
new_rfm.head()
```

	Recency	Frequency	MonetaryValue	rfm_segment
0	326	2	0.00	Clientes Inativos
1	2	182	4310.00	Clientes Campeões
2	75	31	1797.24	Clientes em Risco
3	19	73	1757.55	Clientes Frequentes
4	310	17	334.40	Clientes Inativos

Figura 35 - Exibição do Novo DataFrame RFM.

Fonte: O autor (2025).

O código apresentado na Figura 35 realiza duas ações importantes: primeiro, ele ajusta o índice do DataFrame para o formato inteiro, garantindo que sua estrutura esteja organizada e adequada para manipulação. Em seguida, exibe as primeiras linhas do novo DataFrame RFM já estruturado, permitindo uma verificação inicial dos dados.

Visualizando os Segmentos de Clientes com Scatterplot

Agora vamos criar um gráfico de dispersão (scatterplot) para observar como os clientes estão distribuídos visualmente de acordo com seus valores de Recency e Frequency, usando a cor para representar os segmentos RFM (Figura 36).

```
plt.figure(figsize = (10, 6))
sns.scatterplot(x = 'Recency', y = 'Frequency', hue = 'rfm_segment', data = new_rfm, palette = 'viridis')
plt.title('Customer Cluster based on Recency and Frequency')
plt.show()
```

Figura 36 - Código para Gráfico de Dispersão de Clusters RFM.

Fonte: O autor (2025).

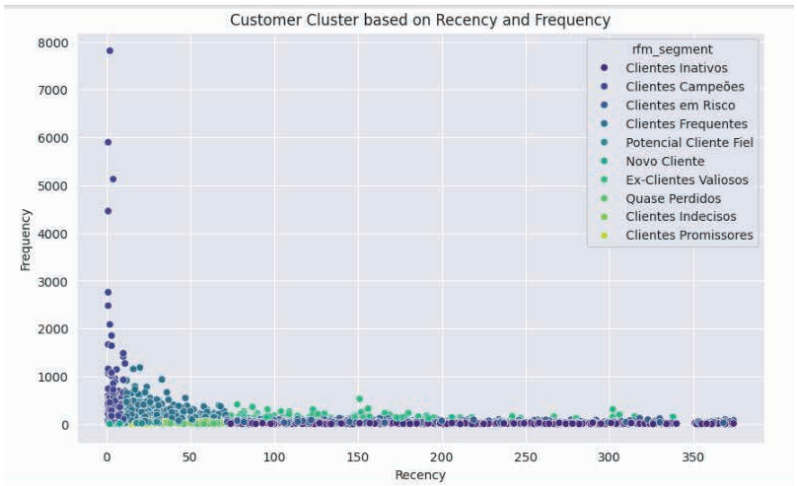

Figura 37 - Gráfico de Dispersão dos Clusters de Clientes.

Fonte: O autor (2025).

Ao observarmos a Figura 37, que apresenta o gráfico de dispersão dos clusters de clientes com base nas variáveis Recency e Frequency, é possível perceber uma concentração significativa de pontos na região inferior esquerda do gráfico. Essa concentração indica que grande parte dos clientes realizou poucas compras (baixa Frequency) e comprou há bastante tempo (alta Recency), o que é típico em bases de clientes onde apenas uma pequena parcela mantém um relacionamento mais ativo com a empresa.

À medida que avançamos no eixo da frequência, notamos a presença de alguns clientes com número muito elevado de compras, embora em menor quantidade. Esses pontos mais afastados representam consumidores altamente engajados, associados a segmentos como Clientes Frequentes ou Clientes Campeões.

As cores aplicadas aos pontos ilustram a divisão dos clientes em diferentes segmentos RFM, permitindo visualizar claramente como cada grupo se distribui em relação ao tempo desde a última compra e à quantidade de transações realizadas.

Gráfico de Barras: Distribuição dos Segmentos RFM

Com os segmentos já atribuídos aos clientes, essa etapa cria um gráfico de barras interativo usando o Plotly, para vermos a quantidade de clientes em cada grupo de forma clara e visual (Figura 38).

```
segments = new_rfm['rfm_segment'].value_counts()

fig = px.bar(
     x = segments.index,
     y = segments.values,
     color = segments.index,
     text = segments.values,
     title = "RFM Segments"
)
fig.update_layout(
    xaxis_title="Segment",
    yaxis_title="Count",
    font=dict(size=15, family="Arial"),
    title_font=dict(size=20, family="Arial")
)
fig.show()
```

Figura 38 - Código Python para Visualização Quantitativa dos Segmentos de Clientes.

Fonte: O autor (2025).

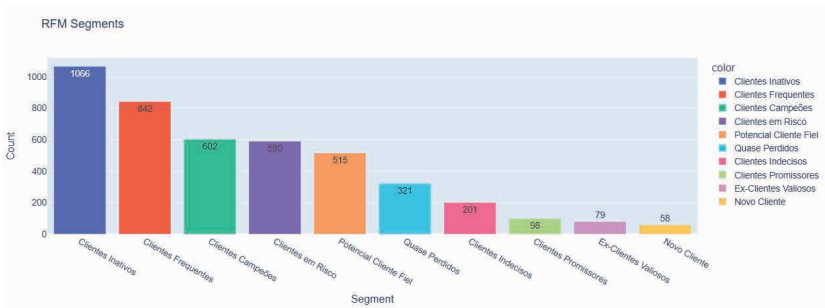

Figura 39 - Distribuição dos Segmentos RFM.

Fonte: O autor (2025).

A Figura 39 mostra o gráfico de barras que apresenta a distribuição da base de clientes por segmento. Observa-se que a maior parte está concentrada em Clientes

Inativos, o que sinaliza a necessidade de ações de reativação. Também é possível identificar grupos relevantes de Clientes Frequentes e Clientes Campeões, que representam os consumidores mais valiosos e devem ser priorizados em estratégias de fidelização. Já os segmentos de Clientes em Risco e Quase Perdidos indicam oportunidades de retenção que podem evitar que esses clientes se tornem totalmente inativos.

Visualizando a Proporção de Clientes por Segmento (Gráfico de Pizza)

Aqui vamos criar um gráfico de pizza com os dados de quantidade de clientes por segmento (Figura 40). Esse tipo de visualização é ideal para entender a participação percentual de cada grupo no total de clientes.

```
[ ]  plt.figure(figsize = (10, 8), )
     explode = (0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0)

     segments.plot(
         kind = 'pie',
         color=segments.index,
         explode = explode,
         autopct = '%1.2f%%')
     plt.axis('equal')
     plt.legend(labels = segments.index, loc = "best")
     plt.show()
```

Figura 40 - Código para Gráfico de Pizza de Segmentos RFM

Fonte: O autor (2025).

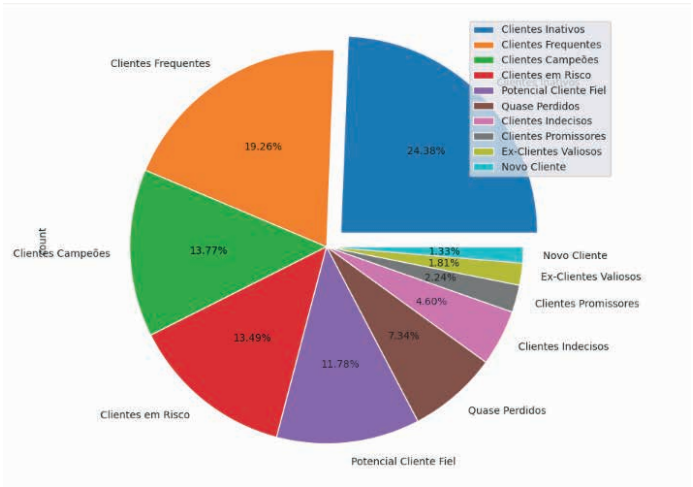

Figura 41 - Gráfico de Pizza da Distribuição de Segmentos RFM.

Fonte: O autor (2025).

A Figura 41 representa um gráfico de pizza que mostra a participação percentual de cada segmento no total de clientes. Nota-se que a maior proporção está concentrada em Clientes Inativos, enquanto Clientes Frequentes e Clientes Campeões também representam parcelas relevantes da base. Já os grupos de Clientes em Risco, Quase Perdidos e Indecisos aparecem em menor proporção, mas ainda assim indicam oportunidades para ações de retenção.

Comparando a Frequência Média por Segmento RFM

O gráfico de barras usa o seaborn para mostrar quais segmentos de clientes compram com mais frequência (Figura 42). Isso nos ajuda a priorizar ações para quem já demonstra mais engajamento.

```
[ ]  sns.set_style("darkgrid")
     colors = sns.color_palette("dark")

     # Create the plot
     plt.figure(figsize=(15, 7))
     sns.barplot(x = "rfm_segment", y = "Frequency", data = new_rfm, palette=colors)
     plt.title("Customer Segments by Frequency", color='black', fontsize=16, fontweight='bold')
     plt.xlabel("Segment", color='black', fontsize=14)
     plt.ylabel("Frequency", color='black', fontsize=14)
     plt.xticks(rotation=45, color='black', fontsize=12)
     plt.yticks(color='black', fontsize=12)
     plt.show()
```

Figura 42- Código para Gráfico de Barras da Frequência por Segmento.

Fonte: O autor (2025).

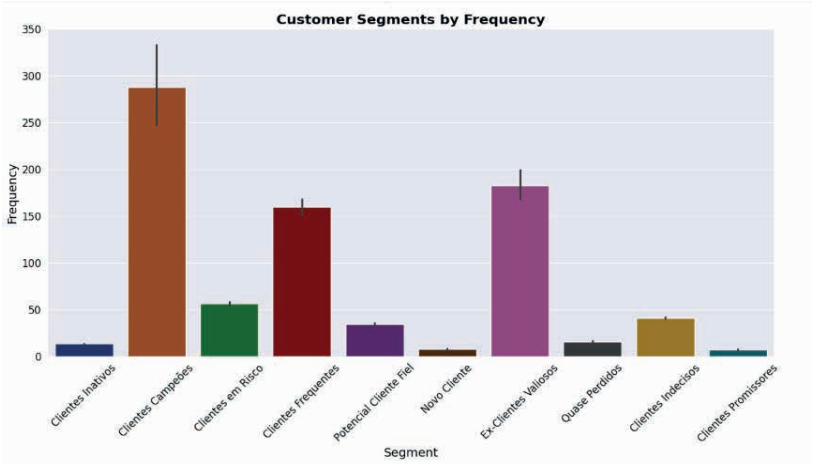

Figura 43 - Frequência por Segmentos de Clientes.

Fonte: O autor (2025).

A Figura 43 representa um gráfico de barras que apresenta a frequência média de compras por segmento de clientes. É possível observar que os grupos de Clientes Campeões, Clientes Frequentes e Ex-Clientes Valiosos se destacam com maior nível de atividade, enquanto segmentos como Clientes Inativos, Promissores e Indecisos aparecem com menor participação, indicando baixo engajamento.

Estatísticas por Segmento: Entendendo o Comportamento de Cada Grupo

Nesta etapa, vamos agrupar os clientes por segmento RFM e calcular estatísticas relevantes para cada métrica: Recência, Frequência e Valor Monetário. O objetivo é obter uma visão geral do comportamento de cada grupo, identificando padrões de compra e potenciais oportunidades de atuação.

```
[ ] new_rfm[["rfm_segment","Recency", "Frequency", "MonetaryValue"]].groupby("rfm_segment").agg(["mean", "count","sum"])
```

	Recency			Frequency			MonetaryValue		
	mean	count	sum	mean	count	sum	mean	count	sum
rfm_segment									
Clientes Campeões	5.280731	602	3179	287.362126	602	172992	6903.407957	602	4155851.590
Clientes Frequentes	32.276722	842	27177	159.731591	842	134494	2575.195285	842	2168314.430
Clientes Inativos	212.132270	1066	226133	13.332083	1066	14212	368.246953	1066	392551.252
Clientes Indecisos	51.213930	201	10294	41.079602	201	8257	799.003985	201	160599.801
Clientes Promissores	22.714286	98	2226	7.122449	98	698	418.561735	98	41019.050
Clientes em Risco	164.706780	590	97177	56.532203	590	33354	949.998797	590	560499.290
Ex-Clientes Valiosos	140.620253	79	11109	182.607595	79	14426	2359.026595	79	186363.101
Novo Cliente	6.689655	58	388	7.517241	58	436	675.133103	58	39157.720
Potencial Cliente Fiel	15.491262	515	7978	34.339806	515	17685	858.925670	515	442346.720
Quase Perdidos	52.239875	321	16769	15.732087	321	5050	410.643209	321	131816.470

Figura 44 - Resumo Estatístico dos Segmentos RFM.

Fonte: O autor (2025).

A tabela mostrada na Figura 44 apresenta as médias de recência, frequência e valor monetário para cada segmento de clientes. Essa visualização é útil para identificar quais grupos concentram maior engajamento e valor financeiro. Observa-se que os Clientes Campeões e Frequentes se destacam como os mais valiosos para a manutenção da receita, enquanto os Inativos e em Risco indicam oportunidades de reativação. Já os Potenciais Clientes Fiéis e Novos Clientes mostram bom potencial para ações de fidelização e crescimento futuro.

Simulando Gasto Futuro

Agora vamos gerar uma nova variável chamada FutureSpending, que simula quanto cada cliente pode vir a gastar futuramente. Essa simulação será baseada no comportamento passado (RFM), usando pesos diferentes para cada métrica (Figura 45).

```
[ ]  import numpy as np
     np.random.seed(42)

     rec_norm = 1 - (rfm["Recency"] / rfm["Recency"].max())
     freq_norm = rfm["Frequency"] / rfm["Frequency"].max()
     monet_norm = rfm["MonetaryValue"] / rfm["MonetaryValue"].max()

     rfm["FutureSpending"] = (
         0.3 * rec_norm +
         0.3 * freq_norm +
         0.4 * monet_norm
     ) * np.random.uniform(0.8, 1.2, len(rfm)) * 3000

     rfm["FutureSpending"] = rfm["FutureSpending"].round(2)
```

Figura 45 - Criação da Variável 'FutureSpending'.

Fonte: O autor (2025).

Separando os Dados para Treinamento e Teste

Antes de treinar o modelo, precisamos dividir o conjunto de dados em duas partes (Figura 46):

Treinamento (80%) – usada para ensinar o modelo.

Teste (20%) – usada para avaliar a performance do modelo em dados que ele nunca viu.

```
[ ]  from sklearn.model_selection import train_test_split

     features = rfm[['Recency', 'Frequency', 'MonetaryValue']]
     target = rfm['FutureSpending']

     X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

Figura 46 - Divisão do Conjunto de Dados em Treino e Teste.

Fonte: O autor (2025).

Treinando um Modelo de Árvore de Decisão

Agora vamos usar o algoritmo Decision Tree Regressor para prever o gasto futuro dos clientes com base nos dados RFM (Figura 47).

```
[ ]  from sklearn.tree import DecisionTreeRegressor
     from sklearn.metrics import mean_squared_error, r2_score

     dtr = DecisionTreeRegressor(random_state=42)
     dtr.fit(X_train, y_train)

     # Fazendo previsões
     dtr_pred = dtr.predict(X_test)
```

Figura 47 - Treinamento e Previsão com DecisionTreeRegressor.

Fonte: O autor (2025).

Avaliando o Desempenho do Modelo

Depois de treinar e fazer previsões com a Árvore de Decisão, vamos avaliar a performance usando duas métricas clássicas

Mean Squared Error (MSE): quanto menor, melhor. Representa o erro médio ao quadrado entre os valores previstos e os reais.

R^2 Score: varia entre 0 e 1 (ou pode ser negativo). Quanto mais próximo de 1, melhor o modelo explica a variabilidade dos dados.

```
[ ]  # Avaliando o modelo
     mse = mean_squared_error(y_test, dtr_pred)
     r2 = r2_score(y_test, dtr_pred)

     print(f"Mean Squared Error: {mse:.2f}")
     print(f"R² Score: {r2:.4f}")
```

```
Mean Squared Error: 15720.41
R² Score: 0.7914
```

Figura 48 - Avaliação do Modelo de Regressão.

Fonte: O autor (2025).

Pela Figura 48 avaliamos o desempenho do modelo de regressão a partir das métricas Mean Squared Error (MSE) e R^2 Score. Essa etapa é importante para entender o quanto o modelo consegue explicar a variação dos dados e o nível de erro nas

previsões. Os resultados indicam um bom poder de explicação e erros relativamente baixos, mostrando que o modelo é uma base consistente para previsões futuras.

Visualizando o Desempenho do Modelo

Depois de avaliar as métricas numéricas, vamos agora comparar graficamente os valores reais com os previstos pelo modelo (Figura 49 e 50).

O modelo de Árvore de Decisão apresentou um bom desempenho, com R^2 de aproximadamente 0.79, indicando que conseguimos capturar boa parte da variação nos gastos futuros com base nos padrões comportamentais dos clientes.

A visualização Real vs Previsto mostrou que o modelo consegue manter uma boa proximidade entre os valores reais e estimados.

A grande vantagem de unir clusterização e modelos preditivos é que conseguimos:

- Entender quem são os clientes (via RFM e segmentação).
- E prever o quanto eles tendem a gastar no futuro (via machine learning).
- Com isso, é possível criar estratégias de marketing personalizadas, investir nos grupos certos e até prever o retorno esperado de cada segmento.

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

     results_df = pd.DataFrame({'Real': y_test, 'Previsto': dtr_pred})
     plt.figure(figsize=(10, 6))
     sns.scatterplot(x='Real', y='Previsto', data=results_df, color='royalblue', alpha=0.6)
     plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r')
     plt.title('Real x Previsto - Future Spending')
     plt.xlabel('Valor Real')
     plt.ylabel('Valor Previsto')
     plt.show()
```

Figura 49 - Código para Gráfico Real x Previsto.

Fonte: O autor (2025).

Figura 50 - Gráfico de Dispersão de Valores Reais vs. Previstos.

Fonte: O autor (2025).

CONCLUSÃO

O avanço das tecnologias da informação e comunicação transformou profundamente a forma como os dados são coletados, processados e utilizados no apoio à tomada de decisão. Nesse contexto, o tutorial atingiu seu propósito central ao demonstrar, por meio de um guia prático, como técnicas de análise de dados podem ser aplicadas ao setor varejista de maneira acessível e didática. A utilização de ferramentas como Python, Google Colab e bibliotecas especializadas, como Pandas, Matplotlib, Seaborn e Scikit-Learn, permitiu não apenas a exploração estruturada de grandes volumes de informações, mas também a construção de modelos capazes de gerar conhecimento estratégico.

Conclui-se, portanto, que a integração entre análise de dados, segmentação inteligente e modelos preditivos constitui um suporte valioso para decisões baseadas em evidências no varejo. O guia elaborado contribui para democratizar esse conhecimento técnico, tornando mais acessível o uso de ferramentas analíticas e fortalecendo a cultura de decisões orientadas por dados. Espera-se que este trabalho estimule profissionais e estudantes a explorar novas possibilidades dentro da ciência de dados, ampliando sua capacidade de transformar informações em insights estratégicos para o negócio.

REFERÊNCIAS

AMARAL, Fernando. **Introdução à ciência de dados: mineração de dados e big data**. 1. ed. rev. [S. l.]: Starlin Altas book, 2016.

Bertolini ,Cristiano.Linguagem de programação I [recurso eletrônico] – 1. ed. – Santa Maria, RS : UFSM, NTE, 2019.

DA SILVA, Martony Demes. **Aplicação da Ferramenta Google Colaboratory para o Ensino da Linguagem Python**. In: ESCOLA REGIONAL DE ENGENHARIA DE SOFTWARE (ERES), 4. , 2020, Evento Online. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2020 . p. 67-76.

GOUVEIA, Fágner Sousa, **O MARKETING E SUA IMPORTANCIA PARA O VAREJO**. REVISTA CIENTÍFICA DO ITPAC, v. 4, p. 12, 01 2011. Disponível em: https://www.yumpu.com/pt/document/read/12578086/o-marketing-e-sua-importancia-para-o-varejo-itpac. Acesso em: 01 out. 2024.

Guardelli, Enrico. **IA no Varejo: Inovação, Eficiência e Personalização**, 2024

IBM. *What is a dataset?* IBM Cloud Education, 2020. Disponível em: https://www.ibm.com/cloud/learn/dataset. Acesso em: 29 jul. 2025.

Kalinowski, Marcos. **Engenharia de Software para Ciência de Dados** Um guia de boas práticas com ênfase na construção de sistemas de Machine Learning em Python, 2023.

KUROSE, James F.; ROSS, Keith W. *Redes de Computadores e a Internet: uma abordagem top-down*. 6. ed. São Paulo: Pearson Prentice Hall, 2014.

LAMBERT, Kenneth A. **Fundamentos de Python: estruturas de dados**. São Paulo: Cengage Learning Brasil, 2022. *E-book.* p.iv. ISBN 9786555584288. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9786555584288/. Acesso em: 07 out. 2024.

LAMBERT, Kenneth A. **Fundamentos de Python: primeiros programas**. São Paulo: Cengage Learning Brasil, 2022. E-book. p.21. ISBN 9786555584301. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9786555584301/. Acesso em: 07 out. 2024.

MCKINNEY, Wes. *Python para análise de dados: tratamento de dados com Pandas, NumPy e IPython*. 2. ed. São Paulo: Novatec, 2018.

MUELLER, John P.; MASSARON, Luca. **Python Para Data Science Para Leigos**. Rio de Janeiro: Editora Alta Books, 2020. E-book. p.19. ISBN 9786555201512. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9786555201512/. Acesso em: 07 out. 2024.

Okada, Renan Shindi. **MANIPULADOR DE ARQUIVOS CSV NA LINGUAGEM JAVA, MONGODB E DOCKER.** 2023.

OLIVEIRA, Hugo Santos. CSVValidation: uma ferramenta para validação de arquivos CSV a partir de metadados. 2015. Dissertação de Mestrado. Universidade Federal de Pernambuco.

Parente, Juracy. **Varejo no Brasil: Gestão e Estratégia** / Juracy Parente, Edgard Barki. – 2. ed. – São Paulo: Atlas, 2017.

PIMENTEL, João Felipe et al. Ciência de dados com reprodutibilidade usando jupyter. **Jornada de Atualização em Informática**, v. 2021, p. 11-59, 2021.

SHARDA, RAMESH. **Business Intelligence e Análise de Dados para Gestão do Negócio** - 4.ed. 2019.

Shen, H. Interactive notebooks: Sharing the code. Nature, 515(7525):151–152. (2014).