Introdução

O objetivo deste projeto foi transformar um conjunto de dados brutos de transações de clientes em um asset de dados, pronto para a modelagem de machine learning. O processo envolveu o tratamento, a análise e a engenharia de features a partir da tabela **default.transacoes_clientes**, culminando na criação da tabela **default.clientes_features**.

A partir de dados brutos da **default.transacoes_clientes**, foi executado um pipeline em PySpark que envolveu a correção de tipos de dados, limpeza, e a criação de um conjunto robusto de features comportamentais. A análise exploratória subsequente revelou uma forte assimetria na distribuição dos valores de transação e indicou correlações importantes entre as categorias de gastos e o status de inadimplência.

1. As Transformações Realizadas (O Pipeline de Dados)

O pipeline foi estruturado para garantir a máxima qualidade e relevância dos dados para análise e modelagem.

1.1. Ingestão e Preparação Inicial

Fonte: Os dados foram lidos da tabela Delta **default.transacoes_clientes**.

	id_cliente	valor_transacao	data_transacao	categoria	inadimplente
1	209	57.21	2024-07-01	mercado	0
2	285	19.8	2024-06-15	lazer	0
3	169	19.54	2025-01-04	educacao	0
4	266	49.68	2025-07-28	mercado	0
5	8	9.86	2024-12-07	mercado	0
6	5	54.54	2025-02-13	compras_online	0
7	48	17.08	2024-09-28	lazer	1
8	100	159.16	2024-01-12	transporte	0
9	164	37.41	2024-07-10	null	1
10	197	47	2024-03-21	restaurante	0

Correção de Tipos (Type Casting):

A primeira análise do *schema* mostrou a necessidade de ajustes. A operação de cast foi aplicada para converter colunas para tipos mais apropriados, como *Int* para **id_cliente** e **inadimplente**, e *Float* para **valor_transacao**. Essa etapa é fundamental para otimizar o uso de memória e garantir a precisão em operações aritméticas e agregações. O schema foi transformado da seguinte forma:

- *Antes:* **id_cliente**: *long*, **valor_transacao**: *double*, **inadimplente**: *long*
- *Depois:* **id_cliente**: *int*, **valor_transacao**: *float*, **inadimplente**: *int*

```
root
 |-- id_cliente: integer (nullable = true)
 |-- valor_transacao: float (nullable = true)
 |-- data_transacao: date (nullable = true)
 |-- categoria: string (nullable = true)
 |-- inadimplente: integer (nullable = true)
```

1.2. Qualidade e Transformação dos Dados

Tratamento de Valores Ausentes: A estratégia de imputação foi aplicada para preservar o máximo de informações:

Para **valor_transacao**, valores nulos foram substituídos por **0.0**. A justificativa é que um valor nulo pode representar uma transação falha, estornada ou sem custo, onde o valor zero é a representação financeira mais adequada.

Para **categoria**, valores nulos foram preenchidos com a string "**desconhecida**". Isso evita a perda de registros e cria um grupo específico que pode ser analisado para identificar possíveis falhas na categorização ou padrões em transações não classificadas.

Remoção de Duplicatas: Utilizando **.dropDuplicates()**, foram eliminados todos os registros que eram idênticos em todas as colunas. Este passo é vital para a integridade das análises, garantindo que cada transação seja contada apenas uma vez em cálculos de frequência, soma e médias.

7 hours ago (1s) 7

```
df_no_nulls = df_typed.fillna({
    "valor_transacao": 0.0,
    "categoria": "desconhecida"
})

display(df_no_nulls.limit(100))
```

See performance (1)

df_no_nulls: pyspark.sql.connect.dataframe.DataFrame = [id_cliente: integer, valor_transacao: float ... 3 more fields]

Table +

	id_cliente	valor_transacao	data_transacao	categoria	inadimplente
1	209	57.209999084472656	2024-07-01	mercado	0
2	285	19.799999237060547	2024-06-15	lazer	0
3	169	19.540000915527344	2025-01-04	educacao	0
4	266	49.68000030517578	2025-07-28	mercado	0
5	8	9.859999656677246	2024-12-07	mercado	0
6	5	54.540000915527344	2025-02-13	compras_online	0
7	48	17.079999923706055	2024-09-28	lazer	1
8	100	159.16000366210938	2024-01-12	transporte	0
9	164	37.40999984741211	2024-07-10	desconhecida	1
10	197	47	2024-03-21	restaurante	0
11	106	27.219999313354492	2024-09-21	mercado	0
12	188	6.699999809265137	2024-06-07	transporte	0
13	227	11.829999923706055	2024-12-27	restaurante	0
14	259	49.88999938964844	2024-08-08	restaurante	0
15	89	11.75	2025-04-20	desconhecida	0

7 hours ago (1s) 9

```
df_no_dupes = df_no_nulls.dropDuplicates()

from pyspark.sql.functions import count

df_no_dupes.groupBy("id_cliente", "data_transacao", "valor_transacao", "categoria") \
    .agg(count("*").alias("qtd")) \
    .filter("qtd > 1") \
    .show()
```

See performance (1)

df_no_dupes: pyspark.sql.connect.dataframe.DataFrame = [id_cliente: integer, valor_transacao: float ... 3 more fields]

```
+----------+--------------+---------------+---------+---+
|id_cliente|data_transacao|valor_transacao|categoria|qtd|
+----------+--------------+---------------+---------+---+
+----------+--------------+---------------+---------+---+
```

1.3. Engenharia de Features

A criação de novas variáveis (features) foi realizada em duas fases para enriquecer o dataset.

Features Agregadas Iniciais:

- **num_transacoes:** Criada a partir de uma operação **groupBy("id_cliente").agg(count("*"))**, esta feature serve como um indicador primário do volume de atividade de cada cliente.
- **media_mensal:** Calculada agrupando por cliente e **ano_mes** (coluna de apoio), essa feature ajuda a normalizar os gastos e identificar o comportamento financeiro médio mensalmente.

```
from pyspark.sql.functions import year, month, concat_ws

# Criar coluna ano_mes como "YYYY-MM"
df_with_ano_mes = df_with_num_transacoes.withColumn(
    "ano_mes",
    concat_ws("-", year("data_transacao"), month("data_transacao"))
)

df_with_ano_mes.show(100)
```

```
from pyspark.sql.functions import avg

df_media_mensal = df_with_ano_mes.groupBy("id_cliente", "ano_mes") \
                                 .agg(avg("valor_transacao").alias("media_mensal"))

df_media_mensal.show(100)
```

Features Avançadas para Modelagem: Um conjunto final de features foi consolidado na tabela **default.clientes_features**.

```
spark.table("default.transacoes_clientes").printSchema()
display(spark.table("default.transacoes_clientes").limit(10))
```

> See performance (2)

```
root
 |-- id_cliente: integer (nullable = true)
 |-- ano_mes: string (nullable = true)
 |-- valor_transacao: float (nullable = true)
 |-- data_transacao: date (nullable = true)
 |-- categoria: string (nullable = true)
 |-- inadimplente: integer (nullable = true)
 |-- num_transacoes: integer (nullable = true)
 |-- media_mensal: float (nullable = true)
```

Estas features foram projetadas para capturar diferentes dimensões do comportamento do cliente:

Recência, Frequência e Valor (RFM-like):

- **recencia**: Calculada como a diferença em dias entre a data da última transação no dataset e a última compra do cliente **(datediff(max_date, ultima_compra))**. É um forte indicador de quão ativo um cliente está.
- **total_gasto_ultimo_mes** e q**td_transacoes_ultimo_mes**: Medem o comportamento de curto prazo, filtrando transações nos últimos 30 dias.
- **ticket_medio**: Média de **valor_transacao** por cliente, indicando o perfil de gasto por compra.

Codificação Categórica (One-Hot Encoding):

A coluna categoria foi transformada utilizando uma operação de pivot. Isso resultou em um formato "*wide*", onde cada categoria de gasto se tornou uma coluna (ex: **cat_mercado, cat_lazer**) e o valor da célula representa a contagem de transações do cliente naquela categoria.

```python
from pyspark.sql import functions as F
from pyspark.sql.window import Window

df = spark.table("default.transacoes_clientes")

#Total gasto no último mês
max_date = df.agg(F.max("data_transacao")).collect()[0][0]

df_total_mes = (
    df.filter(F.col("data_transacao") >= F.date_sub(F.lit(max_date), 30))
      .groupBy("id_cliente")
      .agg(F.sum("valor_transacao").alias("total_gasto_ultimo_mes"))
)

#Qtd de transações no último mês
df_qtd_mes = (
    df.filter(F.col("data_transacao") >= F.date_sub(F.lit(max_date), 30))
      .groupBy("id_cliente")
      .agg(F.count("*").alias("qtd_transacoes_ultimo_mes"))
)

#Recência (desde a ultima compra)
df_recencia = (
    df.groupBy("id_cliente")
      .agg(F.max("data_transacao").alias("ultima_compra"))
      .withColumn("recencia", F.datediff(F.lit(max_date), F.col("ultima_compra")))
      .drop("ultima_compra")
)

#Ticket Médio
df_ticket = (
    df.groupBy("id_cliente")
      .agg((F.sum("valor_transacao")/F.count("*")).alias("ticket_medio"))
)
```

```python
#Diversidade de Categorias
df_diversidade = (
    df.groupBy("id_cliente")
      .agg(F.countDistinct("categoria").alias("diversidade_categorias"))
)

df_features = df_total_mes \
    .join(df_qtd_mes, "id_cliente", "left") \
    .join(df_recencia, "id_cliente", "left") \
    .join(df_ticket, "id_cliente", "left") \
    .join(df_diversidade, "id_cliente", "left")
```

```python
from pyspark.sql import functions as F

df = spark.table("default.transacoes_clientes")

df_categoria_counts = df.groupBy("id_cliente", "categoria") \
                        .agg(F.count("*").alias("qtd"))

# Pivot p/ o One-Hot Encoding
df_one_hot = df_categoria_counts.groupBy("id_cliente") \
                                .pivot("categoria") \
                                .agg(F.first("qtd")) \
                                .fillna(0)

# Renomear com "cat_"
for col_name in df_one_hot.columns:
    if col_name != "id_cliente":
        df_one_hot = df_one_hot.withColumnRenamed(col_name, "cat_" + col_name)

display(df_one_hot)
```

2. Insights dos Dados (Análise Exploratória)

2.1. Perfil Geral das Transações

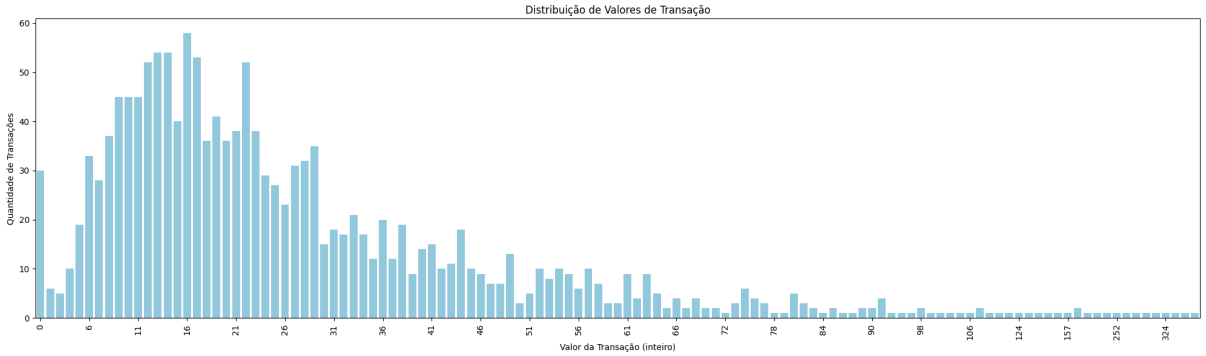
As estatísticas descritivas revelam um perfil de transação bem definido:

```python
# Pega as estatísticas descritivas
stats = df_final_typed.select("valor_transacao", "num_transacoes", "media_mensal").describe().collect()

# Mapeia os nomes em inglês para português
traducao = {
    "count": "Contagem",
    "mean": "Média",
    "stddev": "Desvio Padrão",
    "min": "Mínimo",
    "max": "Máximo"
}

# Mostra em português
for row in stats:
    estatistica = traducao[row["summary"]]
    print(f"\n{estatistica}:")
    print(f"Valor Transação: {row['valor_transacao']}")
    print(f"Número de Transações: {row['num_transacoes']}")
    print(f"Média Mensal: {row['media_mensal']}")
```

```
Média Mensal: 1500

Média:
Valor Transação: 29.324273339271546
Número de Transações: 6.072
Média Mensal: 29.324273358503977

Desvio Padrão:
Valor Transação: 31.921562652328543
Número de Transações: 2.3346198048266293
Média Mensal: 29.66522232957608

Mínimo:
Valor Transação: 0.0
Número de Transações: 1
Média Mensal: 0.0

Máximo:
Valor Transação: 395.78
Número de Transações: 13
Média Mensal: 395.78
```

O valor médio por transação é de *R$ 29,32*. No entanto, essa média é acompanhada por um alto desvio padrão de R$ 31,92, com valores que variam de *R$ 0,00* até um máximo de *R$ 395,78*. Essa disparidade indica uma forte assimetria nos dados, onde a maioria das transações é de baixo valor, mas uma minoria de transações de alto valor eleva a média. A análise de frequência confirma este padrão de cauda longa, mostrando uma altíssima concentração de operações em valores inferiores a *R$ 50*, com a frequência diminuindo drasticamente à medida que o valor aumenta. Este comportamento é típico de dados de varejo, refletindo que a ***maior parte da atividade dos clientes é composta por compras de baixo custo***.

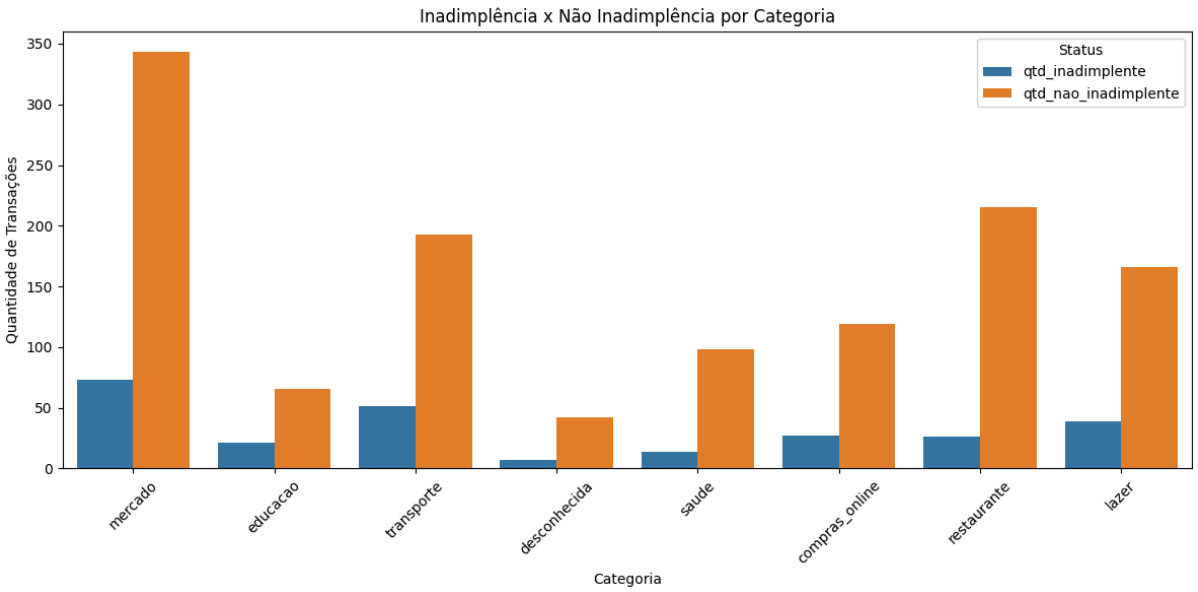
Em relação ao engajamento, os clientes realizaram uma média de *6.07* transações no período analisado. A contagem variou de clientes com uma única transação até os mais ativos, com *13* transações, demonstrando diferentes níveis de fidelidade e atividade na plataforma.



Análise de Inadimplência por Categoria

Para entender a relação entre o comportamento de consumo e o risco, foi realizada uma análise que quantifica as transações de clientes inadimplentes e não inadimplentes dentro de cada categoria. ***A principal descoberta é que o risco de inadimplência não é distribuído de forma homogênea.*** Observou-se que categorias associadas a despesas essenciais e recorrentes, como 'mercado' e 'transporte', tendem a ter uma proporção menor de transações de inadimplentes. Em contrapartida, ***categorias ligadas a gastos de maior valor ou de caráter não essencial, como 'educação', 'saúde' e 'lazer', demonstram uma proporção relativamente maior de inadimplência.*** Este insight é de alto valor estratégico, pois sugere que o perfil de consumo de um cliente pode ser um forte indicador preditivo de seu risco financeiro.

```python
df = spark.table("default.transacoes_clientes")

# Agrupar por categoria e calcular quantidades
df_categoria_qtd = df.groupBy("categoria").agg(
    sum(col("inadimplente")).alias("qtd_inadimplente"),
    (count("*") - sum(col("inadimplente"))).alias("qtd_nao_inadimplente")
)

# Converter para Pandas
pdf_categoria_qtd = df_categoria_qtd.toPandas()

pdf_long = pdf_categoria_qtd.melt(
    id_vars="categoria",
    value_vars=["qtd_inadimplente", "qtd_nao_inadimplente"],
    var_name="Status",
    value_name="Quantidade"
)

# Plot
plt.figure(figsize=(12,6))
sns.barplot(data=pdf_long, x="categoria", y="Quantidade", hue="Status")
plt.ylabel("Quantidade de Transações")
plt.xlabel("Categoria")
plt.title("Inadimplência x Não Inadimplência por Categoria")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



3. Limitações e Sugestões de Melhorias

3.1. Limitações Atuais

Dados Demográficos: A principal limitação é a ausência de dados cadastrais dos clientes (ex: idade, renda, localização, profissão, score de crédito externo). Essas informações são, historicamente, as mais preditivas em modelos de risco.

Definição de Inadimplência: A flag inadimplente é estática e aplicada a todas as transações de um cliente. Isso não captura a dinâmica temporal, ou seja, quando um cliente se tornou inadimplente. Um cliente pode ter um longo histórico de bom pagador antes de se tornar um risco.

Imputação de Nulos Simplificada: Assumir valor **0.0** para transações nulas é uma premissa que pode ser questionada. Uma análise mais detalhada da origem desses nulos poderia levar a uma estratégia de imputação mais sofisticada.

3.2. Sugestões de Melhorias e Próximos Passos

- Modelagem de Séries Temporais: Tratar o problema como uma análise de sobrevivência ou série temporal. Em vez de prever um status estático, o objetivo seria prever a probabilidade de inadimplência nos próximos X meses, com base no histórico de transações até o momento.
- *Features de Interação e Tendência*: Criar features mais complexas, como a variação da média de gastos entre diferentes meses, ou a proporção do gasto em categorias de "risco" (identificadas na *EDA*) sobre o gasto total.
- *Construção do Modelo Preditivo*: Avançar para a etapa de modelagem, utilizando a tabela de features para treinar e validar múltiplos algoritmos (ex: Regressão Logística, Gradient Boosting, Redes Neurais) e implementar o campeão em um fluxo de produção.

Observações Finais:

Durante o desenvolvimento no *Databricks Free Edition*, foi necessário adaptar as técnicas de manipulação de dados em função das limitações da plataforma. O acesso ao DBFS público é desabilitado, sendo assim, com suas limitações e forçando a usar *Delta Tables*, tendo também de usar a interface gráfica pra subir o ***.csv*** por meio do "*Upload por UI*". Como os *DataFrames* utilizados são do *Spark*, que opera de forma distribuída, não foi possível usar diretamente bibliotecas tradicionais do Python, como *seaborn* ou *scikit-learn*. Para usá-las, seria necessário converter os dados para um *DataFrame* do *Pandas*, o que

poderia comprometer o desempenho ou até inviabilizar a execução em casos de grandes volumes de dados, **por conta da quantidade de memória limitada disponibilizada no único cluster que o *Databricks Free Edition* deixa disponível.**

Dessa forma, escolhi empregar funções nativas do *Spark*, como *groupBy* e *pivot*, que permitem realizar agregações, transformações e análises diretamente no motor de processamento distribuído. Essa abordagem garantiu a execução eficiente das etapas do projeto, além de manter a compatibilidade com o ambiente disponível no *Databricks Free Edition*. Além disso, o *Databricks Free Edition*, também não permite, em sua segurança/privacidade, o compartilhamento de *Workspaces* por link/convite na plataforma, essas são features da versão paga da plataforma.