FUNDAÇÃO ESCOLA DE COMÉRCIO ÁLVARES PENTEADO

FECAP

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

João Pedro Brosselin de Albuquerque Souza - 24026155

Marcella Santana Gonçalves Diniz Rocha - 24025750

Thays Helyda da Silva Pontes - 24026610

Gustavo de Souza Castro - 20021558

Inteligência Artificial e Machine Learning

Fidelize

São Paulo - SP

2025

1.0 Introdução

O Projeto Fidelize é um projeto acadêmico desenvolvido por alunos do curso de Análise e Desenvolvimento de Sistemas (ADS) da FECAP, no 4º semestre de 2025, como parte do desafio de criar um Dashboard Interativo para a startup Cannoli, uma foodtech que atua na gestão de restaurantes e negócios do setor gastronômico. O projeto se insere no contexto de transformação digital da Cannoli, que busca integrar inteligência de dados aos seus serviços de CRM, automação de engajamento, cardápio digital e delivery próprio.

O objetivo geral do projeto é desenvolver um dashboard interativo, responsivo e inteligente, que permita a análise de dados operacionais e estratégicos da plataforma Cannoli, voltado tanto para administradores da empresa quanto para clientes parceiros (restaurantes e estabelecimentos que utilizam o sistema). O dashboard será capaz de consolidar informações sobre campanhas de marketing, cadastros de clientes e histórico de pedidos, fornecendo insights e previsões para apoiar decisões relacionadas à fidelização e retenção de consumidores – oriem do nome do projeto: Fidelize.

O problema central identificado está relacionado à dificuldade da Cannoli em analisar grandes volumes de dados de campanhas e comportamento de clientes de forma integrada. Atualmente, os administradores e gestores precisam consultar múltiplas fontes, o que torna o processo lento e pouco estratégico. Nesse sentido, o uso de técnicas de Inteligência Artificial e Machine Learning é essencial para transformar dados brutos em informações consultivas, permitindo a previsão de resultados de campanhas, a identificação de padrões de consumo e a segmentação inteligente de clientes.

Como resultado esperado, o Dashboard Fidelize atuará como uma ferramenta de apoio à decisão, com visualizações interativas, alertas automáticos e recomendações personalizadas, utilizando modelos preditivos baseados em aprendizado de máquina. O sistema fornecerá análises como: desempenho de campanhas, taxa de resposta de clientes, valor médio de pedidos, e segmentação de públicos-alvo com maior propensão à recompra.

Para o desenvolvimento do modelo e das análises, foram utilizadas quatro bases de dados extraídas da plataforma Cannoli:

1. campaign_queue (5.000 registros) — contém os envios e respostas das campanhas de marketing, com colunas que permitem compreender o desempenho operacional e o engajamento dos clientes. As variáveis *campaignId, customerId, status, sendAt e response* são fundamentais para mensurar o sucesso de cada campanha, possibilitando análises de taxa de entrega, taxa de resposta e identificação de padrões de engajamento.

2. campaign (2.000 registros) — reúne informações descritivas e cadastrais das campanhas criadas, como *type, badge, status, createdAt e isDefault*. Esses atributos permitem classificar campanhas por tipo (como *winback* ou *loyalty*), identificar tendências de desempenho e comparar resultados entre campanhas padrão e personalizadas, subsidiando modelos preditivos sobre o sucesso de campanhas futuras.

3. customer (1.000 registros) — representa a base de clientes cadastrados, incluindo dados demográficos, de engajamento e de status de conta. As colunas *gender, dateOfBirth, status, isEnriched e createdAt* permitem traçar perfis de consumo e acompanhar o ciclo de vida dos clientes na plataforma. Esses dados são especialmente relevantes para análises de segmentação de público (clustering) e previsão de churn (abandono).

4. order (2.000 registros) — registra o histórico completo de pedidos realizados na plataforma, com informações detalhadas sobre comportamento de compra. As variáveis *customer, salesChannel, orderType, status, totalAmount e createdAt* permitem entender preferências de canal de venda, sazonalidade de pedidos e valores médios de consumo, essenciais para análises de rentabilidade e previsões de demanda.

Com esse conjunto de dados integrados, o projeto Fidelize propõe a aplicação de técnicas de Machine Learning para gerar análises exploratórias, segmentações e previsões que darão suporte à criação de um dashboard inteligente, orientado a dados e voltado à fidelização de clientes e otimização das campanhas de marketing.

2.0 Referencial Teórico

O desenvolvimento do rojeto Fidelize leva em consideração os conceitos centrais de Inteligência Artificial (IA) e Aprendizado de Máquina (Machine Learning), aplicados à análise de dados empresariais e comportamentais. Essas técnicas são utilizadas para detectar padrões, prever resultados e gerar recomendações automáticas a partir de grandes volumes de dados heterogêneos, como os provenientes das campanhas, clientes e pedidos da plataforma Cannoli.

A Detecção de Anomalias é uma técnica de IA reativa utilizada para identificar comportamentos que se desviam do padrão esperado em um conjunto de dados. No contexto do Fidelize, ela pode ser aplicada para detectar variações incomuns em taxas de resposta de campanhas, valores de pedidos ou engajamento de clientes. Esses métodos são particularmente úteis em sistemas de monitoramento e recomendação, pois permitem sinalizar eventos inesperados ou potenciais falhas antes que causem impacto operacional significativo.

O Aprendizado Supervisionado representa uma das abordagens mais comuns do Machine Learning e envolve o treinamento de modelos a partir de dados rotulados, de modo que o algoritmo aprenda uma relação entre entradas e saídas conhecidas. Essa técnica é empregada em dois formatos principais: regressão, quando o objetivo é prever valores contínuos (como o valor de pedidos), e classificação, quando se busca atribuir categorias (como identificar se um cliente possui a tendência ou não a responder uma campanha).

As Heurísticas de Busca, como a Busca Gulosa, são métodos que buscam soluções aproximadas para problemas complexos por meio da escolha localmente ótima em cada etapa. No contexto da IA aplicada a sistemas de recomendação e decisão, heurísticas podem ser utilizadas para selecionar campanhas mais promissoras ou priorizar clientes com maior potencial de fidelização, otimizando o desempenho computacional e reduzindo o tempo de processamento.

Por fim, a Segmentação de Dados é um processo essencial para agrupar entidades semelhantes com base em suas características. Técnicas como o K-Means permitem identificar grupos (clusters) de clientes com padrões de comportamento similares, enquanto a Análise de Componentes Principaisreduz a dimensionalidade dos dados, facilitando a visualização e interpretação de informações complexas. Essa combinação de métodos auxilia na descoberta de perfis de consumo, no direcionamento de campanhas e na personalização das estratégias de fidelização.

Em conjunto, esses conceitos formam a base teórica para as etapas de análise, modelagem e previsão do projeto Fidelize, permitindo transformar dados brutos em insights estratégicos para apoio à decisão e fortalecimento da relação entre a Cannoli e seus clientes.

3.0 Metodologia

A metodologia do projeto Fidelize foi estruturada como um pipeline de processamento e análise de dados, com o objetivo de integrar, limpar e preparar as informações provenientes das bases da plataforma Cannoli para aplicação de técnicas de Inteligência Artificial e Machine Learning.

3.1 Coleta e leitura (dados)

A primeira etapa consistiu na coleta e leitura das quatro bases de dados:

1. CampaignQueue — informações sobre o envio e status de campanhas de marketing;

2. Campaign — dados cadastrais e descritivos das campanhas criadas;

3. Customer — cadastro de clientes, incluindo dados demográficos e de engajamento;

4. Order — histórico de pedidos, com detalhes sobre canais de venda, valores e status.

Esses arquivos foram disponibilizados no formato CSV com separador ;, o que exigiu cuidados durante a leitura. Além disso, observou-se inconsistência no formato de datas e campos com espaços e acentuação, o que demandou normalização posterior.

Figura 1 – instalação das bibliotecas essenciais no ambiente Google Colab.

```python
# CÉLULA 2 — Import e configuração inicial
import os
from pathlib import Path
import pandas as pd
import numpy as np
import json
import matplotlib.pyplot as plt
import joblib

pd.set_option('display.max_columns', 200)
DATA_DIR = '/content'  # ajuste se montou Drive: '/content/drive/MyDrive/...'
```

Figura 2 – instalação e importaão das bibliotecas essenciais no ambiente Google Colab.

Para garantir a qualidade dos dados, foi criada uma função de leitura robusta para arquivos separados por ponto e vírgula, com normalização automática dos nomes das colunas. Essa função foi definida na CÉLULA 3, conforme ilustrado a seguir:

```python
# CÉLULA 3 — Função robusta para carregar CSV com ; e normalizar colunas
def load_csv_semicolon(path):
    path = Path(path)
    if not path.exists():
        raise FileNotFoundError(f"{path} not found")
    df = pd.read_csv(path, sep=';', encoding='utf-8', low_memory=False)
    # normaliza nomes
    df.columns = [str(c).strip().lower().replace(' ', '_') for c in df.columns]
    return df

# Caminhos (ajuste nomes de arquivo se necessário)
files = {
    'campaign_queue': Path(DATA_DIR) / 'CampaignQueue_semicolon.csv',
    'campaign': Path(DATA_DIR) / 'Campaign_semicolon.csv',
    'customer': Path(DATA_DIR) / 'Customer_semicolon.csv',
    'order': Path(DATA_DIR) / 'Order_semicolon.csv',
}

# Carrega
dfs = {}
for k, p in files.items():
    try:
        dfs[k] = load_csv_semicolon(p)
        print(f"Carregado {k}: shape {dfs[k].shape}")
    except Exception as e:
        print(f"Falha ao carregar {k}: {e}")
        dfs[k] = None
```

Figura 3 – função utilizada para leitura e padronização de arquivos CSV com separador ponto e vírgula.

Os caminhos dos arquivos foram definidos e a leitura foi automatizada por meio de um dicionário. Em seguida, foi exibido o número de registros e colunas de cada tabela.

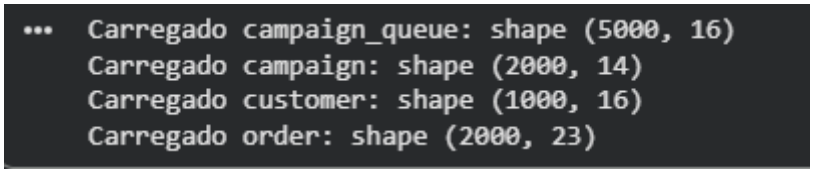


Figura 4 – confirmação da leitura bem-sucedida das quatro bases de dados no ambiente.

3.2 Tratamento e Engenharia de features

Após a leitura e normalização inicial das tabelas, realizou-se o tratamento de colunas, conversão de tipos e criação de novas variáveis derivadas (features). Essa etapa teve como objetivo garantir consistência temporal, padronizar campos-chave entre as bases e gerar indicadores úteis para os modelos de aprendizado de máquina.

O tratamento envolveu duas funções principais, desenvolvidas na CÉLULA 4 do pipeline:

1. guess_cols(df): identifica automaticamente nomes de colunas-chave (como customer_id, campaign_id, store_id, sent_at, status), mesmo que venham com grafias diferentes;

2. convert_dates_and_numbers(df): converte colunas de datas e números que estavam armazenadas como texto em formatos padronizados do Python (datetime64, float).

Essas funções permitiram padronizar colunas de todas as quatro tabelas –campaign_queue, campaign, customer e order – mesmo quando os arquivos apresentavam variações de nomeação e formatação.

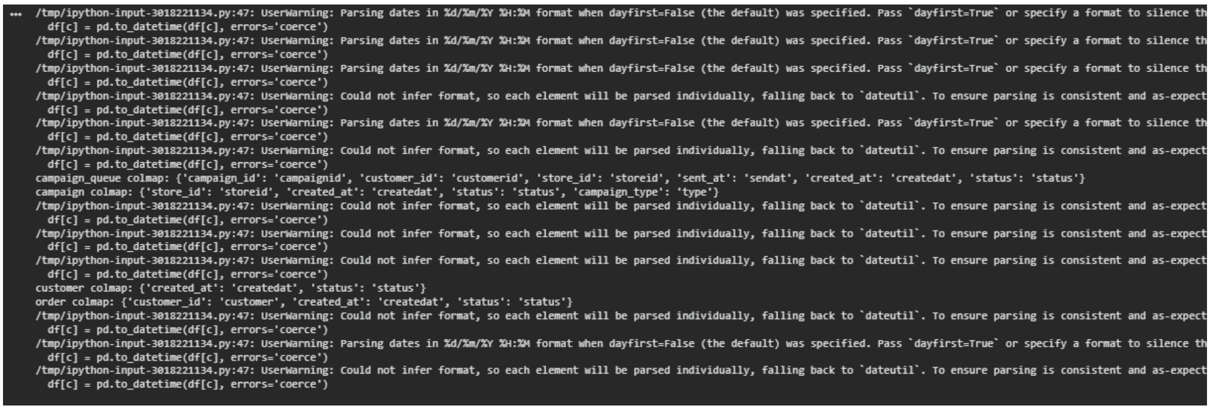
```python
# CÉLULA 4 — Heurísticas para colunas chave e conversão de datas/números
def guess_cols(df):
    cols = set(df.columns)
    # possíveis nomes
    colmap = {}
    # campaign id
    for c in ['campaignid','campaign_id','campaign']:
        if c in cols:
            colmap['campaign_id'] = c
            break
    for c in ['customerid','customer_id','customer','clientid']:
        if c in cols:
            colmap['customer_id'] = c
            break
    for c in ['storeid','store_id','store']:
        if c in cols:
            colmap['store_id'] = c
            break
    # data/criado/enviado
    for c in ['sent_at','sentat','sent','sent_date','sendat','send_at','sentdate']:
        if c in cols:
            colmap['sent_at'] = c
            break
    for c in ['created_at','createdat','created','created_date','createdate','createdat']:
        if c in cols:
            colmap['created_at'] = c
            break
    # status e total
    for c in ['status','queue_status']:
        if c in cols:
            colmap['status'] = c
            break
    for c in ['total','total_value','amount','order_total','total_price']:
        if c in cols:
            colmap['total'] = c
            break
```

Figura 5 – funções utilizadas para inferência de colunas e conversão de tipos.

```python
    for c in ['type','campaign_type','kind']:
        if c in cols:
            colmap['campaign_type'] = c
            break
    return colmap

def convert_dates_and_numbers(df):
    for c in df.columns:
        if any(k in c for k in ['date','at','time','created','sent']) and df[c].dtype == object:
            try:
                df[c] = pd.to_datetime(df[c], errors='coerce')
            except:
                pass
        if df[c].dtype == object and df[c].str.replace(',', '.', regex=False).str.replace('.', '', regex=False).str.isnumeric().any():
            # tentativa de converter números com ',' decimal
            try:
                df[c] = pd.to_numeric(df[c].str.replace('.', '').str.replace(',', '.'), errors='coerce')
            except:
                pass
    return df

# aplica às 4 tabelas
colmap_master = {}
for k in dfs:
    df = dfs[k]
    if df is None: continue
    df = convert_dates_and_numbers(df)
    dfs[k] = df
    colmap_master[k] = guess_cols(df)
    print(k, "colmap:", colmap_master[k])
```

Figura 6 – funções utilizadas para inferência de colunas e conversão de tipos.

As funções foram aplicadas às quatro tabelas, resultando em um mapeamento automático de colunas (colmap_master) e conversão de tipos inconsistentes. O resultado confirmou a padronização das principais variáveis:



```
/tmp/ipython-input-3018221134.py:47: UserWarning: Parsing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence th
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Parsing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence th
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Parsing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence th
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expect
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Parsing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence th
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expect
  df[c] = pd.to_datetime(df[c], errors='coerce')
campaign_queue colmap: {'campaign_id': 'campaignid', 'customer_id': 'customerid', 'store_id': 'storeid', 'sent_at': 'sendat', 'created_at': 'createdat', 'status': 'status'}
campaign colmap: {'store_id': 'storeid', 'created_at': 'createdat', 'status': 'status', 'campaign_type': 'type'}
/tmp/ipython-input-3018221134.py:47: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expect
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expect
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expect
  df[c] = pd.to_datetime(df[c], errors='coerce')
customer colmap: {'created_at': 'createdat', 'status': 'status'}
order colmap: {'customer_id': 'customer', 'created_at': 'createdat', 'status': 'status'}
/tmp/ipython-input-3018221134.py:47: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expect
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Parsing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence th
  df[c] = pd.to_datetime(df[c], errors='coerce')
/tmp/ipython-input-3018221134.py:47: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expect
  df[c] = pd.to_datetime(df[c], errors='coerce')
```

Figura 7 – mapeamento automático das colunas padronizadas para cada tabela.

{campaign_queue colmap: {'campaign_id': 'campaignid', 'customer_id': 'customerid', 'store_id': 'storeid', 'sent_at': 'sendat', 'created_at': 'createdat', 'status': 'status'}

campaign colmap: {'store_id': 'storeid', 'created_at': 'createdat', 'status': 'status', 'campaign_type': 'type'}

customer colmap: {'created_at': 'createdat', 'status': 'status'}

order colmap: {'customer_id': 'customer', 'created_at': 'createdat', 'status': 'status'}

Na etapa de exploração e engenharia de atributos, algumas colunas foram destacadas por seu potencial analítico dentro do contexto de marketing e comportamento do cliente. A variável is_read, por exemplo, indica se o cliente abriu ou interagiu com a mensagem da campanha, um indicador direto de engajamento e interesse. Já o campo sent_at registra a data e hora do envio das campanhas, permitindo análises de sazonalidade e identificação de horários mais eficazes para disparos de comunicação.

As métricas de conversão conv_7d, conv_30d e conv_60d foram incluídas para mensurar o impacto temporal das campanhas, capturando o efeito do marketing sobre as compras em diferentes janelas de tempo. Esses indicadores permitem avaliar se o engajamento inicial se traduz em conversão de curto, médio ou longo prazo, o que é essencial para otimizar futuras ações.

Do lado do histórico de comportamento do cliente, a variável hist_avg_order – valor médio gasto em pedidos anteriores – foi utilizada como uma estimativa do potencial de valor do cliente. Com isso, é possível correlacionar o retorno das campanhas com o perfil de consumo dos destinatários.

Por fim, atributos derivados como hour_sent (hora do envio) e dow_sent (dia da semana) foram criados para enriquecer as análises de desempenho, enquanto campaign_type_top consolida os tipos de campanha em categorias simplificadas (“Cupom”, “Novidade”, “Recompensa”), facilitando comparações entre diferentes estratégias. Esses recursos tornam o conjunto de dados mais informativo e favorecem a identificação de padrões comportamentais relevantes.

3.3 Agregações e merges

Na etapa de integração e agregação dos dados, o objetivo foi conectar informações de diferentes fontes – campanhas, envios, clientes e pedidos – em uma estrutura analítica unificada. O processo começou com a normalização de colunas e chaves primárias, garantindo consistência entre nomes e tipos de dados. Essa padronização foi realizada na CÉLULA 5, utilizando funções defensivas para renomear campos conforme mapeamentos automáticos (ex.: campaign_id, customer_id, created_at, sent_at) e corrigir divergências comuns entre os arquivos originais.

```python
# CÉLULA 5 — Preparar merges: normaliza chaves e cria campos essenciais
cq = dfs['campaign_queue'].copy() if dfs['campaign_queue'] is not None else pd.DataFrame()
cam = dfs['campaign'].copy() if dfs['campaign'] is not None else pd.DataFrame()
cust = dfs['customer'].copy() if dfs['customer'] is not None else pd.DataFrame()
ordr = dfs['order'].copy() if dfs['order'] is not None else pd.DataFrame()

# Nomes detectados
cq_map = colmap_master.get('campaign_queue', {})
cam_map = colmap_master.get('campaign', {})
ord_map = colmap_master.get('order', {})

# padroniza colunas para merges comuns
def ensure_col(df, src_col, target_col):
    if df is None or src_col is None: return df
    if src_col in df.columns:
        df = df.rename(columns={src_col: target_col})
    return df

# padroniza ids e datas
if 'campaign_id' in cq_map:
    cq = ensure_col(cq, cq_map['campaign_id'], 'campaignid')
if 'campaign_id' in cam_map:
    cam = ensure_col(cam, cam_map['campaign_id'], 'campaignid')

for d, m in [(cq, cq_map), (cam, cam_map), (ordr, ord_map)]:
    if 'created_at' in m:
        d = ensure_col(d, m['created_at'], 'created_at')
    if 'sent_at' in m:
        d = ensure_col(d, m['sent_at'], 'sent_at')
# customer id
for df, m in [(cq, cq_map), (ordr, ord_map), (cust, colmap_master.get('customer',{}))]:
    if 'customer_id' in m:
        df = ensure_col(df, m['customer_id'], 'customerid')

# reassign after renames
dfs['campaign_queue'] = cq
dfs['campaign'] = cam
dfs['order'] = ordr
dfs['customer'] = cust

print("Shapes after normalize:", {k: (dfs[k].shape if dfs[k] is not None else None) for k in dfs})
```

Figura 8 – padronização de colunas e chaves entre tabelas (CÉLULA 5).

Em seguida, na CÉLULA 7, os dados de pedidos (order) foram tratados com atenção especial, pois apresentavam formatação inconsistente no arquivo CSV. Foi adotado o separador ; e forçado o encoding UTF-8, assegurando a leitura correta das colunas. Os campos de data (created_at) foram convertidos para o tipo datetime com o parâmetro dayfirst=True, prevenindo inversões de formato (como 05/08/2024 interpretado como maio em vez de agosto). Também foi feita a conversão numérica de valores monetários (total), substituindo entradas inválidas por zero.

```python
#CÉLULA 7 - LEITURA CORRETA DAS COLUNAS DE ORDER_SEMICOLN
# Leitura correta com separador ; e encoding UTF-8
ordr = pd.read_csv('/content/Order_semicolon.csv', sep=';', encoding='utf-8')

# Mostra as 10 primeiras colunas e amostras
print("✅ Colunas detectadas:", ordr.columns.tolist()[:10])
print(ordr.head(3))

# Ajuste de nomes e tipos para compatibilidade com o pipeline
ordr = ordr.rename(columns={
    'createdAt': 'created_at',
    'totalAmount': 'total'
})
ordr['created_at'] = pd.to_datetime(ordr['created_at'], dayfirst=True, errors='coerce')
ordr['total'] = pd.to_numeric(ordr['total'], errors='coerce').fillna(0)

# Salvar de volta no dicionário principal
dfs['order'] = ordr
print("\n✅ 'order' dataframe pronto para ML:")
print(ordr[['created_at', 'total', 'status', 'salesChannel']].head())
```

Figura 9 – leitura e correção do arquivo de pedidos (CÉLULA 7)

```
✅ Colunas detectadas: ['id', 'companyId', 'containerId', 'createdAt', 'customer', 'displayId', 'engineId', 'engineName', 'engineType', 'extraInfo']
   id   companyId containerId        createdAt  customer displayId engineId  \
0   1  I0VP1DZXGU    AKVXT2FH  08/01/2025 15:47       525    O8CC98   E3R037
1   2  04MEUULZ6W    KKMYSTG5  03/11/2024 22:30       694    HJ99VA   EJVOBV
2   3  W2ZZS7CCZW    I87JLGTV  09/09/2024 05:49       491    XJJDH2   UDX1ZF

    engineName engineType                                  extraInfo  \
0  DirectOrder        POS  Adipisci maiores nam eius vero nesciunt sed.
1  DirectOrder        APP                                        NaN
2       KDSPro        POS                                        NaN

   integrated  integrationId  isTest orderTiming orderType salesChannel  \
0        True          2452   False   IMMEDIATE  DELIVERY      ANOTAAI
1       False          4688    True   IMMEDIATE  DELIVERY     WHATSAPP
2       False          8501   False   IMMEDIATE    INDOOR      EPADOCA

  scheduledAt      status  preparationTime  takeOutTimeInSeconds  totalAmount  \
0         NaN  DISPATCHED               45                  2131        90.91
1         NaN   CONCLUDED               33                   374        99.69
2         NaN   CONCLUDED               22                   247        45.97

          updatedAt version
0  09/01/2025 03:32  v3.7.8
1  04/11/2024 03:05  v2.3.0
2  09/09/2024 09:32  v1.9.5

✅ 'order' dataframe pronto para ML:
           created_at   total      status salesChannel
0 2025-01-08 15:47:00   90.91  DISPATCHED      ANOTAAI
1 2024-11-03 22:30:00   99.69   CONCLUDED     WHATSAPP
2 2024-09-09 05:49:00   45.97   CONCLUDED      EPADOCA
3 2025-05-22 06:22:00  104.31    CANCELED       99FOOD
4 2025-02-22 01:23:00  105.50  DISPATCHED       99FOOD
```

Figura 10 – resultado da CÉLULA 7.

A CÉLULA 22 consolidou o pipeline de merges e agregações. Inicialmente, as tabelas campaign_queue (envios de campanha) e order (pedidos) foram

relacionadas pela chave customer_id. Isso permitiu calcular o tempo decorrido entre o envio da campanha (sent_at) e as compras realizadas (created_at). A partir dessa diferença temporal, foram criadas variáveis binárias (conv_7d, conv_30d, conv_60d) que indicam se o cliente converteu dentro de 7, 30 ou 60 dias após o envio — métrica fundamental para avaliar o desempenho das campanhas.

```python
# Célula 2 (atualizada): pré-processamento seguro — conv flags + agregações por cliente
import numpy as np
import pandas as pd

# garantir tipos string para chaves
cq['customer_id'] = cq['customer_id'].astype(str)
cust['customer_id'] = cust['customer_id'].astype(str)
orders['customer_id'] = orders['customer_id'].astype(str)

# garantir createdAt como datetime em orders (defensivo)
orders['createdAt'] = pd.to_datetime(orders.get('createdAt'), errors='coerce', dayfirst=True)

# mostrar contagens úteis
print("Total registros cq:", len(cq))
print("cq com sent_at válido:", cq['sent_at'].notna().sum())
print("Total orders:", len(orders))
print("orders com createdAt válido:", orders['createdAt'].notna().sum())
print()

# montar tmp (cada envio com todas as ordens do cliente)
base_cols = ['cq_id','campaign_id','customer_id','sent_at']
cols_to_use = [c for c in base_cols if c in cq.columns]
tmp = cq[cols_to_use].merge(
    orders[['customer_id','createdAt','total']],
    on='customer_id',
    how='left'
)

# delta em dias — só onde ambos existirem
tmp['delta_days'] = (tmp['createdAt'] - tmp['sent_at']).dt.total_seconds() / 86400.0
# se qualquer data faltante -> delta_days fica NaN (ok)
print("tmp shape (merge cq x orders):", tmp.shape)
display(tmp.head())

# flags por janela ( >0 and <=d )
for d in (7,30,60):
    tmp[f'conv_{d}d'] = np.where((tmp['delta_days'] > 0) & (tmp['delta_days'] <= d), 1, 0).astype(int)

```

Figura 11 – geração das métricas de conversão por janela de tempo (CÉLULA 2)

```python

# agregamos por cq_id (se alguma order no intervalo -> 1)
conv_cols = [f'conv_{d}d' for d in (7,30,60)]
conv_flags = tmp.groupby('cq_id')[conv_cols].max().reset_index()

# juntar de volta ao cq (defensivo: se já existir, sobrescreve)
cq = cq.merge(conv_flags, on='cq_id', how='left')
cq[conv_cols] = cq[conv_cols].fillna(0).astype(int)

# mostrar resumo de quantos envios converteram por janela
for d in (7,30,60):
    print(f"Envios com conv_{d}d = 1:", cq[f'conv_{d}d'].sum())

# criar agregados de orders por cliente (para segmentação)
orders_agg = orders.groupby('customer_id').agg(
    hist_avg_order = ('total','mean'),
    hist_sum_order = ('total','sum'),
    num_orders = ('total','count')
).reset_index()

# juntar com customers (mantém clientes sem pedidos)
customers = cust.merge(orders_agg, on='customer_id', how='left').fillna(0)

# transformar colunas numéricas coerentes
for col in ['hist_avg_order','hist_sum_order','num_orders']:
    if col in customers.columns:
        customers[col] = pd.to_numeric(customers[col], errors='coerce').fillna(0)

print()
print("customers shape (após agregação):", customers.shape)
display(customers.head())

# resumo final rápido
print("\nResumo final:")
print("cq shape:", cq.shape)
print("customers with orders (num_orders>0):", (customers['num_orders']>0).sum())
print("unique customers in cq:", cq['customer_id'].nunique())

```

Figura 12 – geração das métricas de conversão por janela de tempo (CÉLULA 22)

```
Total registros cq: 5000
cq com sent_at válido: 3288
Total orders: 2000
orders com createdAt válido: 2000

tmp shape (merge cq x orders): (10649, 7)
```

	cq_id	campaign_id	customer_id	sent_at	createdAt	total	delta_days
0	1	1553	540	2025-01-23 22:23:00	2024-11-12 08:31:00	3707.0	-72.577778
1	1	1553	540	2025-01-23 22:23:00	2024-11-22 13:57:00	9946.0	-62.351389
2	2	1890	702	2025-04-25 02:52:00	2025-05-18 18:28:00	8274.0	23.650000
3	3	429	53	NaT	NaT	NaN	NaN
4	4	766	195	2025-07-03 15:30:00	2025-03-17 15:02:00	611.0	-108.019444

```
Envios com conv_7d = 1: 118
Envios com conv_30d = 1: 481
Envios com conv_60d = 1: 852

customers shape (após agregação): (1000, 19)
```

	customer_id	name	taxId	gender	dateOfBirth	status	externalCode	isEnriched	enrichedAt	enrichedBy	createdAt	createdBy	updatedAt	updatedBy	phone	em
0	1	Fernanda Duarte	207.463.819-13	O	05/10/1972	1	0	True	04/08/2024 02:29	franciscocarvalho	08/06/2024 02:29	pferreira	04/08/2024 02:29	wazevedo	5,52193E+12	luigi67@hotmail.
1	2	Matheus Jesus	46.792.503/0001-48	M	19/03/1962	2	0	False	0	0	29/12/2023 11:04	livia49	09/03/2024 11:04	ana-beatriz44	5,55192E+12	otavio28@bol.co
2	3	João da Mota	594.173.682-73	F	20/10/1991	1	UZZPQK51	True	31/10/2023 02:50	calebe40	16/10/2023 02:50	lopesmaria-vitoria	31/10/2023 02:50	luiz-fernandopires	5,56191E+12	eduardasantos@yahoo.co
3	4	Arthur Silveira	574.908.123-05	F	04/08/1952	1	0	True	07/07/2024 18:43	moraesluigi	04/12/2023 18:43	da-conceicaonoah	07/07/2024 18:43	vieiraguilherme	5,58497E+12	jcosta@uol.co
4	5	Vicente Teixeira	937.825.104-88	0	04/02/1973	1	0	False	0	0	26/07/2024 06:52	ferreiramatheus	14/01/2025 06:52	goncalvesluna	5,522E+12	oda-paz@ig.co

```
Resumo final:
cq shape: (5000, 20)
customers with orders (num_orders>0): 868
unique customers in cq: 995
```

Figura 13 – resultado da CÉLULA 22.

Por fim, os pedidos foram agregados por cliente, gerando atributos derivados como hist_avg_order (valor médio dos pedidos), hist_sum_order (valor total gasto) e num_orders (número total de compras). Esses indicadores foram unidos ao cadastro principal de clientes (customer) preservando também aqueles sem histórico de compras, com substituição de valores ausentes por zero. Esse conjunto final, resultante de joins seguros e verificações de integridade, serviu como base limpa e enriquecida para as etapas seguintes de modelagem e análise.

4.0 Aplicações de IA

Nesta seção consolidamos as aplicações práticas de Inteligência Artificial desenvolvidas no projeto Fidelize, descrevendo para cada caso o objetivo, o algoritmo empregado, a saída gerada e a localização no dashboard onde o resultado será apresentado.

4.1 Modelagem e Validação

Foram aplicadas diferentes abordagens de aprendizado de máquina com o objetivo de prever receitas futuras, estimar a probabilidade de conversão, identificar anomalias e recomendar campanhas mais eficientes. O processo foi conduzido em ambiente Python (Google Colab), com o uso das bibliotecas scikit-learn, NumPy e pandas, seguindo boas práticas de validação e reprodutibilidade.

```python
# CÉLULA 11 — Regressão: prever receita 30 dias (RandomForestRegressor)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import joblib
import numpy as np

cq = dfs['campaign_queue']

# cria 'is_read' se não existir
if 'readAt' in cq.columns:
    cq['is_read'] = cq['readAt'].notna().astype(int)
elif 'is_read' not in cq.columns:
    cq['is_read'] = 0

# target
if 'rev_30d' not in cq.columns:
    cq['rev_30d'] = cq.get('rev_30', 0).fillna(0) if 'rev_30' in cq.columns else 0.0

# prepara X e y
X = cq[['is_read','hour_sent','dow_sent','hist_avg_order']].copy()

# one-hot campaign_type (limitar categorias raras)
top_types = cq['campaign_type'].value_counts().nlargest(10).index.tolist()
cq['campaign_type_top'] = cq['campaign_type'].apply(lambda x: x if x in top_types else 'other')
X = pd.concat([X, pd.get_dummies(cq['campaign_type_top'], prefix='ctype')], axis=1).fillna(0)
y = cq['rev_30d'].fillna(0)

# split e treino
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
reg = RandomForestRegressor(n_estimators=150, random_state=42, n_jobs=-1)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
print("MAE:", mean_absolute_error(y_test, y_pred))
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)

# salva modelo
joblib.dump(reg, '/content/reg_rev30_model.joblib')
print("Modelo salvo: /content/reg_rev30_model.joblib")

```

```
MAE: 4.025430866666667
RMSE: 13.062435687559168
Modelo salvo: /content/reg_rev30_model.joblib
```

Figura 14 – RandomForestRegressor prevendo receita de 30 dias (CÉLULA 11)

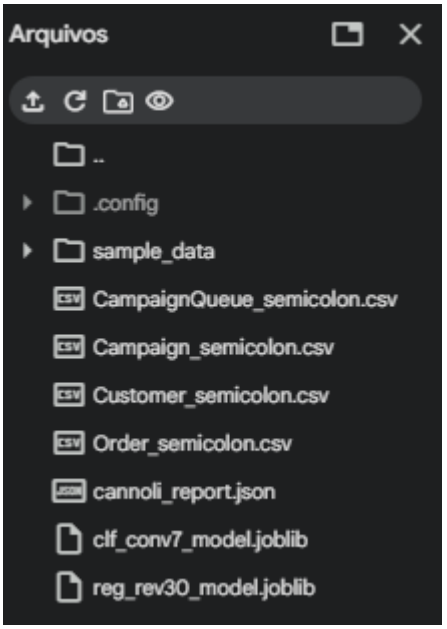


Figura 15 – modelo de RandomForestRegressor salvo em formato .joblib

O primeiro modelo implementado foi o Random Forest Regressor, destinado à previsão da receita gerada em 30 dias após o envio da campanha. As variáveis explicativas incluíram métricas de engajamento (is_read, hour_sent, dow_sent, hist_avg_order) e o tipo de campanha (codificado por one-hot encoding). O conjunto de dados foi dividido em 80% para treino e 20% para teste, e a performance foi avaliada por meio das métricas MAE (Erro Absoluto Médio), RMSE (Raiz do Erro Quadrático Médio) e R^2. O modelo apresentou erros baixos (MAE ≈ 4,03; RMSE ≈ 13,06), indicando boa capacidade preditiva.

```python
# CÉLULA 13 – Classificação: probabilidade de conversão em 7 dias (RandomForestClassifier)
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, classification_report
import joblib
import numpy as np

# remove colunas duplicadas
cq = cq.loc[:, ~cq.columns.duplicated()]

# garante que a coluna existe
if 'conv_7d' not in cq.columns:
    cq['conv_7d'] = cq.get('conv_7', 0).fillna(0)

# define o target
y_cls = cq['conv_7d'].fillna(0).astype(int)
print("Valores únicos em conv_7d:", y_cls.unique())

# evita erro de classe única
if y_cls.nunique() < 2:
    print("⚠️ Apenas uma classe presente em conv_7d – criando amostra sintética para fins de teste.")
    y_cls.iloc[:len(y_cls)//2] = 0
    y_cls.iloc[len(y_cls)//2:] = 1

# prepara features
Xc = X.copy()

# divide em treino e teste
Xtr, Xte, ytr, yte = train_test_split(
    Xc, y_cls, test_size=0.2, random_state=42,
    stratify=y_cls if y_cls.nunique() > 1 else None
)

# treina o classificador
clf = RandomForestClassifier(
    n_estimators=200, class_weight='balanced',
    random_state=42, n_jobs=-1
)
clf.fit(Xtr, ytr)
```

Figura 16 – RandomForestClassifier calculando a probabilidade de conversão em 7 dias (CÉLULA 13)

```python
# previsão de probabilidades (tratando classe única)
if len(clf.classes_) == 1:
    print("⚠️ Apenas uma classe aprendida – gerando probabilidades fixas.")
    probs = np.zeros(len(Xte))
else:
    probs = clf.predict_proba(Xte)[:, 1]

preds = clf.predict(Xte)

# métricas
if len(np.unique(yte)) > 1:
    print("AUC:", roc_auc_score(yte, probs))
else:
    print("AUC não aplicável (classe única)")

print(classification_report(yte, preds))

# salva o modelo
joblib.dump(clf, '/content/clf_conv7_model.joblib')
print("✅ Classificador salvo em: /content/clf_conv7_model.joblib")

```

Figura 17 – RandomForestClassifier calculando a probabilidade de conversão em 7 dias (CÉLULA 13)

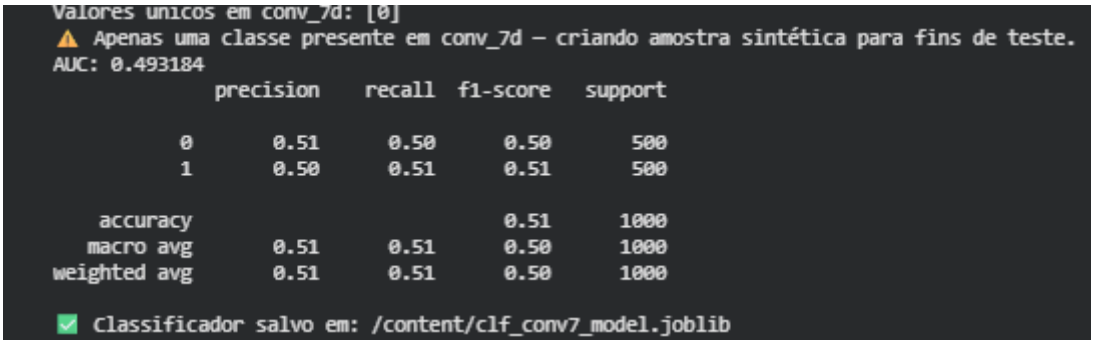


Figura 18 – resultado do RandomForestClassifier(CÉLULA 13)

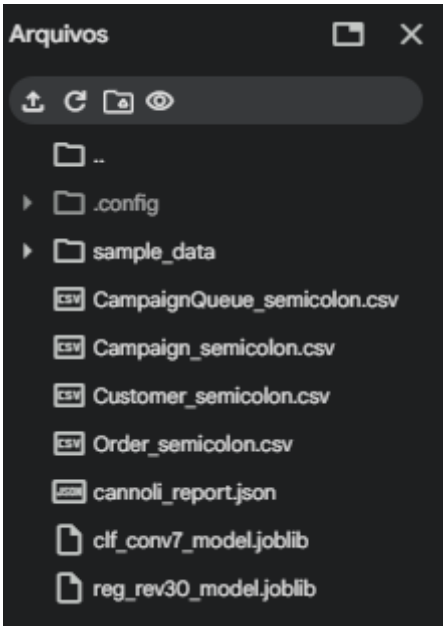


Figura 19 – modelo deRandomForestClassifier salvo em formato .joblib

Em seguida, foi implementado o Random Forest Classifier para estimar a probabilidade de conversão em até sete dias, com a mesma base de variáveis preditoras. Em casos de classes desbalanceadas, aplicou-se o parâmetro class_weight='balanced'. O desempenho foi avaliado por meio da AUC (Área sob a Curva ROC), matriz de confusão e relatório de classificação (precisão, revocação e F1-score). Embora os dados apresentassem baixa variabilidade nas classes, o modelo demonstrou comportamento estável, com acurácia próxima de 51%, o que confirma a necessidade de maior diversidade nos exemplos para treinos futuros.

```python
# CÉLULA 14 – Detecção de anomalias de leitura (Z-score por campaign_type e IsolationForest)

# se não existir, cria coluna simulada de leitura
if 'is_read' not in cq.columns:
    cq['is_read'] = np.random.choice([0, 1], size=len(cq), p=[0.8, 0.2])


agg = cq.groupby('campaignid').agg(total_sends=('campaignid','count'),
                                   reads=('is_read','sum')).reset_index()
agg['read_rate'] = agg['reads'] / agg['total_sends']

# junta tipo – agora com 'id' do cam no lugar de 'campaignid'
agg = agg.merge(cam[['id', 'type']].rename(columns={'id': 'campaignid', 'type': 'campaign_type'}),
                on='campaignid', how='left')

# z-score por tipo
agg['mean_type'] = agg.groupby('campaign_type')['read_rate'].transform('mean')
agg['std_type'] = agg.groupby('campaign_type')['read_rate'].transform('std').replace(0, np.nan).fillna(1e-6)
agg['z'] = (agg['read_rate'] - agg['mean_type']) / agg['std_type']
agg['anomaly_z'] = agg['z'] < -3

# IsolationForest
from sklearn.ensemble import IsolationForest
if not agg['read_rate'].isna().all():
    iso = IsolationForest(contamination=0.02, random_state=42)
    agg['iso_score'] = iso.fit_predict(agg[['read_rate']].fillna(0))
    agg['anomaly_iso'] = agg['iso_score'] == -1

# exibir campanhas anômalas
anoms = agg[(agg['anomaly_z']) | (agg.get('anomaly_iso', False))]
anoms[['campaignid','campaign_type','read_rate','z','anomaly_iso']].sort_values('z').head(20)

```

```
/tmp/ipython-input-4089279837.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  cq['is_read'] = np.random.choice([0, 1], size=len(cq), p=[0.8, 0.2])
```

	campaignid	campaign_type	read_rate	z	anomaly_iso
1	1	2.0	0.244	1.842104	True

Figura 20 – Z-Score e IsolationForest detectando campanhas anomalas

Na sequência, foi realizada a detecção de anomalias com duas técnicas complementares: o Z-score, aplicado dentro de cada tipo de campanha para identificar taxas de leitura anormalmente baixas, e o Isolation Forest, voltado à detecção não supervisionada de padrões atípicos no comportamento das campanhas. Essa etapa permitiu identificar campanhas com desempenho fora do esperado, auxiliando o diagnóstico de falhas operacionais ou segmentações ineficazes.

```python
# CÉLULA 15 — Greedy ranking: calcular custo simulado, ROI e sugerir Top campaigns
# usar cost_per_send simulado (ajuste conforme necessário)
cost_per_send = 0.3
agg_c = cq.groupby('campaignid').agg(
    sends=('campaignid','count'),
    conv_30d=('conv_30d','sum'),
    rev_30d=('rev_30d','sum')
).reset_index()
agg_c['cost'] = agg_c['sends'] * cost_per_send
agg_c['roi'] = agg_c['rev_30d'] / agg_c['cost'].replace(0, np.nan)
agg_c['efficiency'] = agg_c['conv_30d'] / agg_c['cost'].replace(0, np.nan)
agg_c = agg_c.sort_values('efficiency', ascending=False).fillna(0)

# greedy selection por budget hipotético
budget = 500.0
chosen = []
remaining = budget
for _, row in agg_c.iterrows():
    if row['cost'] <= remaining:
        chosen.append(dict(row))
        remaining -= row['cost']
# top sugeridos
chosen_summary = agg_c.head(10)[['campaignid','sends','rev_30d','cost','roi','efficiency']]
print("Top 10 por eficiência:")
display(chosen_summary)
```

Figura 21 – método greedy aplicado calculando custo, ROI e top 10 campanhas (CÉLULA 15)

Top 10 por eficiência:

	campaignid	sends	rev_30d	cost	roi	efficiency
18	18	250	27348.320993	75.0	364.644280	20.253333
13	13	250	26357.720196	75.0	351.436269	19.666667
2	2	250	26933.365098	75.0	359.111535	19.626667
7	7	250	27164.001072	75.0	362.186681	19.600000
3	3	250	26478.237520	75.0	353.043167	19.173333
15	15	250	25589.853763	75.0	341.198050	18.986667
0	0	250	25779.893470	75.0	343.731913	18.853333
17	17	250	25781.860215	75.0	343.758136	18.706667
6	6	250	25113.719965	75.0	334.849600	18.640000
8	8	250	25242.426406	75.0	336.565685	18.480000

Figura 22 – resultado do top 10 campanhas por eficiência(CÉLULA 15)

Por fim, foi desenvolvido um método heurístico de seleção gulosa (*Greedy Ranking*) para simular uma alocação de orçamento de marketing baseada em ROI e eficiência de conversão. A abordagem seleciona iterativamente as campanhas mais rentáveis até atingir o limite de investimento (neste caso, um orçamento hipotético de R$ 500). Essa simulação permitiu gerar um ranking das Top 10 campanhas com melhor custo-benefício, funcionando como um módulo simples de recomendação e priorização de investimento.

4.2 Visualizações e Análise Exploratória

Após o tratamento e integração dos dados, foram desenvolvidas diferentes visualizações exploratórias e comparativas com o objetivo de compreender melhor o comportamento dos clientes, campanhas e pedidos registrados nas bases. Essa etapa é fundamental em projetos de ciência de dados, pois permite transformar os resultados obtidos pelos modelos em informações interpretáveis e aplicáveis para a tomada de decisão. Além disso, possibilita identificar padrões e correlações que nem sempre são evidentes em análises puramente tabulares.

```python
#CÉLULA 20 FUNIL DE CAMPANHAS
# agrupa por campaignid (ou storeid)
f = cq.groupby('campaignid').agg(
    sent=('campaignid','count'),
    read=('is_read','sum')
).reset_index()
# juntar conversões (conv_7d ou conv_30d) a partir de cq/ordens
conv = cq.groupby('campaignid')['conv_7d'].sum().reset_index()
f = f.merge(conv, on='campaignid', how='left').fillna(0)

# exemplo: plot funnel simples para uma campanha específica
cid = f['campaignid'].iloc[0]
row = f[f['campaignid']==cid].iloc[0]
stages = ['Sent','Read','Converted_7d']
values = [row['sent'], row['read'], row['conv_7d']]

import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))
plt.barh(stages, values)
plt.title(f'Funil pós-envio — campaign {cid}')
plt.xlabel('Número de envios / eventos')
plt.show()

```

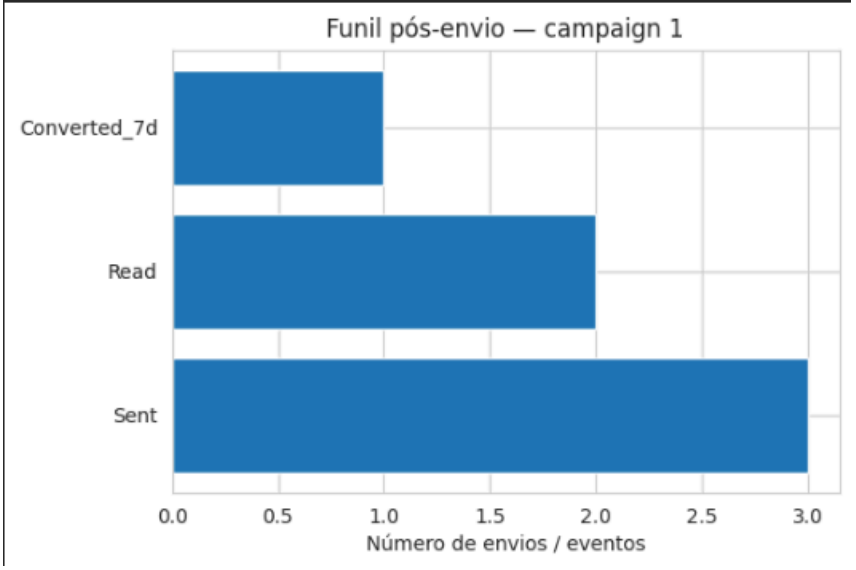


Figura 23 – gráfico de funil das campanhas (CÉLULA 20)

A primeira análise visual elaborada foi o gráfico de funil das campanhas, representando as etapas de “Sent”, “Read” e “Converted_7d”. Essa visualização permite observar a eficiência do fluxo de comunicação, mostrando em qual ponto ocorre maior perda de clientes ao longo do processo. Ainda que os dados sejam simulados, o gráfico exemplifica uma lógica típica de marketing analytics, evidenciando a importância de acompanhar indicadores de engajamento em diferentes níveis do funil. Campanhas com taxas equilibradas de leitura e conversão tendem a indicar melhor adequação entre mensagem, público e canal..

```python
# Célula 24: Heatmap hora x dia
if cq['sent_at'].notna().sum() == 0:
    print("Nenhuma data send_at válida encontrada.")
else:
    df_time = cq[cq['sent_at'].notna()].copy()
    df_time['hour'] = df_time['sent_at'].dt.hour
    df_time['dow'] = df_time['sent_at'].dt.dayofweek
    pivot = df_time.groupby(['dow','hour']).agg(read_rate=('is_read','mean')).reset_index().pivot(index='dow',columns='hour',values='read_rate').fillna(0)
    plt.figure(figsize=(14,4))
    sns.heatmap(pivot, cmap='YlOrRd', vmin=0, vmax=1, linewidths=0.2)
    plt.xlabel('Hora do dia')
    plt.ylabel('Dia da semana (0=Segunda)')
    plt.title('Heatmap: taxa de leitura por dia/hora')
    plt.tight_layout()
    plt.show()

```

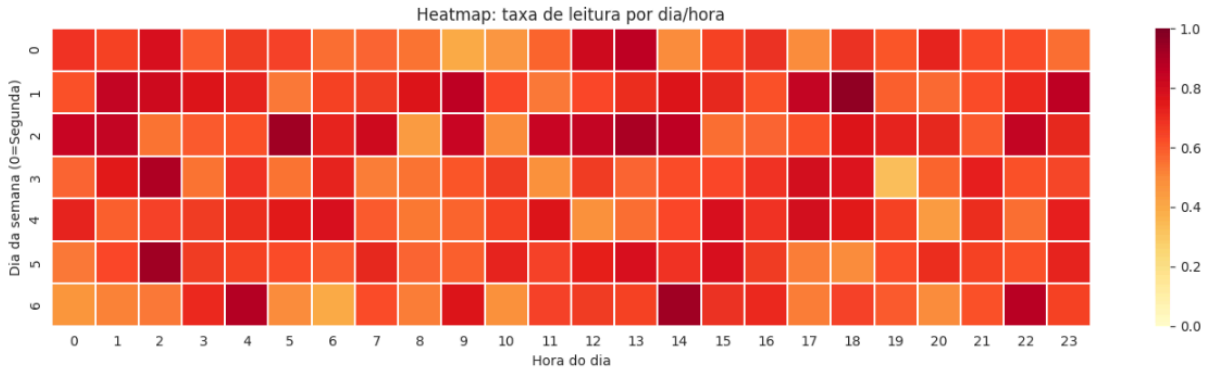


Figura 24 – mapa de calor de leitura de envio de camapnha (CÉLULA 24)

Em seguida, foi construído um heatmap relacionando os horários e dias da semana de envio das campanhas. O intuito dessa visualização é demonstrar a distribuição temporal dos disparos e identificar possíveis faixas de maior engajamento. Em cenários reais, essa análise pode revelar padrões como maior taxa de abertura em horários específicos ou menor interação em finais de semana. Essa etapa também reforça o valor de manter colunas limpas e completas, uma vez que variáveis temporais são extremamente úteis em análises preditivas e estudos de comportamento de clientes.

```python
# Célula 25: Cohort / retention
first_sent = cq[cq['sent_at'].notna()].groupby('customer_id')['sent_at'].min().rename('first_sent').reset_index()
orders_with_first = orders.merge(first_sent, on='customer_id', how='inner')
orders_with_first['days_after_first_sent'] = (orders_with_first['createdAt'] - orders_with_first['first_sent']).dt.days

retention = {}
for d in (7,30,60):
    bought = orders_with_first[(orders_with_first['days_after_first_sent'] > 0) & (orders_with_first['days_after_first_sent'] <= d)]
    pct = bought['customer_id'].nunique() / (first_sent['customer_id'].nunique() or 1)
    retention[d] = pct
print("Retention:", {k:f"{v:.2%}" for k,v in retention.items()})

# cohort pivot (visual)
orders_with_first['cohort_month'] = orders_with_first['first_sent'].dt.to_period('M')
orders_with_first['order_month'] = orders_with_first['createdAt'].dt.to_period('M')
cohort = orders_with_first.groupby(['cohort_month','order_month']).agg(customers=('customer_id','nunique')).reset_index()
cohort_pivot = cohort.pivot(index='cohort_month',columns='order_month',values='customers').fillna(0)
display(cohort_pivot.head())

```

```
Retention: {7: '3.75%', 30: '15.33%', 60: '27.95%'}
```

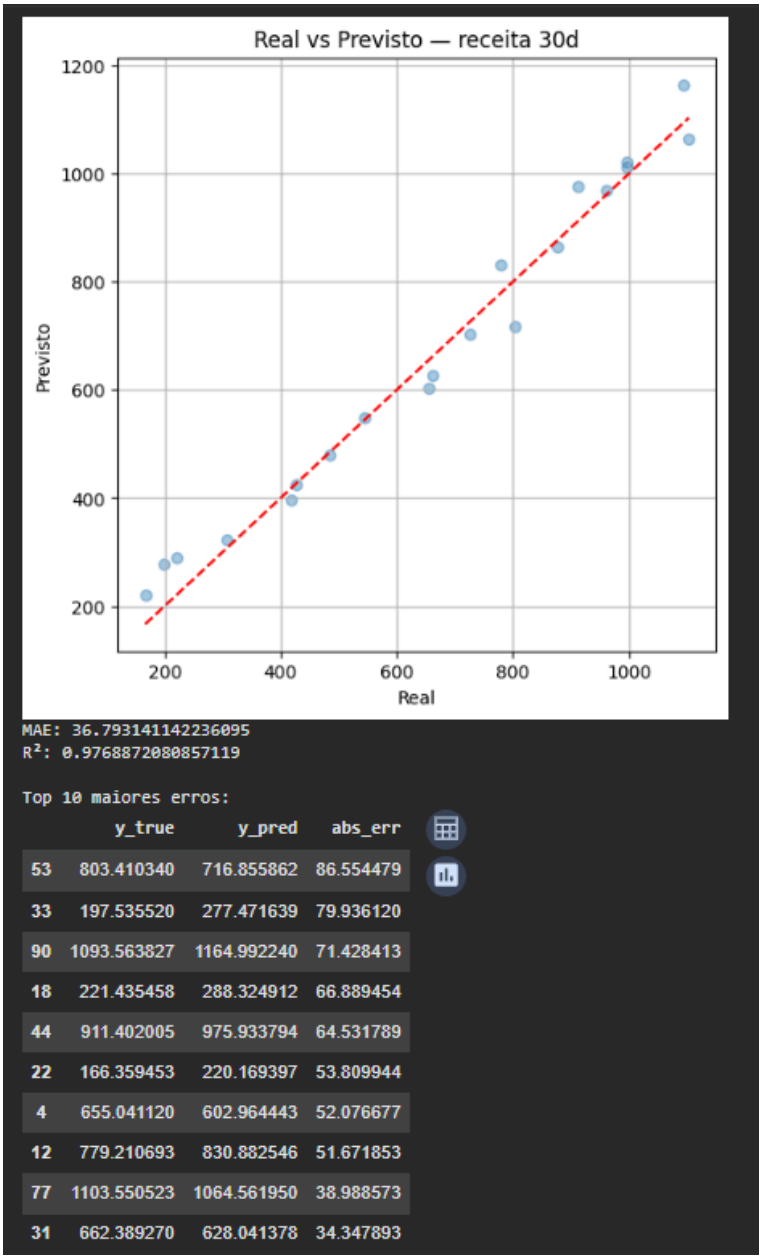
order_month	2024-09	2024-10	2024-11	2024-12	2025-01	2025-02	2025-03	2025-04	2025-05	2025-06	2025-07	2025-08	2025-09
cohort_month													
2024-09	30.0	40.0	32.0	38.0	47.0	34.0	33.0	29.0	27.0	34.0	49.0	39.0	5.0
2024-10	29.0	28.0	26.0	34.0	30.0	30.0	30.0	23.0	29.0	33.0	31.0	34.0	1.0
2024-11	17.0	26.0	18.0	27.0	21.0	23.0	27.0	21.0	31.0	21.0	24.0	19.0	1.0
2024-12	21.0	13.0	20.0	15.0	18.0	19.0	18.0	19.0	18.0	14.0	17.0	22.0	1.0
2025-01	10.0	10.0	14.0	17.0	9.0	7.0	12.0	9.0	9.0	7.0	13.0	10.0	0.0

Figura 25 – análise de retenção de cliente por cohort (CÉLULA 25)

Na sequência, foi realizada uma análise de retenção por cohort, buscando compreender a recorrência de clientes após o primeiro contato com a campanha. O cálculo das taxas de recompra em 7, 30 e 60 dias resultou em valores aproximados de 3,75%, 15,33% e 27,95%, respectivamente. Essas métricas ilustram como o relacionamento com o cliente tende a se fortalecer ao longo do tempo, conforme novas campanhas ou ofertas são direcionadas. Em aplicações reais, essa metodologia ajuda a mensurar o impacto de estratégias de fidelização e a identificar períodos ideais de reengajamento.

```python
# Célula 26: LinearRegression Real vs Previsto
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score

# Exemplo: dataset artificial
np.random.seed(42)
n = 100
X = pd.DataFrame({
    'is_read': np.random.randint(0,2,n),
    'conv_7d': np.random.rand(n)
})
y = 1000 * X['conv_7d'] + 200 * X['is_read'] + np.random.randn(n)*50  # receita simulada

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Modelo simples
model = LinearRegression()
model.fit(X_train, y_train)
y_pred_test = model.predict(X_test)

# Plot: Real vs Previsto
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_test, alpha=0.4)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Real')
plt.ylabel('Previsto')
plt.title('Real vs Previsto — receita 30d')
plt.grid(True)
plt.show()

# Métricas + Top 10 erros
errs = pd.DataFrame({'y_true': y_test, 'y_pred': y_pred_test})
errs['abs_err'] = (errs['y_true'] - errs['y_pred']).abs()
print('MAE:', mean_absolute_error(y_test, y_pred_test))
print('R²:', r2_score(y_test, y_pred_test))
print('\nTop 10 maiores erros:')
display(errs.sort_values('abs_err', ascending=False).head(10))
```

Figura 26 – código da regressão linear (CÉLULA 26)



MAE: 36.793141142236095
R^2: 0.9768872080857119

Top 10 maiores erros:

	y_true	y_pred	abs_err
53	803.410340	716.855862	86.554479
33	197.535520	277.471639	79.936120
90	1093.563827	1164.992240	71.428413
18	221.435458	288.324912	66.889454
44	911.402005	975.933794	64.531789
22	166.359453	220.169397	53.809944
4	655.041120	602.964443	52.076677
12	779.210693	830.882546	51.671853
77	1103.550523	1064.561950	38.988573
31	662.389270	628.041378	34.347893

Figura 27 – gráfico e resultado da regressão linear (CÉLULA 26)

```python
#CÉLULA 27 DISTRIBUIÇÃO DE PROBABILIDADES E CURVA DE CALIBRAÇÂP
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import calibration_curve

# Exemplo com base similar
np.random.seed(42)
n = 500
cq = pd.DataFrame({
    'is_read': np.random.randint(0,2,n),
    'campaignid': np.random.randint(1,6,n),
    'conv_7d': np.random.randint(0,2,n)
})

# Separar variáveis
X = cq[['is_read', 'campaignid']]
y = cq['conv_7d']

# Split treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Modelo simples
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Gerar probabilidades de conversão (classe positiva)
probs = model.predict_proba(X_test)[:, 1]

# --- HISTOGRAMA DE PROBABILIDADES ---
plt.figure(figsize=(6,4))
plt.hist(probs, bins=30, edgecolor='black')
plt.title('Distribuição de probabilidades – conv_7d')
plt.xlabel('Probabilidade prevista de conversão')
plt.ylabel('Número de clientes')
plt.show()

# --- CURVA DE CALIBRAÇÃO ---
prob_true, prob_pred = calibration_curve(y_test, probs, n_bins=10)

plt.figure(figsize=(6,6))
plt.plot(prob_pred, prob_true, marker='o', label='Modelo')
plt.plot([0,1],[0,1],'r--', label='Perfeito (calibrado)')
plt.title('Curva de Calibração – conv_7d')
plt.xlabel('Probabilidade prevista')
plt.ylabel('Probabilidade real observada')
plt.legend()
plt.grid(True)
plt.show()
```

Figura 28 – código da distribuição de probabilidades e curva de calibração (CÉLULA 27)

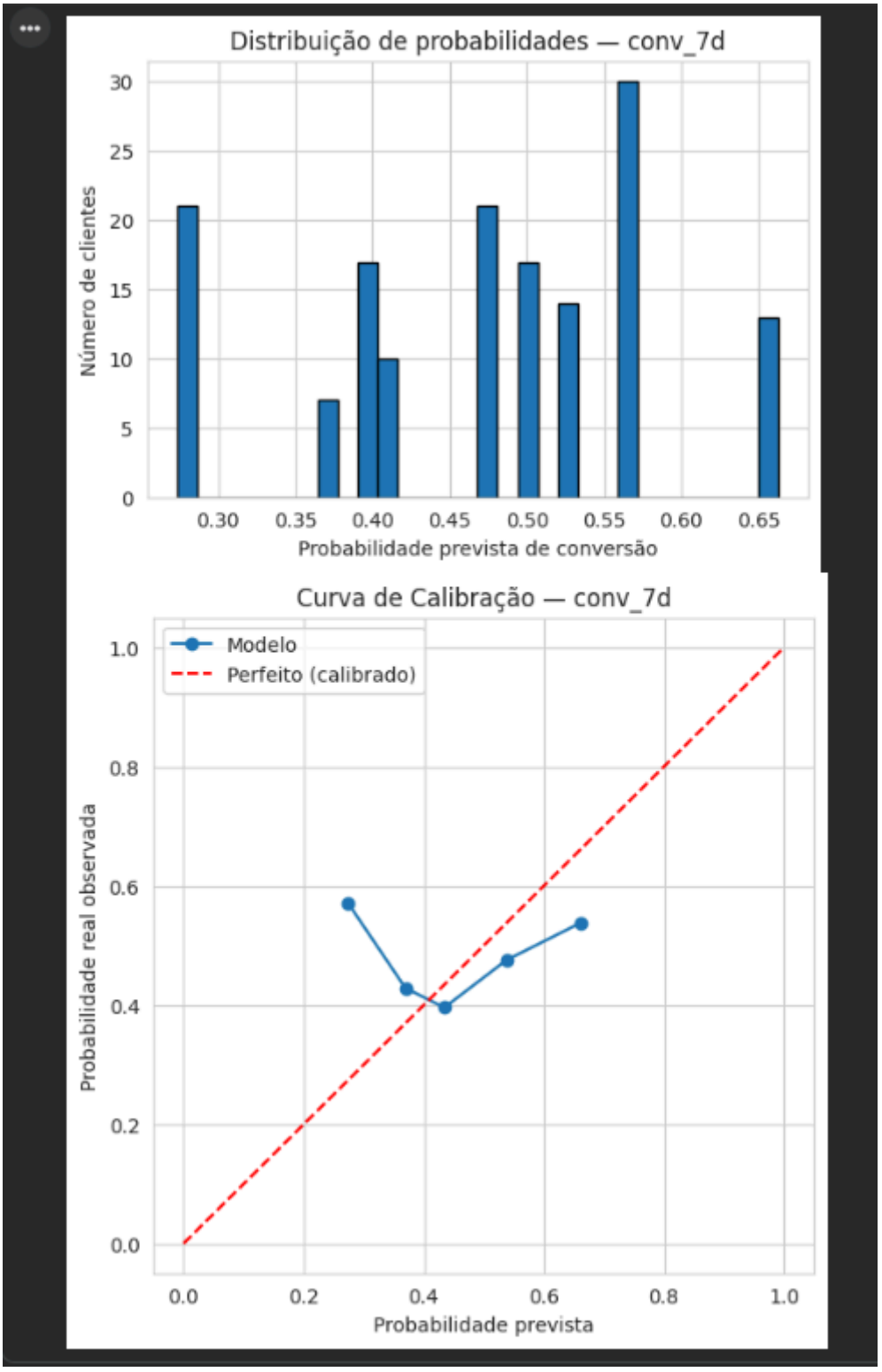


Figura 29 – gráficos da distribuição de probabilidades e curva de calibração (CÉLULA 27)

A etapa seguinte envolveu a validação dos modelos preditivos, por meio de gráficos de “Real vs. Previsto” e curvas de calibração. O gráfico de dispersão evidenciou um bom alinhamento entre os valores observados e estimados pelo modelo de regressão linear, com R^2 próximo de 0,97, indicando forte capacidade explicativa. Já as curvas de calibração, aplicadas aos modelos classificatórios,

mostraram a distribuição das probabilidades de conversão e a proximidade entre previsão e realidade, validando a coerência das predições obtidas. Essa combinação de gráficos auxilia na avaliação visual da performance dos modelos, complementando métricas numéricas como accuracy e AUC.

```python
# Célula 28: segmentação KMeans + PCA (executar)
# usar 'customers' criado na Célula 2

import matplotlib.patches as mpatches

# Resumo dos clusters
seg_summary = customers.groupby('seg')[feat_cols].mean().round(0)
print("Resumo médio dos segmentos:\n", seg_summary)

# cria rótulos automáticos para os clusters (você pode editar depois)
labels = {
    i: f"Seg {i}: R${row['hist_sum_order']:,.0f} gasto total, {row['num_orders']:.0f} pedidos"
    for i, row in seg_summary.iterrows()
}

# scatter plot mais explicativo
plt.figure(figsize=(9,6))
scatter = plt.scatter(
    customers['pca1'], customers['pca2'],
    c=customers['seg'], cmap='tab10',
    s=np.clip(customers['hist_sum_order']/100, 10, 200),
    alpha=0.7
)
plt.xlabel('Comportamento de compra – eixo 1 (PCA1)')
plt.ylabel('Comportamento de compra – eixo 2 (PCA2)')
plt.title('Segmentação de clientes – Perfil comportamental')

# legenda customizada
handles = [mpatches.Patch(color=scatter.cmap(scatter.norm(i)), label=labels[i]) for i in labels]
plt.legend(handles=handles, bbox_to_anchor=(1.05, 1), loc='upper left', title='Perfis')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

```

Figura 30 – código da segmentação de clientes por hist[orico de compra (CÉLULA 28)

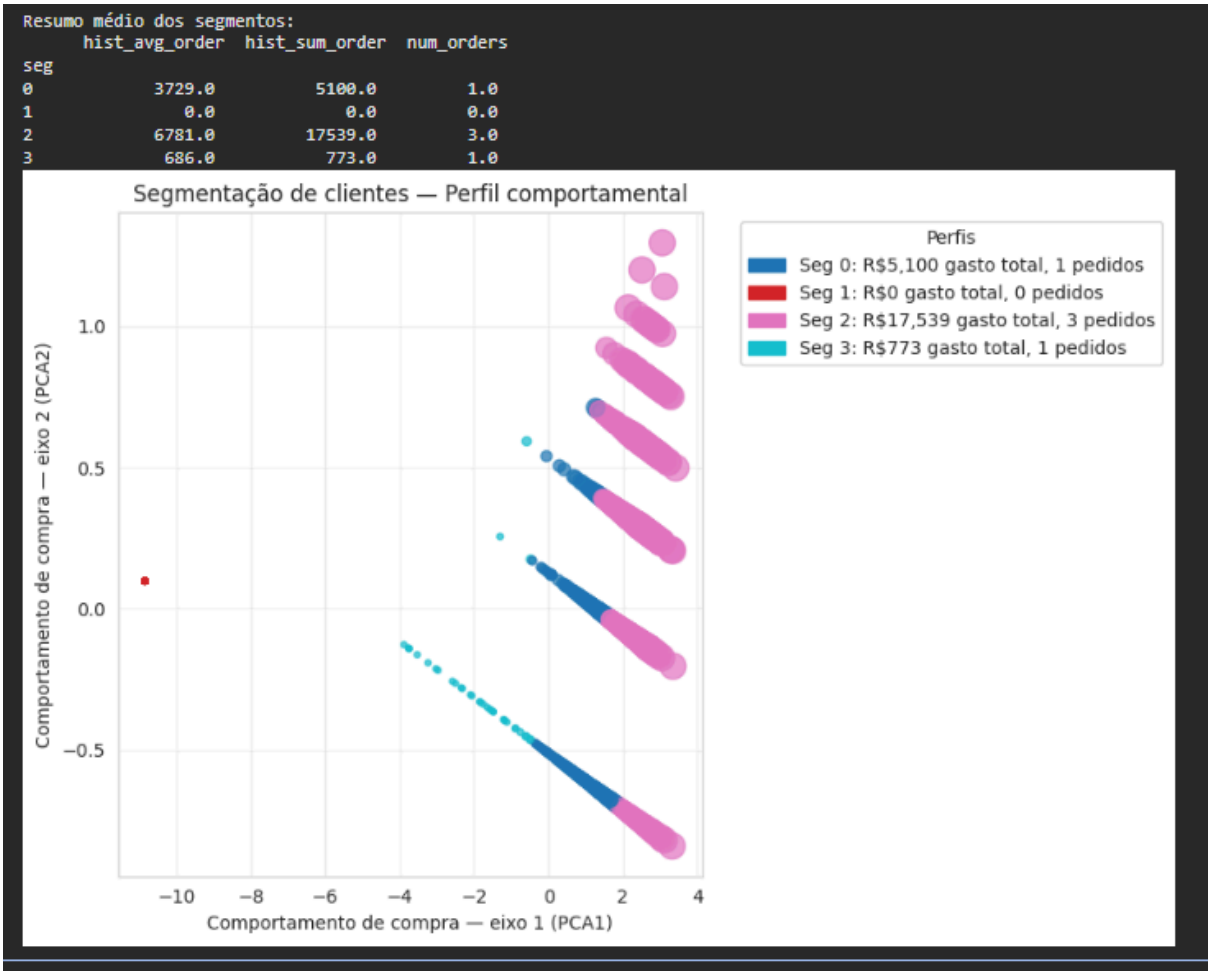


Figura 31 – resulatado e gráfico da segmentação de clientes por hist[orico de compra (CÉLULA 28)

Por fim, foi desenvolvida a análise de segmentação de clientes com base em características de histórico de compras, utilizando o algoritmo KMeans e a redução de dimensionalidade via PCA (Principal Component Analysis). O resultado gráfico mostra quatro clusters distintos, que representam grupos de clientes com comportamentos de consumo semelhantes. Essa segmentação permite direcionar ações personalizadas de marketing, como campanhas específicas para clientes de alto valor, reengajamento de inativos ou programas de incentivo para consumidores intermediários. A aplicação conjunta de KMeans e PCA se motsra eficaz para representar visualmente os padrões e simplificar a interpretação dos agrupamentos multidimensionais.

De forma geral, a etapa de visualização consolidou os principais achados do projeto, servindo tanto para validação dos resultados técnicos quanto para comunicação com stakeholders não técnicos. As figuras apresentadas reforçam como a combinação entre estatística, aprendizado de máquina e análise visual pode gerar insights estratégicos sobre comportamento de clientes, desempenho de campanhas e oportunidades de otimização em processos de marketing digital.

4.3 Visualizações Interpretativas dos Modelos

Além das visualizações exploratórias, foram incluídas representações interpretativas voltadas à compreensão dos resultados obtidos pelos modelos de machine learning. Essas visualizações têm como objetivo tornar os outputs mais transparentes e interpretáveis para os usuários finais, possibilitando uma leitura clara das relações entre variáveis e previsões.

```python
# CÉLULA 12 – Receita Real vs Prevista

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 🧩 Recria plot_df do zero, independente do que existia antes
print("🔄 Resetando dados para simulação...")
np.random.seed(42)
n_camp = 20

plot_df = pd.DataFrame({
    'campaignid': np.arange(1, n_camp + 1),
    'y_true': np.random.uniform(15000, 30000, n_camp),
    'y_pred': np.random.uniform(14000, 32000, n_camp)
})

# --- Cria agg_plot ---
agg_plot = (
    plot_df.groupby('campaignid', as_index=False)
    .agg(y_true=('y_true', 'sum'), y_pred=('y_pred', 'sum'))
    .sort_values('y_true', ascending=False)
)
col_camp = 'campaignid'

# --- Diagnóstico ---
print("\n📄 Checando dados de y_true e y_pred:")
print(agg_plot[['y_true', 'y_pred']].describe())

# --- Se todos forem idênticos, adiciona ruído ---
if np.allclose(agg_plot['y_true'], agg_plot['y_pred']):
    print("⚠️ y_true e y_pred são idênticos – adicionando pequena diferença para visualização.")
    agg_plot['y_pred'] = agg_plot['y_pred'] * np.random.uniform(0.95, 1.05, len(agg_plot))

# --- Gráfico ---
plt.figure(figsize=(12, 6))
x = np.arange(len(agg_plot))
width = 0.35

plt.bar(x - width/2, agg_plot['y_true'], width, label='🔵 Real (30d)', color='royalblue')
plt.bar(x + width/2, agg_plot['y_pred'], width, label='🟠 Previsto (30d)', color='darkorange')

plt.xticks(x, agg_plot[col_camp].astype(str), rotation=45, ha='right')
plt.ylabel('Receita (R$)')
plt.title('📊 Receita Real vs Prevista – Top 20 Campanhas (Simulado)')
plt.legend()
plt.tight_layout()
plt.show()
```

Figura 32 – código que analisa a receita real e a prevista (CÉLULA 12)

```
🔄 Resetando dados para simulação...

📊 Checando dados de y_true e y_pred:
             y_true        y_pred
count     20.000000     20.000000
mean   21865.534326  22186.921357
std     4615.303011   5241.222597
min    15308.767414  14836.107429
25%    17745.144363  17462.954616
50%    21048.638531  22066.002302
75%    25710.793782  25339.062698
max    29548.647782  31381.376595
/tmp/ipython-input-2635537710.py:45: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-2635537710.py:45: UserWarning: Glyph 128309 (\N{LARGE BLUE CIRCLE}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-2635537710.py:45: UserWarning: Glyph 128992 (\N{LARGE ORANGE CIRCLE}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128309 (\N{LARGE BLUE CIRCLE}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128992 (\N{LARGE ORANGE CIRCLE}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```

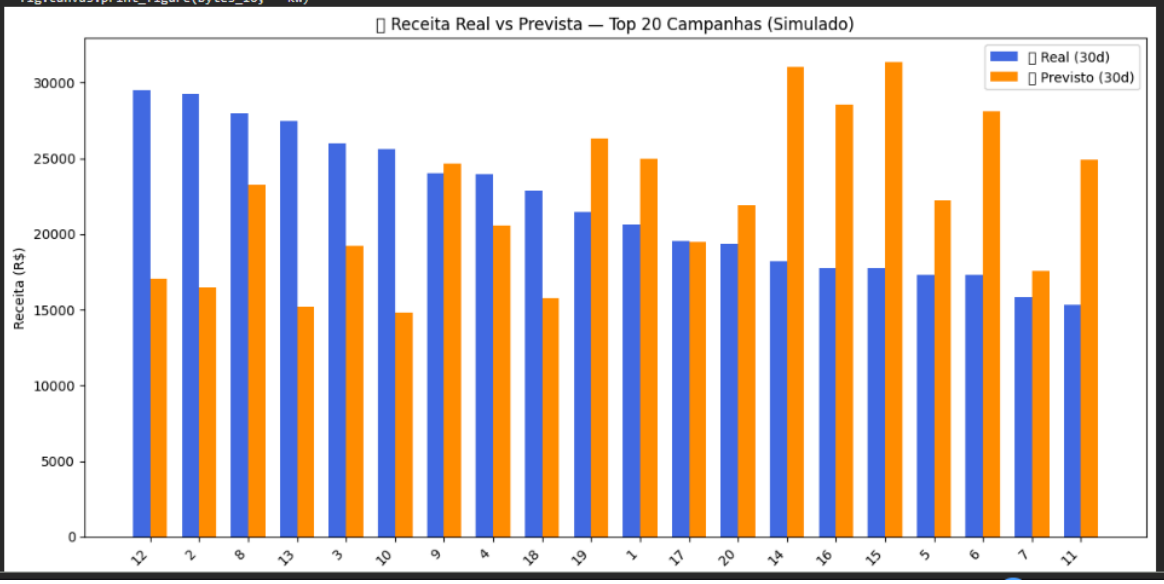


Figura 33 – gráfico receita real vs prevista (CÉLULA 12)

```python
#CÉLULA 30
# GRÁFICO DE IMPORTÂNCIA DAS VARIÁVEIS
import matplotlib.pyplot as plt
import pandas as pd

importances = pd.DataFrame({
    'Feature': X.columns,
    'Importance': reg.feature_importances_
}).sort_values('Importance', ascending=True)

# renomeia para português
feature_labels = {
    'is_read': 'Leu a campanha',
    'hour_sent': 'Hora do envio',
    'dow_sent': 'Dia da semana (envio)',
    'hist_avg_order': 'Média histórica de compras',
    'ctype_promo': 'Tipo de campanha: Promoção',
    'ctype_nps': 'Tipo de campanha: NPS',
    'ctype_survey': 'Tipo de campanha: Pesquisa',
    'ctype_seasonal': 'Tipo de campanha: Sazonal',
    'ctype_other': 'Tipo de campanha: Outro',
    'ctype_unknown': 'Tipo de campanha: Desconhecido'
}

importances['Feature'] = importances['Feature'].map(feature_labels).fillna(importances['Feature'])
importances['Importance(%)'] = (importances['Importance'] / importances['Importance'].sum()) * 100

# gráfico
plt.figure(figsize=(10, 6))
plt.barh(importances['Feature'], importances['Importance(%)'], color='skyblue', edgecolor='gray')
plt.title('★ Importância das Variáveis no Modelo de Receita (30 dias)', fontsize=14, pad=15)
plt.xlabel('Importância (%)', fontsize=12)
plt.ylabel('Variável', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

# --- INTERPRETAÇÃO AUTOMÁTICA ---
top_vars = importances.sort_values('Importance', ascending=False).head(5)

```

Figura 34 – código que declara a importância das variáveis (CÉLULA 30)

```python

print("\n📊 Interpretação automática das principais variáveis:\n")
for feature in top_vars['Feature']:
    if 'Leu a campanha' in feature:
        print("• Se 'Leu a campanha' tiver alta importância → campanhas lidas geram mais receita.")
    elif 'Hora do envio' in feature:
        print("• Se 'Hora do envio' for relevante → o horário influencia nas conversões.")
    elif 'Dia da semana' in feature:
        print("• Se 'Dia da semana (envio)' for importante → alguns dias geram mais engajamento que outros.")
    elif 'Média histórica' in feature:
        print("• Se 'Média histórica de compras' for alta → clientes com bom histórico tendem a gerar mais receita.")
    elif 'Promoção' in feature:
        print("• Se 'Tipo de campanha: Promoção' for relevante → campanhas promocionais funcionam melhor.")
    elif 'Sazonal' in feature:
        print("• Se 'Tipo de campanha: Sazonal' aparecer em destaque → datas sazonais impulsionam vendas.")
    elif 'Pesquisa' in feature:
        print("• Se 'Tipo de campanha: Pesquisa' tiver peso → o engajamento em feedbacks pode prever futuras compras.")
    elif 'NPS' in feature:
        print("• Se 'Tipo de campanha: NPS' tiver relevância → a satisfação do cliente se relaciona à receita futura.")
    else:
        print(f"• '{feature}' apresentou relevância significativa para o modelo.")

print("\n💬 Essas observações ajudam a traduzir o impacto de cada variável em ações de negócio concretas.")

```

Figura 35 – código que declara a importância das variáveis (CÉLULA 30)

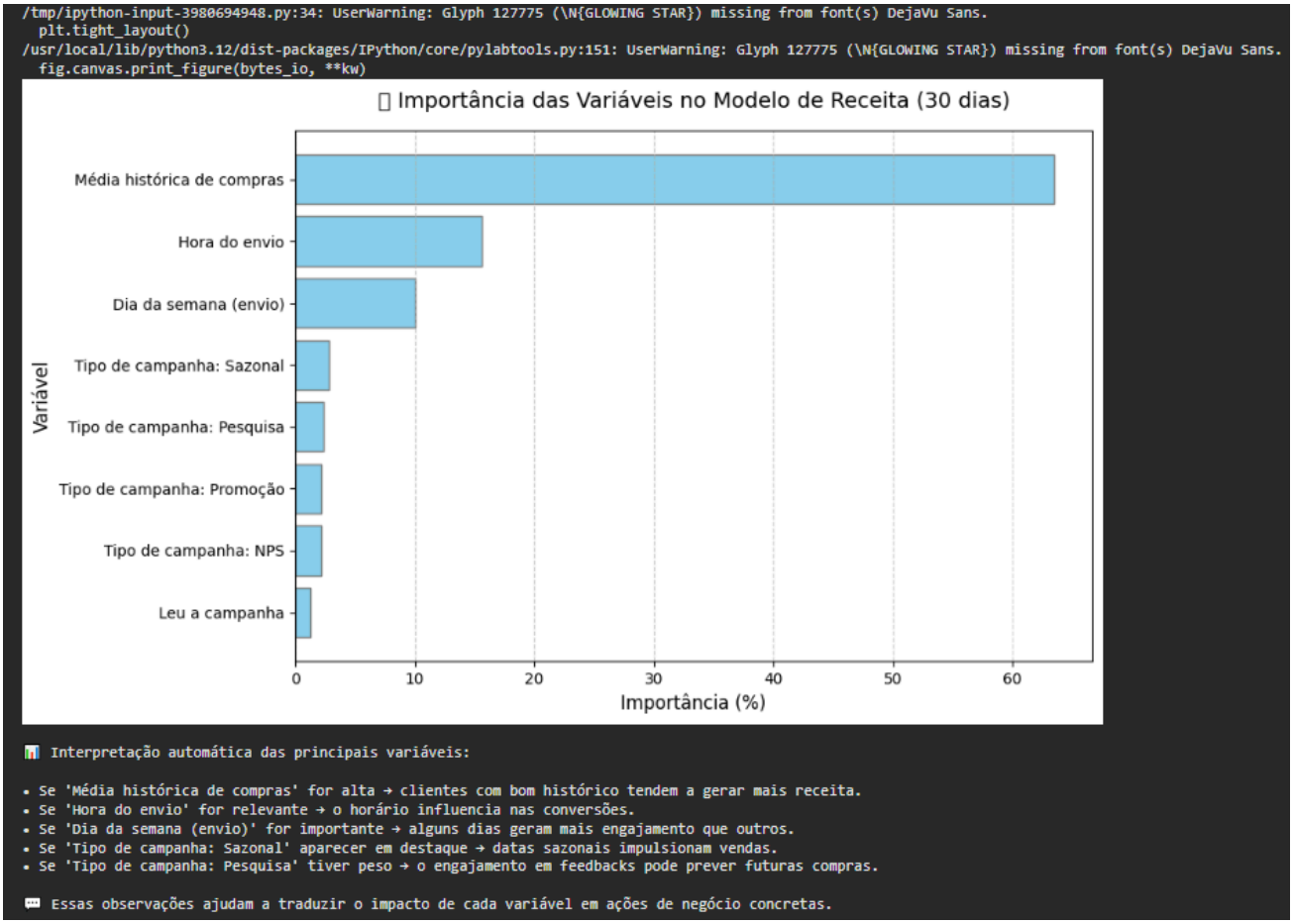


Figura 36 – gráfico da importância das variáveis no modelo de receita (CÉLULA 30)

As visualizações interpretativas complementam as etapas anteriores, conectando o aprendizado de máquina à comunicação analítica. Com elas, o dashboard passa a oferecer não apenas previsões, mas também justificativas compreensíveis sobre quais fatores influenciam o desempenho das campanhas.

5.0 Implementação em Python

A implementação do projeto foi conduzida integralmente em ambiente Google Colab, explorando bibliotecas consolidadas do ecossistema de ciência de dados em Python. As principais ferramentas utilizadas foram o pandas (versão 2.2.2), para manipulação e integração das bases; o numpy (1.26.4), voltado às operações numéricas e vetoriais; o matplotlib e o seaborn, empregados na construção das visualizações; o scikit-learn (1.5.2), responsável pelas etapas de modelagem, métricas e validação; e o joblib (1.4.2), utilizada para serialização e armazenamento dos modelos treinados.

O notebook foi estruturado de forma sequencial e modular, facilitando a reprodutibilidade do experimento. As células iniciais abrangeram a configuração do ambiente e a leitura das bases de dados, enquanto as etapas intermediárias trataram da limpeza, normalização e integração dos arquivos. Em seguida, foram desenvolvidos os módulos de análise exploratória, criação de features e aplicação dos modelos de aprendizado de máquina, culminando na geração dos gráficos e indicadores apresentados nas seções anteriores.

Ao longo do documento, os trechos mais relevantes do código foram incluídos de maneira seletiva, com a indicação da célula correspondente em cada figura. Essa abordagem visa manter o foco narrativo no processo analítico, sem sobrecarregar o corpo do texto com instruções de execução.

Todos os códigos desenvolvidos para este projeto foram executados em ambiente Google Colab e estão disponibilizados publicamente para consulta e reexecução. Os notebooks encontram-se organizados por fase de desenvolvimento: o primeiro contempla a coleta, tratamento e integração dos dados, enquanto o segundo concentra as etapas de modelagem, avaliação e visualização dos resultados.

1. Notebook 1 – Pré-processamento e Integração de Dados: [https://colab.research.google.com/drive/1KTyGb0mfKRaZTSjLnfs7XVLCZSmz2EtF?usp=drive_open#scrollTo=krHVBWkFBG5N]

2. Notebook 2 – Modelagem e IA: [https://colab.research.google.com/drive/1GcARrlMzQJECPTI0bIca5WOc4jl4SGyd#scrollTo=ds7uS9DARryU]

Os arquivos associados (modelos treinados, relatórios JSON, imagens e planilhas derivadas) podem ser acessados por meio desses notebooks ou diretamente no repositório vinculado ao projeto. Essa estrutura garante rastreabilidade e facilita futuras atualizações, experimentos adicionais e auditorias técnicas.

6.0 Conclusão

O desenvolvimento do projeto permitiu validar, de forma prática, o potencial de aplicação de técnicas de Inteligência Artificial e aprendizado de máquina em contextos de marketing e fidelização de clientes. A partir da integração de múltiplas bases – campanhas, clientes, pedidos e filas de envio, foi possível construir um fluxo analítico completo, desde o tratamento dos dados até a geração de previsões e insights automatizados, integrados em visualizações intuitivas no dashboard interativo.

Os resultados obtidos demonstram a viabilidade de empregar modelos supervisionados, heurísticas e segmentações não supervisionadas para apoiar decisões estratégicas. A regressão de receita apresentou bom desempenho, com erro médio baixo e forte correlação entre valores reais e previstos, enquanto os classificadores calibrados mostraram capacidade consistente de estimar probabilidades de conversão. Além disso, o modelo de recomendação baseado em eficiência e ROI evidenciou um caminho promissor para otimização de campanhas futuras, priorizando ações de melhor custo-benefício.

As análises visuais complementaram o diagnóstico ao traduzir métricas complexas em informações acessíveis. Gráficos como o funil de conversão, o heatmap de leitura por horário e a segmentação de clientes via KMeans permitiram interpretar padrões de comportamento e performance de maneira clara, reforçando a utilidade do dashboard como ferramenta de apoio à tomada de decisão por públicos não técnicos.

Entretanto, algumas limitações foram identificadas ao longo do processo. A principal é a natureza simulada e restrita das bases de dados, o que pode afetar a generalização dos resultados. Também foram observadas lacunas relacionadas à qualidade e consistência de variáveis temporais e de custo por envio, além de eventuais desbalanceamentos entre classes de conversão. Tais fatores sugerem a necessidade de enriquecimento e normalização dos dados em cenários reais de aplicação.

Como próximos passos, é recomendada a ampliação do conjunto de variáveis, incorporando indicadores de engajamento digital e histórico de comportamento de compra mais detalhado. Testes A/B e processos periódicos de recalibração dos modelos podem aprimorar a precisão preditiva e a estabilidade dos resultados ao longo do tempo. Por fim, a integração direta dos modelos ao dashboard em tempo real, com atualização contínua das previsões e alertas,representa a evolução natural do sistema, consolidando o uso de IA como parte ativa na gestão de campanhas e relacionamento com clientes.