

1. Configuração e inicialização Spark

1.1. Máquina local

```
In [2]: import warnings
warnings.filterwarnings('ignore')

import pyspark.sql.functions as F
from pyspark.sql import types as T
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import math

import findspark
findspark.init()

from pyspark.sql import SparkSession

from math import isnan, isinf
from pyspark.sql.window import Window
from pyspark.ml import Pipeline
from pyspark.ml.stat import ChiSquareTest, Correlation
from pyspark.ml.feature import StringIndexer, VectorAssembler, QuantileDiscretizer, OneHotEncoder, StandardScaler
from pyspark.sql.types import DecimalType, StringType, IntegerType, DoubleType, FloatType
from scipy import stats
```

```
In [3]: from pyspark.sql import SparkSession

spark = (
    SparkSession.builder
    .appName("datamaster_case_gustavo")
    .master("local[*]")
    .config("spark.driver.memory", "24g")
    .config("spark.driver.maxResultSize", "8g")
    .config("spark.executor.memory", "4g")
    .config("spark.executor.cores", "4")
    .config("spark.sql.shuffle.partitions", "200")
    .config("spark.sql.adaptive.enabled", "true")
    .config("spark.sql.adaptive.coalescePartitions.enabled", "true")
    .config("spark.sql.execution.arrow.pyspark.enabled", "false")
    .getOrCreate()
)
```

1.2. Google Colab

```
In [1]: # 1. Instala Python 3.11 e Java 11
!apt-get update -y
!apt-get install -y python3.11 python3.11-distutils openjdk-11-jdk-headless

# 2. Instala o PIP para o Python 3.11
!curl -sS https://bootstrap.pypa.io/get-pip.py | python3.11

# 3. Instala PySpark e Findspark no Python 3.11
!python3.11 -m pip install -q pyspark findspark

Get:1 https://cli.github.com/packages stable InRelease [3,917 B]
Get:2 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease [3,632 B]
Get:3 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ Packages [85.0 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:5 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:6 https://r2u.stat.illinois.edu/ubuntu jammy InRelease [6,555 B]
Get:7 https://cli.github.com/packages stable/main amd64 Packages [356 B]
Get:8 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:9 https://r2u.stat.illinois.edu/ubuntu jammy/main all Packages [9,712 kB]
Get:10 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease [18.1 kB]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [62.6 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:13 https://ppa.launchpadcontent.net/ubuntujamvis/ppa/ubuntu jammy InRelease [24.6 kB]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [6,388 kB]
Get:15 https://r2u.stat.illinois.edu/ubuntu jammy/main amd64 Packages [2,892 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [70.9 kB]
Get:17 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main amd64 Packages [38.8 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [4,014 kB]
Get:19 https://ppa.launchpadcontent.net/ubuntujamvis/ppa/ubuntu jammy/main amd64 Packages [75.3 kB]
Get:20 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [6,607 kB]
Get:21 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [3,677 kB]
Get:22 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [1,297 kB]
Get:23 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1,608 kB]
Fetched 37.0 MB in 4s (10.3 MB/s)
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu jammy
InRelease' does not seem to provide it (sources.list entry misspelt?)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'python3-distutils' instead of 'python3.11-distutils'
python3-distutils is already the newest version (3.10.8-1~22.04).
python3-distutils set to manually installed.
The following additional packages will be installed:
  libpython3.11-minimal libpython3.11-stdlib openjdk-11-jre-headless
  python3.11-minimal
Suggested packages:
  openjdk-11-demo openjdk-11-source libnss-mdns fonts-dejavu-extra
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei
  | fonts-wqy-zenhei fonts-indic python3.11-venv binfmt-support
The following NEW packages will be installed:
  libpython3.11-minimal libpython3.11-stdlib openjdk-11-jdk-headless
  openjdk-11-jre-headless python3.11 python3.11-minimal
0 upgraded, 6 newly installed, 0 to remove and 54 not upgraded.
Need to get 122 MB of archives.
After this operation, 279 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 openjdk-11-jre-headless amd64 11.0.30+7-1ubuntu1~22.04 [42.6
MB]
Get:2 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main amd64 libpython3.11-minimal amd64 3.11.14-1+jammy1
[887 kB]
Get:3 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main amd64 python3.11-minimal amd64 3.11.14-1+jammy1
[2,358 kB]
Get:4 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main amd64 libpython3.11-stdlib amd64 3.11.14-1+jammy1
[1,927 kB]
Get:5 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main amd64 python3.11 amd64 3.11.14-1+jammy1 [94.3 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 openjdk-11-jdk-headless amd64 11.0.30+7-1ubuntu1~22.04 [73.7
MB]
Fetched 122 MB in 9s (13.3 MB/s)
Selecting previously unselected package libpython3.11-minimal:amd64.
(Reading database ... 117540 files and directories currently installed.)
Preparing to unpack .../0-libpython3.11-minimal_3.11.14-1+jammy1_amd64.deb ...
Unpacking libpython3.11-minimal:amd64 (3.11.14-1+jammy1) ...
Selecting previously unselected package python3.11-minimal.
Preparing to unpack .../1-python3.11-minimal_3.11.14-1+jammy1_amd64.deb ...
Unpacking python3.11-minimal (3.11.14-1+jammy1) ...
Selecting previously unselected package libpython3.11-stdlib:amd64.
Preparing to unpack .../2-libpython3.11-stdlib_3.11.14-1+jammy1_amd64.deb ...
Unpacking libpython3.11-stdlib:amd64 (3.11.14-1+jammy1) ...
Selecting previously unselected package openjdk-11-jre-headless:amd64.
Preparing to unpack .../3-openjdk-11-jre-headless_11.0.30+7-1ubuntu1~22.04_amd64.deb ...
Unpacking openjdk-11-jre-headless:amd64 (11.0.30+7-1ubuntu1~22.04) ...
```

```
In [2]: import os

# Aponta para o Java 11 que instalamos
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"

# FORÇA o PySpark a usar o executável do Python 3.11
os.environ["PYSPARK_PYTHON"] = "python3.11"
os.environ["PYSPARK_DRIVER_PYTHON"] = "python3.11"

# Adiciona o Java ao PATH
os.environ["PATH"] = "/usr/lib/jvm/java-11-openjdk-amd64/bin:" + os.environ["PATH"]

print("Configurações aplicadas para Python 3.11")
!python3.11 --version
!java -version
```

```
Configurações aplicadas para Python 3.11
Python 3.11.14
openjdk version "11.0.30" 2026-01-20
OpenJDK Runtime Environment (build 11.0.30+7-post-Ubuntu-1ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.30+7-post-Ubuntu-1ubuntu122.04, mixed mode, sharing)
```

```
In [6]: import os
import subprocess
from pyspark.sql import SparkSession

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["PATH"] = "/usr/lib/jvm/java-11-openjdk-amd64/bin:" + os.environ["PATH"]
os.environ["PYSPARK_PYTHON"] = "python3.11"
os.environ["PYSPARK_DRIVER_PYTHON"] = "python3.11"

try:
    spark = (
        SparkSession.builder
        .appName("teste_minimo_2g")
        .master("local[2]") # só 2 cores
        .config("spark.driver.memory", "2g") # memória MÍNIMA
        .getOrCreate()
    )
    print("✅ FUNCIONOU com 2GB!")
    display(spark)
except Exception as e:
    print("❌ Ainda deu erro com 2GB:")
    print(e)
```

❌ Ainda deu erro com 2GB:
[JAVA_GATEWAY_EXITED] Java gateway process exited before sending its port number.

```
In [7]: import findspark
findspark.init()

from pyspark.sql import SparkSession
import os

# Reforçando as variáveis de ambiente dentro da célula de inicialização
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["PATH"] = "/usr/lib/jvm/java-11-openjdk-amd64/bin:" + os.environ["PATH"]

try:
    spark = (
        SparkSession.builder
        .appName("datamaster_case_gustavo")
        .master("local[*]")
        .config("spark.driver.memory", "32g")
        .config("spark.sql.shuffle.partitions", "24")
        .config("spark.sql.execution.arrow.pyspark.enabled", "true")
        .config("spark.driver.maxResultSize", "4g")
        .getOrCreate()
    )
    print("✅ SUCESSO: O Spark foi conectado!")
    display(spark)
except Exception as e:
    print("❌ ERRO CRÍTICO:")
    print(e)
```

❌ ERRO CRÍTICO:
[JAVA_GATEWAY_EXITED] Java gateway process exited before sending its port number.

```
In [3]: # Verifica a memória RAM total disponível
from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print(f"Sua máquina tem {ram_gb:.2f} GB de RAM total.")

# Verifica a quantidade de núcleos de CPU
import multiprocessing
cores = multiprocessing.cpu_count()
print(f"Sua máquina tem {cores} núcleos de CPU.")
```

Sua máquina tem 54.75 GB de RAM total.
Sua máquina tem 8 núcleos de CPU.

```
In [7]: # CUIDADO: Isso encerra a sessão e desliga a máquina imediatamente.
# Use apenas quando tiver certeza que o processamento pesado acabou.
from google.colab import runtime
runtime.unassign()
```

```
In [ ]:
import pyspark.sql.functions as F
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

import findspark
findspark.init()

from pyspark.sql import SparkSession

from math import isnan, isinf
from pyspark.sql.window import Window
from pyspark.ml import Pipeline
from pyspark.ml.stat import Correlation, ChiSquareTest
from pyspark.ml.feature import StringIndexer, VectorAssembler, QuantileDiscretizer, OneHotEncoder, StandardScaler
from pyspark.sql.types import DecimalType, StringType, IntegerType, DoubleType, FloatType
from scipy import stats
```

2. Carregando bases iniciais

```
In [3]:
base_path = r"C:\Users\Gustavo\Downloads\datamaster\dados\parquet" # ajuste para o seu caminho

df_logs = spark.read.parquet(f"{base_path}/user_logs.parquet")
df_transactions = spark.read.parquet(f"{base_path}/transactions.parquet")
df_members = spark.read.parquet(f"{base_path}/members.parquet")
```

3. Funções Utilizadas

3.1. Análises gerais

```
In [3]:
def verificar_duplicatas(df, col_id="msno", col_safra="safra", visualizar=False):
    """
    Identifica e conta linhas duplicadas para a mesma chave (ID + Safra).
    """

    # Define a janela abrangendo todas as linhas do grupo
    window_spec = Window.partitionBy(col_id, col_safra)\n        .rowsBetween(Window.unboundedPreceding, Window.unboundedFollowing)

    # Adiciona contagem e filtra duplicados
    df_duplicados = df.withColumn("contagem_janela", F.count("*").over(window_spec)).filter("contagem_janela > 1")

    if visualizar:
        df_duplicados.show(10)

    # Retorna a quantidade total de linhas duplicadas encontradas
    return df_duplicados.count()
```

```
In [4]:
def verificar_mudanca_estado(df, col_alvo, col_id="msno", col_safra="safra", visualizar=False):
    """
    Verifica mudanças de estado em uma variável ao longo do tempo (safras).
    """

    # Definir a especificação da janela
    window_spec = Window.partitionBy(col_id).orderBy(col_safra)

    # Lag para criar as colunas de estado anterior da variável e detecção de mudança
    df_historico = (
        df.withColumn(f"{col_alvo}_anterior", F.lag(col_alvo).over(window_spec))
        .withColumn("mudou",
                   F.when(F.col(f'{col_alvo}_anterior').isNull(), False)
                   .otherwise(F.col(col_alvo) != F.col(f'{col_alvo}_anterior'))))

    # Caso o usuário deseje visualizar as mudanças
    if visualizar:
        # Mostrar quantas mudanças ocorreram por safra
        print("Contagem de mudanças por safra:")
        df_historico.filter(F.col("mudou") == True).groupBy(col_safra, "mudou").count().orderBy(col_safra, "mudou").show(30,
truncate=False)
        # Mostrar quantas vezes o usuário mudou
        print("Mudanças por usuário:")
        df_historico.filter(F.col("mudou") ==
True).groupBy(col_id).agg(F.sum(F.col("mudou").cast("int")).alias("total_mudancas")).show(10, truncate=False)
```

```
In [5]: def calcular_distribuicao(df, colunas_alvo, col_id="msno", n_show=10, agrupar_por_safra=False):
    """
    Gera agregação de contagem e percentual sobre uma variável alvo.
    """
    # 1. Calcula o total da base para o percentual global
    total_base = df.count()

    if agrupar_por_safra:
        # 2. Agrupamento inicial por safra e alvo
        df_result = df.groupBy("safra", *colunas_alvo).agg(
            F.count(col_id).alias("total")
        )

        # 3. Define a janela para calcular o total por safra (denominador do pct_safra)
        window_safra = Window.partitionBy("safra")

        # 4. Adiciona os cálculos de percentual
        df_result = df_result.withColumn(
            "pct_safra",
            F.round((F.col("total") / F.sum("total").over(window_safra)) * 100, 2)
        ).withColumn(
            "pct_total",
            F.round((F.col("total") / F.lit(total_base)) * 100, 2)
        ).orderBy("safra", F.desc("total"))

    else:
        # Resultado geral sem partição de safra
        df_result = df.groupBy(colunas_alvo).agg(
            F.count(col_id).alias("total"),
            F.round((F.count(col_id) / F.lit(total_base)) * 100, 2).alias("pct_total")
        ).orderBy(F.desc("total"))

    df_result.show(n_show, truncate=False)
    return df_result # Retornar o DF é uma boa prática para análises posteriores
```

```
In [6]: def correlation_matrix(df, colunas, plot=True, top_n=15, figsize=(20, 8)):
    """
    Calcula e visualiza matriz de correlação com layout otimizado.

    Parâmetros:
    -----
    df : Spark DataFrame
    colunas : Lista com nomes das colunas numéricas
    plot : bool - Se True, gera visualização
    top_n : int - Número máximo de features a exibir (ordenadas por variância ou IV)
    figsize : tuple - Tamanho da figura
    """

    n = len(colunas)

    # =====
    # CASO 1: Par de variáveis (Scatter Plot)
    # =====
    if n == 2:
        col1, col2 = colunas

        pearson_val = df.corr(col1, col2, method="pearson")
        spearman_val = df.corr(col1, col2, method="spearman")

        print(f"📊 Correlação de Pearson: {pearson_val:+.4f}")
        print(f"📊 Correlação de Spearman: {spearman_val:+.4f}")

        if plot:
            amostra_pd = df.select(col1, col2).sample(
                False,
                fraction=min(1.0, 50000/df.count())
            ).toPandas()

            fig, ax = plt.subplots(figsize=(10, 7))

            # Scatter com densidade
            scatter = ax.scatter(
                amostra_pd[col1],
                amostra_pd[col2],
                alpha=0.4,
                s=20,
                c=amostra_pd[col1],
                cmap='viridis',
                edgecolors='none'
            )

            # Linha de tendência
            z = np.polyfit(amostra_pd[col1], amostra_pd[col2], 1)
            p = np.poly1d(z)
            ax.plot(
                amostra_pd[col1].sort_values(),
                p(amostra_pd[col1].sort_values()),
                "r--",
                alpha=0.8,
                linewidth=2,
                label=f'Tendência: y = {z[0]:.3f}x + {z[1]:.3f}'
            )

            ax.set_xlabel(col1, fontsize=12, fontweight='bold')
            ax.set_ylabel(col2, fontsize=12, fontweight='bold')
            ax.set_title(
                f"Dispersão: {col1} vs {col2}\n"
                f"Pearson: {pearson_val:+.3f} | Spearman: {spearman_val:+.3f}",
                fontsize=14,
                fontweight='bold',
                pad=20
            )
            ax.grid(True, alpha=0.3, linestyle='--')
            ax.legend(loc='best', fontsize=10)

            plt.colorbar(scatter, ax=ax, label=col1)
            plt.tight_layout()
            plt.show()
            plt.close()

    # =====
    # CASO 2: Matriz de correlação (> 2 variáveis)
    # =====
    else:
        # Limitar número de features se necessário
        if n > top_n:
            print(f"⚠️ Muitas features ({n}). Exibindo apenas top {top_n} por variância.")

        # Calcular variância de cada coluna
        variancias = []
        for col in colunas:
            var = df.select(F.variance(col)).collect()[0][0]
            variancias.append((col, var if var else 0))
```

```

# Ordenar por variância e pegar top_n
colunas_top = [col for col, _ in sorted(variancias, key=lambda x: x[1], reverse=True)[:top_n]]
else:
    colunas_top = colunas

# Preparar vetor
assembler = VectorAssembler(
    inputCols=colunas_top,
    outputCol="features",
    handleInvalid="skip"
)
df_vector = assembler.transform(df).select("features")

# Calcular matrizes
def get_corr_matrix(method):
    matrix = Correlation.corr(df_vector, "features", method=method).collect()[0][0]
    return pd.DataFrame(matrix.toArray(), index=colunas_top, columns=colunas_top)

print("Calculando matriz de Pearson...")
matrix_pearson = get_corr_matrix("pearson")

print("Calculando matriz de Spearman...")
matrix_spearman = get_corr_matrix("spearman")

if plot:
    # =====
    # VISUALIZAÇÃO OTIMIZADA
    # =====

    # Truncar nomes longos para melhor legibilidade
    labels_curtos = [
        col[:30] + '...' if len(col) > 30 else col
        for col in colunas_top
    ]

    fig, axes = plt.subplots(1, 2, figsize=figsize)

    # ----- PEARSON -----
    mask_pearson = np.triu(np.ones_like(matrix_pearson, dtype=bool), k=1)

    sns.heatmap(
        matrix_pearson,
        annot=True,
        fmt=".2f",
        cmap='RdBu_r', # Vermelho = positivo, Azul = negativo
        center=0,
        vmin=-1,
        vmax=1,
        square=True,
        linewidths=0.5,
        cbar_kws={
            "shrink": 0.8,
            "label": "Correlação"
        },
        ax=axes[0],
        xticklabels=labels_curtos,
        yticklabels=labels_curtos,
        annot_kws={"size": 8}
    )

    axes[0].set_title(
        "Matriz de Pearson\n(Correlação Linear)",
        fontsize=14,
        fontweight='bold',
        pad=15
    )
    axes[0].set_xticklabels(
        axes[0].get_xticklabels(),
        rotation=45,
        ha='right',
        fontsize=9
    )
    axes[0].set_yticklabels(
        axes[0].get_yticklabels(),
        rotation=0,
        fontsize=9
    )

    # ----- SPEARMAN -----
    mask_spearman = np.triu(np.ones_like(matrix_spearman, dtype=bool), k=1)

    sns.heatmap(
        matrix_spearman,
        annot=True,
        fmt=".2f",
        cmap='PiYG', # Rosa/Verde para Spearman
        center=0,

```

```

vmin=-1,
vmax=1,
square=True,
lineweights=0.5,
cbar_kws={
    "shrink": 0.8,
    "label": "Correlaç&atilde;o"
},
ax=axes[1],
xticklabels=labels_curtos,
yticklabels=labels_curtos,
annot_kws={"size": 8}
)

axes[1].set_title(
    "Matriz de Spearman\n(Correlaç&atilde;o Monotônica)",
    fontsize=14,
    fontweight='bold',
    pad=15
)
axes[1].set_xticklabels(
    axes[1].get_xticklabels(),
    rotation=45,
    ha='right',
    fontsize=9
)
axes[1].set_yticklabels(
    axes[1].get_yticklabels(),
    rotation=0,
    fontsize=9
)

plt.tight_layout()
plt.show()
plt.close()

# =====
# VISUALIZAÇÃO ALTERNATIVA: Apenas Triângulo Superior
# =====

print("\n

```

```

        annot=True,
        fmt=".2f",
        cmap='PiYG',
        center=0,
        vmin=-1,
        vmax=1,
        square=True,
        linewidths=0.5,
        cbar_kws={"shrink": 0.8},
        ax=axes[1],
        xticklabels=labels_curtos,
        yticklabels=labels_curtos,
        annot_kws={"size": 8}
    )

    axes[1].set_title(
        "Spearman (Triângulo Inferior)",
        fontsize=14,
        fontweight='bold',
        pad=15
    )
    axes[1].set_xticklabels(
        axes[1].get_xticklabels(),
        rotation=45,
        ha='right',
        fontsize=9
    )
    axes[1].set_yticklabels(
        axes[1].get_yticklabels(),
        rotation=0,
        fontsize=9
    )

plt.tight_layout()
plt.show()
plt.close()

return matrix_pearson, matrix_spearman

```

```
In [7]: def plot_correlation_clusters(matrix_pearson, threshold=0.85, figsize=(16, 12), max_pairs=30):
    """
    Visualiza apenas pares com alta correlação ( $|r| \geq \text{threshold}$ ).
    Versão otimizada para evitar sobreposição de labels.

    Parâmetros:
    -----
    matrix_pearson : pd.DataFrame - Matriz de correlação
    threshold : float - Threshold mínimo de correlação
    figsize : tuple - Tamanho da figura
    max_pairs : int - Número máximo de pares a exibir
    """

    # Extrair pares com alta correlação
    pares_altos = []
    n = len(matrix_pearson)

    for i in range(n):
        for j in range(i+1, n):
            corr_val = matrix_pearson.iloc[i, j]
            if abs(corr_val) >= threshold:
                pares_altos.append({
                    'feature_1': matrix_pearson.index[i],
                    'feature_2': matrix_pearson.columns[j],
                    'correlacao': corr_val
                })

    if not pares_altos:
        print(f"✅ Nenhum par com |correlação| >= {threshold}")
        return

    # Criar DataFrame e ordenar
    df_pares = pd.DataFrame(pares_altos).sort_values('correlacao', key=abs, ascending=False)

    # Limitar número de pares se necessário
    if len(df_pares) > max_pairs:
        print(f"⚠️ Limitando visualização aos top {max_pairs} pares (de {len(df_pares)} encontrados)")
        df_pares = df_pares.head(max_pairs)

    print(f"\n⌚ Encontrados {len(pares_altos)} pares com |correlação| >= {threshold}\n")
    print(df_pares.to_string(index=False))

    # =====
    # VISUALIZAÇÃO MELHORADA
    # =====

    fig, ax = plt.subplots(figsize=figsize)

    # Criar labels mais limpos
    def truncar_nome(nome, max_len=35):
        """Trunca nome mantendo início e fim"""
        if len(nome) <= max_len:
            return nome
        else:
            # Manter início e fim
            metade = (max_len - 3) // 2
            return f"{nome[:metade]}...{nome[-metade:]}""

    # Labels formatados (um por linha, sem "vs")
    labels = []
    for _, row in df_pares.iterrows():
        feat1 = truncar_nome(row['feature_1'], max_len=40)
        feat2 = truncar_nome(row['feature_2'], max_len=40)
        labels.append(f"{feat1}\n ↔ {feat2}")

    # Cores por intensidade
    def get_color(corr):
        abs_corr = abs(corr)
        if abs_corr >= 0.99:
            return '#8B0000' # Vermelho escuro (perfeita)
        elif abs_corr >= 0.95:
            return '#DC143C' # Vermelho (crítica)
        elif abs_corr >= 0.90:
            return '#FF8C00' # Laranja (muito alta)
        else:
            return '#32CD32' # Verde (alta)

    colors = [get_color(c) for c in df_pares['correlacao']]

    # Criar barras horizontais
    y_pos = np.arange(len(df_pares))
    bars = ax.bars(y_pos, df_pares['correlacao'], color=colors, alpha=0.85, height=0.7)

    # Configurar eixos
    ax.set_yticks(y_pos)
    ax.set_yticklabels(labels, fontsize=9, family='monospace')
    ax.set_xlabel('Correlação de Pearson', fontsize=13, fontweight='bold')
```

```

ax.set_xlim([-1.05, 1.15]) # Espaço extra para valores

# Título
ax.set_title(
    f'Pares com Alta Correlação (|r| ≥ {threshold})\n'
    f'Total: {len(pares_altos)} pares | Exibindo: {len(df_pares)} pares',
    fontsize=15,
    fontweight='bold',
    pad=20
)

# Linhas de referência
ax.axvline(x=0, color='black', linestyle='-', linewidth=1.2, alpha=0.7)
ax.axvline(x=threshold, color='red', linestyle='--', linewidth=1.5, alpha=0.4,
           label=f'Threshold = ±{threshold}')
ax.axvline(x=-threshold, color='red', linestyle='--', linewidth=1.5, alpha=0.4)
ax.axvline(x=0.95, color='orange', linestyle=':', linewidth=1.2, alpha=0.4,
           label='Crítico = ±0.95')
ax.axvline(x=-0.95, color='orange', linestyle=':', linewidth=1.2, alpha=0.4)

# Grid
ax.grid(axis='x', alpha=0.3, linestyle='--', linewidth=0.5)
ax.set_axisbelow(True)

# Adicionar valores nas barras
for i, (bar, val) in enumerate(zip(bars, df_pares['correlacao'])):
    # Posição do texto
    x_pos = val + 0.03 if val > 0 else val - 0.03
    ha = 'left' if val > 0 else 'right'

    # Cor do texto (branco se barra escura)
    text_color = 'white' if abs(val) >= 0.95 else 'black'

    # Texto dentro da barra se correlação alta
    if abs(val) >= 0.90:
        x_pos = val - 0.03 if val > 0 else val + 0.03
        ha = 'right' if val > 0 else 'left'
        text_color = 'white'

    ax.text(
        x_pos,
        i,
        f'{val:.3f}',
        va='center',
        ha=ha,
        fontsize=10,
        fontweight='bold',
        color=text_color
    )

# Legenda
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor="#8B0000", label='Perfeita (≥ 0.99)'),
    Patch(facecolor="#DC143C", label='Crítica (0.95 - 0.99)'),
    Patch(facecolor="#FF8C00", label='Muito Alta (0.90 - 0.95)'),
    Patch(facecolor="#32CD32", label='Alta (0.85 - 0.90')),
]
ax.legend(
    handles=legend_elements,
    loc='lower right',
    fontsize=10,
    framealpha=0.9,
    title='Intensidade'
)

plt.tight_layout()
plt.show()
plt.close()

# -----
# TABELA RESUMIDA POR CATEGORIA
# -----

print("\n" + "="*100)
print("📊 RESUMO POR CATEGORIA DE CORRELAÇÃO")
print("="*100)

categorias = [
    ('🔴 PERFEITA (≥ 0.99)', 0.99, 1.01),
    ('🟡 CRÍTICA (0.95 - 0.99)', 0.95, 0.99),
    ('🟠 MUITO ALTA (0.90 - 0.95)', 0.90, 0.95),
    ('🟢 ALTA (0.85 - 0.90)', 0.85, 0.90),
]
]

for titulo, min_val, max_val in categorias:
    subset = df_pares[
        (df_pares['correlacao'].abs() >= min_val) &

```

```

        (df_pares['correlacao'].abs() < max_val)
    ]

    if len(subset) > 0:
        print(f"\n{titulo} ({len(subset)} pares)")
        print("-"*100)

        for idx, row in subset.iterrows():
            feat1 = row['feature_1'][:45]
            feat2 = row['feature_2'][:45]
            corr = row['correlacao']

            print(f" • {feat1:47s} ↔ {feat2:47s} | r = {corr:+.4f}")

    print("\n" + "="*100)

```

```

In [8]: def calcular_cramer_v(df, col1, col2):
    # Verificação de segurança: não rodar em colunas com muitos valores únicos (numéricas)
    # Se a coluna tiver mais de 100 valores distintos, provavelmente não é uma categoria útil
    distinct_count = df.select(col1).distinct().count()
    if distinct_count > 100:
        print(f"Notei {col1}: muitos valores únicos ({distinct_count}). Verifique se é quantitativa.")
        return np.nan

    # Indexação
    idx1, idx2 = col1 + "_idx", col2 + "_idx"
    df_indexed = StringIndexer(inputCol=col1, outputCol=idx1, handleInvalid="skip").fit(df).transform(df)
    df_indexed = StringIndexer(inputCol=col2, outputCol=idx2, handleInvalid="skip").fit(df_indexed).transform(df_indexed)

    # Assembler para o Teste
    assembler = VectorAssembler(inputCols=[idx1], outputCol="features")
    test_data = assembler.transform(df_indexed).select("features", idx2)

    # Chi-Square Test
    res = ChiSquareTest.test(test_data, "features", idx2).head()

    # --- CORREÇÃO DO ACESSO AOS ATRIBUTOS ---
    chi2 = float(res.statistics[0]) # Estatística Chi2 correta
    n = test_data.count()

    # Resgate do número de categorias pelos metadados ou contagem
    r = df_indexed.select(idx1).distinct().count()
    k = df_indexed.select(idx2).distinct().count()

    if n == 0 or min(r-1, k-1) <= 0: return 0.0

    v = np.sqrt(chi2 / (n * min(r-1, k-1)))
    return v

```

```
In [9]: def matriz_cramer_v_spark_optimized(df, colunas_categoricas, figsize=(20, 16)):
    """
    Constrói uma matriz de correlação Cramer's V para variáveis categóricas.
    Otimizada para visualização de até 20+ variáveis.

    Parameters:
    -----
    df : pyspark.sql.DataFrame
        DataFrame com as variáveis categóricas
    colunas_categoricas : list
        Lista de nomes das colunas categóricas
    figsize : tuple
        Tamanho da figura (width, height)

    Returns:
    -----
    pd.DataFrame : Matriz de correlação Cramer's V
    """

    n_cols = len(colunas_categoricas)
    matrix = np.ones((n_cols, n_cols))

    print(f"✉️ Iniciando indexação de {n_cols} colunas categóricas...")

    # =====
    # 1. INDEXAÇÃO EM BATCH (Performance)
    # =====
    indexers = [
        StringIndexer(
            inputCol=c,
            outputCol=c+"_idx",
            handleInvalid="keep" # Mantém categorias novas como "unknown"
        )
        for c in colunas_categoricas
    ]

    df_indexed = df
    models = []
    for idx, indexer in enumerate(indexers):
        model = indexer.fit(df_indexed)
        df_indexed = model.transform(df_indexed)
        models.append(model)

    print(f"✅ Indexação concluída!\n")

    # =====
    # 2. CÁLCULO DO CRAMER'S V (Triângulo Superior)
    # =====
    total_pairs = (n_cols * (n_cols - 1)) // 2
    current_pair = 0

    for i in range(n_cols):
        for j in range(i + 1, n_cols):
            current_pair += 1
            col1 = colunas_categoricas[i]
            col2 = colunas_categoricas[j]
            col1_idx = col1 + "_idx"
            col2_idx = col2 + "_idx"

            print(f"Calculando: {col1} ↔ {col2}")

            # Preparar dados para ChiSquareTest
            assembler = VectorAssembler(inputCols=[col1_idx], outputCol="features")
            test_data = assembler.transform(df_indexed).select("features", col2_idx)

            # Executar teste Qui-Quadrado
            try:
                res = ChiSquareTest.test(test_data, "features", col2_idx).head()
                chi2 = float(res.statistics[0])
                n = test_data.count()

                # Número de categorias (r e k)
                r = len(models[i].labels)
                k = len(models[j].labels)

                # Cálculo do V de Cramer
                if n > 0 and min(r-1, k-1) > 0:
                    v = np.sqrt(chi2 / (n * min(r-1, k-1)))
                else:
                    v = 0.0

            except Exception as e:
                print(f"⚠️ Erro ao calcular {col1} vs {col2}: {e}")
                v = 0.0

            # Preencher matriz (simétrica)
            matrix[i, j] = v
```

```

matrix[j, i] = v

print(f"\n ✅ Cálculo concluído!\n")

# =====
# 3. CRIAR DATAFRAME E VISUALIZAÇÃO OTIMIZADA
# =====

cramer_df = pd.DataFrame(
    matrix,
    index=colunas_categoricas,
    columns=colunas_categoricas
)

# Plotar Heatmap Otimizado
fig, ax = plt.subplots(figsize=figsize)

# Máscara para o triângulo superior (opcional, deixa mais limpo)
mask = np.triu(np.ones_like(cramer_df, dtype=bool), k=1)

sns.heatmap(
    cramer_df,
    annot=True,           # Mostrar valores
    fmt='.2f',            # 2 casas decimais
    cmap='RdYlGn_r',      # Colormap: Vermelho (alta correlação) -> Verde (baixa)
    vmin=0,
    vmax=1,
    center=0.5,          # Centro da escala
    square=True,          # Células quadradas
    linewidths=0.5,       # Linhas entre células
    cbar_kws={
        "shrink": 0.8,
        "label": "Cramer's V"
    },
    mask=mask,             # Oculta triângulo superior (opcional)
    ax=ax
)

# Ajustes de visualização
plt.title(
    "Matriz de Associação: Cramer's V (Variáveis Categóricas)",
    fontsize=16,
    fontweight='bold',
    pad=20
)
plt.xlabel("", fontsize=12)
plt.ylabel("", fontsize=12)

# Rotacionar labels para melhor legibilidade
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(rotation=0, fontsize=10)

plt.tight_layout()
plt.show()

return cramer_df

```

```
In [10]: def matriz_cramer_v_spark(df, colunas_categoricas):
    """
    Constrói uma matriz de correlação Cramer's V para uma lista de colunas.
    """
    n_cols = len(colunas_categoricas)
    # Inicializa a matriz com 1s na diagonal (correlação de uma var com ela mesma)
    matrix = np.ones((n_cols, n_cols))

    print(f"Iniciando indexação de {n_cols} colunas...")

    # 1. Indexar todas as colunas de uma vez para ganhar performance
    indexers = [StringIndexer(inputCol=c, outputCol=c+"_idx", handleInvalid="skip") for c in colunas_categoricas]

    # Aplicamos a indexação em cadeia
    df_indexed = df
    models = []
    for indexer in indexers:
        model = indexer.fit(df_indexed)
        df_indexed = model.transform(df_indexed)
        models.append(model)

    # 2. Iterar sobre os pares (apenas o triângulo superior para economizar processamento)
    for i in range(n_cols):
        for j in range(i + 1, n_cols):
            col1 = colunas_categoricas[i]
            col2 = colunas_categoricas[j]
            col1_idx = col1 + "_idx"
            col2_idx = col2 + "_idx"

            print(f"Calculando Cramer's V: {col1} vs {col2}...")

            # Preparar dados para o ChiSquareTest
            assembler = VectorAssembler(inputCols=[col1_idx], outputCol="features")
            test_data = assembler.transform(df_indexed).select("features", col2_idx)

            # Executar teste
            res = ChiSquareTest.test(test_data, "features", col2_idx).head()
            chi2 = float(res.statistics[0])
            n = test_data.count()

            # Número de categorias de cada coluna (r e k) extraídos dos modelos indexadores
            r = len(models[i].labels)
            k = len(models[j].labels)

            # Cálculo do V de Cramer
            if n > 0 and min(r-1, k-1) > 0:
                v = np.sqrt(chi2 / (n * min(r-1, k-1)))
            else:
                v = 0.0

            # Preencher a matriz de forma simétrica
            matrix[i, j] = v
            matrix[j, i] = v

    # Criar DataFrame Pandas para facilitar a visualização
    cramer_df = pd.DataFrame(matrix, index=colunas_categoricas, columns=colunas_categoricas)

    # Plotar o Heatmap
    plt.figure(figsize=(10, 8))
    sns.heatmap(cramer_df, annot=True, cmap='YlGnBu', fmt='%.2f', vmin=0, vmax=1)
    plt.title("Matriz de Associação: Cramer's V (Variáveis Categóricas)")
    plt.show()

    return cramer_df
```

```
In [11]: def identificar_outliers(df, coluna, negativo=True, minimo_definido=0):
    # Calcula os quantis de forma aproximada para performance
    quantis = df.approxQuantile(coluna, [0.25, 0.75], 0.05)
    q1, q3 = quantis[0], quantis[1]
    iqr = q3 - q1

    limite_superior = q3 + 1.5 * iqr

    # Se negativo for True, permite valores negativos no limite inferior --> dependendo da variável
    if negativo:
        limite_inferior = q1 - 1.5 * iqr
    else:
        limite_inferior = max(0, (q1 - 1.5 * iqr))

    if minimo_definido != 0:
        limite_inferior = minimo_definido

outliers = df.filter((F.col(coluna) < limite_inferior) | (F.col(coluna) > limite_superior))
print(f"Variável {coluna}:")
print(f"Limites: [{limite_inferior}, {limite_superior}]")
print(f"Total de outliers (#): {outliers.count()}")
print(f"Total de outliers (%): {(outliers.count() / df.count()) * 100:.2f}%")
```

```
In [12]: def plot_tendencia_temporal(df, col_valor, aggregation="mean", col_safra="safra", categories=None):
    """
    df: Spark DataFrame
    col_valor: Variável numérica para calcular a média
    col_safra: Coluna de tempo (ex: 'safra')
    categories: Nome da coluna categórica para quebrar as linhas (ex: 'gender') ou None
    """
    # Definir a lista de colunas para o agrupamento
    group_cols = [col_safra]
    if categories:
        group_cols.append(categories)

    # Agregação no Spark por safra (e categoria, se houver) e calculamos a média. Transformar em Pandas para plotar
    if aggregation == "mean":
        stats_safra = (df.groupBy(group_cols)
                        .agg(F.mean(col_valor).alias(f"{aggregation}"))
                        .orderBy(col_safra)
                        .toPandas())
    elif aggregation == "sum":
        stats_safra = (df.groupBy(group_cols)
                        .agg(F.sum(col_valor).alias(f"{aggregation}"))
                        .orderBy(col_safra)
                        .toPandas())
    elif aggregation == "max":
        stats_safra = (df.groupBy(group_cols)
                        .agg(F.max(col_valor).alias(f"{aggregation}"))
                        .orderBy(col_safra)
                        .toPandas())
    elif aggregation == "min":
        stats_safra = (df.groupBy(group_cols)
                        .agg(F.min(col_valor).alias(f"{aggregation}"))
                        .orderBy(col_safra)
                        .toPandas())
    elif aggregation == "med":
        stats_safra = (df.groupBy(group_cols)
                        .agg(F.expr("percentile_approx({}, 0.5)".format(col_valor)).alias(f"{aggregation}"))
                        .orderBy(col_safra)
                        .toPandas())

    # Tratar a safra como categoria (texto)
    stats_safra[col_safra] = stats_safra[col_safra].astype(str)

    # 4. Configuração do Plot
    plt.figure(figsize=(14, 6))

    # Se 'categories' for informada, o Seaborn cria uma linha para cada categoria usando 'hue'
    sns.lineplot(
        data=stats_safra,
        x=col_safra,
        y=f"{aggregation}",
        hue=categories if categories else None,
        marker='o',
        sort=False # Garante que a ordem das safras do Spark seja mantida
    )

    # Ajustes finos de visualização
    titulo = f"{aggregation} de {col_valor} por {col_safra}"
    if categories:
        titulo += f" (Quebrado por {categories})"
        plt.legend(title=categories, bbox_to_anchor=(1.02, 1), loc='upper left')

    plt.title(titulo)
    plt.xticks(rotation=45)
    plt.grid(True, axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
```

```
In [13]: def integridade_entre_bases(df_base, df_comparacao, nome_relacao, chave=["msno", "safra"]):
    """
    Rastreia a perda de dados entre tabelas (ex: membros vs logs).
    Essencial para justificar a volumetria final do modelo.
    """
    total_base = df_base.select(chave).distinct().count()
    com_match = df_base.join(df_comparacao, chave, "inner").select(chave).distinct().count()
    perda = ((total_base - com_match) / total_base) * 100

    print(f"--- Integridade: {nome_relacao} ---")
    print(f"Registros na base: {total_base} | Registros correspondentes: {com_match}")
    print(f"Taxa de Perda: {perda:.2f}%\n")
```

```

In [14]: def plot_boxplot(df, coluna_categorica, coluna_alvo, agrupar_por_safra=False, table=False):
    """
    Plota boxplots utilizando estatísticas calculadas no Spark, incluindo a média.
    """

    # 1. Definimos as colunas de agrupamento
    cols_base = ['safra'] if agrupar_por_safra else []

    for col_cat in coluna_categorica:
        print(f"Processando estatísticas para: {col_cat}...")

    # 2. Calculamos os quartis E a média no Spark
    df_stats = (df
        .groupBy(cols_base + [col_cat])
        .agg(
            F.percentile_approx(coluna_alvo, [0.0, 0.25, 0.5, 0.75, 1.0], 10000).alias("stats"),
            F.mean(coluna_alvo).alias("avg") # <-- Adicionado média
        )
        .select(*cols_base, col_cat,
            F.col("stats")[0].alias("min"),
            F.col("stats")[1].alias("q1"),
            F.col("stats")[2].alias("med"),
            F.col("stats")[3].alias("q3"),
            F.col("stats")[4].alias("max"),
            F.col("avg").alias("mean")) # <-- Selecionando a média
        .toPandas())

    # 3. Lógica de Plotagem
    if agrupar_por_safra:
        safras = sorted(df_stats['safra'].unique())
        for s in safras:
            df_safra = df_stats[df_stats['safra'] == s]
            _desenhar_boxplot_estatistico(df_safra, col_cat, coluna_alvo, f"Safra {s}", show_table=table)
    else:
        _desenhar_boxplot_estatistico(df_stats, col_cat, coluna_alvo, "Visão Consolidada", show_table=table)

def _desenhar_boxplot_estatistico(df_plot, col_cat, col_alvo, subtítulo, show_table=False):
    stats_list = []
    df_plot = df_plot.sort_values(by=col_cat)

    for _, row in df_plot.iterrows():
        stats_list.append({
            "label": str(row[col_cat]),
            "whislo": row["min"],
            "q1": row["q1"],
            "med": row["med"],
            "q3": row["q3"],
            "whishi": row["max"],
            "mean": row["mean"] # <-- Matplotlib usa essa chave para desenhar a média
        })

    if not stats_list:
        return

    # Renderização do Gráfico
    plt.figure(figsize=(12, 6))
    ax = plt.gca()
    # showmeans=True ativa a exibição do marcador da média
    ax.bxp(stats_list, showfliers=False, showmeans=True,
           meanprops={"marker": "o", "markerfacecolor": "white", "markeredgecolor": "black", "markersize": 6})

    plt.title(f"Boxplot: {col_alvo} por {col_cat} ({subtítulo})")
    plt.ylabel(col_alvo)
    plt.xticks(rotation=45)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()

    # Renderização da Tabela
    if show_table:
        print(f"\n--- Estatísticas: {col_cat} ({subtítulo}) ---")
        # Adicionado "mean" na visualização da tabela
        cols_tabela = [col_cat, "min", "q1", "med", "mean", "q3", "max"]
        df_resumo = df_plot[cols_tabela].copy()

        try:
            from IPython.display import display
            display(df_resumo)
        except ImportError:
            print(df_resumo.to_string(index=False))
        print("\n" + "="*50 + "\n")

```

3.2. Tratamento de dados

3.2.1. Agrupamento

```
In [15]: def agregar_logs(df_logs):
    """
    Agrega a tabela de logs no nível (msno, safra).

    Premissa: A base de logs já vem agregada por mês, mas esta função
    garante robustez caso surjam duplicatas em produção.

    Regra de agregação:
    - Variáveis numéricas de uso: SUM (acumulação de comportamento)

    Args:
        df_logs: DataFrame Spark com logs brutos

    Returns:
        DataFrame agregado no nível (msno, safra)
    """

    # Colunas esperadas (contrato de dados)
    expected_cols = ["msno", "safra", "num_25", "num_50", "num_75", "num_985", "num_100", "num_unq", "total_secs"]

    # Selecionar apenas colunas esperadas (proteção contra novas features)
    df_logs = df_logs.select(*expected_cols)

    # Verificar duplicatas
    duplicates = df_logs.groupBy("msno", "safra").count().filter("count > 1")

    if duplicates.count() > 0:
        print(f"⚠️ LOGS: {duplicates.count()} duplicatas detectadas. Aplicando agregação...")

    df_logs_agg = (
        df_logs
        .groupBy("msno", "safra")
        .agg(
            # Variáveis de uso: acumulação (SUM)
            F.sum("num_25").alias("num_25"),
            F.sum("num_50").alias("num_50"),
            F.sum("num_75").alias("num_75"),
            F.sum("num_985").alias("num_985"),
            F.sum("num_100").alias("num_100"),
            F.sum("num_unq").alias("num_unq"),
            F.sum("total_secs").alias("total_secs")))
    )

    print("✅ LOGS: Agregação concluída.")
    return df_logs_agg

else:
    print("✅ LOGS: Base já está no nível (msno, safra). Nenhuma agregação necessária.")
    return df_logs
```

```
In [16]: def agregar_transactions(df_transactions):
    """
    Agrega a tabela de transações no nível (msno, safra).

    Premissa: Podem existir múltiplas transações por cliente no mesmo mês
    (upgrades, downgrades, reprocessamentos).

    Regras de agregação:
    - Flags: MAX (ocorreu ao menos uma vez?)
    - Valores monetários: SUM (acumulação financeira)
    - Atributos de plano e categóricas: LAST (último estado observado)
    - Datas: MAX para transaction_date, LAST para membership_expire_date

    Args:
        df_transactions: DataFrame Spark com transações brutas

    Returns:
        DataFrame agregado no nível (msno, safra)
    """

    # Colunas esperadas (contrato de dados)
    expected_cols = ["msno", "safra", "payment_method_id", "payment_plan_days", "plan_list_price",
                     "actual_amount_paid", "is_auto_renew", "transaction_date", "membership_expire_date", "is_cancel"]

    # Selecionar apenas colunas esperadas (proteção contra novas features)
    df_transactions = df_transactions.select(*expected_cols)

    # Verificar duplicatas
    duplicates = df_transactions.groupBy("msno", "safra").count().filter("count > 1")

    if duplicates.count() > 0:
        print(f"⚠️ TRANSACTIONS: {duplicates.count()} duplicatas detectadas. Aplicando agregação...")

    # Criar ranking por data (última transação = 1)
    w_order = Window.partitionBy("msno", "safra").orderBy(F.col("transaction_date").desc())
    df_transactions = df_transactions.withColumn("_rank", F.row_number().over(w_order))

    df_tx_agg = (
        df_transactions
            .groupBy("msno", "safra")
            .agg(
                # Flags: MAX (ocorrência)
                F.max("is_auto_renew").alias("is_auto_renew"),
                F.max("is_cancel").alias("is_cancel"),

                # Valores monetários: SUM (acumulação)
                F.sum("actual_amount_paid").alias("actual_amount_paid"),

                # Atributos de plano: LAST (último estado)
                F.max(F.when(F.col("_rank") == 1, F.col("payment_method_id"))).alias("payment_method_id"),
                F.max(F.when(F.col("_rank") == 1, F.col("payment_plan_days"))).alias("payment_plan_days"),
                F.max(F.when(F.col("_rank") == 1, F.col("plan_list_price"))).alias("plan_list_price"),

                # Datas
                F.max("transaction_date").alias("transaction_date"),
                F.max(F.when(F.col("_rank") == 1, F.col("membership_expire_date"))).alias("membership_expire_date")
            )
    )

    print("✅ TRANSACTIONS: Agregação concluída.")
    return df_tx_agg
else:
    print("✅ TRANSACTIONS: Base já está no nível (msno, safra). Nenhuma agregação necessária.")
    return df_transactions
```

```
In [17]: def agregar_members(df_members):
    """
        Agrega a tabela de membros no nível (msno, safra).

        Premissa: Atributos de membros são majoritariamente estáticos, mas podem
        existir múltiplas linhas por cliente em casos de mudança de cidade,
        reativação, ou inconsistências de dados.

        Regras de agregação:
        - Atributos demográficos: LAST (último estado observado)
        - Flag de status: MAX (está ativo ao menos uma vez?)
        - Data de registro: MIN (primeira ocorrência)

        Args:
            df_members: DataFrame Spark com dados de membros

        Returns:
            DataFrame agregado no nível (msno, safra)
    """

    # Colunas esperadas (contrato de dados)
    expected_cols = ["msno", "safra", "registration_init_time", "city", "bd", "gender", "registered_via", "is_ativo"]

    # Selecionar apenas colunas esperadas (proteção contra novas features)
    df_members = df_members.select(*expected_cols)

    # Converter safra para integer (está como string no schema)
    df_members = df_members.withColumn("safra", F.col("safra").cast("integer"))

    # Verificar duplicatas
    duplicates = df_members.groupBy("msno", "safra").count().filter("count > 1")

    if duplicates.count() > 0:
        print(f"⚠️ MEMBERS: {duplicates.count()} duplicatas detectadas. Aplicando agregação...")

        # Criar ranking por data de registro (mais recente = 1)
        # Se não houver coluna de atualização, usamos registration_init_time como proxy
        w_order = Window.partitionBy("msno", "safra").orderBy(F.col("registration_init_time").desc())
        df_members = df_members.withColumn("_rank", F.row_number().over(w_order))

        df_members_agg = (
            df_members
            .groupBy("msno", "safra")
            .agg(
                # Data de registro: MIN (primeira ocorrência)
                F.min("registration_init_time").alias("registration_init_time"),

                # Atributos demográficos: LAST (último estado)
                F.max(F.when(F.col("_rank") == 1, F.col("city"))).alias("city"),
                F.max(F.when(F.col("_rank") == 1, F.col("bd"))).alias("bd"),
                F.max(F.when(F.col("_rank") == 1, F.col("gender"))).alias("gender"),
                F.max(F.when(F.col("_rank") == 1, F.col("registered_via"))).alias("registered_via"),

                # Flag de status: MAX (está ativo?)
                F.max("is_ativo").alias("is_ativo"))
        )

        print("✅ MEMBERS: Agregação concluída.")
        return df_members_agg

    else:
        print("✅ MEMBERS: Base já está no nível (msno, safra). Nenhuma agregação necessária.")
        return df_members
```

3.2.2. Outliers e Categorização Estatística

```
In [18]: def aplicar_winsorizacao(df, colunas, p_inf=0.01, p_sup=0.99):
    for col in colunas:
        # Calcula os limites baseados em percentis
        limites = df.approxQuantile(col, [p_inf, p_sup], 0.001) # 0.001 é a precisão/erro aceitável
        inf, sup = limites[0], limites[1]

        print(f"Coluna {col}: Limite Inferior={inf}, Limite Superior={sup}")

        # Aplica o Capping
        df = df.withColumn(f"{col}_win",
                           F.when(F.col(col) < inf, inf)
                           .when(F.col(col) > sup, sup)
                           .otherwise(F.col(col)))
    )
    return df
```

```
In [19]: def segment_by_percentile(df, col_to_group, table_name, num_buckets=4):
    """
        Categoriza uma variável dinamicamente com base em percentis calculados no Spark.

    Parâmetros:
    - df: DataFrame do Spark.
    - col_to_group: Nome da coluna numérica a ser segmentada (ex: 'total_plays').
    - table_name: Nome da tabela de origem para compor a flag (ex: 'logs').
    - num_buckets: Número de divisões desejadas.
        4 para Quartis (25%, 50%, 75%, 100%)
        10 para Decisões (10%, 20%, ..., 100%)

    Variáveis esperadas no DataFrame:
    - f"flag_has_{table_name)": Coluna binária (0/1) que indica presença de dados.
    - {col_to_group}: A variável numérica que será processada.
    """

    # 1. Definimos a coluna de flag
    flag_col = f"flag_has_{table_name}"

    # 2. Calculo dinamico dos percentis
    # Criamos uma lista de probabilidades: ex para 4 buckets -> [0.25, 0.50, 0.75, 1.0]
    probabilities = [i / num_buckets for i in range(1, num_buckets + 1)]

    # Coletamos os valores de corte (stats)
    # Nota: Filtramos apenas onde a flag é 1 para não enviesar os percentis com zeros/nulos
    cut_off_points = (df
        .filter(F.col(flag_col).isin(1))
        .agg(F.percentile_approx(col_to_group, probabilities).alias("pts"))
        .collect()[0]["pts"])

    # 3. A lógica do .when dinamico, começando com a condicao base para desconhecidos/inativos
    case_expression = F.when(F.col(flag_col).isin(0), "unknown")

    # Iteramos sobre os pontos de corte para criar as faixas
    # Ex: para Quartis, teremos Tier 1 (até 25%), Tier 2 (até 50%), etc.
    for i, point in enumerate(cut_off_points):
        tier_label = f"tier_{i+1}"
        # A primeira faixa pega do menor valor até o primeiro ponto
        if i == 0:
            case_expression = case_expression.when(F.col(col_to_group) <= point, tier_label)
        else:
            # As demais pegam entre o ponto anterior e o atual
            case_expression = case_expression.when(
                (F.col(col_to_group) > cut_off_points[i-1]) & (F.col(col_to_group) <= point),
                tier_label)

    # Caso escape de alguma lógica (fallback)
    case_expression = case_expression.otherwise(f"tier_{num_buckets}")

    # Retornamos o dataframe com a nova coluna
    return df.withColumn(f"{col_to_group}_group", case_expression)
```

3.2.3. Tendência - Janela temporal (3 e 6 meses)

```
In [20]: def features_de_tendencia_num(df, variables, id_col="msno", time_col="safra", windows=[3, 6], keep_intermediate=False):
    original_cols = df.columns
    generated_cols = []

    for var in variables:
        # manter raw
        raw_col = f"{var}_raw"
        df = df.withColumn(raw_col, F.col(var))
        generated_cols.append(raw_col)

        for n in windows:
            w = Window.partitionBy(id_col).orderBy(time_col)

            # lags
            lags = []
            for i in range(n):
                lag_col = f"{var}_lag_{i}"
                df = df.withColumn(lag_col, F.lag(var, i).over(w))
                lags.append(lag_col)

            # contadores
            cnt_null = f"{var}_cnt_null_{n}"
            cnt_zero = f"{var}_cnt_zero_{n}"

            df = (df
                  .withColumn(cnt_null, sum(F.col(c).isNull().cast("int") for c in lags))
                  .withColumn(cnt_zero, sum((F.col(c) == 0).cast("int") for c in lags)))

            ref = f"{var}_lag_0"
            arr = f"array({','.join(lags)})"

            # métricas raw
            mean_raw = f"{var}_mean_{n}_raw"
            min_raw = f"{var}_min_{n}_raw"
            max_raw = f"{var}_max_{n}_raw"

            df = (df
                  .withColumn(
                      mean_raw,
                      F.expr(f"""
                            aggregate(filter({arr}, x -> x is not null), 0D, (acc, x) -> acc + x) / size(filter({arr}, x -> x is not
null)))
                  """
                  .withColumn(min_raw, F.expr(f"array_min(filter({arr}, x -> x is not null))"))
                  .withColumn(max_raw, F.expr(f"array_max(filter({arr}, x -> x is not null))")))

            # métricas finais (com sentinela)
            for metric, raw_metric in zip(
                ["mean", "min", "max"],
                [mean_raw, min_raw, max_raw]
            ):
                final_col = f"{var}_{metric}_{n}"
                generated_cols.append(final_col)

                df = df.withColumn(final_col,
                                   F.when(F.col(cnt_null) == n, F.lit(-99998))
                                   .when(F.col(cnt_zero) == n, F.lit(-99999))
                                   .when((F.col(ref) == 0) & (F.col(cnt_null) < n), F.lit(-99995))
                                   .when((F.col(ref).isNotNull()) & (F.col(cnt_null) == n - 1), F.lit(-99996))
                                   .when((F.col(ref).isNotNull()) & (F.col(cnt_zero) == n - 1), F.lit(-99997))
                                   .otherwise(F.col(raw_metric)))

            # razões
            for metric in ["mean", "min", "max"]:
                denom = f"{var}_{metric}_{n}"
                ratio = f"{var}_ratio_ref_{metric}_{n}"
                generated_cols.append(ratio)

                df = df.withColumn(ratio,
                                   F.when(F.col(denom) <= 0, F.col(denom))
                                   .otherwise(F.col(ref) / F.col(denom)))

    # =====
    # CONTROLE DE SAÍDA
    # =====
    if not keep_intermediate:
        final_cols = list(dict.fromkeys(
            original_cols + generated_cols))
        df = df.select(final_cols)

    return df
```

```
In [21]: def features_de_tendencia_cat(df, flags, id_col="msno", time_col="safra", windows=[3, 6]):  
    for flag in flags:  
        df = df.withColumn(f"${flag}_raw", F.col(flag))  
  
    for n in windows:  
        w = Window.partitionBy(id_col).orderBy(time_col)  
  
        lags = []  
        for i in range(n):  
            lag_col = f"${flag}_lag_{i}"  
            df = df.withColumn(lag_col, F.lag(flag, i).over(w))  
            lags.append(lag_col)  
  
        arr = f"array({','.join(lags)})"  
  
        df = (df  
              .withColumn(f"${flag}_sum_{n}",  
                         F.expr(f"aggregate(filter({arr}), x -> x is not null), 0, (acc, x) -> acc + x)"))  
              .withColumn(f"${flag}_mean_{n}", F.col(f"${flag}_sum_{n}") / F.lit(n))  
              .withColumn(f"${flag}_max_{n}", F.expr(f"array_max(filter({arr}, x -> x is not null))))"))  
  
    return df
```

Objetivo geral

O objetivo desta função é capturar **comportamentos temporais dos usuários**, indo além do valor pontual do mês de referência.

Para cada cliente (`msno`) e safra (`yyyyMM`), são construídas variáveis que representam:

- o **nível atual** de uma métrica,
- o **histórico recente** dessa métrica,
- e a **relação entre o comportamento atual e o passado**.

Essa abordagem permite ao modelo identificar:

- padrões de crescimento ou queda,
- usuários novos vs recorrentes,
- mudanças abruptas de comportamento,
- ausência estrutural de informação.

Estrutura geral da função

A função recebe:

- uma lista de variáveis contínuas (ex: `num_100` , `total_secs` , `avg_secs_per_play`);
- janelas temporais fixas (`n = 3` e `n = 6` meses);
- e gera automaticamente **todas as transformações** para todas as variáveis listadas.

Para cada variável e cada janela temporal, são criados:

1. valores defasados (lags),
2. métricas estatísticas históricas,
3. razões entre o valor atual e o histórico,
4. versões com e sem códigos sentinela,
5. mantendo sempre o valor original (raw).

Lags temporais

Para cada variável `X` , são criados:

- `X_lag_0` : valor no mês de referência
- `X_lag_1` : valor no mês anterior
- ...
- `X_lag_(n-1)` : valor $n-1$ meses atrás

Esse lags formam a base para todas as transformações seguintes.

Variáveis `*_raw`

As variáveis com sufixo `_raw` representam o **valor matemático puro**, sem qualquer regra semântica adicional.

Exemplos:

- `num_100_raw`
- `num_100_mean_6_raw`
- `total_secs_max_3_raw`

Características:

- ignoram valores nulos quando possível;
- não utilizam códigos sentinela;
- podem assumir valores nulos ou zero naturalmente.

Uso principal:

- regressão linear,
- análises estatísticas,
- inspeção e debug.

Métricas históricas criadas

Para cada variável `X` e janela `n` , são criadas:

- `X_mean_n_raw` : média dos últimos `n` meses
- `X_min_n_raw` : mínimo dos últimos `n` meses
- `X_max_n_raw` : máximo dos últimos `n` meses

Essas métricas descrevem o **nível típico**, o **pior cenário** e o **melhor cenário recente** do usuário.

Variáveis sem `_raw` (com semântica)

As variáveis sem o sufixo `_raw` aplicam **regras semânticas** para diferenciar situações estruturalmente distintas, que numericamente poderiam parecer iguais.

Essas variáveis utilizam **códigos sentinelas negativos**, que carregam significado comportamental:

Código	Significado
----- -----	
-99998	Não há registros em nenhum dos últimos n meses
-99999	Todos os valores nos últimos n meses são zero
-99995	Valor atual é zero, mas havia histórico válido
-99996	Valor atual existe, mas histórico é todo nulo
-99997	Valor atual existe, mas histórico é todo zero

Esses códigos permitem ao modelo (especialmente árvores):

- distinguir ausência de uso vs ausência de dados,
- identificar usuários novos,
- detectar interrupções de comportamento.

Razões (variáveis de mudança de comportamento)

Para cada métrica histórica, são criadas razões do tipo:

- X_ratio_ref_mean_n
- X_ratio_ref_min_n
- X_ratio_ref_max_n

Essas variáveis comparam o valor atual (`lag_0`) com o histórico recente.

Interpretação:

- valor ≈ 1 → comportamento estável
- valor > 1 → aumento recente
- valor < 1 → queda recente

Caso o denominador seja inválido (zero, nulo ou sentinela), a razão herda o código sentinela, evitando divisões inválidas.

Separação entre estatística e semântica

Um ponto central do design é **não misturar estatística com semântica**.

Por isso:

- variáveis `_raw` são mantidas limpas e contínuas;
- variáveis sem `_raw` carregam contexto comportamental.

Essa separação permite:

- uso seguro em regressão linear (via `_raw`);
- exploração rica de padrões em modelos baseados em árvore (via sentinelas).

Variáveis de flags (binárias)

Para variáveis categóricas/binárias (ex: `is_cancel` , `is_auto_renew`), são criadas transformações específicas:

- `flag_sum_n` : número de ocorrências nos últimos n meses
- `flag_mean_n` : frequência relativa
- `flag_max_n` : ocorreu pelo menos uma vez

Essas variáveis capturam **persistência, recorrência e reincidência** de eventos.

Benefícios da abordagem

Essa arquitetura de features:

- reduz dependência de um único mês,
- melhora estabilidade temporal,
- captura mudanças de comportamento,
- é robusta a dados faltantes,
- funciona bem tanto para regressão quanto para árvores.

Além disso, permite seleção posterior de variáveis sem refazer o pipeline, mantendo rastreabilidade e flexibilidade.

Observação final

Nem todas as variáveis criadas necessariamente entrarão no modelo final.

A função foi desenhada para **gerar um espaço rico de candidatos**, permitindo que a seleção seja feita de forma empírica, baseada em validação e performance.

4. EDA - _Exploratory Data Analysis_

4.1. Logs

4.1.1. dictionary

Definition: daily user logs describing listening behaviors of a user. Data collected until 3/31/2017.

```
* msno: user id
* date: format %Y%m%d
* num_25: # of songs played less than 25% of the song
length
* num_50: # of songs played between 25% to 50% of the
song length
* num_75: # of songs played between 50% to 75% of
the song length
* num_985: # of songs played between 75% to 98.5% of
the song length
* num_100: # of songs played over 98.5% of the song
length
* num_unq: # of unique songs played
* total_secs: total seconds played
```

4.1.2. general info

```
In [12]: total_logs = df_logs.count()
total_logs
```

```
Out[12]: 26758971
```

```
In [11]: df_logs.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- num_25: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- num_100: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_secs: double (nullable = true)
```

```
In [12]: df_logs.groupBy("safra").count().orderBy("safra").show(5, truncate=False)
```

```
+-----+-----+
|safra |count |
+-----+-----+
|201501|937789|
|201502|933040|
|201503|944739|
|201504|939930|
|201505|924216|
+-----+-----+
only showing top 5 rows
```

```
In [13]: df_logs.show(10, truncate=False)
```

msno	safra	num_25	num_50	num_75	num_985	num_100	num_unq	total_secs
SwlrSivYHoKF9V5wm1YYYAnjHpd9y30PjI9rDUhGJ3k=	201701 121.0	28.0	14.0	29.0	704.0	827.0	184606.903	
rE5wSmHEF1Dhu55zhkiGB1HvotdlShcIMGXv6Vcq02A=	201605 26.0	2.0	5.0	6.0	462.0	256.0	119439.485	
hx+cyaQ/Jcdr/Z5foa/Cn0PXUzC/F7Q0/NQvWQS1Qt=	201611 161.0	71.0	49.0	34.0	668.0	891.0	204791.242	
53QW6B70J23X2UCvxaaUppjyE0b6X9nzP79W4huZv+0=	201502 37.0	9.0	3.0	9.0	408.0	447.0	101186.041	
/0S1N/oRyxGLZlxnW5r0jfo0ZA1s9EH23ahuDNuqz8=	201506 205.0	49.0	23.0	21.0	225.0	489.0	69957.524	
qB/zteXKa0k3hzFCoIUD6wrTp57hnreDX4Vvon25Mfm=	201509 52.0	10.0	10.0	20.0	308.0	264.0	81703.877	
7btpx0qza1gg0ggSw81L05zDYyDj07dXgmwVzYmI2Q=	201610 159.0	53.0	30.0	43.0	1075.0	881.0	30535.79099999997	
kgEhriaQTydVKQ1xn+ZzKQzf4sQ1aod5zcEg5ksyWrE=	201609 88.0	13.0	14.0	13.0	588.0	325.0	159511.737	
8uQ6M70Zdwlsuzo0BRZ6siIPZfBoG43bRvlm+My36B6k=	201509 62.0	27.0	15.0	6.0	1389.0	1427.0	324691.01099999994	
T2gUh1bhFMoSFA9jFI/BkuyE/EPA6oneabYQiGBy9wU=	201702 36.0	23.0	5.0	9.0	216.0	241.0	56296.34000000001	

only showing top 10 rows

First assumptions:

- * The variables "num_percent" are related to songs that can be REPEATED, which means, they only interfere in "num_unq" variable with the rule that their sum cannot be below num_unq --> it would not make sense since if we listen a new song we add another value to num_unq and add this one value in one of num_percent variables AND if we listen again to this song we DO NOT add in num_unq, BUT add in one of num_percent.

Feature Engineering: construir a variavel percentual de musicas finalizadas = num_100/sum(num_25, num_50, num_75, num_985)

Duplicatas e aparicoes

```
In [52]: print("Número de duplicatas encontradas: " + str(verificar_duplicatas(df_logs)))
```

Número de duplicatas encontradas: 0

```
In [20]: logs_contagem = df_logs.groupBy('msno').agg(F.count('*').alias('contagem'))  
logs_contagem.select("contagem").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "99.5%", "max").show()
```

summary	contagem
count	5234111
mean	5.112419472953477
stddev	7.579316644302186
min	1
25%	1
50%	1
75%	4
99.5%	26
max	26

```
In [55]: contagem_logs = logs_contagem.count()  
calcular_distribuicao(logs_contagem, "contagem", contagem_logs, col_id="contagem")
```

contagem	total	pct_total
1	2940775	56.18
2	678146	12.96
26	330373	6.31
3	193813	3.7
4	115104	2.2
5	86369	1.65
6	75191	1.44
7	68650	1.31
8	62998	1.2
13	53738	1.03

only showing top 10 rows

Aproximadamente 56% da base total aparece somente uma vez na base de logs.

```
In [15]: contagem_safra_logs = df_logs.select("safra").distinct().count()  
print(f"O número de safras na base é: {contagem_safra_logs}")
```

O número de safras na base é: 26

4.1.3. feature: num_25

```
In [25]: df_logs.select("num_25").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	num_25
count	26758971
mean	95.42601163549973
stddev	175.28825956456888
min	0.0
1%	0.0
10%	2.0
25%	9.0
50%	40.0
75%	114.0
95%	369.0
99.5%	975.0
max	111864.0

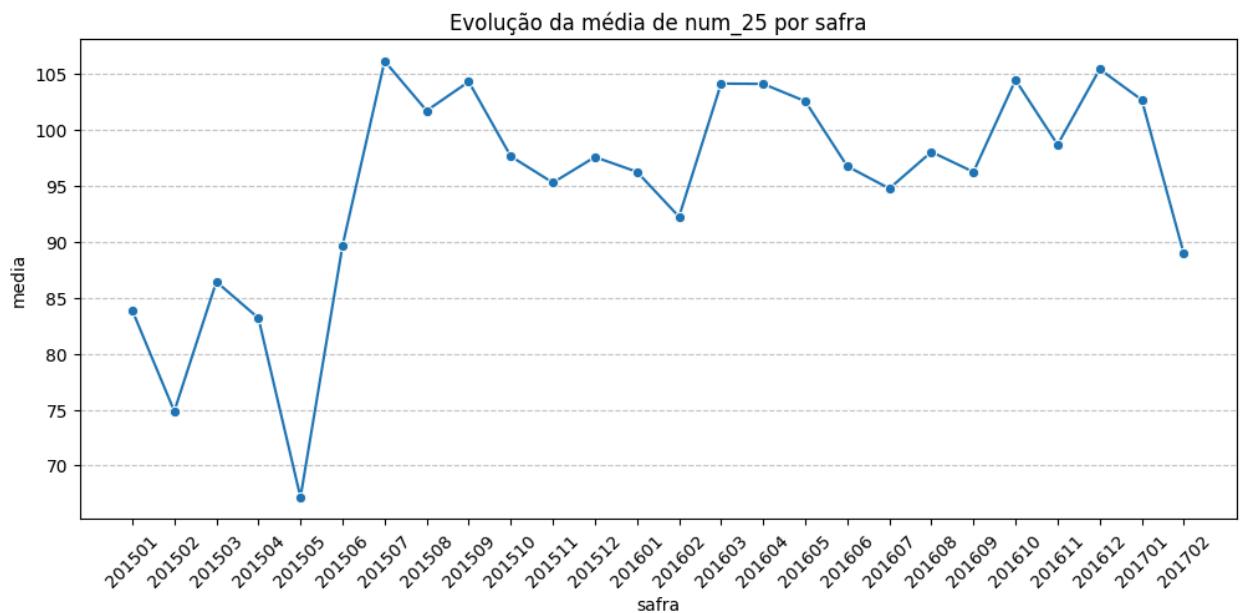
```
In [13]: percentage = (df_logs.filter(F.col("num_25") > 975).select("num_25").count() / total_logs) * 100
print(f'{percentage:.2f}% dos registros têm num_25 > 975')
```

0.50% dos registros têm num_25 > 975

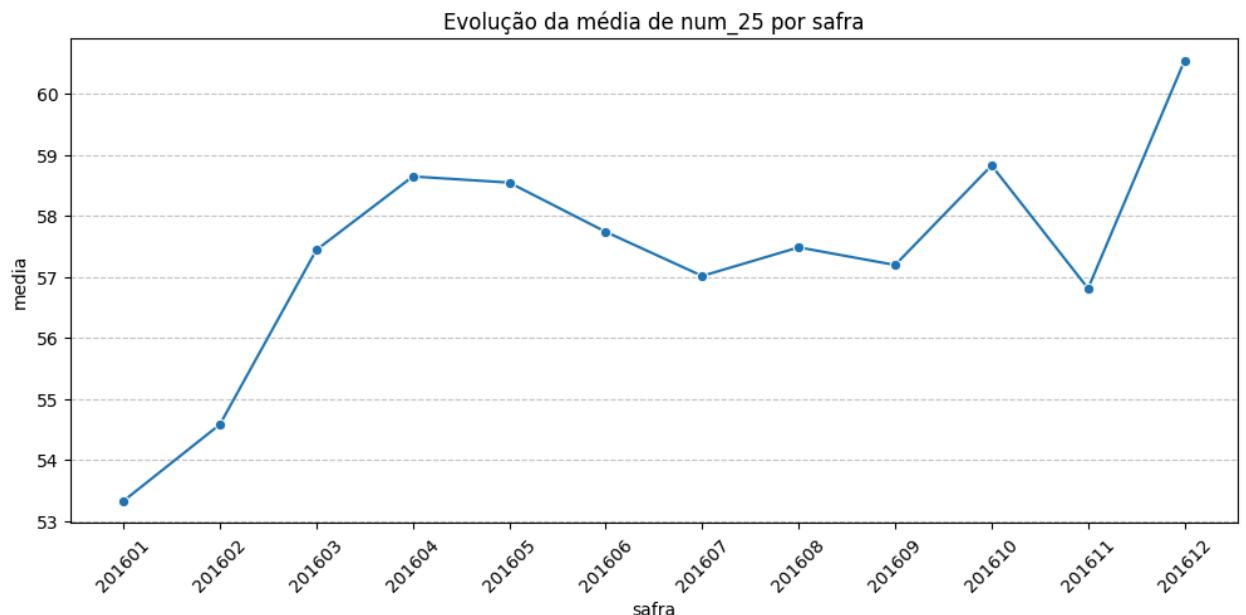
```
In [27]: identificar_outliers(df_logs, "num_25", negativo=False)
```

Variável num_25:
Limites: [0, 246.5]
Total de outliers (#): 2636926
Total de outliers (%): 9.85%

```
In [21]: plot_tendencia_temporal(df_logs, col_valor="num_25", col_safra="safra")
```



```
In [15]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & F.col("num_25").between(0, 247)), col_valor="num_25", col_safra="safra")
```



4.1.4. feature: num_50

```
In [24]: df_logs.select("num_50").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	num_50
count	26758971
mean	24.03566975725636
stddev	39.12554591867814
min	0.0
1%	0.0
10%	0.0
25%	3.0
50%	11.0
75%	30.0
95%	89.0
99.5%	222.0
max	8875.0

```
In [19]: percentage = (df_logs.filter((F.col("num_50") > 30)).select("num_50").count() / total_logs) * 100  
print(f"{percentage:.2f}% dos registros têm num_50 > 30")
```

24.81% dos registros têm num_50 > 30

```
In [30]: percentage = (df_logs.filter((F.col("num_50").between(0, 30))).select("num_50").count() / total_logs) * 100  
print(f"{percentage:.2f}% dos registros têm num_50 entre 0 e 30")
```

75.19% dos registros têm num_50 entre 0 e 30

```
In [18]: percentage = (df_logs.filter((F.col("num_50").between(30, 100))).select("num_50").count() / total_logs) * 100  
print(f"{percentage:.2f}% dos registros têm num_50 entre 30 e 100")
```

21.73% dos registros têm num_50 entre 30 e 100

```
In [31]: percentage = (df_logs.filter((F.col("num_50").between(0, 100))).select("num_50").count() / total_logs) * 100  
print(f"{percentage:.2f}% dos registros têm num_50 entre 0 e 100")
```

96.09% dos registros têm num_50 entre 0 e 100

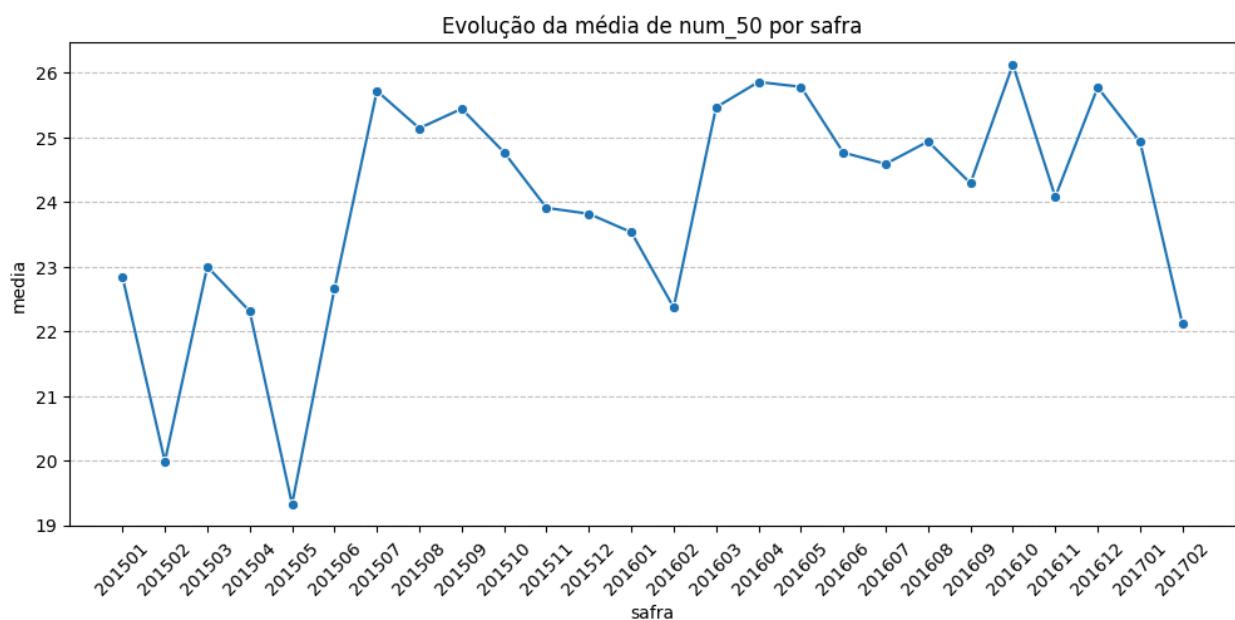
```
In [23]: percentage = (df_logs.filter(F.col("num_50") > 222).select("num_50").count() / total_logs) * 100  
print(f"{percentage:.2f}% dos registros têm num_50 > 222")
```

0.50% dos registros têm num_50 > 222

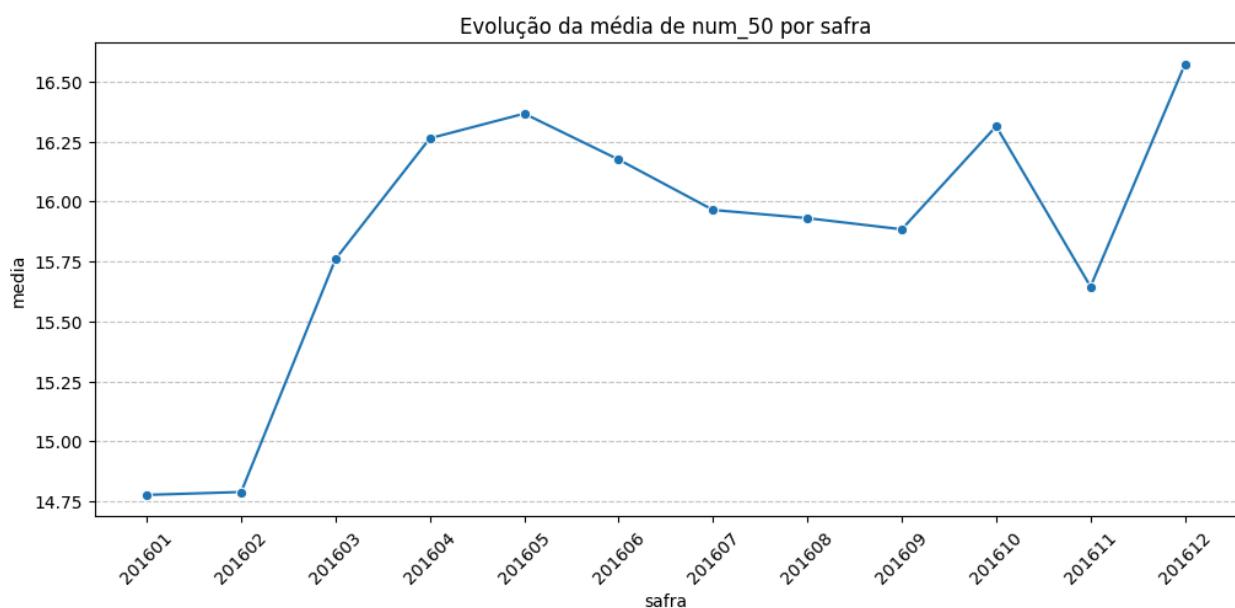
```
In [28]: identificar_outliers(df_logs, "num_50", negativo=False)
```

Variável num_50:
Limites: [0, 68.0]
Total de outliers (#): 2215100
Total de outliers (%): 8.28%

```
In [29]: plot_tendencia_temporal(df_logs, col_valor="num_50", col_safra="safra")
```



```
In [17]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & F.col("num_50").between(0, 68)), col_valor="num_50", col_safra="safra")
```



4.1.5. feature: num_75

```
In [26]: df_logs.select("num_75").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	num_75
count	26758971
mean	14.903521327483034
stddev	22.62672151325168
min	0.0
1%	0.0
10%	0.0
25%	2.0
50%	7.0
75%	20.0
95%	54.0
99.5%	127.0
max	3485.0

```
In [28]: percentage = (df_logs.filter((F.col("num_75").between(0, 100))).select("num_75").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm num_75 entre 0 e 100')
```

98.99% dos registros têm num_75 entre 0 e 100

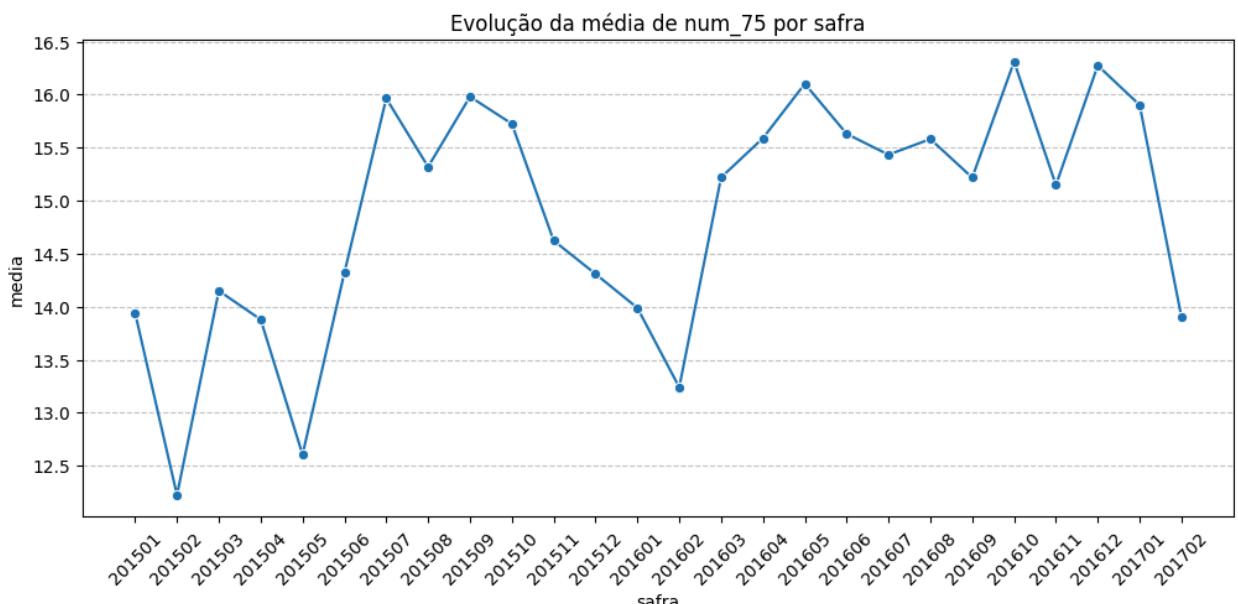
```
In [29]: percentage = (df_logs.filter(F.col("num_75") > 127).select("num_75").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm num_75 > 127')
```

0.49% dos registros têm num_75 > 127

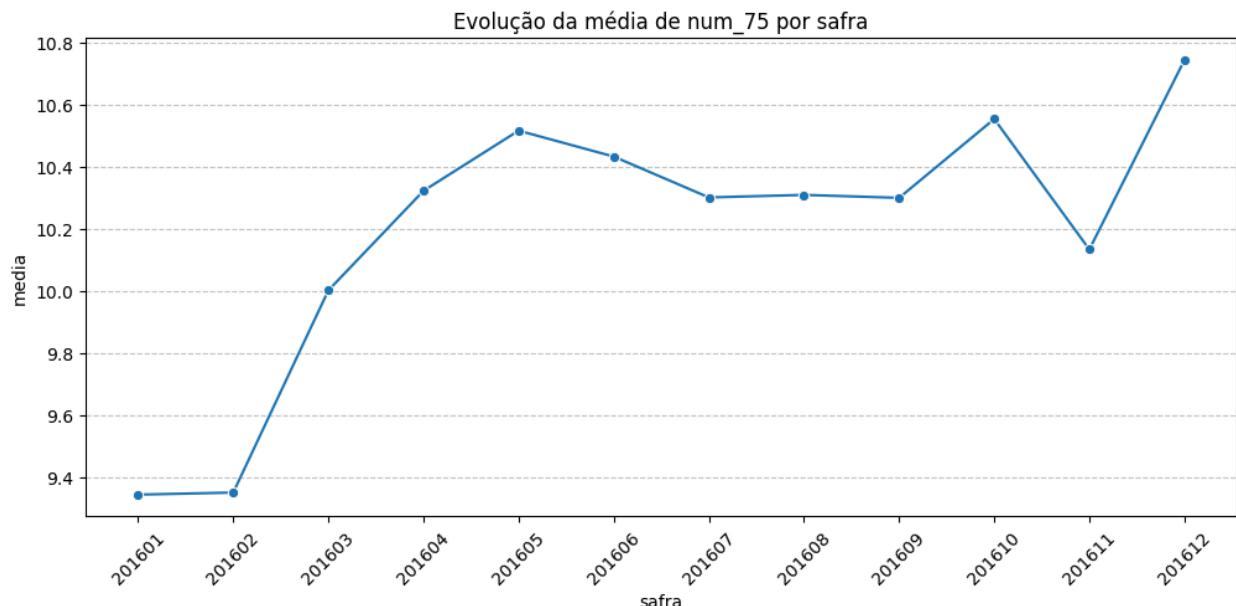
```
In [30]: identificar_outliers(df_logs, "num_75", negativo=False)
```

Variável num_75:
Limites: [0, 42.0]
Total de outliers (#): 2208148
Total de outliers (%): 8.25%

```
In [31]: plot_tendencia_temporal(df_logs, col_valor="num_75", col_safra="safra")
```



```
In [18]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & F.col("num_75").between(0, 42)),  
    col_valor="num_75", col_safra="safra")
```



4.1.6. feature: num_985

```
In [32]: df_logs.select("num_985").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%",  
"max").show()
```

summary	num_985
count	26758971
mean	16.538671797207748
stddev	37.409156681476375
min	0.0
1%	0.0
10%	0.0
25%	1.0
50%	7.0
75%	20.0
95%	60.0
99.5%	164.0
max	37698.0

```
In [33]: percentage = (df_logs.filter((F.col("num_985").between(0, 100))).select("num_985").count() / total_logs) * 100  
print(f"{percentage:.2f}% dos registros têm num_985 entre 0 e 100")
```

98.38% dos registros têm num_985 entre 0 e 100

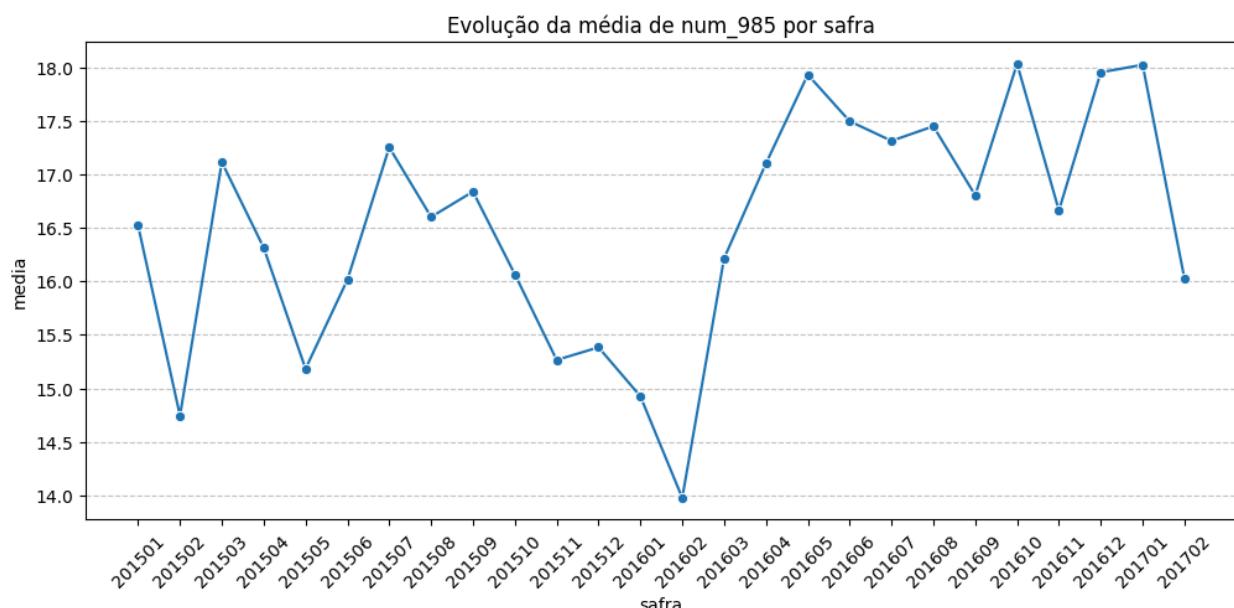
```
In [34]: percentage = (df_logs.filter(F.col("num_985") > 164).select("num_985").count() / total_logs) * 100  
print(f"{percentage:.2f}% dos registros têm num_985 > 127")
```

0.49% dos registros têm num_985 > 127

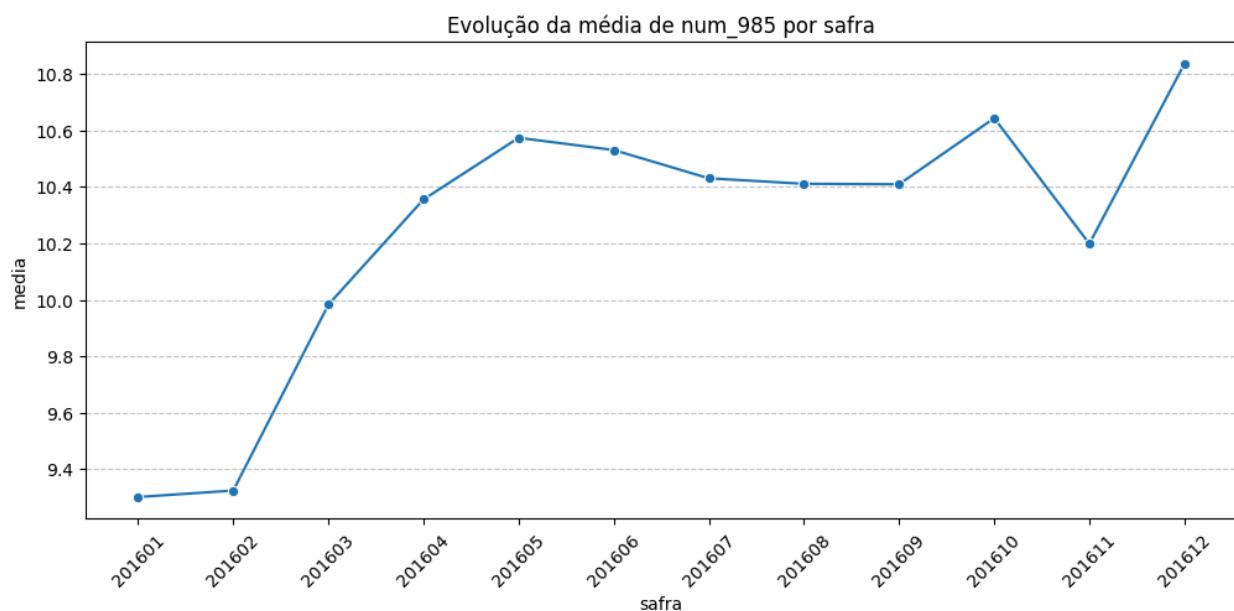
```
In [32]: identificar_outliers(df_logs, "num_985", negativo=False)
```

Variável num_985:
Limites: [0, 44.5]
Total de outliers (#): 2339963
Total de outliers (%): 8.74%

```
In [33]: plot_tendencia_temporal(df_logs, col_valor="num_985", col_safra="safra")
```



```
In [19]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & F.col("num_985").between(0, 43)), col_valor="num_985", col_safra="safra")
```



4.1.7. feature: num_100

```
In [35]: df_logs.select("num_100").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	num_100
count	26758971
mean	450.1598216538297
stddev	725.3139035411615
min	0.0
1%	0.0
10%	2.0
25%	37.0
50%	211.0
75%	564.0
95%	1697.0
99.5%	4365.0
max	196741.0

```
In [37]: percentage = (df_logs.filter(F.col("num_100").between(0, 1700)).select("num_100").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm num_100 entre 0 e 1700')
```

95.02% dos registros têm num_100 entre 0 e 1700

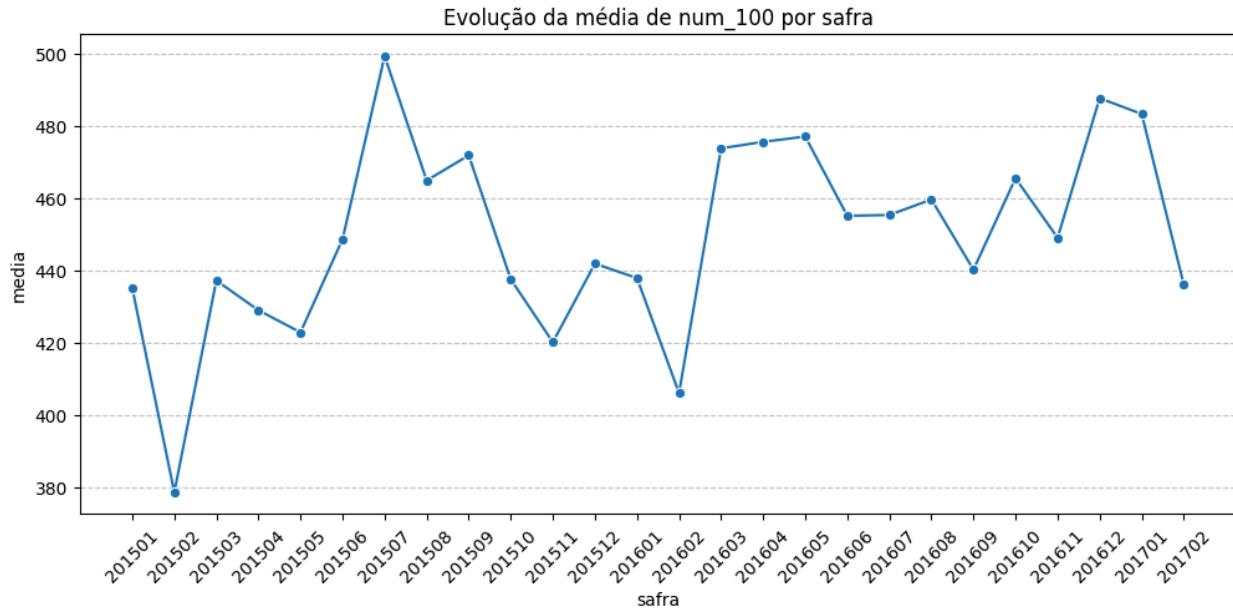
```
In [39]: percentage = (df_logs.filter(F.col("num_100") > 4365).select("num_100").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm num_100 > 4365')
```

0.50% dos registros têm num_100 > 4365

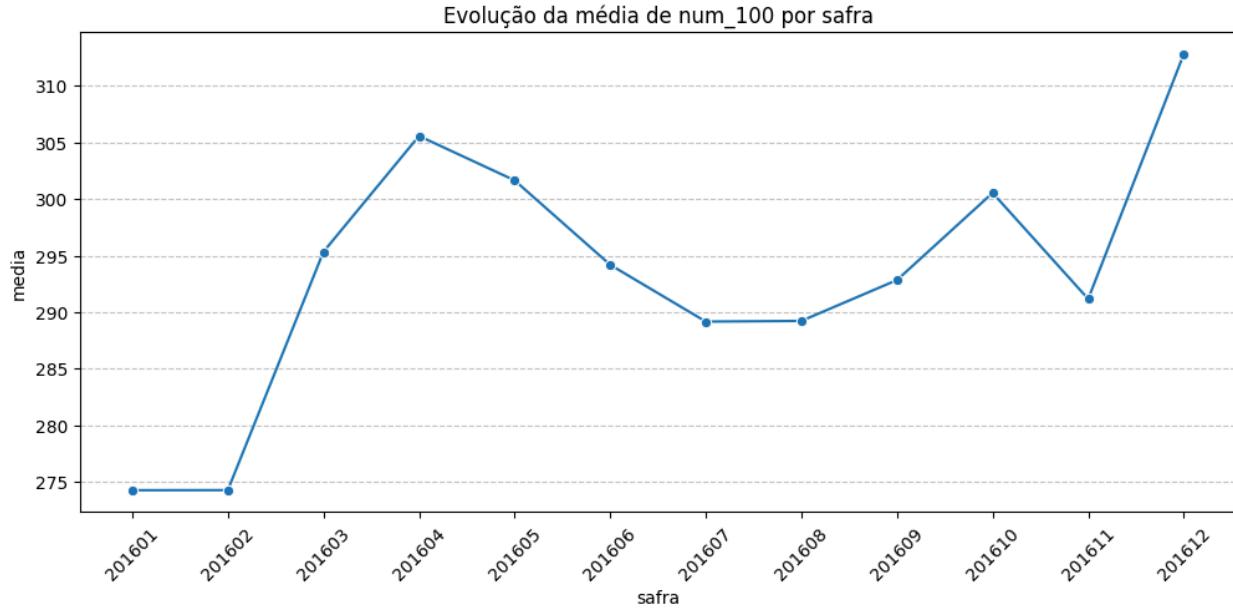
```
In [34]: identificar_outliers(df_logs, "num_100", negativo=False)
```

Variável num_100:
Limites: [0, 1277.5]
Total de outliers (#): 2190020
Total de outliers (%): 8.18%

```
In [35]: plot_tendencia_temporal(df_logs, col_valor="num_100", col_safra="safra")
```



```
In [20]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & F.col("num_100").between(0, 1278)),  
col_valor="num_100", col_safra="safra")
```



4.1.8. feature: num_unq

```
In [42]: df_logs.filter(F.col("num_unq").isNull()).show(1)
```

```
+---+-----+-----+-----+-----+-----+-----+
|msno|safra|num_25|num_50|num_75|num_985|num_100|num_unq|total_secs|
+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+-----+-----+-----+
```

```
In [40]: df_logs.select("num_unq").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+-----+
|summary|      num_unq|
+-----+-----+
| count| 26758971|
| mean| 440.91930526775485|
| stddev| 590.6059571748677|
| min| 1.0|
| 1%| 1.0|
| 10%| 7.0|
| 25%| 50.0|
| 50%| 239.0|
| 75%| 600.0|
| 95%| 1559.0|
| 99.5%| 3398.0|
| max| 32706.0|
+-----+-----+
```

```
In [42]: percentage = (df_logs.filter((F.col("num_unq").between(0, 3400))).select("num_unq").count() / total_logs) * 100
print(f'{percentage:.2f}% dos registros têm num_unq entre 0 e 3400')
```

99.50% dos registros têm num_unq entre 0 e 3400

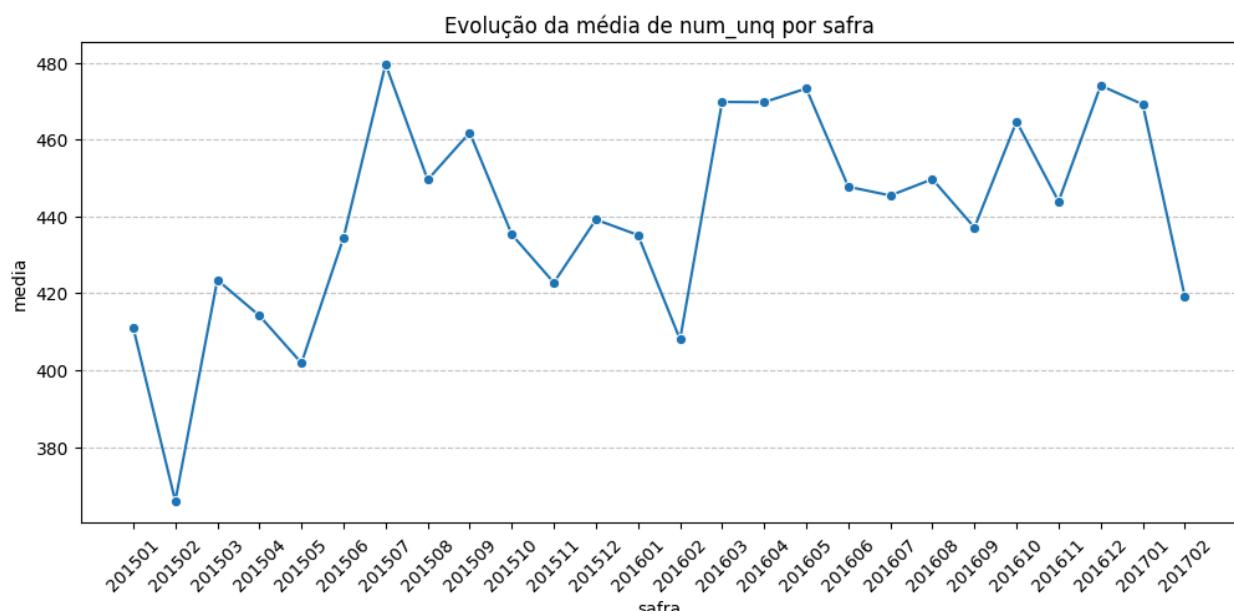
```
In [43]: percentage = (df_logs.filter(F.col("num_unq") > 3398).select("num_unq").count() / total_logs) * 100
print(f'{percentage:.2f}% dos registros têm num_unq > 3398')
```

0.50% dos registros têm num_unq > 3398

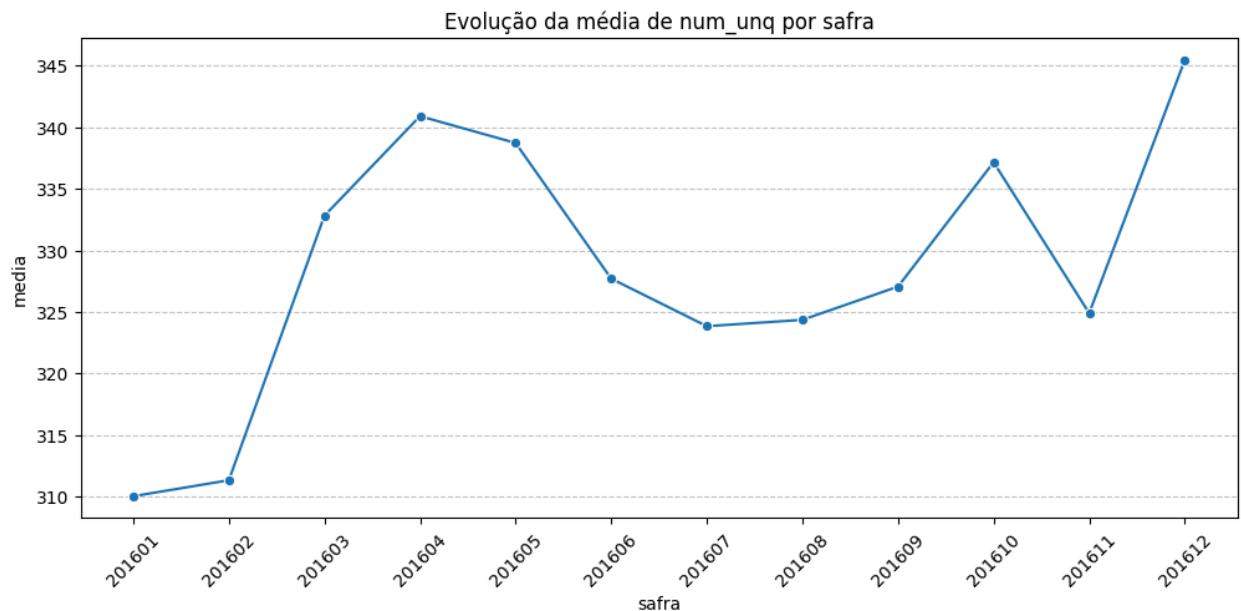
```
In [36]: identificar_outliers(df_logs, "num_unq", negativo=False)
```

```
Variável num_unq:
Limites: [0, 1345.5]
Total de outliers (#): 1842121
Total de outliers (%): 6.88%
```

```
In [37]: plot_tendencia_temporal(df_logs, col_valor="num_unq", col_safra="safra")
```



```
In [21]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & F.col("num_unq").between(0, 1346)),  
    col_valor="num_unq", col_safra="safra")
```



4.1.9. feature: total_secs

```
In [41]: df_logs.filter(F.col("total_secs").isNull()).show(1)
```

```
+-----+-----+-----+-----+-----+-----+-----+  
|msno|safra|num_25|num_50|num_75|num_985|num_100|num_unq|total_secs|  
+-----+-----+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+-----+
```

```
In [44]: df_logs.select("total_secs").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%",  
"max").show()
```

```
+-----+-----+  
|summary|      total_secs|  
+-----+-----+  
| count| 26758971|  
| mean|-2.11717488634759...|  
| stddev|6.839043702349422E14|  
| min|-2.39807672957245...|  
| 1%|     28.797|  
| 10%|    1041.48|  
| 25%|   11021.553|  
| 50%| 58359.429000000004|  
| 75%| 152111.487|  
| 95%| 441495.728|  
| 99.5%| 1072298.1889999998|  
| max|9.223372037135594E15|  
+-----+-----+
```

```
In [46]: percentage = (df_logs.filter((F.col("total_secs") < 0)).select("total_secs").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm total_secs menor do que 0')
```

```
0.16% dos registros têm total_secs menor do que 0
```

```
In [45]: df_logs.filter(F.col("total_secs") > 0).select("total_secs").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+-----+
|summary|      total_secs|
+-----+-----+
| count| 26716625|
| mean|1.596254162738099...|
| stddev|1.210745784107657...|
| min|      0.001|
| 1%|      35.082|
| 10%|     1069.775|
| 25%| 11162.85300000001|
| 50%| 58566.95500000001|
| 75%| 152332.517|
| 95%| 441852.025|
| 99.5%| 1072298.1889999998|
| max|9.223372037135594E15|
+-----+-----+
```

```
In [38]: identificar_outliers(df_logs, "total_secs", negativo=False)
```

```
Variável total_secs:  
Limites: [0, 344114.6870000003]  
Total de outliers (#): 2146972  
Total de outliers (%): 8.02%
```

```
In [42]: (344114/(60*60*24)) # conversão para dias
```

```
Out[42]: 3.982800925925926
```

OBS: Depois de verificar correlação entre as variáveis, seria interessante levantar o seguinte ponto para feature_store: ver qual o valor de num_percent maior e atribuir um peso em cima dessas categorias.

4.1.10. feature: total_days --> from total_secs

```
In [23]: df_logs = df_logs.withColumn("total_days", F.col("total_secs") / (60*60*24))
```

```
In [50]: df_logs.select("total_days").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show(truncate=False)
```

```
+-----+-----+
|summary|total_days   |
+-----+-----+
|count | 26758971   |
|mean  |-2.4504338962349823E8|
|stddev| 7.915559840682272E9|
|min   |-2.775551770338489E12|
|1%   | 3.3329861111111114E-4|
|10%  | 0.012054166666666666 |
|25%  | 0.12756427083333333 |
|50%  | 0.6754563541666667 |
|75%  | 1.7605496180555555 |
|95%  | 5.10990425925926 |
|99.5%| 12.41085866898148 |
|max   | 1.0675199117055086E11|
+-----+-----+
```

Possível separar entre quem ouviu por faixas de dias. Menos de um dia, entre 1 e 3 dias... faz sentido? Ou seria melhor manter somente o valor da variável numérica?

```
In [48]: identificar_outliers(df_logs, "total_days", negativo=False)
```

```
Variável total_days:  
Limites: [0, 3.982808877314815]  
Total de outliers (#): 2146972  
Total de outliers (%): 8.02%
```

```
In [51]: percentage = (df_logs.filter((F.col("total_days").between(0, 15))).select("total_days").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm total_days entre 0 e 15')
```

```
99.64% dos registros têm total_days entre 0 e 15
```

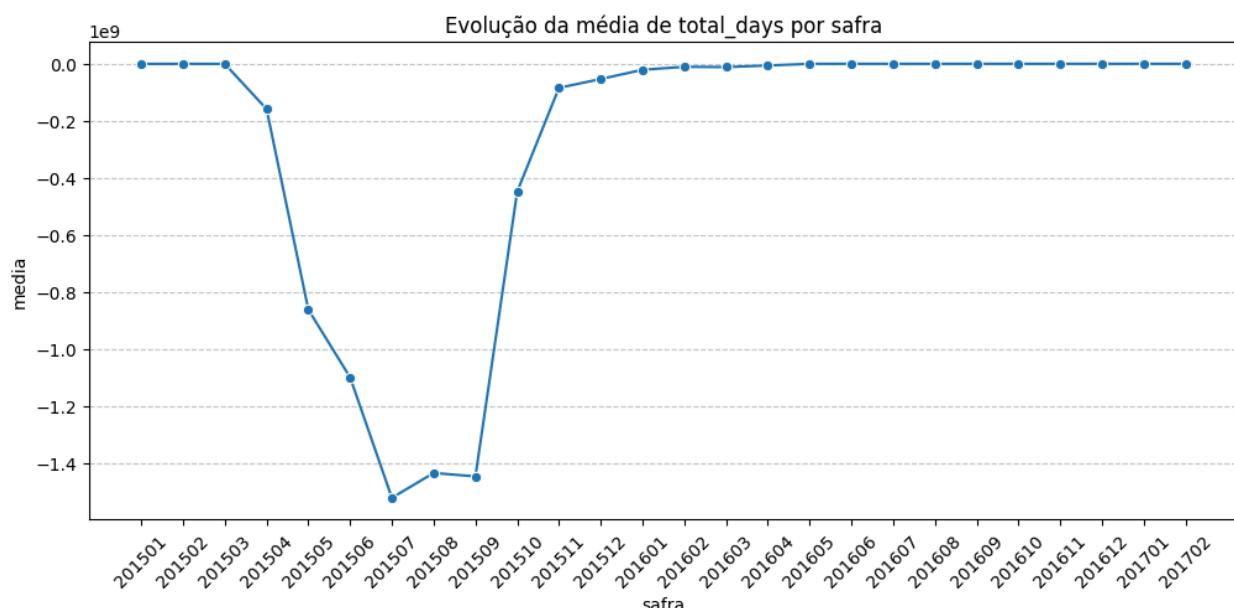
```
In [24]: percentage = (df_logs.filter((F.col("total_days").between(0, 4))).select("total_days").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm total_days entre 0 e 4')
```

92.04% dos registros têm total_days entre 0 e 4

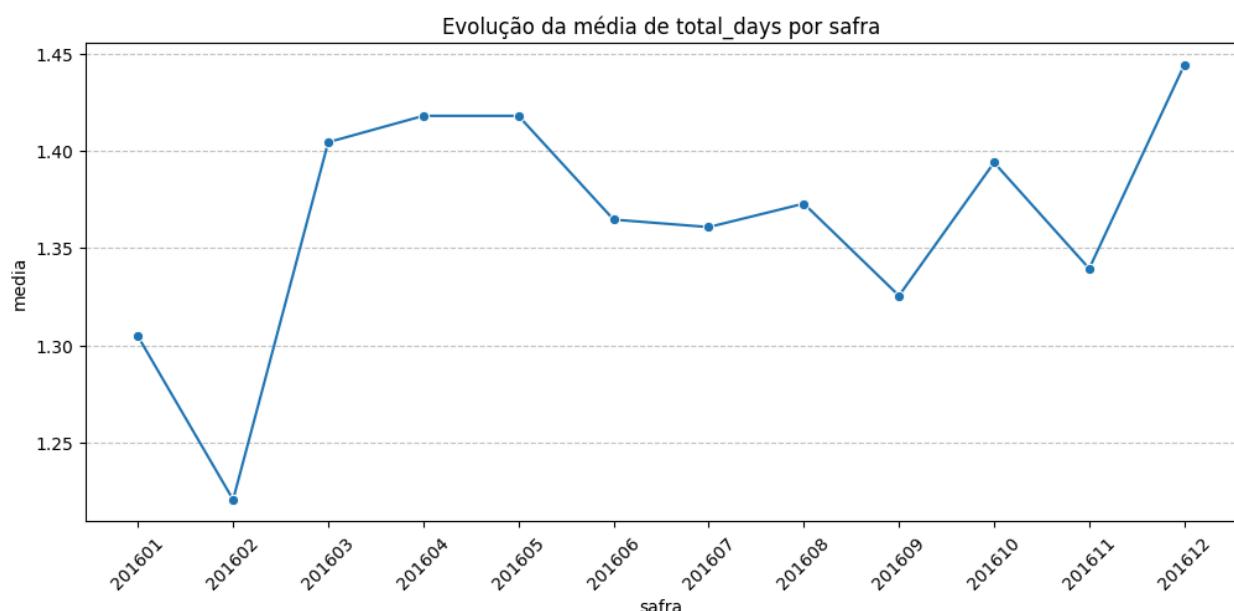
```
In [25]: percentage = (df_logs.filter((F.col("total_days").between(0, 1))).select("total_days").count() / total_logs) * 100  
print(f'{percentage:.2f}% dos registros têm total_days entre 0 e 1')
```

59.44% dos registros têm total_days entre 0 e 1

```
In [52]: plot_tendencia_temporal(df_logs, col_valor="total_days", col_safra="safra")
```



```
In [31]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & F.col("total_days").between(0, 15)),  
col_valor="total_days", col_safra="safra")
```



4.1.11. feature: completed_songs --> from num_numbers

```
In [39]: df_logs = df_logs.withColumn("total_plays", F.col("num_25") + F.col("num_50") + F.col("num_75") + F.col("num_985") +  
F.col("num_100"))  
  
# Cálculo da completude com tratamento de divisão por zero  
df_logs = df_logs.withColumn("completed_songs_rate",  
    F.when(F.col("total_plays") > 0, F.col("num_100") / F.col("total_plays")).otherwise(0.0))  
  
df_logs = df_logs.drop("total_plays")
```

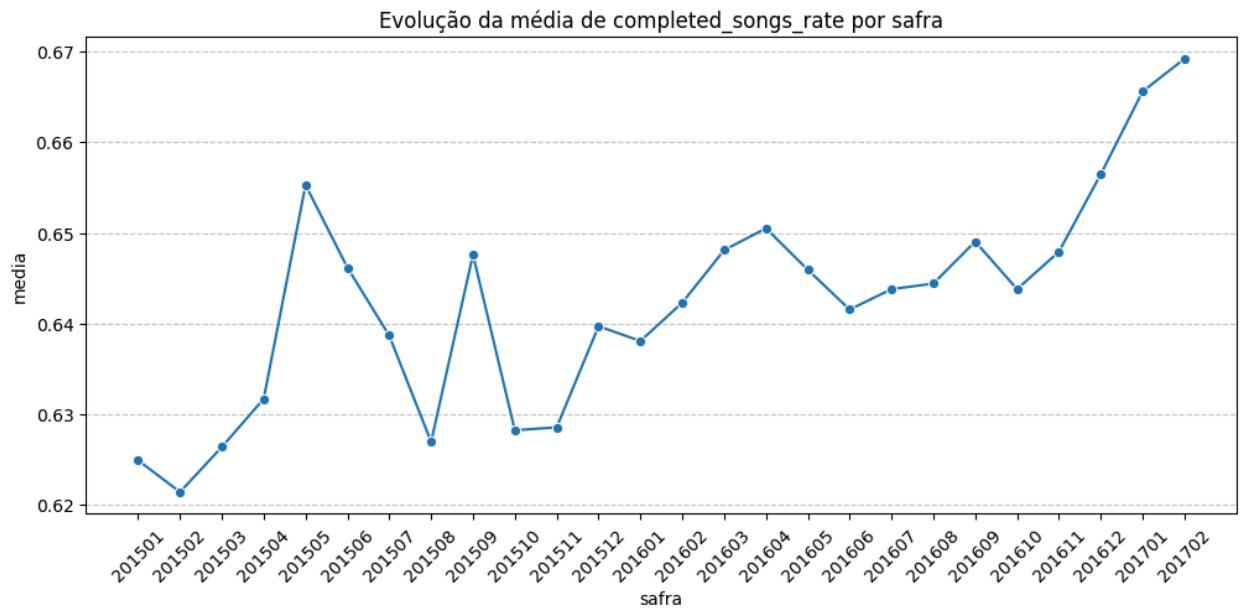
```
In [34]: df_logs.select("completed_songs_rate").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show(truncate=False)
```

```
+-----+-----+
|summary|completed_songs_rate|
+-----+-----+
|count | 26758971 |
|mean  | 0.6429295856112768 |
|stddev| 0.2662612402357104 |
|min   | 0.0 |
|1%   | 0.0 |
|10%  | 0.225 |
|25%  | 0.5 |
|50%  | 0.7058823529411765 |
|75%  | 0.8463576158940397 |
|95%  | 0.9670958512160229 |
|99.5% | 1.0 |
|max   | 1.0 |
+-----+-----+
```

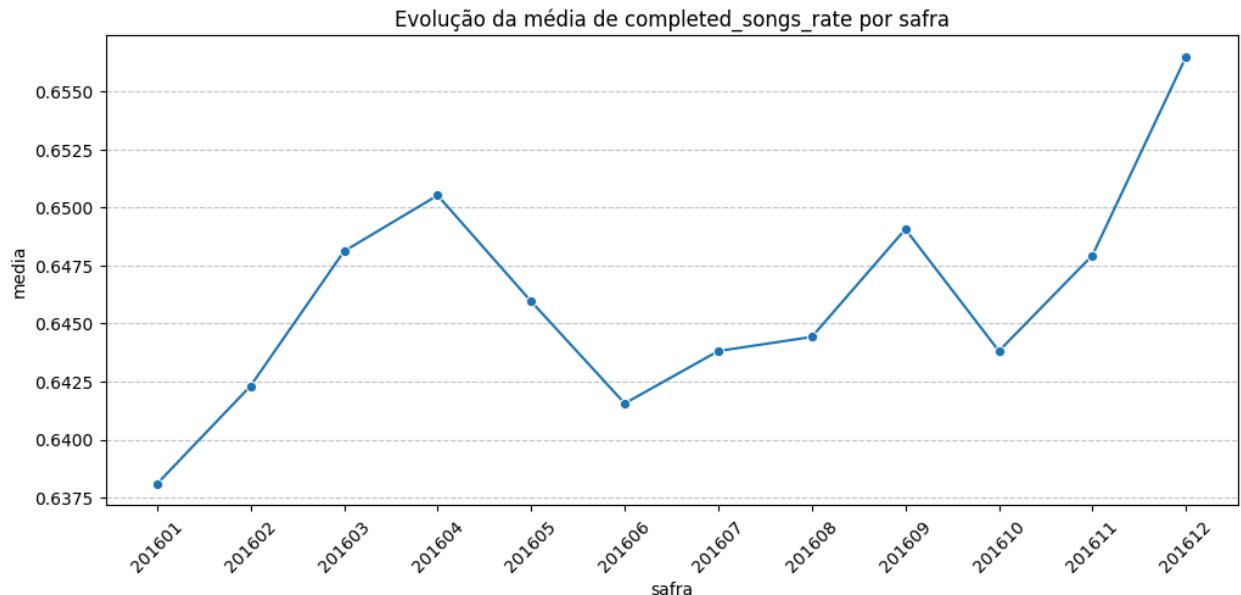
```
In [45]: identificar_outliers(df_logs, "completed_songs_rate", negativo=False)
```

```
Variável completed_songs_rate:  
Limites: [0, 1.3333333333333335]  
Total de outliers (#): 0  
Total de outliers (%): 0.00%
```

```
In [38]: plot_tendencia_temporal(df_logs, col_valor="completed_songs_rate", col_safra="safra")
```



```
In [37]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612))), col_valor="completed_songs_rate", col_safra="safra")
```



4.1.12. feature: direct_cost: $50 + 0.0051 \text{ num_unq} + 0.0001 \text{ total_secs}$

```
In [47]: df_logs = df_logs.withColumn("direct_cost", (50 + (0.0051 * F.col("num_unq")) + (0.0001 * F.col("total_secs"))))  
df_logs = df_logs.withColumn("direct_cost", F.when(F.col("direct_cost") < 50.0, 50.0).otherwise(F.col("direct_cost"))))
```

```
In [48]: df_logs.select("direct_cost").summary("count", "mean", "stddev", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show(truncate=False)
```

summary direct_cost	
count	26758971
mean	1593780.3351396902
stddev	1.2097874054439375E9
min	50.0
1%	50.0094893
10%	50.1433909
25%	51.3857325
50%	57.144163500000005
75%	68.3760931
95%	101.787799
99.5%	172.99474699999996
max	9.223372037680117E11

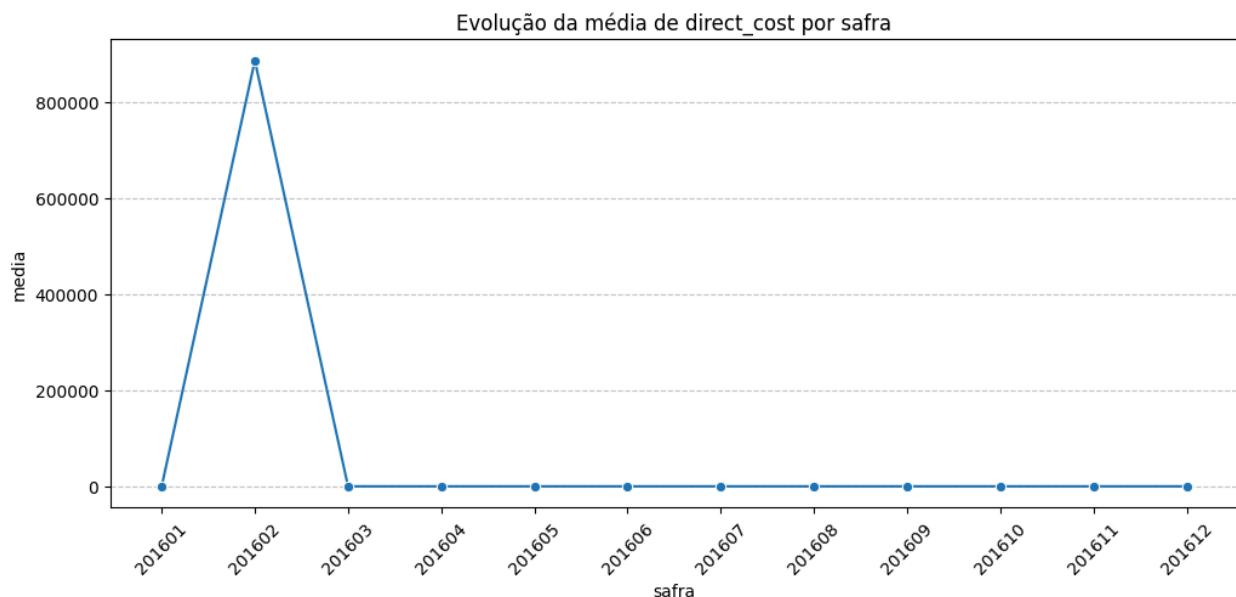
```
In [49]: identificar_outliers(df_logs, "direct_cost")
```

Variável direct_cost:
Limites: [27.221311700000008, 91.50429409999998]
Total de outliers (#): 2030113
Total de outliers (%): 7.59%

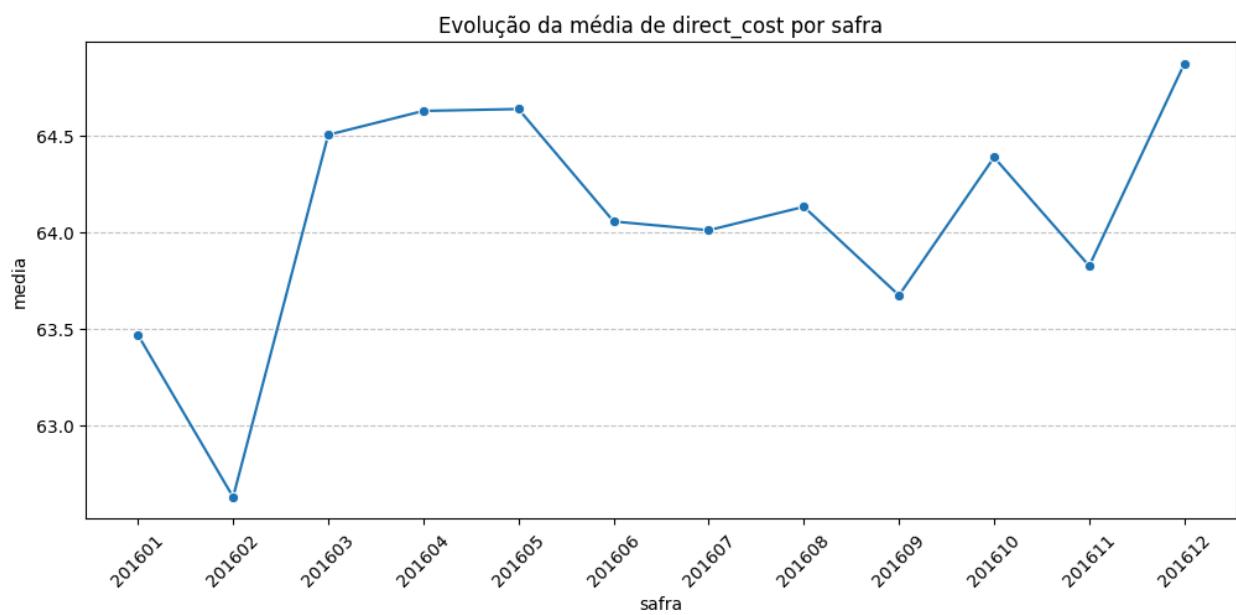
```
In [51]: plot_tendencia_temporal(df_logs, col_valor="direct_cost", col_safra="safra")
```



```
In [52]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612))), col_valor="direct_cost", col_safra="safra")
```



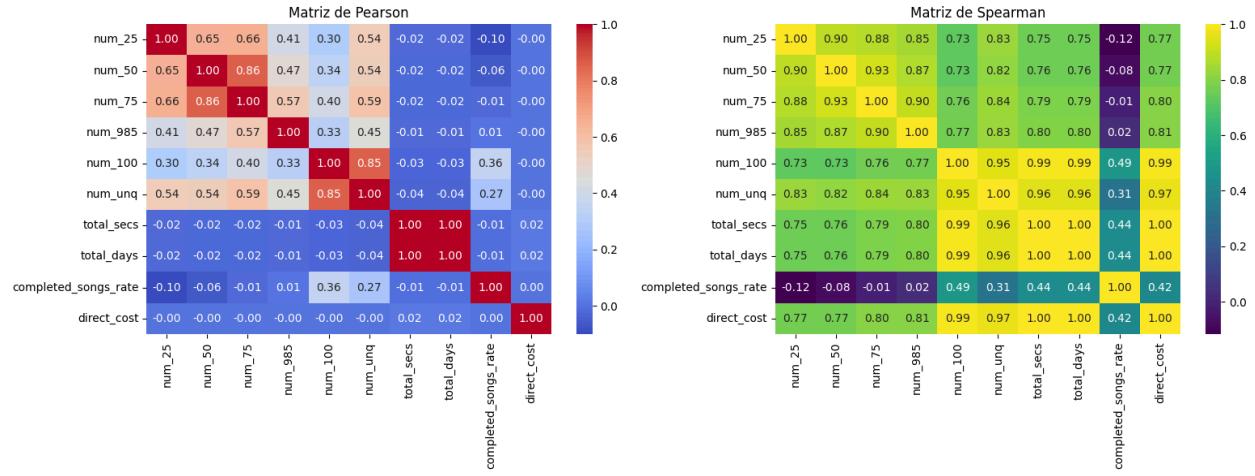
```
In [56]: plot_tendencia_temporal(df_logs.filter((F.col("safra").between(201601, 201612)) & (F.col("direct_cost") < 200)), col_valor="direct_cost", col_safra="safra")
```



4.1.13. Correlação

```
#### 4.1.13.1 Sem tratamento de outliers
```

In [58]: analise_correlacao_quantitativa(df_logs, ["num_25", "num_50", "num_75", "num_985", "num_100", "num_unq", "total_secs", "total_days", "completed_songs_rate", "direct_cost"])



```

(
    num_25      num_50      num_75      num_985     num_100  \
num_25      1.000000  0.650072  0.660435  0.407206  0.298916
num_50      0.650072  1.000000  0.863763  0.468252  0.342047
num_75      0.660435  0.863763  1.000000  0.565739  0.402806
num_985     0.407206  0.468252  0.565739  1.000000  0.325132
num_100     0.298916  0.342047  0.402806  0.325132  1.000000
num_unq     0.544709  0.542394  0.588550  0.454277  0.854030
total_secs   -0.022581 -0.023214 -0.024492 -0.014503 -0.027573
total_days   -0.022581 -0.023214 -0.024492 -0.014503 -0.027573
completed_songs_rate -0.100234 -0.064219 -0.014394  0.011163  0.357497
direct_cost   -0.000330 -0.000192 -0.000211 -0.000165 -0.000469

                                         num_unq  total_secs  total_days  completed_songs_rate  \
num_25      0.544709  -0.022581  -0.022581      -0.100234
num_50      0.542394  -0.023214  -0.023214      -0.064219
num_75      0.588550  -0.024492  -0.024492      -0.014394
num_985     0.454277  -0.014503  -0.014503      0.011163
num_100     0.854030  -0.027573  -0.027573      0.357497
num_unq     1.000000  -0.035933  -0.035933      0.268697
total_secs   -0.035933  1.000000  1.000000      -0.011009
total_days   -0.035933  1.000000  1.000000      -0.011009
completed_songs_rate  0.268697  -0.011009  -0.011009      1.000000
direct_cost   -0.000465  0.017730  0.017730      0.000068

                                         direct_cost
num_25      -0.000330
num_50      -0.000192
num_75      -0.000211
num_985     -0.000165
num_100     -0.000469
num_unq     -0.000465
total_secs   0.017730
total_days   0.017730
completed_songs_rate  0.000068
direct_cost   1.000000 ,
                                         num_25  num_50  num_75  num_985  num_100  \
num_25      1.000000  0.897406  0.879727  0.851440  0.729546
num_50      0.897406  1.000000  0.925688  0.873317  0.725842
num_75      0.879727  0.925688  1.000000  0.901034  0.762779
num_985     0.851440  0.873317  0.901034  1.000000  0.772803
num_100     0.729546  0.725842  0.762779  0.772803  1.000000
num_unq     0.831967  0.818311  0.837759  0.833530  0.951500
total_secs   0.752346  0.755528  0.789534  0.798762  0.989632
total_days   0.752346  0.755528  0.789534  0.798762  0.989632
completed_songs_rate -0.115572 -0.076090 -0.012331  0.023995  0.488532
direct_cost   0.768631  0.768042  0.799572  0.806474  0.987314

                                         num_unq  total_secs  total_days  completed_songs_rate  \
num_25      0.831967  0.752346  0.752346      -0.115572
num_50      0.818311  0.755528  0.755528      -0.076090
num_75      0.837759  0.789534  0.789534      -0.012331
num_985     0.833530  0.798762  0.798762      0.023995
num_100     0.951500  0.989632  0.989632      0.488532
num_unq     1.000000  0.956376  0.956376      0.310327
total_secs   0.956376  1.000000  1.000000      0.441323
total_days   0.956376  1.000000  1.000000      0.441323
completed_songs_rate  0.310327  0.441323  0.441323      1.000000
direct_cost   0.965759  0.999090  0.999090      0.420097

                                         direct_cost
num_25      0.768631
num_50      0.768042
num_75      0.799572
num_985     0.806474

```

```
num_100          0.987314
num_unq         0.965759
total_secs       0.999000
total_days       0.999090
completed_songs_rate 0.420097
direct_cost      1.000000 )
```

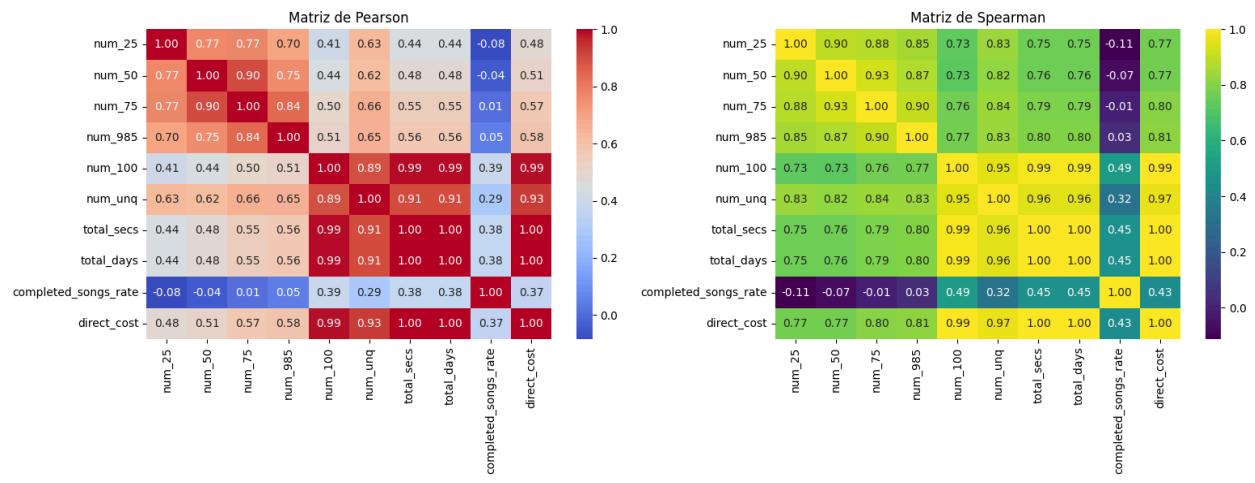
4.1.13.2 Com tratamento de outliers

```
In [15]: df_logs_fix = aplicar_winsorizacao(df_logs, ["num_25", "num_50", "num_75", "num_985", "num_100", "num_unq", "total_secs"])
```

```
Coluna num_25: Limite Inferior=0.0, Limite Superior=746.0
Coluna num_50: Limite Inferior=0.0, Limite Superior=174.0
Coluna num_75: Limite Inferior=0.0, Limite Superior=101.0
Coluna num_985: Limite Inferior=0.0, Limite Superior=122.0
Coluna num_100: Limite Inferior=0.0, Limite Superior=3487.0
Coluna num_unq: Limite Inferior=1.0, Limite Superior=2819.0
Coluna total_secs: Limite Inferior=28.11, Limite Superior=881683.002
```

```
In [16]: df_logs_fix = (df_logs_fix
    .withColumn("total_days", F.col("total_secs") / (60*60*24))
    .withColumn("completed_songs_rate",
        F.when(F.col("num_25") + F.col("num_50") + F.col("num_75") + F.col("num_985") + F.col("num_100") > 0),
        (F.col("num_100") / (F.col("num_25") + F.col("num_50") + F.col("num_75") + F.col("num_985") + F.col("num_100")))).otherwise(0.0))
    .withColumn("direct_cost", (50 + (0.0051 * F.col("num_unq")) + (0.0001 * F.col("total_secs"))))
    .withColumn("direct_cost", F.when(F.col("direct_cost") < 50.0, 50.0).otherwise(F.col("direct_cost")))
)
```

In [17]: analise_correlacao_quantitativa(df_logs_fix, ["num_25", "num_50", "num_75", "num_985", "num_100", "num_unq", "total_secs", "total_days", "completed_songs_rate", "direct_cost"])



```

(
    num_25      num_50      num_75      num_985     num_100  \
num_25  1.000000  0.770349  0.774383  0.698305  0.405307
num_50  0.770349  1.000000  0.902022  0.752818  0.437046
num_75  0.774383  0.902022  1.000000  0.844033  0.499662
num_985 0.698305  0.752818  0.844033  1.000000  0.510703
num_100  0.405307  0.437046  0.499662  0.510703  1.000000
num_unq  0.634334  0.616295  0.661861  0.653989  0.893450
total_secs 0.444207  0.483376  0.546357  0.560823  0.989622
total_days 0.444207  0.483376  0.546357  0.560823  0.989622
completed_songs_rate -0.083729 -0.043263  0.013206  0.048805  0.393819
direct_cost  0.477942  0.509059  0.570208  0.581520  0.987300

                                         num_unq  total_secs  total_days  completed_songs_rate  \
num_25  0.634334  0.444207  0.444207          -0.083729
num_50  0.616295  0.483376  0.483376          -0.043263
num_75  0.661861  0.546357  0.546357          0.013206
num_985 0.653989  0.560823  0.560823          0.048805
num_100  0.893450  0.989622  0.989622          0.393819
num_unq  1.000000  0.905069  0.905069          0.289868
total_secs 0.905069  1.000000  1.000000          0.376775
total_days 0.905069  1.000000  1.000000          0.376775
completed_songs_rate 0.289868  0.376775  0.376775          1.000000
direct_cost  0.930209  0.997989  0.997989          0.368394

                                         direct_cost
num_25  0.477942
num_50  0.509059
num_75  0.570208
num_985 0.581520
num_100 0.987300
num_unq 0.930209
total_secs 0.997989
total_days 0.997989
completed_songs_rate 0.368394
direct_cost 1.000000 ,
                                         num_25  num_50  num_75  num_985  num_100  \
num_25  1.000000 0.897402 0.879724 0.851452 0.729554
num_50  0.897402 1.000000 0.925685 0.873325 0.725849
num_75  0.879724 0.925685 1.000000 0.901039 0.762787
num_985 0.851452 0.873325 0.901039 1.000000 0.772804
num_100 0.729554 0.725849 0.762787 0.772804 1.000000
num_unq 0.831967 0.818306 0.837756 0.833524 0.951502
total_secs 0.752397 0.755581 0.789587 0.798808 0.989690
total_days 0.752397 0.755581 0.789587 0.798808 0.989690
completed_songs_rate -0.112714 -0.072906 -0.008767 0.028163 0.493048
direct_cost  0.772491 0.771909 0.803529 0.810356 0.991888

                                         num_unq  total_secs  total_days  completed_songs_rate  \
num_25  0.831967 0.752397 0.752397          -0.112714
num_50  0.818306 0.755581 0.755581          -0.072906
num_75  0.837756 0.789587 0.789587          -0.008767
num_985 0.833524 0.798808 0.798808          0.028163
num_100 0.951502 0.989690 0.989690          0.493048
num_unq 1.000000 0.956436 0.956436          0.315152
total_secs 0.956436 1.000000 1.000000          0.446130
total_days 0.956436 1.000000 1.000000          0.446130
completed_songs_rate 0.315152 0.446130 0.446130          1.000000
direct_cost 0.970487 0.997391 0.997391          0.426506

                                         direct_cost
num_25  0.772491
num_50  0.771909
num_75  0.803529
num_985 0.810356

```

```

num_100          0.991888
num_unq          0.970487
total_secs       0.997391
total_days       0.997391
completed_songs_rate 0.426506
direct_cost      1.000000 )

```

Conclusões para Feature Selection

- * **Manter:** total_secs , num_unq , completed_songs_rate .
- * **Descartar:** total_days (redundante), num_100 (colinearidade extrema com tempo), num_50 e num_75 (redundantes com num_25).

> Observação: Manter num_unq com total_secs apresenta multicolinearidade, dado que são próximas entre si. No entanto, cada uma tem um peso diferente no custo final: num_unq pesa sobre a diversidade e total_secs sobre o volume de infraestrutura. O fato de a correlação de Spearman (0.96) ser maior que a de Pearson (0.91) sugere que a relação entre elas é extremamente consistente em termos de ranking (quem ouve mais tempo quase sempre ouve mais músicas únicas), mesmo que essa relação não seja perfeitamente linear em todos os níveis de consumo.

4.2. Transactions

4.2.1. dictionary

Definition: transactions of users up until 3/31/2017.

```

* msno: user id
* payment_method_id: payment method
* payment_plan_days: length of membership plan in days
* plan_list_price: in New Taiwan Dollar (NTD)
* actual_amount_paid: in New Taiwan Dollar
(NTD)
* is_auto_renew: automatic renovation on/off
* transaction_date: format %Y%m%d
* membership_expire_date: format %Y%m%d
* is_cancel: whether or not the user canceled the
membership in this transaction.

```

4.2.2. general info

```
In [1]: total_transactions = df_transactions.count()
total_transactions
```

```
-----
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 total_transactions = df_transactions.count()
      2 total_transactions

NameError: name 'df_transactions' is not defined
```

```
In [ ]: df_transactions.printSchema()
```

```

root
|-- msno: string (nullable = true)
|-- payment_method_id: string (nullable = true)
|-- payment_plan_days: string (nullable = true)
|-- plan_list_price: string (nullable = true)
|-- actual_amount_paid: string (nullable = true)
|-- is_auto_renew: string (nullable = true)
|-- transaction_date: string (nullable = true)
|-- membership_expire_date: string (nullable = true)
|-- is_cancel: string (nullable = true)
|-- safra: integer (nullable = true)
```

```
In [ ]: df_transactions.show(10, truncate=False)
```

msno	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	safra
++IZseRQiQS9aaSkH6cMYU6bGDcxUiAi/tH67sC5s= 38				410		1788			0
20151121	20170104	0	201511						
++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc= 41				30		149			1
20150526	20150626	0	201505						
++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc= 41				30		149			1
20150926	20151026	0	201509						
++/9R3sX37CjxbY/AaGvbwr3QkwElKBCtSvVzhCBDDok= 41				30		149			1
20160615	20160715	0	201606						
++/Gw1B9K+X01B3hLTloeUK2Q1Ca2m+Bj8TrzGf7djI= 40				31		149			1
20150113	20150216	0	201501						
++0nOC7BmrUTtcSboR0Rfg6ZXTajnBDt1f/SEgh6ONo= 16				30		149			1
20151012	20151111	0	201510						
++0t0Uy2D3r1pRVxg28G3r3l5PfhFlCPMGElwHqbYL8= 35				7		0			0
20150614	20150615	0	201506						
++1e0qPCRmzyBjMGvAJeaurjI1AFz4Mify6fk2eeccbY= 35				7		0			0
20161101	20161108	0	201611						
++2wdWRV3Thy9HzYRjtKxlNsaa55oDiDc7arR1guiypc= 41				30		149			1
20151008	20151015	1	201510						
++3A6JMzYJeron30GTcDostfXoAl8rTBuB2M8GeVdNU= 41				30		149			1
20150611	20150711	0	201506						

only showing top 10 rows

Duplicatas e aparicoes

```
In [ ]: print("Número de duplicatas encontradas: " + str(verificar_duplicatas(df_transactions)))
```

Número de duplicatas encontradas: 0

```
In [ ]: trans_contagem = df_transactions.groupBy('msno').agg(F.count('*').alias('contagem'))
trans_contagem.select("contagem").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "99.5%", "max").show()
```

summary	contagem
count	2363626
mean	8.762902845035551
stddev	8.580922690781978
min	1
25%	1
50%	5
75%	15
99.5%	26
max	26

```
In [ ]: contagem_trans = trans_contagem.count()
calcular_distribuicao(trans_contagem, "contagem", contagem_trans, col_id="contagem")
```

contagem	total	pct_total
1	665663	28.16
2	313299	13.26
26	182281	7.71
3	112866	4.78
4	80113	3.39
19	79431	3.36
14	65923	2.79
22	64390	2.72
8	63216	2.67
5	61995	2.62

only showing top 10 rows

```
In [ ]: contagem_safra_trans = df_transactions.select("safra").distinct().count()
print(f"O número de safras na base é: {contagem_safra_trans}")
```

O número de safras na base é: 26

4.2.3. features: plan_list_price + actual_amount_paid

```
In [39]: df_transactions.select("plan_list_price").distinct().orderBy("plan_list_price").show(100)
```

```
+-----+  
|plan_list_price|  
+-----+  
| 0|  
| 1|  
| 10|  
| 100|  
| 1000|  
| 105|  
| 1150|  
| 119|  
| 120|  
| 1200|  
| 124|  
| 126|  
| 129|  
| 131|  
| 134|  
| 143|  
| 149|  
| 15|  
| 150|  
| 1520|  
| 1599|  
| 1788|  
| 180|  
| 1825|  
| 2000|  
| 210|  
| 265|  
| 298|  
| 30|  
| 300|  
| 35|  
| 350|  
| 400|  
| 44|  
| 447|  
| 450|  
| 477|  
| 480|  
| 50|  
| 500|  
| 536|  
| 596|  
| 600|  
| 699|  
| 70|  
| 760|  
| 799|  
| 800|  
| 894|  
| 930|  
| 99|  
+-----+
```

```
In [16]: df_transactions.groupBy("plan_list_price", "actual_amount_paid").count().orderBy(F.desc("count")).show(40)
```

plan_list_price	actual_amount_paid	count
149	149	11233604
99	99	4774736
129	129	1096720
0	149	764403
180	180	656705
0	0	562167
149	0	484549
150	150	360929
149	119	270324
894	894	107881
1788	1788	78940
100	100	75569
536	536	42850
119	119	29195
0	129	25410
0	119	23335
480	480	22736
0	150	12462
1599	1599	11369
477	477	10752
799	799	6220
300	300	5342
1200	1200	5275
298	298	5167
930	930	5026
120	120	4462
35	35	4144
0	1788	3869
447	447	3805
134	134	3567
0	894	3528
149	129	2129
450	450	1334
149	99	1323
0	536	1279
500	500	1094
0	134	1058
699	699	887
120	0	859
50	50	748

only showing top 40 rows

```
In [52]: calcular_distribuicao(df_transactions, ["plan_list_price", "actual_amount_paid"], n_show=10)
```

plan_list_price	actual_amount_paid	total	pct_total
149	149	11233604	54.24
99	99	4774736	23.05
129	129	1096720	5.3
0	149	764403	3.69
180	180	656705	3.17
0	0	562167	2.71
149	0	484549	2.34
150	150	360929	1.74
149	119	270324	1.31
894	894	107881	0.52

only showing top 10 rows

```
Out[52]: DataFrame[plan_list_price: string, actual_amount_paid: string, total: bigint, pct_total: double]
```

Nem tudo o que deveriam pagar foi efetivamente pago em alguns casos, bem como aparentemente houveram pagamentos que não deveriam ter sido executados.

```
In [55]: df_transactions_anl = df_transactions.withColumn("payment_diff", F.col("actual_amount_paid") - F.col("plan_list_price"))
df_transactions_anl = df_transactions_anl.withColumn("flag_pag_diff", F.when(F.col("payment_diff") != 0, 1).otherwise(0))
```

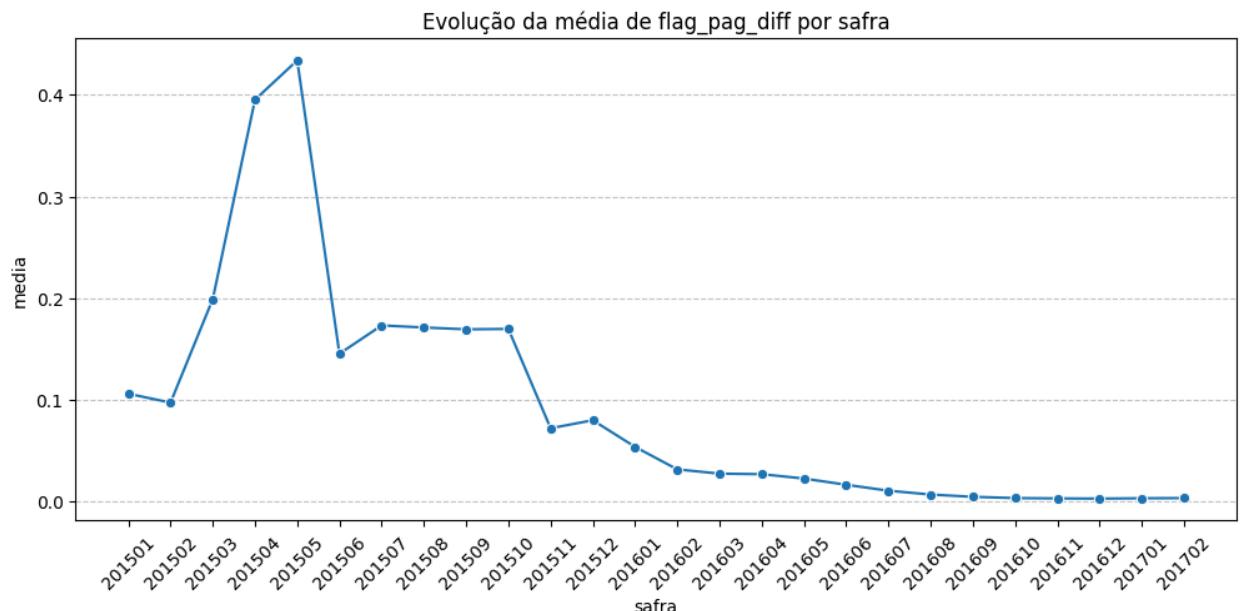
```
In [56]: percentage = df_transactions_anl.filter(F.col("flag_pag_diff") == 1).count() / df_transactions_anl.count() * 100
print(f'{percentage:.2f}% dos registros têm divergência no pagamento.')
```

7.71% dos registros têm divergência no pagamento.

```
In [63]: flag_pag_diff_per_safra = calcular_distribuicao(df_transactions_anl, ["flag_pag_diff"], n_show=60, agrupar_por_safra=True)
```

safra	flag_pag_diff	total	pct_safra	pct_total
201501 0	490695	89.41	2.37	
201501 1	58097	10.59	0.28	
201502 0	492343	90.29	2.38	
201502 1	52960	9.71	0.26	
201503 0	501826	80.1	2.42	
201503 1	124662	19.9	0.6	
201504 0	341355	60.46	1.65	
201504 1	223227	39.54	1.08	
201505 0	323465	56.59	1.56	
201505 1	248087	43.41	1.2	
201506 0	663052	85.47	3.2	
201506 1	112685	14.53	0.54	
201507 0	550109	82.69	2.66	
201507 1	115171	17.31	0.56	
201508 0	585194	82.89	2.83	
201508 1	120781	17.11	0.58	
201509 0	593645	83.07	2.87	
201509 1	120965	16.93	0.58	
201510 0	565009	83.03	2.73	
201510 1	115456	16.97	0.56	
201511 0	761390	92.81	3.68	
201511 1	58955	7.19	0.28	
201512 0	792430	92.02	3.83	
201512 1	68677	7.98	0.33	
201601 0	810857	94.65	3.91	
201601 1	45859	5.35	0.22	
201602 0	767444	96.86	3.71	
201602 1	24856	3.14	0.12	
201603 0	754395	97.28	3.64	
201603 1	21074	2.72	0.1	
201604 0	753535	97.33	3.64	
201604 1	20634	2.67	0.1	
201605 0	766440	97.77	3.7	
201605 1	17516	2.23	0.08	
201606 0	791680	98.38	3.82	
201606 1	13049	1.62	0.06	
201607 0	914446	98.96	4.42	
201607 1	9586	1.04	0.05	
201608 0	960079	99.34	4.64	
201608 1	6371	0.66	0.03	
201609 0	978356	99.56	4.72	
201609 1	4284	0.44	0.02	
201610 0	1030687	99.69	4.98	
201610 1	3211	0.31	0.02	
201611 0	1091882	99.72	5.27	
201611 1	3059	0.28	0.01	
201612 0	966029	99.74	4.66	
201612 1	2518	0.26	0.01	
201701 0	985678	99.71	4.76	
201701 1	2898	0.29	0.01	
201702 0	882826	99.69	4.26	
201702 1	2740	0.31	0.01	

```
In [62]: plot_tendencia_temporal(df_transactions_anl, col_valor="flag_pag_diff", col_safra="safra")
```



```
In [64]: flag_pag_diff_per_safra.filter(F.col("flag_pag_diff").isin([0])).groupBy("flag_pag_diff").agg(F.mean("pct_safra").alias("media_pgt_correto")).orderBy("flag_pag_diff").show(1)
```

flag_pag_diff	media_pgt_correto
0	90.67346153846152

90% dos pagamentos feitos estão concordantes com o valor cobrado. Os demais são casos de erro?

4.2.4. feature: payment_method_id

Verificar se o método de pagamento dos clientes mudou ao longo do período

```
In [22]: verificar_mudanca_estado(df_transactions, col_alvo="payment_method_id", visualizar=True)
```

Contagem de mudanças por safra:

safra	mudou	count
201502	true	3637
201503	true	4998
201504	true	7434
201505	true	8885
201506	true	24154
201507	true	13619
201508	true	13737
201509	true	14670
201510	true	13727
201511	true	13463
201512	true	19871
201601	true	14609
201602	true	22316
201603	true	13574
201604	true	12695
201605	true	13231
201606	true	14387
201607	true	27179
201608	true	25557
201609	true	24162
201610	true	28990
201611	true	36827
201612	true	20930
201701	true	17245
201702	true	14828

+-----+-----+

Mudanças por usuário:

msno	total_mudancas
++hTNyKbQJOnbwH4zStU+NGBhqxsUwQ++qQwLaZJliA= 1	
++wAgGfr8qio0G5N+4vbhDT5Mnv6uql117T5AnNxPTc= 2	
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 1	
+03f1Gdx1ny5j09r7j09K1WGamqEGQvXxG5DfDK+5Sg= 1	
+03Ty76kp19253f7CsQxW69sd3437dnf+aHP++cE4EY= 1	
+0ISc0kpp1AoLWDzLGjm7g2W6yqJrVR/dCgc14+XRn0= 1	
+0JC575nvKJGCKJYoPhdAtd9HU807Shpkf1N+jWMQwg= 1	
+0PFkru2UpsbwRoqF6eHpLNNUbfhJax1z1NjPyJwZ1Y= 2	
+0WreIqj9Sk7/BExszmjOoDZNtrWT84wgqGH1hrzy84= 1	
+0odXIbHS3RcBVjn+wPz2oNE86/rfLomwA5CcCEfBnA= 2	

+-----+-----+

only showing top 10 rows

A mudança no método de pagamento é fator relevante para o cancelamento da assinatura do cliente?

```
In [23]: # Janela por usuário para ver histórico
window_msno = Window.partitionBy("msno").orderBy("transaction_date")

df_trans_features = (df_transactions
    # Criar coluna do método de pagamento anterior para mapear mudança e verificar se afeta na taxa de cancelamento
    .withColumn("payment_method_id_anterior", F.lag("payment_method_id").over(window_msno))
    .withColumn("mudou_metodo_pagto",
        F.when(F.col("payment_method_id_anterior").isNull(), 0)
        .when(F.col("payment_method_id") != F.col("payment_method_id_anterior"), 1)
        .otherwise(0))
)

# Resumo de relevância: Média de cancelamento para quem muda método
df_trans_features_agg = df_trans_features.groupBy("safra", "mudou_metodo_pagto").agg(
    F.round(F.avg("is_cancel") * 100, 2).alias("taxa_cancelamento_media"),
    F.count("msno").alias("volumetria")).orderBy("safra", "mudou_metodo_pagto")

df_trans_features_agg.groupBy("mudou_metodo_pagto").agg(
    F.round(F.avg("taxa_cancelamento_media"), 2).alias("taxa_cancelamento_media_geral"),
    F.sum("volumetria").alias("volumetria_total")).orderBy("mudou_metodo_pagto").show(10, truncate=False)
```

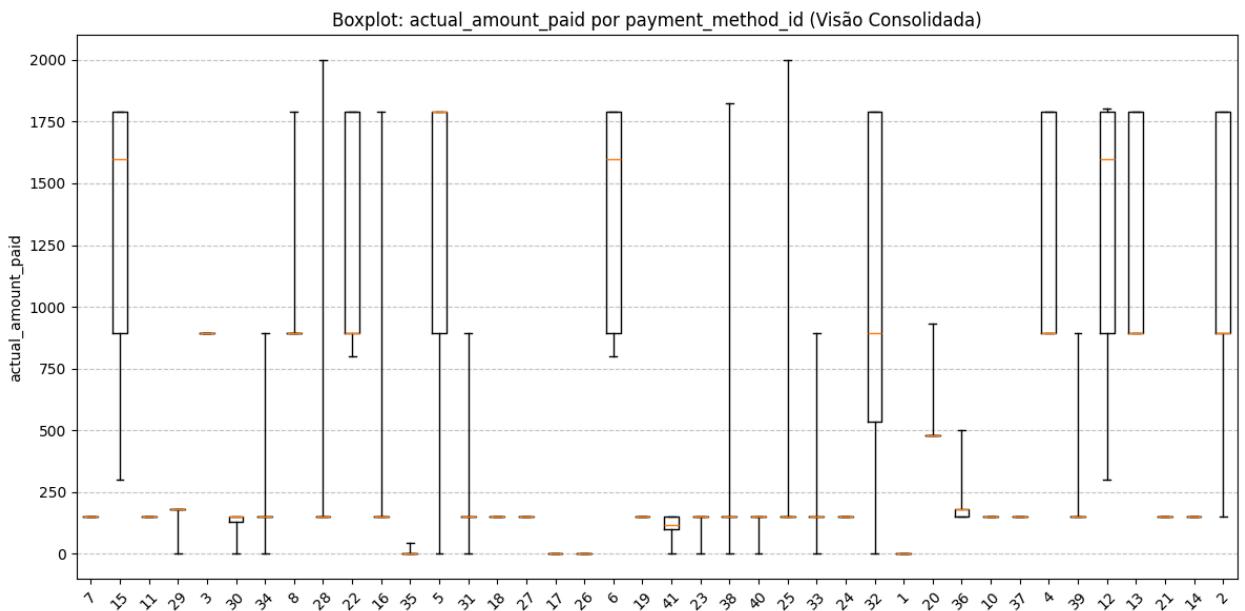
mudou_metodo_pagto	taxa_cancelamento_media_geral	volumetria_total
0	3.43	20287500
1	0.66	424725

A mudança no método de pagamento não tem indícios de ser relevante para o cancelamento.

Pendente: verificar a margem líquida média de quem mudou o método de pagamento e de quem não mudou

In [42]: `plot_boxplot(df_transactions, ["payment_method_id"], "actual_amount_paid", agrupar_por_safra=False)`

Processando estatísticas para: payment_method_id...



Método A: mediana = 100 → variação enorme

Método B: mediana = 1000 → variação pequena

Isso permite comparar métodos baratos e caros na mesma escala.

Conceito estatístico: por que IQR? Por que não média e desvio padrão?

- * Média é puxada por outliers
- * Desvio padrão explode em distribuições assimétricas
- * Preços e pagamentos não são normais

Por que IQR?

- * Robusto a outliers
- * Reflete comportamento típico
- * Muito usado em análise exploratória e boxplots

```
In [14]: stats_pm = (df_transactions
    .groupBy("payment_method_id")
    .agg(
        F.expr("percentile_approx(actual_amount_paid, 0.25)").alias("q1"),
        F.expr("percentile_approx(actual_amount_paid, 0.75)").alias("q3"),
        F.expr("percentile_approx(actual_amount_paid, 0.5)").alias("median"))
    .withColumn("iqr", F.col("q3") - F.col("q1"))
    .withColumn("iqr_rel",
        F.when(F.col("median") > 0, F.col("iqr") / F.col("median"))
        .otherwise(F.lit(0)))
)
)
```

```
In [15]: IQR_FIXO = 1e-6 # IQR proximo a zero: Todos os pagamentos têm o mesmo valor
IQR_REL_BAIXO = 0.25 # Até 25% de variação relativa é considerado "estável": 50% central dos dados varia no máximo ±12.5% da mediana
MEDIANA_ALTA = 500 # Baseado no que foi visto nos graficos boxplot, parece razoável considerar alta a mediana acima de 500
```

```
In [20]: stats_pm = stats_pm.withColumn("payment_method_price_profile",
    F.when(F.col("iqr") <= IQR_FIXO, "fixo")
    .when((F.col("iqr_rel") <= IQR_REL_BAIXO) & (F.col("median") < MEDIANA_ALTA), "baixo_estavel")
    .when((F.col("iqr_rel") <= IQR_REL_BAIXO) & (F.col("median") >= MEDIANA_ALTA), "alto_estavel")
    .when((F.col("iqr_rel") > IQR_REL_BAIXO) & (F.col("median") < MEDIANA_ALTA), "misto_baixo")
    .otherwise("misto_alto"))
```

```
In [21]: df_transactions_enriched = (df_transactions
    .join(
        stats_pm.select("payment_method_id", "payment_method_price_profile"),
        on="payment_method_id",
        how="left")
)
```

Agrupamento de métodos de pagamento por perfil de preço (IQR)

Neste projeto, os métodos de pagamento foram agrupados com base na variabilidade do valor efetivamente pago (`actual_amount_paid`), utilizando a métrica robusta Interquartile Range (IQR). Essa abordagem permite identificar métodos associados a preços fixos, planos estáveis de baixo valor, planos estáveis de alto valor e métodos com alta heterogeneidade de cobrança.

Optou-se por utilizar o valor efetivamente pago em vez do preço de tabela (`plan_list_price`), pois a margem líquida é diretamente impactada pela receita real recebida, incorporando efeitos de descontos, promoções e campanhas comerciais. O preço de tabela foi mantido como variável complementar, pois representa a política comercial e o tipo estrutural de plano contratado.

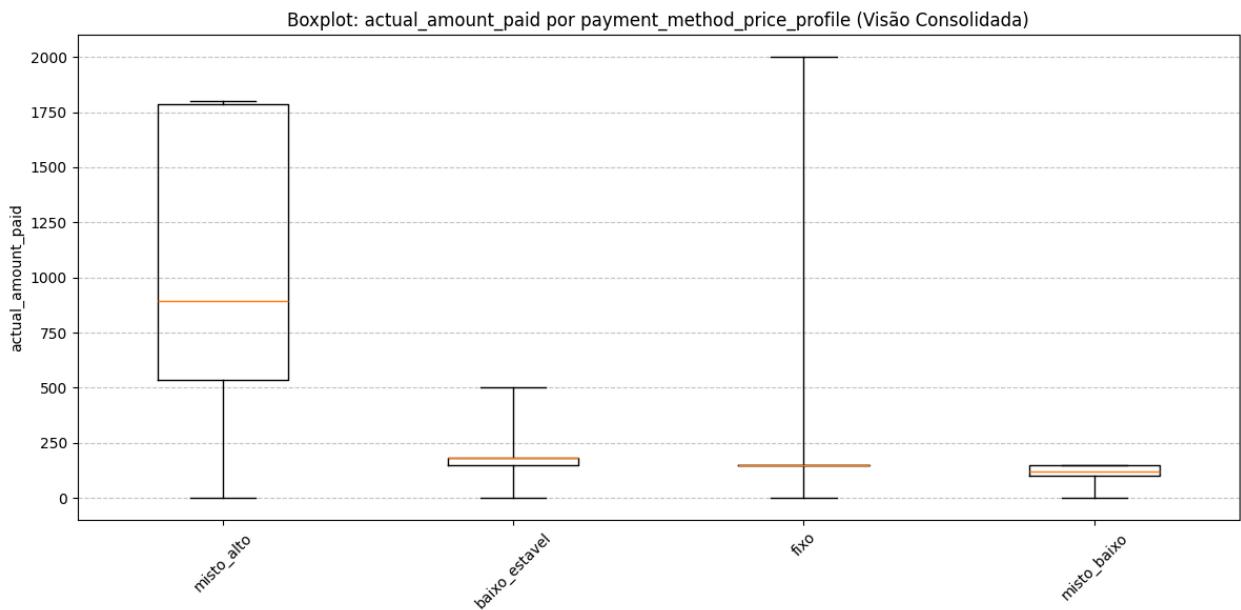
O agrupamento foi construído exclusivamente com variáveis disponíveis no momento da transação, garantindo viabilidade de implantação em produção e evitando target leakage.

Target leakage ocorre quando informações que só estariam disponíveis após o evento a ser previsto (como consumo futuro ou margem líquida no mês seguinte) são utilizadas durante o treinamento do modelo, resultando em métricas artificialmente otimistas e modelos inviáveis em cenários reais.

Dessa forma, o agrupamento de métodos de pagamento é tratado como uma feature estática de lookup, reduzindo dimensionalidade, controlando variância e aumentando interpretabilidade, sem introduzir viés indevido na modelagem.

```
In [22]: plot_boxplot(df_transactions_enriched, ["payment_method_price_profile"], "actual_amount_paid", agrupar_por_safra=False)
```

Processando estatísticas para: payment_method_price_profile...



```
In [23]: calcular_distribuicao(df_transactions_enriched, ["payment_method_price_profile"], n_show=10, agrupar_por_safra=False)
```

payment_method_price_profile	total	pct_total
misto_baixo	11026911	53.24
fixo	18541108	41.24
baixo_estavel	967633	4.67
misto_alto	176573	0.85

Out[23]: DataFrame[payment_method_price_profile: string, total: bigint, pct_total: double]

Agrupamento de métodos de pagamento por perfil de preço

Com base na variabilidade e na escala do valor efetivamente pago (`actual_amount_paid`), os métodos de pagamento foram agrupados em quatro perfis distintos, utilizando métricas robustas (IQR e mediana) e apenas informações disponíveis no momento da transação, garantindo viabilidade de implantação em produção e ausência de target leakage.

Definição dos grupos

1. Fixo

Métodos de pagamento com valor de cobrança praticamente constante, caracterizados por IQR próximo de zero. Representam planos ou políticas com preço único e rígido, nos quais a grande maioria das transações ocorre em um valor específico, com raros desvios pontuais. Esse grupo apresenta alta previsibilidade de receita e baixo risco de variação na margem.

2. Baixo Estável

Métodos associados a valores baixos de cobrança, com variabilidade relativa controlada. A mediana do valor pago é baixa e o IQR relativo indica estabilidade no preço típico, refletindo planos básicos ou mensais com pouca influência de promoções. Esse grupo apresenta comportamento consistente e boa interpretabilidade para modelos lineares.

3. Misto Baixo

Métodos de pagamento com valor típico baixo, porém com maior heterogeneidade. Embora a maioria das transações ocorra em valores reduzidos, existem eventos ocasionais de cobrança mais elevada, indicando uso do mesmo método para diferentes contextos comerciais, como upgrades pontuais ou campanhas específicas. Esse grupo concentra o maior volume de transações e apresenta maior variância de comportamento.

4. Misto Alto

Métodos caracterizados por valores típicos elevados e alta variabilidade. A mediana do valor pago é alta e a dispersão é significativa, indicando uso irregular do método para planos premium, renovações longas ou operações específicas de alto valor. Apesar do baixo volume, esse grupo é relevante para análise de risco e impacto na margem, pois concentra transações com grande influência financeira.

Conclusão

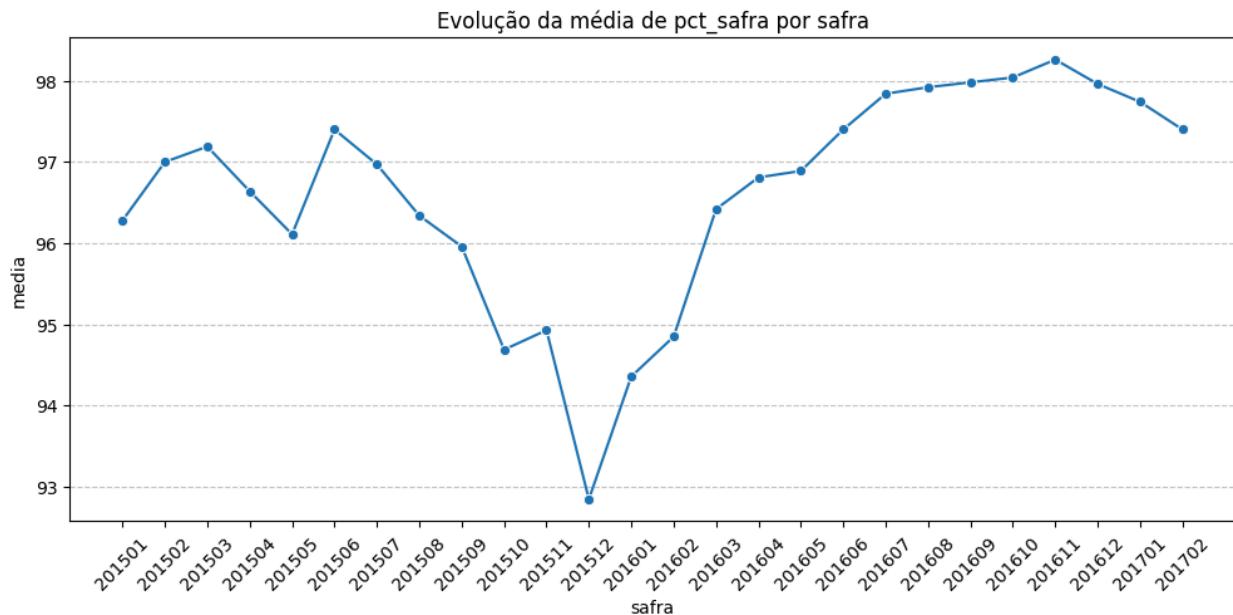
A subdivisão do grupo heterogêneo em perfis de preço baixo e alto permitiu capturar diferenças relevantes de escala e comportamento de cobrança, sem aumentar excessivamente a complexidade do modelo. O resultado final equilibra interpretabilidade, controle de variância e aderência ao contexto de negócio, tornando o agrupamento adequado para uso como feature explicativa em modelos de previsão de margem líquida.

4.2.5. feature: is_cancel

```
In [ ]: is_cancel_general = calcular_distribuicao(df_transactions, "is_cancel")
is_cancel_per_safra = calcular_distribuicao(df_transactions, "is_cancel", agrupar_por_safra=True)
```

```
+-----+-----+
|is_cancel|  total|pct_total|
+-----+-----+
|      0|20031271|   96.71|
|      1| 680954|    3.29|
+-----+-----+
+-----+-----+-----+-----+
| safra|is_cancel| total|pct_safra|pct_total|
+-----+-----+-----+-----+
|201501|      0|528398|   96.28|    2.55|
|201501|      1| 20394|    3.72|    0.1|
|201502|      0|528952|   97.0|    2.55|
|201502|      1| 16351|    3.0|    0.08|
|201503|      0|608867|   97.19|    2.94|
|201503|      1| 17621|    2.81|    0.09|
|201504|      0|545598|   96.64|    2.63|
|201504|      1| 18984|    3.36|    0.09|
|201505|      0|549342|   96.11|    2.65|
|201505|      1| 22210|    3.89|    0.11|
+-----+-----+-----+-----+
only showing top 10 rows
```

```
In [ ]: plot_tendencia_temporal(is_cancel_per_safra.filter(F.col("is_cancel").isin([0])), "pct_safra")
```



```
In [ ]: is_cancel_per_safra.filter(F.col("is_cancel").isin([0])).groupBy("is_cancel").agg(F.mean("pct_safra").alias("media_ativos")).orderBy("is_cancel").show(20)
```

```
+-----+
|is_cancel|  media_ativos|
+-----+
|      0|96.62423076923079|
+-----+
```

Aproximadamente 96,62% de todos os registros tem plano ativo. Todos eles sao clientes presentes na tabela members?

```
In [23]: calcular_distribuicao(df_transactions.filter(F.col("is_cancel") == 1), ["actual_amount_paid"], n_show=20)
```

	actual_amount_paid	total	pct_total
149	367676	53.99	
0	233580	34.3	
99	42775	6.28	
129	20907	3.07	
180	13207	1.94	
119	941	0.14	
100	901	0.13	
120	826	0.12	
134	125	0.02	
131	14	0.0	
41	1	0.0	
894	1	0.0	

Out[23]: DataFrame[actual_amount_paid: string, total: bigint, pct_total: double]

```
In [25]: calcular_distribuicao(df_transactions.filter(F.col("is_cancel") == 1), ["plan_list_price", "actual_amount_paid"], n_show=20)
```

	plan_list_price	actual_amount_paid	total	pct_total
149	149	357834	52.55	
149	0	228592	33.57	
99	99	42772	6.28	
129	129	20553	3.02	
180	180	13207	1.94	
0	149	9822	1.44	
0	0	4294	0.63	
100	100	897	0.13	
120	120	826	0.12	
120	0	694	0.1	
119	119	679	0.1	
149	119	257	0.04	
0	129	244	0.04	
134	134	117	0.02	
149	129	110	0.02	
119	149	20	0.0	
131	131	14	0.0	
0	134	8	0.0	
0	119	5	0.0	
0	180	4	0.0	

only showing top 20 rows

Out[25]: DataFrame[plan_list_price: string, actual_amount_paid: string, total: bigint, pct_total: double]

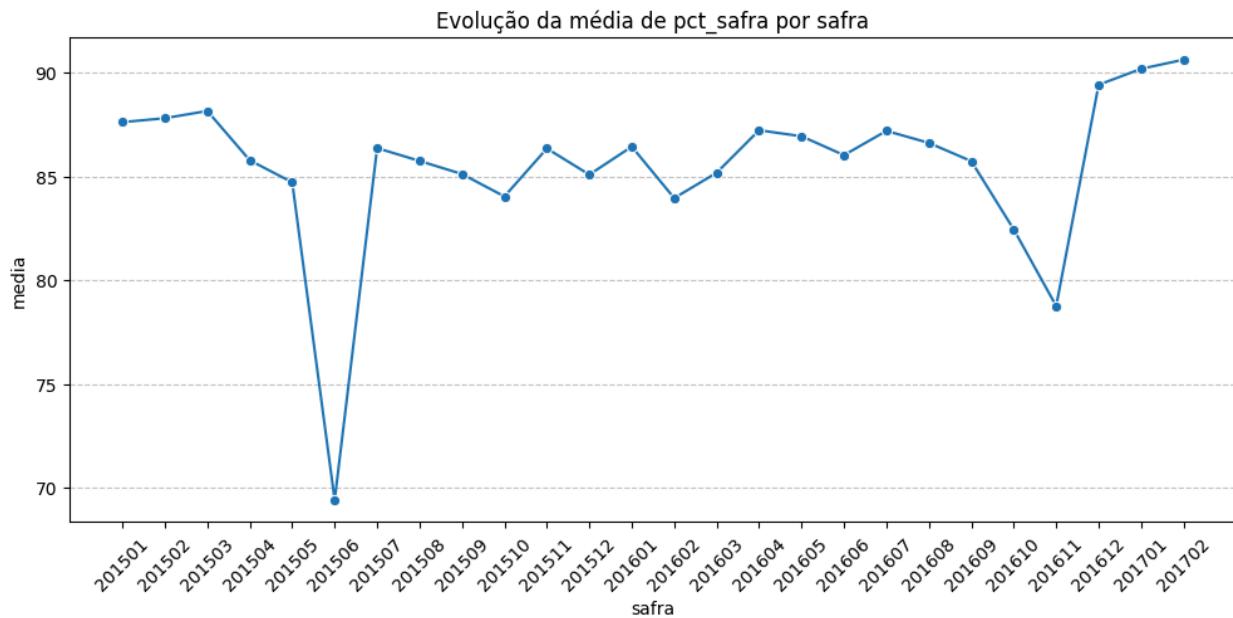
Aparentemente é possível pagar e cancelar, bem como não pagar e também cancelar.

4.2.6. feature: is_auto_renew

```
In [46]: is_auto_renew_general = calcular_distribuicao(df_transactions, "is_auto_renew")
is_auto_renew_per_safra = calcular_distribuicao(df_transactions, ["is_auto_renew"], agrupar_por_safra=True)
```

```
+-----+-----+-----+
|is_auto_renew|total |pct_total|
+-----+-----+-----+
|1          |17696175|85.44   |
|0          |3016050 |14.56   |
+-----+-----+-----+
+-----+-----+-----+-----+
|safra |is_auto_renew|total |pct_safra|pct_total|
+-----+-----+-----+-----+
|201501|1      |480841|87.62  |2.32    |
|201501|0      |67951 |12.38  |0.33    |
|201502|1      |478828|87.81  |2.31    |
|201502|0      |66475  |12.19  |0.32    |
|201503|1      |552320|88.16  |2.67    |
|201503|0      |74168  |11.84  |0.36    |
|201504|1      |484295|85.78  |2.34    |
|201504|0      |80287  |14.22  |0.39    |
|201505|1      |484227|84.72  |2.34    |
|201505|0      |87325  |15.28  |0.42    |
+-----+-----+-----+-----+
only showing top 10 rows
```

```
In [48]: plot_tendencia_temporal(is_auto_renew_per_safra.filter(F.col("is_auto_renew").isin([1])), "pct_safra")
```



```
In [50]: is_auto_renew_per_safra.filter(F.col("is_auto_renew").isin([1])).groupBy("is_auto_renew").agg(F.mean("pct_safra").alias("media_auto_renew")).orderBy("is_auto_renew").show(20)
```

```
+-----+-----+
|is_auto_renew| media_auto_renew|
+-----+-----+
|1          |85.50038461538462|
+-----+-----+
```

Aproximadamente 85,5% de todos os registros tem auto renovacao ativa.

4.2.7. feature: payment_plan_days

```
In [64]: calcular_distribuicao(df_transactions, "payment_plan_days", agrupar_por_safra=False)
```

payment_plan_days	total	pct_total
30	18251936	88.12
0	848204	4.1
31	743859	3.59
7	517635	2.5
195	107816	0.52
410	78425	0.38
180	51019	0.25
100	23828	0.12
10	22538	0.11
90	11518	0.06

only showing top 10 rows

```
Out[64]: DataFrame[payment_plan_days: int, total: bigint, pct_total: double]
```

Agrupando tipo de plano

```
In [16]: df_transactions = df_transactions.withColumn("payment_plan_days", F.col("payment_plan_days").cast("int"))
```

```
# 1. Categorização de Grupos de Compromisso
df_trans_analise = df_transactions.withColumn("grupo_plano",
    F.when(F.col("payment_plan_days") < 30, "01. Curto Prazo (<30d)")
    .when(F.col("payment_plan_days").isin(30, 31), "02. Mensal (30-31d)")
    .when(F.col("payment_plan_days") > 31, "03. Longo Prazo (>31d)")
    .otherwise("04. Indefinido"))

# 2. Agregação de Métricas de Negócio
resumo_planos = df_trans_analise.groupBy("grupo_plano").agg(
    F.count("msno").alias("total_transacoes"),
    F.round((F.count("msno") / F.lit(total_transactions)) * 100, 2).alias("pct_transacoes"),
    F.round(F.avg("plan_list_price"), 2).alias("valor_plano_medio"),
    F.round(F.avg("actual_amount_paid"), 2).alias("valor_pago_medio"),
    # Calculando o valor proporcional por dia
    F.round(F.avg(F.col("actual_amount_paid")) / F.when(F.col("payment_plan_days") == 0, 1).otherwise(F.col("payment_plan_days"))),
    2).alias("ticket_diario_medio"),
    F.round(F.avg("is_auto_renew").cast(DecimalType(20, 10)) * 100, 4).alias("pct_auto_renovacao"),
    F.round(F.avg("is_cancel").cast(DecimalType(20, 10)) * 100, 4).alias("pct_cancelamento")
).orderBy("grupo_plano")

resumo_planos.show(truncate=False)
```

grupo_plano	total_transacoes	pct_transacoes	valor_plano_medio	valor_pago_medio	ticket_diario_medio	pct_auto_renovacao	pct_cancelamento
01. Curto Prazo (<30d)	1398223	6.75	0.16	96.2	96.06	56.5530	
1.0283							
02. Mensal (30-31d)	18995795	91.71	136.1	131.86	4.39	88.9957	
3.5091							
03. Longo Prazo (>31d)	318207	1.54	1002.81	1002.81	4.18	0.0000	
0.0003							

Validacao

```
In [ ]: (df_trans_analise
    .groupBy("grupo_plano")
    .agg(
        F.min("payment_plan_days").alias("min_days"),
        F.max("payment_plan_days").alias("max_days"),
        F.countDistinct("payment_plan_days").alias("qtd_variacoes"))
    .orderBy("min_days")
    .show(10, truncate=False)
)
```

grupo_plano	min_days	max_days	qtd_variacoes
01. Curto Prazo (<30d) 0	21	9	1
02. Mensal (30-31d)	30	31	2
03. Longo Prazo (>31d) 35	450	26	1

Análise de "Planos de Valor Zero"

Nenhum valor cobrado e nenhum valor pago

```
In [ ]: df_trans_analise\
    .filter(
        (F.col("plan_list_price").isin(0))
        & (F.col("actual_amount_paid").isin(0)))
    )\
    .orderBy("safra").show(20, truncate=False)
```

msno	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	safra	payment_diff	flag_pag_diff	grupo_plano
GDPa9EDRpnnRz7wfV0NCQFdSE6sBJGuAMK2wqAgMpew= 26	0	1	0	0	0	0	0	0	20150115	20151230	0	01. Curto Prazo (<30d)
V+ASDGD+6JVAmRaJqrkLiPSxkZzIMrvV4hILmTgDpE= 17	0	410	0	0	0	0	0	0	20150120	20150123	0	03. Longo Prazo (>31d)
D7pfprXzXS501jssPUwu70MJSFSJH7g3f0Wjhaqw7LTQ= 26	0	1	0	0	0	0	0	0	20150109	20150514	0	01. Curto Prazo (<30d)
WQHyqtktMTQTHmvQWAEl5oyqeLLQP/IuHMR6tw9HgZ/w= 17	0	195	0	0	0	0	0	0	20150126	20150129	0	03. Longo Prazo (>31d)
HU67Zrg0+a8RZhHGH5HGvADCh0rTGJX62UYziQ02BQ= 26	0	1	0	0	0	0	0	0	20150121	20150222	0	01. Curto Prazo (<30d)
pw9d+NJ3YNix5D5SUWx064dDU618TYD1JfHAH9X6Z78= 38	0	30	0	0	0	0	0	0	20150131	20150524	0	02. Mensal (30-31d)
wxX+1MnfAJsXGprhzQ5K1gDmPfHej/jzLx2QgFzz7wc= 38	0	180	0	0	0	0	0	0	20150107	20150706	0	03. Longo Prazo (>31d)
09eILP59ACsce0Zn9kdJdv5BX8eKVXlFGwr1s6s6ynQ= 38	0	90	0	0	0	0	0	0	20150117	20150418	0	03. Longo Prazo (>31d)
AlpVSe4HMOcW/PhW/4A08+ow1qTAwCeBplWqHC2T6OI= 17	0	410	0	0	0	0	0	0	20150128	20150205	0	03. Longo Prazo (>31d)
5+0/ugN6UCvAXzbq3rik1loBvYLyLtRsGV5zGTzBSJjkC= 17	0	410	0	0	0	0	0	0	20150112	20150115	0	03. Longo Prazo (>31d)
EMHJDqm6oKe4zQAcL7LQr+a/kWhtEUmB03Y2kbD= 38	0	180	0	0	0	0	0	0	20150129	20150728	0	03. Longo Prazo (>31d)
DI+WEEQuN4xR7Q1K/8SnjtJqMz8yEafxuwna5yA/UTk= 38	0	30	0	0	0	0	0	0	20150124	20150223	0	02. Mensal (30-31d)
vwfcrxF4hvfwW5yxv1mVGUOagiThjXS4H1vgSOCq4Ps= 17	0	395	0	0	0	0	0	0	20150112	20150125	0	03. Longo Prazo (>31d)
8WPFJZ+y2oW0021iDtSk78VmCoN0ytYBqcjaQx5Y0ks= 17	0	395	0	0	0	0	0	0	20150122	20150208	0	03. Longo Prazo (>31d)
856J4KErD5z3H6E+203dzrl6NoijVix4hYyd7iGbVew= 26	0	1	0	0	0	0	0	0	20150116	20150217	0	01. Curto Prazo (<30d)
Mi6Drr8Gb/hm6aoYkr16ApJkw7iTirra/04G60iQsE= 26	0	1	0	0	0	0	0	0	20150123	20150224	0	01. Curto Prazo (<30d)
kFX3JdzTHniwqZgZrffljE65La5xdjp8FBVdLo5bqM= 38	0	180	0	0	0	0	0	0	20150122	20150721	0	03. Longo Prazo (>31d)
Q06Wh/2i0ZuSMbT900yI3LUAgx0t2FEICMFMT1LP/9U= 17	0	410	0	0	0	0	0	0	20150114	20150118	0	03. Longo Prazo (>31d)
V/+sdgVRqjd5DeJIFkXEryk7k31wK1+p4VbX3zuJl0= 38	0	180	0	0	0	0	0	0	20150101	20150630	0	03. Longo Prazo (>31d)
S99dhoZY9eJG1GKNPKQ27C+pR9HgZNUpWf1qRys4P8g= 38	0	180	0	0	0	0	0	0	20150121	20150720	0	03. Longo Prazo (>31d)

only showing top 20 rows

```
In [ ]: df_trans_analise
    .filter(
        (F.col("plan_list_price").isin(0))
        & (F.col("actual_amount_paid").isin(0))
        & (F.col("msno").isin("S99dhoZY9eJG1GKNPkQ27C+pR9HgZNUpWf1qRYs4P8g=")))
    )\
    .orderBy("safra").show(20, truncate=False)

+-----+-----+-----+-----+-----+
|msno |payment_method_id|payment_plan_days|plan_list_price|actual_amount_paid|is_auto_renew|transaction_date|membership_expire_date|is_cancel|safra |payment_diff|flag_pag_diff|grupo_plano      |
+-----+-----+-----+-----+-----+
|S99dhoZY9eJG1GKNPkQ27C+pR9HgZNUpWf1qRYs4P8g=|38           |180          |0             |0             |0            |20150101|03. Longo Prazo (>31d)|0         |20150121 |20150720 |0             |0             |
+-----+-----+-----+-----+-----+
```



```
In [ ]: trans_contagem = df_trans_analise.filter((F.col("plan_list_price").isin(0)) &
(F.col("actual_amount_paid").isin(0))).groupBy('msno').agg(F.count('*').alias('contagem'))
trans_contagem.select("contagem").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "99.5%", "max").show()

+-----+-----+
|summary| contagem|
+-----+-----+
| count| 529090|
| mean| 1.0625167740838042|
| stddev| 0.3081890497067433|
| min| 1|
| 25%| 1|
| 50%| 1|
| 75%| 1|
| 99.5%| 3|
| max| 9|
+-----+-----+
```


O P75 mostra que os casos de transações grátis ocorreram apenas uma vez


```
In [ ]: df_trans_analise.filter((F.col("plan_list_price").isin(0)) & (F.col("actual_amount_paid").isin(0))).groupBy("safra")\
.agg(
    F.count("msno").alias("total_transacoes_gratis"),
    F.round((F.count("msno") / F.lit(total_transactions)) * 100, 4).alias("pct_transacoes_gratis"))\
.show(20, truncate=False)
```


safra	total_transacoes_gratis	pct_transacoes_gratis
201505	8693	0.042
201702	2162	0.0104
201701	5017	0.0242
201501	218	0.0011
201509	4916	0.0237
201609	38849	0.1876
201512	15880	0.0767
201604	476	0.0023
201511	5626	0.0272
201611	140203	0.6769
201502	338	0.0016
201503	451	0.0022
201603	3838	0.0185
201508	2569	0.0124
201607	27497	0.1328
201610	83359	0.4025
201606	12392	0.0598
201507	810	0.0039
201601	2986	0.0144
201506	154292	0.7449

only showing top 20 rows

Mudanca de estado ao longo do tempo

```
In [68]: verificar_mudanca_estado(df_transactions, col_alvo="payment_plan_days", visualizar=True)
```

Contagem de mudanças por safra:

safra	mudou	count
201502	true	4004
201503	true	6221
201504	true	181046
201505	true	413728
201506	true	227428
201507	true	25546
201508	true	18998
201509	true	12293
201510	true	12146
201511	true	81854
201512	true	18349
201601	true	11426
201602	true	21723
201603	true	9835
201604	true	8628
201605	true	8805
201606	true	9314
201607	true	21309
201608	true	19260
201609	true	18395
201610	true	23665
201611	true	27820
201612	true	13091
201701	true	9067
201702	true	8013

Mudanças por usuário:

msno	total_mudancas
++4RuqBw0Ss6bQU4oMxaR1bBPoWzoEiIZaxPM04Y4+U=	2
++6P09mCSJSh+Ft2pvZ0FWTrtcI3v1A7h3/coh8dBKw=	2
++UyRqjARgvFXB6YdIVvIKnDWvPiCPo4yBb5tKJrjEc=	2
++Yw58ksb6lxxkqx EaMZFCiAAEnmmNy90SYjiI5vVU=	2
++ZahLG+SvfBC1jmIaj/KN0o4/ueRqnVLi2/gARCdLs=	2
++Zm4BQcxL8/QaQnMAd5D9L750tpVIuhSiJHSS4aPus=	2
++pSqq0qSB81a0m+RTW6NLTqsMVQ0egh4Rs5+GOSJrQ=	2
++qTmh4qA8N9/jpTo4sNdz0oQt8ZepzMQ5cfqcFG1GI=	2
++ywLqSa3Ts36aYwZ2FUpf8ru0Cf4f/OgVfiiz0Qnt4=	2
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0\$9dwUQ+g=	2

only showing top 10 rows

```
In [70]: verificar_mudanca_estado(df_trans_analise, col_alvo="grupo_plano", visualizar=True)
```

Contagem de mudanças por safra:

safra	mudou	count
201502	true	2700
201503	true	4438
201504	true	179306
201505	true	381153
201506	true	216546
201507	true	24698
201508	true	9665
201509	true	10626
201510	true	10558
201511	true	80158
201512	true	15418
201601	true	9507
201602	true	18518
201603	true	7882
201604	true	7288
201605	true	7328
201606	true	8080
201607	true	19614
201608	true	17680
201609	true	16751
201610	true	22028
201611	true	26474
201612	true	12589
201701	true	8735
201702	true	7744

Mudanças por usuário:

msno	total_mudancas
++4RuqBw0Ss6bQU4oMxaR1bBPoWzoEiIZaxPM04Y4+U=	2
++6P09mCSJSh+Ft2pvZ0FWTrtcI3v1A7h3/coh8dBKw=	2
++UyRqjARgvFXB6YdIVvIKnDWvPiCpo4yBb5tKJrjEc=	2
++Yw58kbsb6lxxkqx EaMZFCiAAEnmmNy90SYji5vVU=	2
++ZahLG+SvFBc1jmIaj/kN0o4/ueRqnVLi2/gARCdLs=	2
++Zm4BQcxL8/QaQnMD5D9L750tpVIuhSiJHSS4aPus=	2
++pSgq0qSB81a0m+RTW6NLTqsMVQ0egh4Rs5+GOSJrQ=	2
++qTmh4qA8N9/jpTo4sNdz0Qt8ZepzMQ5cfqcFG1GI=	2
++ywLqSa3Ts36aYwZ2FUpf8ru0Cf4f/OgVfiizQnt4=	2
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	2

only showing top 10 rows

```
In [72]: df_trans_analise.filter(F.col("msno").isin("+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=")).select("msno", "safra", "payment_plan_days", "grupo_plano").orderBy("safra").show(20, truncate=False)
```

msno	safra	payment_plan_days	grupo_plano
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201501	31	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201502	31	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201503	31	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201504	0	01. Curto Prazo (<30d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201505	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201506	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201507	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201509	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201510	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201511	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201512	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201601	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201602	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201603	30	02. Mensal (30-31d)
+/H9aTMCu1Y64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=	201604	30	02. Mensal (30-31d)

```
In [74]: df_trans_analise.filter(F.col("msno").isin("+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g=")).orderBy("safra").show(20, truncate=False)
```

msno	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	safra	payment_diff	flag_pag_diff	grupo_plano
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 40	20150124	20150223	0	201501 -149.0	31	1	149	0	02. Mensal (30-31d)		1	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 40	20150224	20150323	0	201502 0.0	31	0	149	149	02. Mensal (30-31d)		1	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 40	20150324	20150423	0	201503 0.0	31	0	149	149	02. Mensal (30-31d)		1	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 40	20150424	20150523	0	201504 149.0	30	1	0	149	01. Curto Prazo (<30d)		1	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 40	20150524	20150623	0	201505 0.0	30	0	149	149	02. Mensal (30-31d)		1	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 40	20150624	20150723	0	201506 0.0	30	0	149	0	02. Mensal (30-31d)		1	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 40	20150726	20150824	1	201507 -149.0	30	1	149	02. Mensal (30-31d)		1		
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20150904	20151004	0	201509 0.0	30	0	150	150	02. Mensal (30-31d)		0	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20151007	20151106	0	201510 0.0	30	0	150	150	02. Mensal (30-31d)		0	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20151110	20151210	0	201511 0.0	30	0	150	150	02. Mensal (30-31d)		0	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20151209	20160109	0	201512 0.0	30	0	150	150	02. Mensal (30-31d)		0	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20160109	20160208	0	201601 0.0	30	0	150	150	02. Mensal (30-31d)		0	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20160212	20160313	0	201602 0.0	30	0	150	150	02. Mensal (30-31d)		0	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20160316	20160415	0	201603 0.0	30	0	150	150	02. Mensal (30-31d)		0	
+/H9aTMCulY64Kzq7YxYXXoPnVgGk0nc/p0S9dwUQ+g= 36	20160421	20160521	0	201604 0.0	30	0	180	180	02. Mensal (30-31d)		0	

```
In [75]: df_trans_analise.filter(F.col("msno").isin("++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U=")).orderBy("safra").show(20, truncate=False)
```

msno	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	safra	payment_diff	flag_pag_diff	grupo_plano
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		129		129			1			
20150113	20150213	0	201501 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		129		129			1			
20150213	20150313	0	201502 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		129		129			1			
20150313	20150413	0	201503 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		129		129			1			
20150413	20150513	0	201504 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		0		0		01. Curto Prazo (<30d)						
20150513	20150613	0	201505 129.0	1		129						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		129		129			1			
20150613	20150713	0	201506 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20150713	20150813	0	201507 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20150813	20150913	0	201508 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20150913	20151013	0	201509 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20151013	20151113	0	201510 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20151113	20151213	0	201511 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20151213	20160113	0	201512 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160113	20160213	0	201601 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160213	20160313	0	201602 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160313	20160413	0	201603 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160413	20160513	0	201604 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160513	20160613	0	201605 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160613	20160713	0	201606 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160713	20160813	0	201607 0.0	0		02. Mensal (30-31d)						
++4RuqBw0Ss6bQU4oMxaRlbBPoWzoEiIZaxPM04Y4+U= 41		30		149		149			1			
20160813	20160913	0	201608 0.0	0		02. Mensal (30-31d)						

only showing top 20 rows

Suspeita:

- * Casos em que o cliente pagou (actual_amount_paid <> 0) mas nao veio cobranca (plan_list_price == 0), marca payment_plan_days = 0.
- * Casos em que o cliente nao paga (actual_amount_paid == 0) mas veio cobranca (plan_list_price <> 0), marca is_cancel = 1.

```
In [20]: calcular_distribuicao(df_trans_analise.filter((~F.col("actual_amount_paid").isin(0)) & (F.col("plan_list_price").isin(0))), ["payment_plan_days"])
```

payment_plan_days	total	pct_total
0	837847 100.0	
180	2	0.0

```
Out[20]: DataFrame[payment_plan_days: int, total: bigint, pct_total: double]
```

```
In [19]: calcular_distribuicao(df_trans_analise.filter((F.col("actual_amount_paid").isin(0)) & (~F.col("plan_list_price").isin(0))), ["is_cancel"])
```

is_cancel	total	pct_total
0	256390	52.79
1	229286	47.21

```
Out[19]: DataFrame[is_cancel: string, total: bigint, pct_total: double]
```

```
In [21]: calcular_distribuicao(df_trans_analise.filter((F.col("actual_amount_paid").isin(0)) & (F.col("is_cancel").isin(0))), ["plan_list_price"])
```

plan_list_price	total	pct_total
0	557873	68.51
149	255957	31.43
180	266	0.03
120	165	0.02
699	2	0.0

```
Out[21]: DataFrame[plan_list_price: string, total: bigint, pct_total: double]
```

A análise revelou que `payment_plan_days = 0` representa um estado administrativo de ausência de cobrança, e não um plano de curta duração. Além disso, ciclos sem pagamento não correspondem necessariamente a cancelamentos, indicando a existência de isenções e falhas de cobrança. Com base nisso, no processo de *feature engineering* devo distinguir explicitamente estados de cobrança, utilizando `payment_plan_days` apenas em ciclos válidos de cobrança e introduzindo flags específicas para isenção e falha de cobrança, reduzindo ruído e viés estrutural no modelo.

Levar para feature engineering:

```
In [ ]: df = df.withColumn("flag_cobranca_valida", # Para construcao do grupo_plano_final, portanto, nao precisa inserir na base final
    F.when((F.col("payment_plan_days") > 0) & (F.col("plan_list_price") > 0), 1) \
    .otherwise(0))

df = df.withColumn("flag_sem_cobranca",
    F.when(F.col("payment_plan_days") == 0, 1).otherwise(0))

df = df.withColumn("flag_isencao",
    F.when(
        (F.col("actual_amount_paid") == 0) &
        (F.col("plan_list_price") == 0) &
        (F.col("is_cancel") == 0), 1) \
    .otherwise(0))

df = df.withColumn(
    "flag_falha_cobranca",
    F.when(
        (F.col("actual_amount_paid") == 0) &
        (F.col("plan_list_price") > 0) &
        (F.col("is_cancel") == 0), 1) \
    .otherwise(0))

df = df.withColumn("flag_cancelamento",
    F.when(F.col("is_cancel") == 1, 1).otherwise(0))
```

```
In [ ]: df = df.withColumn("grupo_plano_final",
    F.when(F.col("flag_cobranca_valida") == 0, "00. Sem Cobranca")
    .when(F.col("payment_plan_days") < 30, "01. Curto Prazo")
    .when(F.col("payment_plan_days").isin(30, 31), "02. Mensal")
    .when(F.col("payment_plan_days") > 31, "03. Longo Prazo"))
```

O `payment_plan_days` só é interpretado como tipo de plano quando existe cobrança válida (`payment_plan_days > 0` e `plan_list_price > 0`). Casos fora dessa condição representam estados administrativos do ciclo, como isenção, falha de cobrança ou ausência de cobrança. Para evitar ruído e viés estrutural no modelo, foram criadas flags específicas que distinguem esses estados, permitindo que o modelo aprenda separadamente comportamento de plano e eventos excepcionais.

Apesar do aumento no número de variáveis binárias, as flags criadas representam estados mutuamente exclusivos do ciclo de cobrança e refletem regras de negócio reais. Esse tipo de feature engineering reduz ambiguidade semântica, melhora interpretabilidade e diminui viés estrutural, sendo preferível ao uso direto de variáveis contínuas que misturam múltiplos fenômenos.

Mais sugestões para Feature Engineering:

1. daily_revenue_efficiency (Eficiência de Receita Diária)

Significado: É a normalização do valor pago pelo tempo de serviço. Ela responde: "Quantos NTD este cliente deixa na mesa por cada dia de acesso?"

Impacto no Modelo: Como o custo operacional possui um componente diário (tempo em segundos), esta métrica permite comparar se um plano de 410 dias é proporcionalmente mais barato para a empresa do que um plano de 30 dias. É uma variável fundamental para identificar o "lucro bruto diário" antes da subtração do custo de logs.

2. revenue_stability_index (Índice de Estabilidade de Receita) Significado: Um score que combina a duração do plano com o comportamento de renovação automática (is_auto_renew). Score 2 (Alta): Usuários que ou já pagaram por um longo período ou estão no débito automático mensal. Score 1 (Média): Usuários mensais que precisam tomar ação manual de pagar todo mês. Score 0 (Baixa): Usuários de pacotes curtos e eventuais. Impacto no Modelo: Ajuda a estimar a volatilidade da margem. Planos de estabilidade 2 têm uma variância muito menor na previsão de rentabilidade para o mês seguinte (\$M+1\$).

```
In [ ]: df = df.withColumn("daily_revenue_efficiency",
    F.when(F.col("flag_cobranca_valida").isin(1), F.col("actual_amount_paid") / F.col("payment_plan_days")))

df = df.withColumn("revenue_stability_index",
    F.when((F.col("grupo_plano_final").isin("03. Longo Prazo")) & (F.col("is_auto_renew").isin(1)), 2) # Alta estabilidade
    .when((F.col("grupo_plano_final").isin("02. Mensal")) & (F.col("is_auto_renew").isin(1)), 1) # Média estabilidade
    .otherwise(0)) # Baixa ou inexistente
```

4.2.8. feature: actual_amount_paid

```
In [5]: df_transactions = df_transactions.withColumn("actual_amount_paid", F.col("actual_amount_paid").cast("float"))
```

```
In [6]: df_transactions.select("actual_amount_paid").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "max").show()
```

	summary actual_amount_paid
count	20712225
mean	142.83468555406287
stddev	133.60944276809502
min	0.0
25%	99.0
50%	149.0
75%	149.0
max	2000.0

```
In [7]: df_transactions.groupBy("actual_amount_paid").count().orderBy(F.desc("actual_amount_paid")).show()
```

actual_amount_paid	count
2000.0	119
1825.0	2
1802.0	1
1800.0	1
1799.0	1
1788.0	82809
1599.0	11991
1520.0	13
1200.0	5313
1150.0	95
1000.0	741
930.0	5063
894.0	111410
890.0	1
849.0	1
800.0	11
799.0	6422
760.0	2
699.0	887
600.0	3

only showing top 20 rows

```
In [13]: ((df_transactions.filter(F.col("actual_amount_paid") > 149).count()) / (df_transactions.count())) * 100
```

```
Out[13]: 6.521515674921454
```

Por representar aproximadamente 6,5% do total de transações, talvez faça sentido considerar como ponto de alavancagem e remover?

PENDENTE: aplicar a fórmula para capturar casos de alavancagem e ver se a quantidade diminui. O filtro atual considera apenas o valor mais comum ate 75% dos dados ordenados.

4.2.9. feature: membership_expire_date

```
In [36]: calcular_distribuicao(df_transactions, "membership_expire_date")
```

```
+-----+-----+
|membership_expire_date|total |pct_total|
+-----+-----+
|20150615             |173138|0.84   |
|20170228             |129307|0.62   |
|20161031             |127679|0.62   |
|20161130             |127035|0.61   |
|20160831             |125082|0.6    |
|20170131             |124755|0.6    |
|20160930             |123540|0.6    |
|20161231             |122891|0.59   |
|20160229             |115762|0.56   |
|20170331             |112799|0.54   |
+-----+-----+
only showing top 10 rows
```

```
Out[36]: DataFrame[membership_expire_date: string, total: bigint, pct_total: double]
```

```
In [37]: df_transactions_anl = (df_transactions
    .withColumn("membership_expire_date", F.to_date(F.col("membership_expire_date"), "yyyyMMdd"))
    .withColumn("membership_expire_year_month", F.date_format(F.col("membership_expire_date"), "yyyyMM").cast("int"))
)
```

```
In [41]: calcular_distribuicao(df_transactions_anl, "membership_expire_year_month")
```

```
+-----+-----+-----+
|membership_expire_year_month|total |pct_total|
+-----+-----+-----+
|201611                |1129184|5.45   |
|201612                |1013338|4.89   |
|201609                |1006833|4.86   |
|201702                |1004932|4.85   |
|201701                |1001690|4.84   |
|201610                |997294 |4.82   |
|201703                |997231 |4.81   |
|201603                |860866 |4.16   |
|201608                |858715 |4.15   |
|201601                |829898 |4.01   |
+-----+-----+-----+
only showing top 10 rows
```

```
Out[41]: DataFrame[membership_expire_year_month: int, total: bigint, pct_total: double]
```

```
In [38]: df_transactions_anl.select("membership_expire_year_month").summary("count", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+
|summary|membership_expire_year_month|
+-----+
|  count|              20712225|
|  min|            197001|
|  1%|            201502|
| 10%|            201505|
| 25%|            201510|
| 50%|            201605|
| 75%|            201610|
| 95%|            201702|
| 99.5%|           201703|
|  max|           201703|
+-----+
```

```
In [30]: df_transactions_anl.filter(F.col("membership_expire_year_month") < "201501").groupBy("membership_expire_year_month").count().show(10, truncate=False)
```

```
+-----+-----+
|membership_expire_year_month|count|
+-----+-----+
|201402 |147 |
|201302 |69 |
|201303 |356 |
|201305 |147 |
|201309 |196 |
|201406 |190 |
|201312 |216 |
|201403 |166 |
|201211 |7 |
|201311 |222 |
+-----+-----+
only showing top 10 rows
```

```
In [32]: df_transactions_anl.filter(F.col("membership_expire_year_month") < "201501").select("msno", "safra", "membership_expire_year_month", "transaction_date").show(10, truncate=False)
```

```
+-----+-----+-----+-----+
|msno |safra |membership_expire_year_month|transaction_date|
+-----+-----+-----+-----+
|+3UG/uRcLrwhCK0LtHqdm0H05NL1ZJzY+0G33pufsjE=|201512|201310 |20151223 |
|+ZoXRYo4IxvYq2xYPivjn9qug+dTHdNPveettAzX60M=|201501|201411 |20150127 |
|+oMJt7y1ez1nsOhmUj6928jafEs3Vt4C8d6UFpzfIG8=|201512|201408 |20151223 |
|+q4cNHUod+HpoCT1IOFZHfYdb6eSu9oHzJplVeiyGs=|201501|201412 |20150123 |
|/GUW0qZoq/CQ+ygEqOghs0U90f4FtIsS6HHpVKMeJxE=|201512|201303 |20151224 |
|/RKh5ZwSeKPPAqmWIIBbfC31HnC7FQJdzwFY2sCrsg0=|201507|197001 |20150706 |
|/f3tTQOEzDOD79J4CGZ4k4PA51InD94qVK4t39VdJ/U=|201512|201302 |20151223 |
|/xJpJyntvneH0q26eMtovv7RgXZMBQChNg+XpHqYGr4=|201701|197001 |20170128 |
|/yayVtROn6ra+H0Tn5dkerdMNAvH5vrarw4jD9uNFKg=|201512|201407 |20151223 |
|077nGY7U4E+08RNFWadpVmyMM8nH7EiDS79vhcsHyZQ=|201512|201405 |20151224 |
+-----+-----+-----+
only showing top 10 rows
```

Menos de 1% da base tem membership_expire_date menor do que jan/2015. Sugestao para feature engineering: construir flag expire_date_valida

4.2.10. feature: transaction_date

```
In [42]: calcular_distribuicao(df_transactions, "transaction_date")
```

```
+-----+-----+
|transaction_date|total |pct_total|
+-----+-----+
|20160831 |203233|0.98 |
|20161031 |202971|0.98 |
|20160930 |199635|0.96 |
|20160731 |198135|0.96 |
|20161130 |196293|0.95 |
|20161231 |196158|0.95 |
|20170131 |193562|0.93 |
|20160131 |192898|0.93 |
|20150831 |183190|0.88 |
|20150331 |179309|0.87 |
+-----+-----+
only showing top 10 rows
```

```
Out[42]: DataFrame[transaction_date: string, total: bigint, pct_total: double]
```

```
In [33]: df_transactions_anl = (df_transactions
    .withColumn("transaction_date", F.to_date(F.col("transaction_date"), "yyyyMMdd"))
    .withColumn("transaction_date_year_month", F.date_format(F.col("transaction_date"), "yyyyMM").cast("int"))
)
```

```
In [44]: calcular_distribuicao(df_transactions_anl, "transaction_date_year_month")
```

transaction_date_year_month	total	pct_total
201611	1094941	5.29
201610	1033898	4.99
201701	988576	4.77
201609	982640	4.74
201612	968547	4.68
201608	966450	4.67
201607	924032	4.46
201702	885566	4.28
201512	861107	4.16
201601	856716	4.14

only showing top 10 rows

```
Out[44]: DataFrame[transaction_date_year_month: int, total: bigint, pct_total: double]
```

```
In [35]: df_transactions_anl.select("transaction_date_year_month").summary("count", "min", "1%", "10%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	transaction_date_year_month
count	20712225
min	201501
1%	201501
10%	201504
25%	201509
50%	201603
75%	201609
95%	201701
99.5%	201702
max	201702

Aparentemente não tem problemas na data de transação. Sugestão para feature engineering com a data de expiração de membro:

```
In [ ]: df = df.withColumn("months_to_expire",
    F.when(F.col("membership_expire_year_month") >= 201501,
           (F.col("membership_expire_year_month") - F.col("transaction_date_year_month"))))
```

A data de expiração da assinatura apresenta valores inconsistentes anteriores ao início da base (<201501), representando menos de 1% dos registros e indicando ruído estrutural ou valores default de sistema. Para evitar viés, esses casos foram identificados por uma flag específica. A informação de expiração foi transformada em uma variável relativa ao momento da transação (meses até a expiração), que é causal, interpretável e adequada para uso em modelos preditivos.

4.2.10.1 Verificar se as transações ocorrem em dias específicos

```
In [35]: df_trans_anl = df_transactions.withColumn("payment_day", F.dayofmonth(F.to_date(F.col("transaction_date"), "yyyyMMdd"))))
```

```
In [36]: calcular_distribuicao(df_trans_anl, ["payment_day"], n_show=31)
```

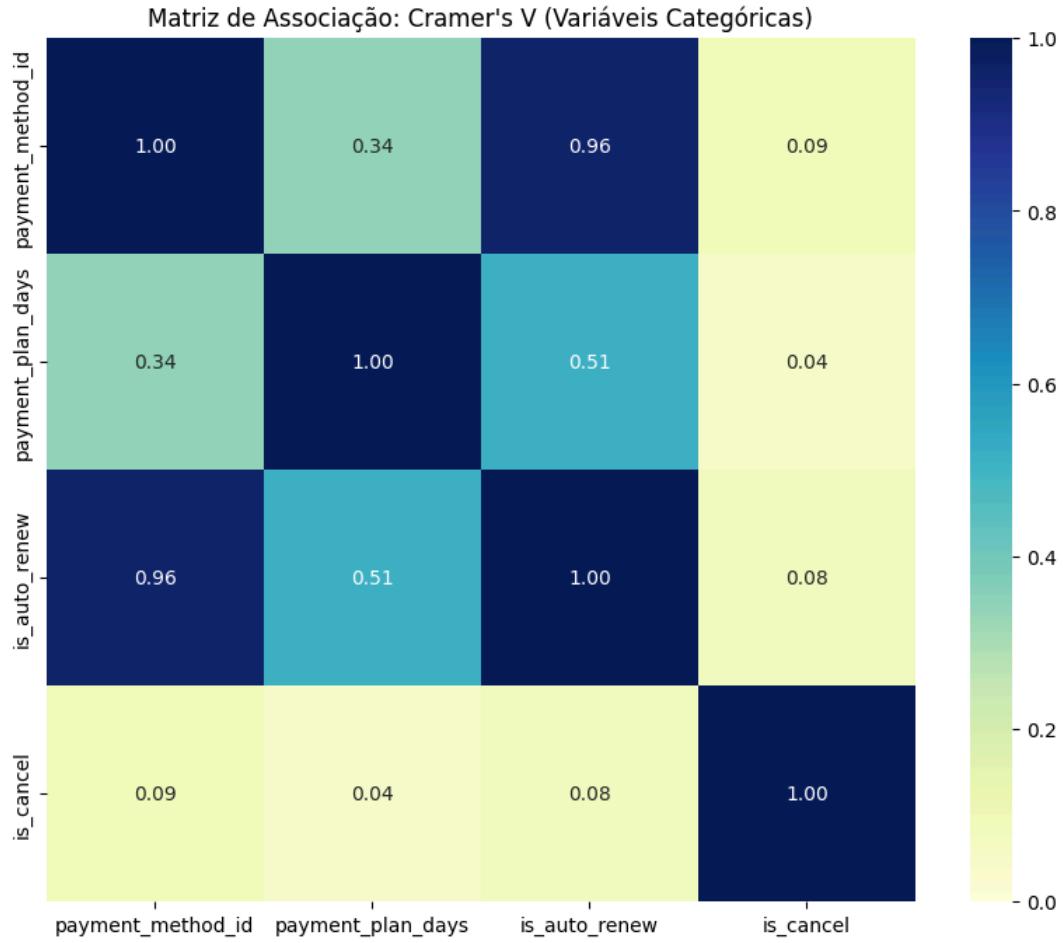
payment_day	total	pct_total
31	2344707	11.32
30	1266583	6.12
2	721991	3.49
1	651033	3.14
6	625118	3.02
9	625054	3.02
4	621873	3.0
5	616576	2.98
11	613119	2.96
21	612745	2.96
16	610511	2.95
10	606955	2.93
20	605181	2.92
19	603929	2.92
7	601510	2.9
25	597525	2.88
3	596424	2.88
22	595443	2.87
8	595416	2.87
12	594892	2.87
15	593247	2.86
26	591987	2.86
18	590881	2.85
17	590719	2.85
14	590027	2.85
13	589354	2.85
27	578336	2.79
24	577453	2.79
23	574118	2.77
28	475467	2.3
29	254051	1.23

```
Out[36]: DataFrame[payment_day: int, total: bigint, pct_total: double]
```

4.2.11. Cramer V

```
In [22]: matriz_cramer_v_spark(df_transactions, ["payment_method_id", "payment_plan_days", "is_auto_renew", "is_cancel"])
```

Iniciando indexação de 4 colunas...
 Calculando Cramer's V: payment_method_id vs payment_plan_days...
 Calculando Cramer's V: payment_method_id vs is_auto_renew...
 Calculando Cramer's V: payment_method_id vs is_cancel...
 Calculando Cramer's V: payment_plan_days vs is_auto_renew...
 Calculando Cramer's V: payment_plan_days vs is_cancel...
 Calculando Cramer's V: is_auto_renew vs is_cancel...



	payment_method_id	payment_plan_days	is_auto_renew	is_cancel
payment_method_id	1.000000	0.342212	0.957676	0.089751
payment_plan_days	0.342212	1.000000	0.510796	0.043592
is_auto_renew	0.957676	0.510796	1.000000	0.076111
is_cancel	0.089751	0.043592	0.076111	1.000000

A análise de associação entre variáveis categóricas mostrou forte dependência estrutural entre `payment_method_id` e `is_auto_renew` (Cramer's V == 0.96), indicando que o método de pagamento carrega regras de negócio que praticamente determinam a ativação de renovação automática.

```
In [ ]: df_payment_method_stats = df_transactions.groupBy("payment_method_id")\
    .agg(
        F.count("*").alias("total"),
        F.avg(F.col("is_auto_renew")).alias("pct_auto_renew"))\
    .orderBy(F.desc("pct_auto_renew"))

df_payment_method_stats .show(50, truncate=False)
```

payment_method_id	total	pct_auto_renew
7	1822	1.0
11	2114	1.0
30	150972	1.0
27	60532	1.0
19	32045	1.0
23	40594	1.0
40	2160126	1.0
24	12620	1.0
10	1316	1.0
37	991185	1.0
21	22839	1.0
14	13591	1.0
41	11026911	0.9999999093127713
39	1437059	0.9999972165373864
34	728384	0.9998421162463755
33	402388	0.9982429893535593
31	248960	0.9965134961439589
18	15709	0.9952893245909988
16	10301	0.9335986797398311
36	816661	0.3916312888701677
29	108713	0.18688657290296468
5	469	0.0255863539445629
32	144037	6.942660566382249E-6
15	1464	0.0
3	206	0.0
8	649	0.0
28	92752	0.0
22	19973	0.0
35	492261	0.0
17	5081	0.0
26	2795	0.0
6	462	0.0
38	1618337	0.0
25	11741	0.0
1	11	0.0
20	27977	0.0
4	15	0.0
12	3796	0.0
13	6305	0.0
2	52	0.0

4.2.11.1 Ideia para feature engineering:

```
In [26]: df_payment_method_stats = df_payment_method_stats.withColumn("payment_method_group",
    F.when(F.col("pct_auto_renew") >= 0.95, "auto_only")
    .when(F.col("pct_auto_renew") >= 0.70, "mostly_auto")
    .when(F.col("pct_auto_renew") >= 0.30, "mixed")
    .when(F.col("pct_auto_renew") >= 0.05, "mostly_manual")
    .otherwise("manual_only"))
```

```
In [27]: df_transactions = (df_transactions
    .join(
        df_payment_method_stats.select("payment_method_id", "payment_method_group"),
        on="payment_method_id",
        how="left"))
```

```
In [28]: calcular_distribuicao(df_transactions, "payment_method_group", n_show=10)
```

payment_method_group	total	pct_total
auto_only	17348167	83.76
manual_only	2428383	11.72
mixed	816661	3.94
mostly_manual	108713	0.52
mostly_auto	10301	0.05

```
Out[28]: DataFrame[payment_method_group: string, total: bigint, pct_total: double]
```

Pensando que `is_auto_renew == 1` tem como media em todo o dataframe aproximadamente 85% dos casos, talvez `payment_method_group` nao seja a melhor para usar, mas sim a propria `is_auto_renew`, dado que a distribuicao fica similar.

4.3. Members

4.3.1 dictionary

Definition: User information. Note that not every user in the dataset is available.

```
* msno
* city
* bd: age. Note: this column has outlier values ranging
from -7000 to 2015, please use your judgement.
* gender
* registered_via: registration method
* registration_init_time: format %Y%m%d
```

4.3.2. general info

```
In [86]: total_members = df_members.count()
total_members
```

```
Out[86]: 63867246
```

```
In [87]: df_members.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: string (nullable = true)
 |-- registration_init_time: string (nullable = true)
 |-- city: string (nullable = true)
 |-- bd: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- registered_via: string (nullable = true)
 |-- is_ativo: integer (nullable = true)
```

```
In [11]: df_members.show(10, truncate=False)
```

msno	safra	registration_init_time	city	bd	gender	registered_via	is_ativo
++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc=	201612 20140927	1 0 NULL 7 1					
++AwGzubug3gT6J+0STBGMdWKxaM+UFZTI8Tcmq4To=	201607 20150322	1 0 NULL 9 0					
++/Gw1B9K+X01B3hLTloeUK2QlCa2m+BJ8TrzGf7djI=	201601 20121217	15 32 male 3 1					
++02XbtviomSxcIBUHMO1jkjRxdicTXSfiVqLdsr5lo=	201603 20131112	14 21 male 7 0					
++000Bq04sB/9ZcOS+pajpYL2Hin9jcanc/8bKzKFuE=	201610 20141021	5 33 male 3 0					
++2AQgVgYUAqJDw684tbDqDffUeKhqydyQmbr81z91Q=	201608 20150416	18 23 male 3 0					
++2gRJ7i2Mb06qUG6rGfFnu/Fcv+hdX4YvTkZD+PUsk=	201608 20140616	11 33 male 7 0					
++3brN43Yd6GURegTBR85oMQcJrgW1+/N488Rjj75FY=	201604 20100512	14 19 male 9 0					
++4FwgRp7pHuuQWpaUFrCTgJbXVwNTTQpLB2bM1A31U=	201605 20141010	1 0 NULL 9 0					
++4KsBMCDgIrwmv5w2g2cCXhzCsJwDyB5r96/FrwGek=	201604 20141031	1 0 NULL 9 0					

only showing top 10 rows

Duplicatas e aparicoes

```
In [46]: print("Número de duplicatas encontradas: " + str(verificar_duplicatas(df_members)))
```

Número de duplicatas encontradas: 0

```
In [28]: members_contagem = df_members.groupBy('msno').agg(F.count('*').alias('contagem'))  
members_contagem.select("contagem").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "99.5%", "max").show()
```

summary	contagem
count	6287789
mean	10.157345610674913
stddev	3.242783103114158
min	1
25%	9
50%	12
75%	12
99.5%	12
max	12

```
In [51]: contagem_members = members_contagem.count()  
calcular_distribuicao(members_contagem, "contagem", contagem_members, col_id="contagem", n_show=20)
```

contagem	total	pct_total
12	4294184	68.29
11	208170	3.31
10	195618	3.11
7	189359	3.01
6	189273	3.01
8	184404	2.93
9	178329	2.84
5	176584	2.81
1	173444	2.76
3	170302	2.71
2	165057	2.63
4	163065	2.59

```
In [30]: contagem_safra_members = df_members.select("safra").distinct().count()  
print(f'O número de safras na base é: {contagem_safra_members}'")
```

O número de safras na base é: 12

4.3.3. tipagem

```
In [88]: df_members = (df_members  
    .withColumn("bd", F.col("bd").cast("integer"))  
    .withColumn("is_ativo", F.col("is_ativo").cast("integer"))  
    .withColumn("registered_via", F.col("registered_via").cast("integer")))
```

4.3.4. feature: registered_via

```
In [89]: calcular_distribuicao(df_members, ["registered_via"], n_show=25)
```

registered_via	total	pct_total
4	19648219	30.76
3	18640987	29.19
9	16776651	26.27
7	8407395	13.16
11	269614	0.42
8	47588	0.07
5	18715	0.03
2	16847	0.03
16	10564	0.02
13	7546	0.01
14	7111	0.01
17	6630	0.01
19	5629	0.01
6	3380	0.01
1	254	0.0
10	104	0.0
-1	12	0.0

```
Out[89]: DataFrame[registered_via: int, total: bigint, pct_total: double]
```

Pendente: verificar na margem liquida as estatísticas para cada categoria e então agrupar

```
In [15]: verificar_mudanca_estado(df_members, col_alvo="registered_via", visualizar=True)
```

Contagem de mudanças por safra:

safra	mudou	count

Mudanças por usuário:

msno	total_mudancas

4.3.5. feature: bd --> idade

```
In [90]: calcular_distribuicao(df_members, ["bd"], n_show=25)
```

bd	total	pct_total
0	39509864	61.86
22	1220332	1.91
21	1196188	1.87
20	1183935	1.85
27	1136339	1.78
23	1109043	1.74
24	1066096	1.67
26	1027742	1.61
25	1010945	1.58
18	961760	1.51
19	955064	1.5
28	926093	1.45
29	920441	1.44
17	838237	1.31
30	792655	1.24
32	726893	1.14
31	705358	1.1
33	659476	1.03
34	624997	0.98
37	605437	0.95
35	595797	0.93
36	575431	0.9
38	478553	0.75
39	432434	0.68
16	401278	0.63

only showing top 25 rows

```
Out[90]: DataFrame[bd: int, total: bigint, pct_total: double]
```

```
In [91]: df_members.select("bd").summary("count", "mean", "stddev", "min", "1%", "25%", "50%", "75%", "99.5%", "max").show()
```

summary	bd
count	63867246
mean	11.430160210759675
stddev	19.041318387984273
min	-7168
1%	0
25%	0
50%	0
75%	24
99.5%	61
max	2016

```
In [99]: print("Percentual do total da base: " + str((df_members.filter(F.col("bd").between(0, 61)).count()) / (total_members) * 100) + "%")  
df_members.filter(F.col("bd").between(0, 61)).select("bd").summary("count", "mean", "stddev", "min", "1%", "25%", "50%", "75%", "99%", "max").show()
```

Percentual do total da base: 99.5290089696368%

summary	bd
count	63566437
mean	11.090138605692182
stddev	15.397906043029254
min	0
1%	0
25%	0
50%	0
75%	24
99%	52
max	61

```
In [100]: print("Percentual do total da base: " + str((df_members.filter(F.col("bd").between(1, 61)).count()) / (total_members) * 100) + "%")
df_members.filter(F.col("bd").between(1, 61)).select("bd").summary("count", "mean", "stddev", "min", "1%", "25%", "50%", "75%", "99%", "max").show()
```

```
Percentual do total da base: 37.6665262817188%
+-----+
|summary|      bd|
+-----+-----+
| count|     24056573|
| mean|29.304281910810822|
| stddev| 9.630283475662145|
| min|      1|
| 1%|     16|
| 25%|    22|
| 50%|    27|
| 75%|    35|
| 99%|    56|
| max|    61|
+-----+
```

Dado que o P1 da variável (removendo os zeros e maiores de 61) é 16 + o fato de que serviços de assinatura digital frequentemente adotam 13 ou 16 anos como idade mínima por questões de conformidade com leis de proteção de dados (LGPD), 16 anos foi decidido como idade mínima permitida.

```
In [103]: print("Porcentagem de usuários com idade igual a 0:", (df_members.filter(F.col("bd").isin(0)).count() / total_members) * 100)
print("Porcentagem de usuários com idade menor do que 0:", (df_members.filter(F.col("bd") < 0).count() / total_members) * 100)
print("Porcentagem de usuários com idade entre 1 e 60 anos:", (df_members.filter(F.col("bd").between(1, 60)).count() / total_members) * 100)
print("Porcentagem de usuários com idade entre 61 e 80 anos:", (df_members.filter(F.col("bd").between(61, 80)).count() / total_members) * 100)
print("Porcentagem de usuários com idade maior do que 80 anos:", (df_members.filter(F.col("bd") > 80).count() / total_members) * 100)
```

```
Porcentagem de usuários com idade igual a 0: 61.862482687917996
Porcentagem de usuários com idade menor do que 0: 0.004819371732421342
Porcentagem de usuários com idade entre 1 e 60 anos: 37.61010455969872
Porcentagem de usuários com idade entre 61 e 80 anos: 0.37236457635890546
Porcentagem de usuários com idade maior do que 80 anos: 0.1502288042919527
```

```
In [107]: identificar_outliers(df_members, "bd", negativo=False)
```

```
Variável bd:
Limites: [0, 57.5]
Total de outliers (#): 470971
Total de outliers (%): 0.74%
```

```
In [108]: identificar_outliers(df_members, "bd", negativo=False, minimo_definido=16)
```

```
Variável bd:
Limites: [16, 57.5]
Total de outliers (#): 40166084
Total de outliers (%): 62.89%
```

Como qualquer tratamento atrelaria um viés na dataprep, entendo que seja melhor trabalhar com agrupamentos, classificando outliers como "Desconhecido".

```
In [35]: df_members_teste = df_members.withColumn("faixa_etaria",
    F.when(F.col("bd").between(16, 21), "01.Adolescente")\
    .when(F.col("bd").between(22, 35), "02.Jovem Adulto")\
    .when(F.col("bd").between(36, 50), "03.Adulto")\
    .when(F.col("bd").between(51, 80), "04.Meia Idade")\
    .otherwise("05.Desconhecido"))
```

```
In [36]: calcular_distribuicao(df_members_teste, "faixa_etaria")
```

```
+-----+-----+-----+
|faixa_etaria|total |pct_total|
+-----+-----+-----+
|05.Desconhecido|39794138|62.31   |
|02.Jovem Adulto|12522207|19.61   |
|01.Adolescente |5536462 |8.67    |
|03.Adulto      |4951923 |7.75    |
|04.Meia Idade  |1062516  |1.66    |
+-----+-----+-----+
```

```
Out[36]: DataFrame[faixa_etaria: string, total: bigint, pct_total: double]
```

4.3.6. feature: gender

```
In [112]: calcular_distribuicao(df_members, ["gender"], n_show=5)
```

```
+-----+-----+-----+
|gender|total |pct_total|
+-----+-----+-----+
|NULL  |38210177|59.83   |
|male   |13075425|20.47   |
|female |12581644|19.7    |
+-----+-----+-----+
```

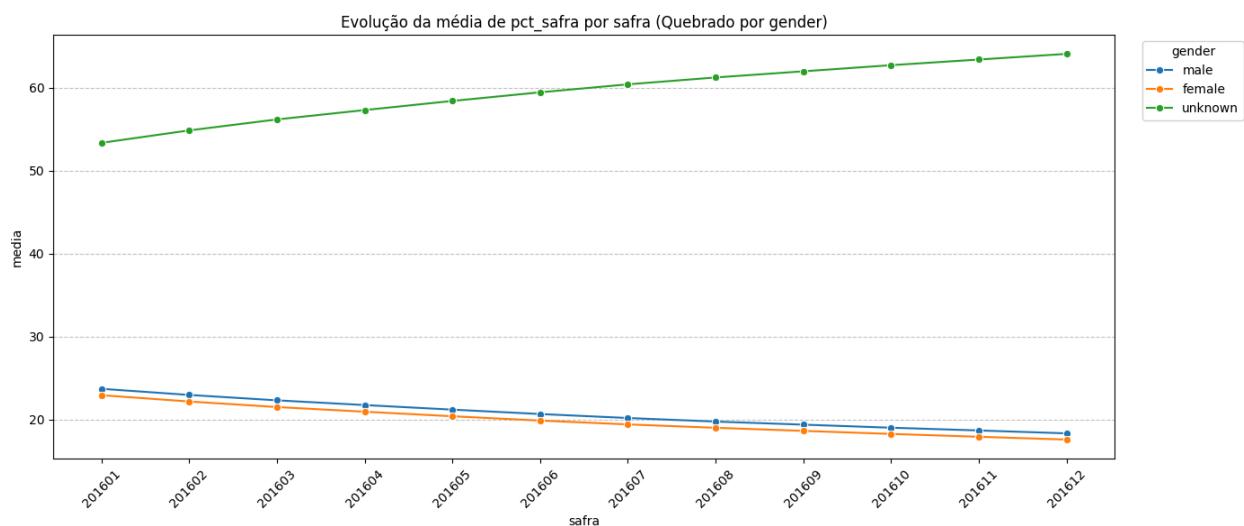
Out[112]: DataFrame[gender: string, total: bigint, pct_total: double]

```
In [19]: gender_per_safra = calcular_distribuicao(df_members, ["gender"], n_show=100, agrupar_por_safra=True)
```

```
+-----+-----+-----+-----+
|safra |gender|total |pct_safra|pct_total|
+-----+-----+-----+-----+
|201601|NULL  |2290971|53.35   |3.59    |
|201601|male   |1018241|23.71   |1.59    |
|201601|female |984972 |22.94   |1.54    |
|201602|NULL  |2469561|54.85   |3.87    |
|201602|male   |1034270|22.97   |1.62    |
|201602|female |998523 |22.18   |1.56    |
|201603|NULL  |2639013|56.17   |4.13    |
|201603|male   |1048445|22.32   |1.64    |
|201603|female |1010514|21.51   |1.58    |
|201604|NULL  |2793991|57.3    |4.37    |
|201604|male   |1060627|21.75   |1.66    |
|201604|female |1021683|20.95   |1.6     |
|201605|NULL  |2955526|58.4    |4.63    |
|201605|male   |1072726|21.2    |1.68    |
|201605|female |1032453|20.4    |1.62    |
|201606|NULL  |3120875|59.44   |4.89    |
|201606|male   |1085229|20.67   |1.7     |
|201606|female |1043960|19.88   |1.63    |
|201607|NULL  |3285146|60.4    |5.14    |
|201607|male   |1098063|20.19   |1.72    |
|201607|female |1056128|19.42   |1.65    |
|201608|NULL  |3438762|61.23   |5.38    |
|201608|male   |1109603|19.76   |1.74    |
|201608|female |1067556|19.01   |1.67    |
|201609|NULL  |3581269|61.97   |5.61    |
|201609|male   |1120684|19.39   |1.75    |
|201609|female |1077033|18.64   |1.69    |
|201610|NULL  |3730748|62.71   |5.84    |
|201610|male   |1131565|19.02   |1.77    |
|201610|female |1086975|18.27   |1.7     |
|201611|NULL  |3875692|63.39   |6.07    |
|201611|male   |1142532|18.69   |1.79    |
|201611|female |1096121|17.93   |1.72    |
|201612|NULL  |4028623|64.07   |6.31    |
|201612|male   |1153440|18.34   |1.81    |
|201612|female |1105726|17.59   |1.73    |
+-----+-----+-----+-----+
```

```
In [20]: gender_per_safra = gender_per_safra.withColumn("gender", F.when(F.col("gender").isNull(), "unknown").otherwise(F.col("gender"))))
```

```
In [21]: plot_tendencia_temporal(gender_per_safra, col_valor="pct_safra", categories="gender")
```



```
In [22]: verificar_mudanca_estado(df_members, "gender", visualizar=True)
```

Contagem de mudanças por safra:

```
+-----+-----+
|safra|mudou|count|
+-----+-----+
+-----+-----+
```

Mudanças por usuário:

```
+-----+
|msno|total_mudancas|
+-----+
+-----+
```

Pendente: verificar margem liquida por categoria. Ideia: separar numa possivel variavel binaria gender_registered --> pessoas com genero cadastrado tem maior/menor margem liquida do que as que nao tem este cadastro?

4.3.7. feature: city

```
In [23]: calcular_distribuicao(df_members, "city")
```

```
+-----+-----+
|city|total    |pct_total|
+-----+-----+
|1   |41955263|65.69  |
|5   |4271348  |6.69   |
|13  |3650065  |5.72   |
|4   |2730740  |4.28   |
|22  |2328925  |3.65   |
|15  |2123349  |3.32   |
|6   |1492818  |2.34   |
|14  |1004416  |1.57   |
|12  |740584   |1.16   |
|9   |538369   |0.84   |
+-----+-----+
only showing top 10 rows
```

```
Out[23]: DataFrame[city: string, total: bigint, pct_total: double]
```

Pendente: checar valores de margem liquida para ver como fazer o agrupamento entre cidades.

Ideia possivel tambem: criar uma categorica binaria: cidade 1 ou nao.

Checando transicoes de cidade

```
In [24]: verificar_mudanca_estado(df_members, "city", visualizar=True)
```

Contagem de mudanças por safra:

```
+-----+-----+
|safra|mudou|count|
+-----+-----+
+-----+-----+
```

Mudanças por usuário:

```
+-----+
|msno|total_mudancas|
+-----+
+-----+
```

4.3.8. feature: registration_init_time

```
In [11]: df_members.select("registration_init_time").distinct().count()
```

```
Out[11]: 4663
```

```
In [74]: df_members.select("registration_init_time").summary("count", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+
|summary|registration_init_time|
+-----+-----+
| count | 63867246 |
| min | 20040326 |
| 1% | 2.0051023E7 |
| 5% | 2.0081015E7 |
| 25% | 2.013102E7 |
| 50% | 2.0150316E7 |
| 75% | 2.0151225E7 |
| 95% | 2.0160714E7 |
| 99.5% | 2.0161118E7 |
| max | 20161231 |
+-----+-----+
```

Sao muitas e diversas datas de registro, as quais vao de 26/03/2004 ate 31/12/2016. Checar agrupamento por ANO, ANOMES, talvez ANOMESSEMANA

```
In [ ]: df_registration_init_time = (df_members
          .select("msno", "registration_init_time")
          .withColumn("ano_registration", F.year(F.to_date(F.col("registration_init_time").cast("string"),
          "yyyyMMdd"))))

df_registration_init_time.groupBy("ano_registration").agg(
    F.count("msno").alias("total_members"),
    F.round((F.count("msno") / F.lit(total_members)) * 100, 2).alias("pct_total")
).orderBy("ano_registration").show(50)
```

```
+-----+-----+-----+
|ano_registration|total_members|pct_total|
+-----+-----+-----+
| 2004 | 314808 | 0.49 |
| 2005 | 496188 | 0.78 |
| 2006 | 647436 | 1.01 |
| 2007 | 1077960 | 1.69 |
| 2008 | 812280 | 1.27 |
| 2009 | 763596 | 1.21 |
| 2010 | 1380900 | 2.16 |
| 2011 | 2148612 | 3.36 |
| 2012 | 3398280 | 5.32 |
| 2013 | 6296664 | 9.86 |
| 2014 | 11709312 | 18.33 |
| 2015 | 19446300 | 30.45 |
| 2016 | 15374910 | 24.07 |
+-----+-----+-----+
```

Possiveis agregacoes:

```
In [25]: df_registration_init_time = df_registration_init_time.withColumn("ano_registration_final", F.when(
    F.col("ano_registration") <= 2010, F.lit("2004-2010")).otherwise(F.col("ano_registration")))

df_registration_init_time.groupBy("ano_registration_final").agg(
    F.count("msno").alias("total_members"),
    F.round((F.count("msno") / F.lit(total_members)) * 100, 2).alias("pct_total")
).orderBy(("ano_registration_final")).show(50)
```

	ano_registration_final	total_members	pct_total
2004-2010	5493168	8.6	
2011	2148612	3.36	
2012	3398280	5.32	
2013	6296664	9.86	
2014	11709312	18.33	
2015	19446300	30.45	
2016	15374910	24.07	

```
In [24]: df_registration_init_time = df_registration_init_time.withColumn("ano_registration_final", F.when(
    F.col("ano_registration") <= 2013, F.lit("2004-2013")).otherwise(F.col("ano_registration")))

df_registration_init_time.groupBy("ano_registration_final").agg(
    F.count("msno").alias("total_members"),
    F.round((F.count("msno") / F.lit(total_members)) * 100, 2).alias("pct_total")
).orderBy(("ano_registration_final")).show(50)
```

	ano_registration_final	total_members	pct_total
2004-2013	17336724	27.14	
2014	11709312	18.33	
2015	19446300	30.45	
2016	15374910	24.07	

```
In [23]: df_registration_init_time = df_registration_init_time.withColumn("ano_registration_final",
    F.when(F.col("ano_registration") <= 2011, F.lit("2004-2011"))\
    .when(F.col("ano_registration").isin(2012, 2013), F.lit("2012-2013"))\
    .otherwise(F.col("ano_registration")))

df_registration_init_time.groupBy("ano_registration_final").agg(
    F.count("msno").alias("total_members"),
    F.round((F.count("msno") / F.lit(total_members)) * 100, 2).alias("pct_total")
).orderBy(("ano_registration_final")).show(50)
```

	ano_registration_final	total_members	pct_total
2004-2011	7641780	11.97	
2012-2013	9694944	15.18	
2014	11709312	18.33	
2015	19446300	30.45	
2016	15374910	24.07	

Pendente: verificar margem líquida para cada ano de registro e capturar informações para melhor agrupamento.

4.3.9. feature: is_ativo

```
In [28]: calcular_distribuicao(df_members, ["is_ativo"])
calcular_distribuicao(df_members, ["is_ativo"], agrupar_por_safra=True)
```

```
+-----+-----+-----+
|is_ativo|total |pct_total|
+-----+-----+-----+
|0      |52624381|82.4    |
|1      |11242865|17.6    |
+-----+-----+-----+
+-----+-----+-----+-----+
|safra |is_ativo|total |pct_safra|pct_total|
+-----+-----+-----+-----+
|201601|0      |3405521|79.31   |5.33
|201601|1      |888663 |20.69   |1.39
|201602|0      |3584238|79.61   |5.61
|201602|1      |918116 |20.39   |1.44
|201603|0      |3826321|81.45   |5.99
|201603|1      |871651 |18.55   |1.36
|201604|0      |4044736|82.95   |6.33
|201604|1      |831565 |17.05   |1.3
|201605|0      |4211052|83.21   |6.59
|201605|1      |849653 |16.79   |1.33
+-----+-----+-----+-----+
only showing top 10 rows
```

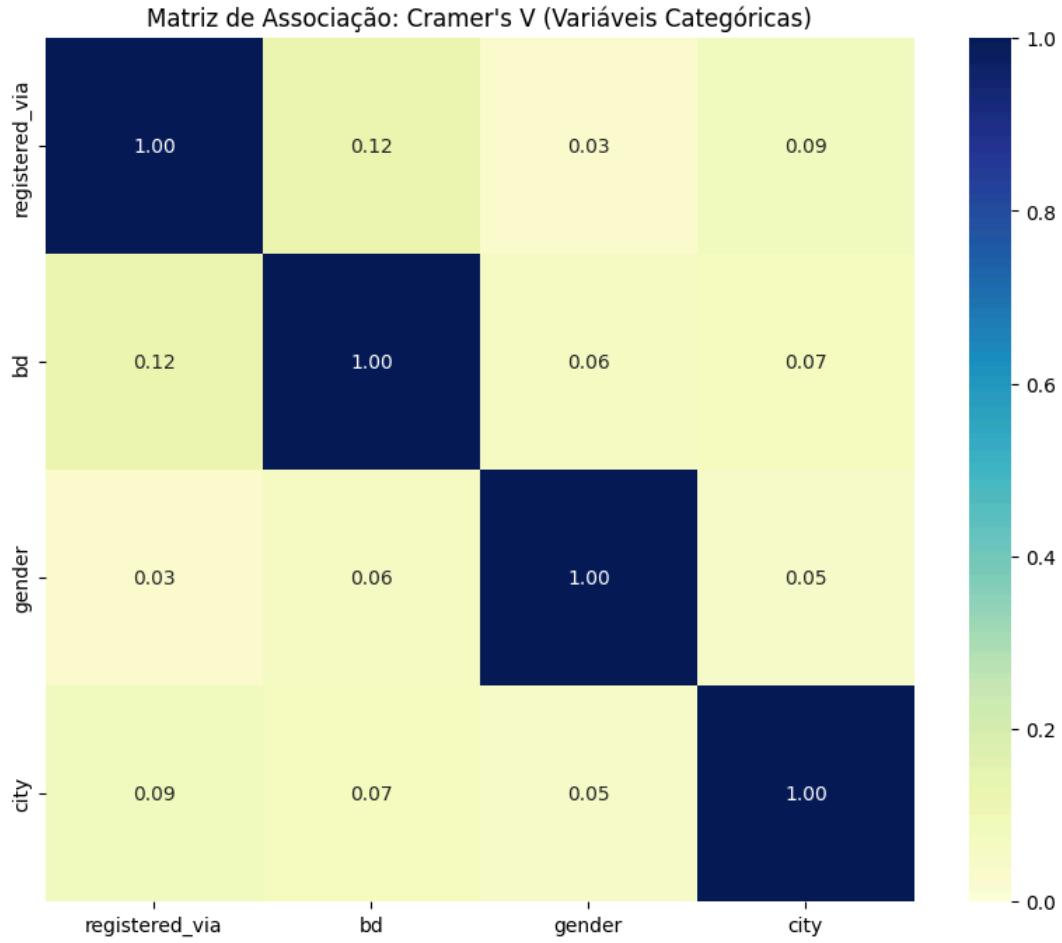
```
Out[28]: DataFrame[safra: string, is_ativo: int, total: bigint, pct_safra: double, pct_total: double]
```

Vou usar, basicamente, somente 17,6% do dataframe, dado que o foco preciso ser no publico ativo.

4.3.10. Cramer V

```
In [29]: matriz_cramer_v_spark(df_members.filter(F.col("is_ativo").isin(1)), ["registered_via", "bd", "gender", "city"])
```

```
Iniciando indexação de 4 colunas...
Calculando Cramer's V: registered_via vs bd...
Calculando Cramer's V: registered_via vs gender...
Calculando Cramer's V: registered_via vs city...
Calculando Cramer's V: bd vs gender...
Calculando Cramer's V: bd vs city...
Calculando Cramer's V: gender vs city...
```



	registered_via	bd	gender	city
registered_via	1.000000	0.119932	0.031408	0.085063
bd	0.119932	1.000000	0.062330	0.070997
gender	0.031408	0.062330	1.000000	0.048697
city	0.085063	0.070997	0.048697	1.000000

Relação fracaissima entre variáveis categóricas do dataframe de members. Isso indica que podemos considerar aplicá-las ao mesmo tempo na engenharia de variáveis ou até na seleção de variáveis para o modelo.

4.4. Relação entre tabelas

4.4.1. tabela members + logs

```
In [74]: integridade_entre_bases(df_members, df_logs, "Membros x Logs")
```

```
-- Integridade: Membros x Logs --
Registros na base: 63867246 | Registros correspondentes: 13103573
Taxa de Perda: 79.48%
```

```
In [26]: integrade_entre_bases(df_members.filter(F.col("is_ativo").isin(1)), df_logs, "Membros ativos x Logs")
```

```
--- Integridade: Membros ativos x Logs ---
Registros na base: 11242865 | Registros correspondentes: 9975880
Taxa de Perda: 11.27%
```

```
In [24]: df_atv_members_logs = df_members.filter(F.col("is_ativo").isin(1)).join(df_logs, on=["msno", "safra"], how="inner")
```

```
#### registered_via
```

```
In [ ]: calcular_distribuicao(df_atv_members_logs, ["registered_via"], n_show=25)
```

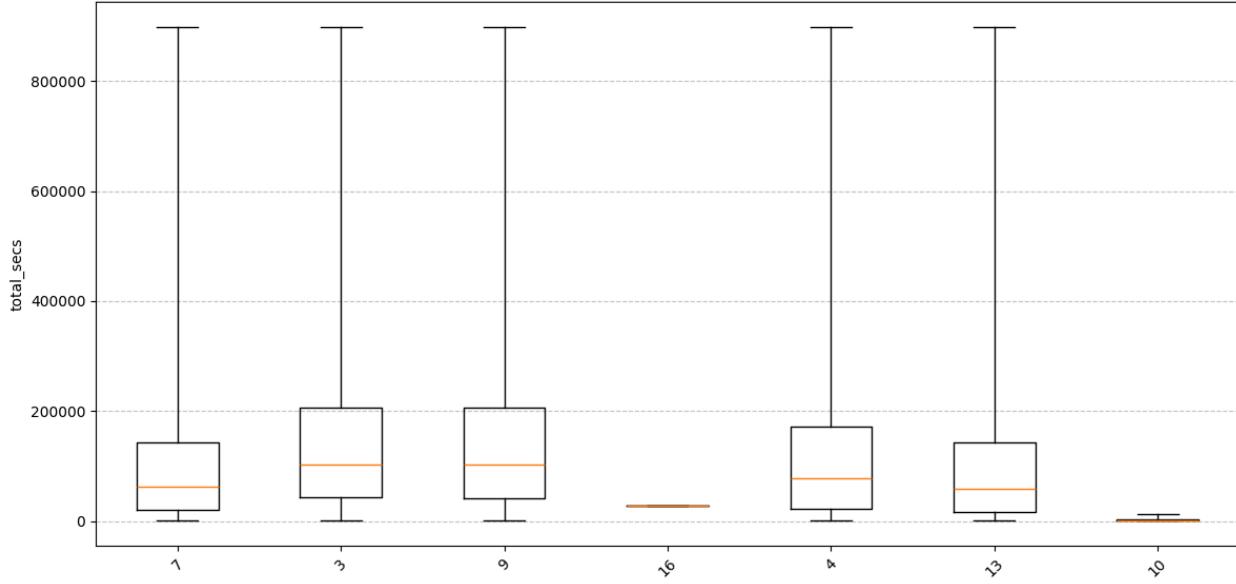
registered_via	total	pct_total
7	4320557	43.31
9	3477122	34.86
3	1585188	15.89
4	586714	5.88
13	6293	0.06
10	5	0.0
16	1	0.0

```
DataFrame[registered_via: string, total: bigint, pct_total: double]
```

```
In [ ]: df_atv_members_logs = aplicar_winsorizacao(df_atv_members_logs, ["total_secs"])
plot_boxplot(df_atv_members_logs, ["registered_via"], "total_secs", agrupar_por_safra=False)
```

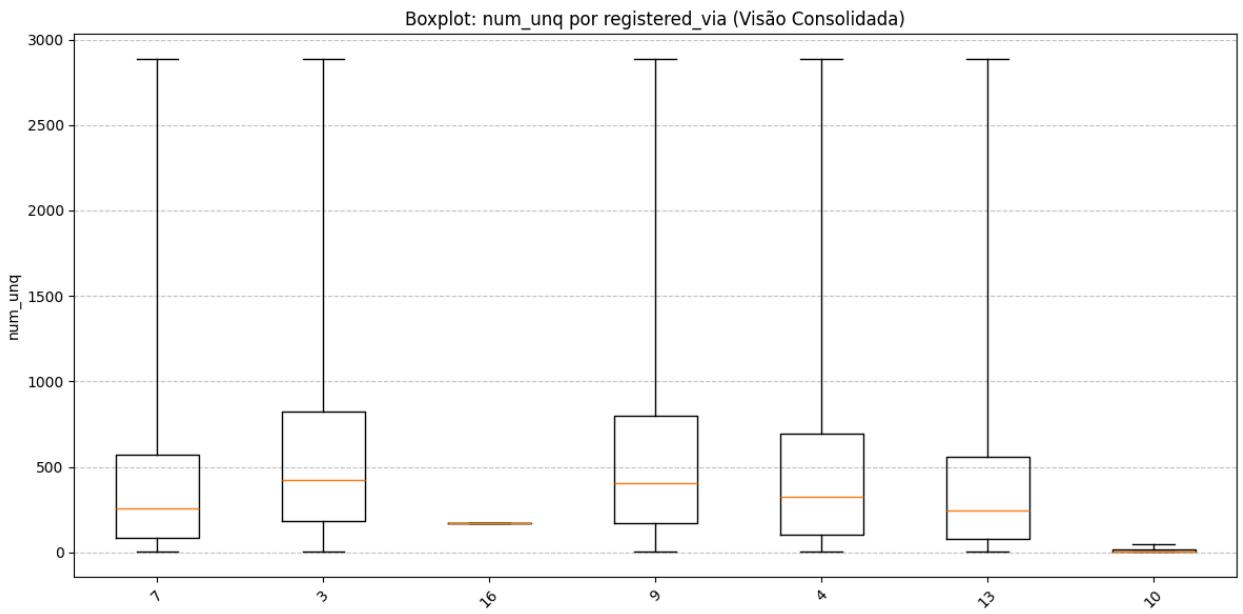
```
Coluna total_secs: Limite Inferior=245.213, Limite Superior=898842.207999999
Processando estatísticas para: registered_via...
```

Boxplot: total_secs por registered_via (Visão Consolidada)



```
In [ ]: df_atv_members_logs = aplicar_winsorizacao(df_atv_members_logs, ["num_unq"])
plot_boxplot(df_atv_members_logs, ["registered_via"], "num_unq", agrupar_por_safra=False)
```

Coluna num_unq: Limite Inferior=2.0, Limite Superior=2888.0
Processando estatísticas para: registered_via...



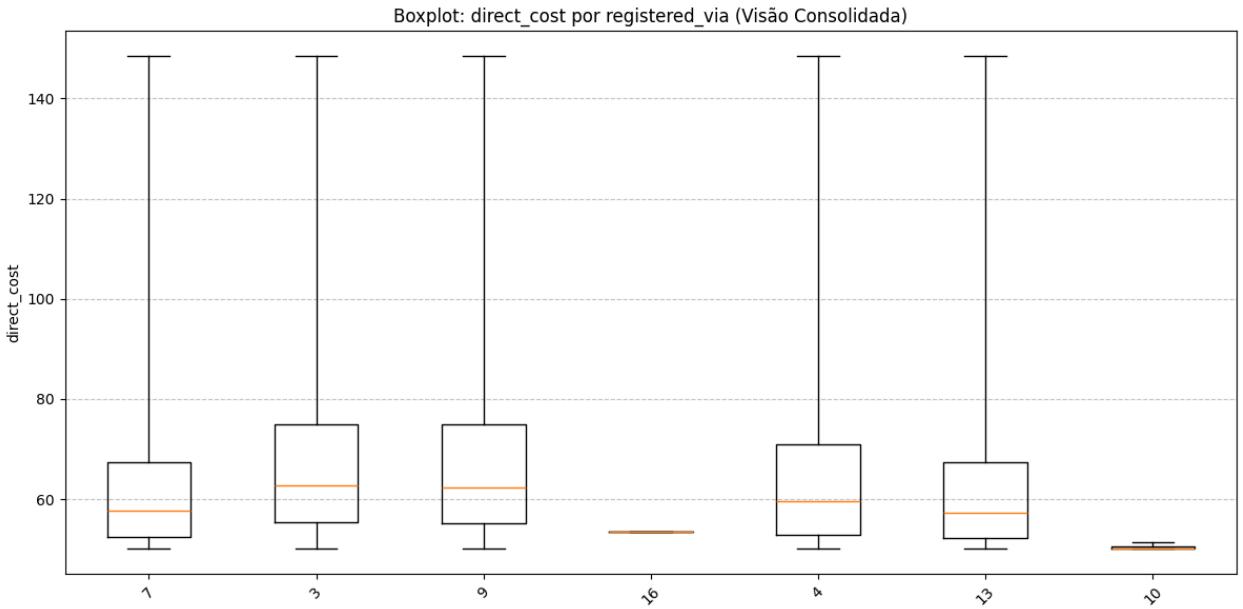
```
In [58]: df_atv_members_logs = df_atv_members_logs.withColumn("direct_cost", (50 + (0.0051 * F.col("num_unq")) + (0.0001 * F.col("total_secs"))))
```

```
In [59]: df_atv_members_logs = aplicar_winsorizacao(df_atv_members_logs, ["direct_cost"])
```

Coluna direct_cost: Limite Inferior=50.040212, Limite Superior=148.5311526

```
In [61]: plot_boxplot(df_atv_members_logs, ["registered_via"], "direct_cost", agrupar_por_safra=False)
```

Processando estatísticas para: registered_via...

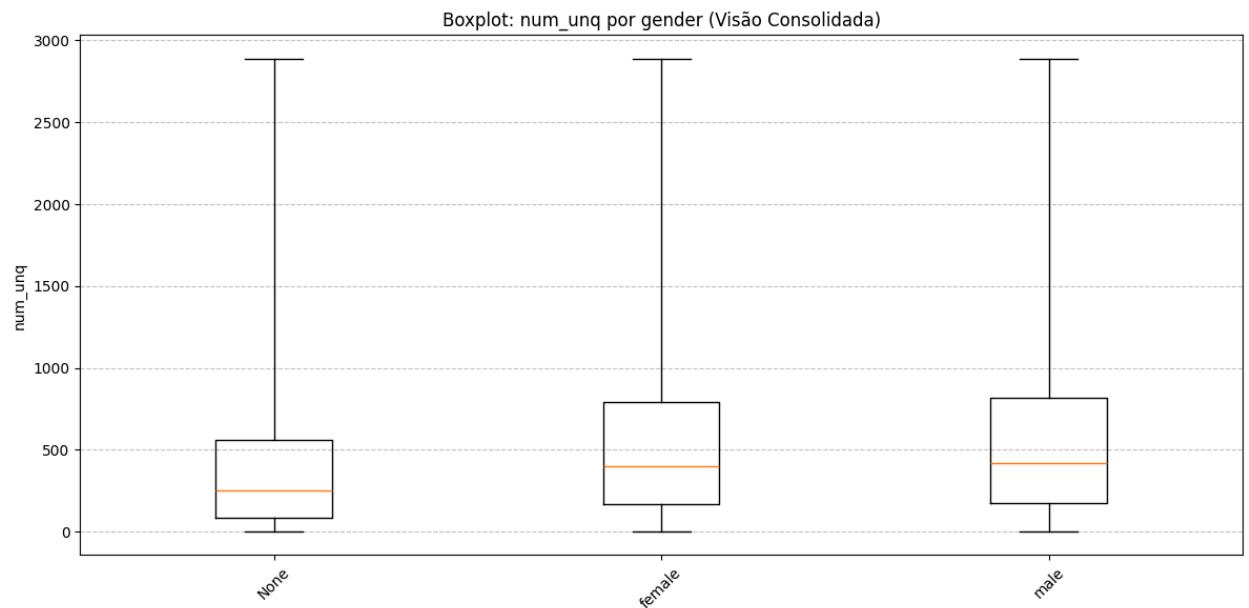


Possibilidade de agrupar pela similaridade de mediana, criando uma flag binaria (mediana acima e abaixo de 60). No entanto, faz sentido eu fazer isso baseado no custo, que compõe a target? Entendo que sim, porque caracteriza parte do processo de EDA, e não estou usando como variável preditiva, o que caracterizaria target leakage.

```
#### gender
```

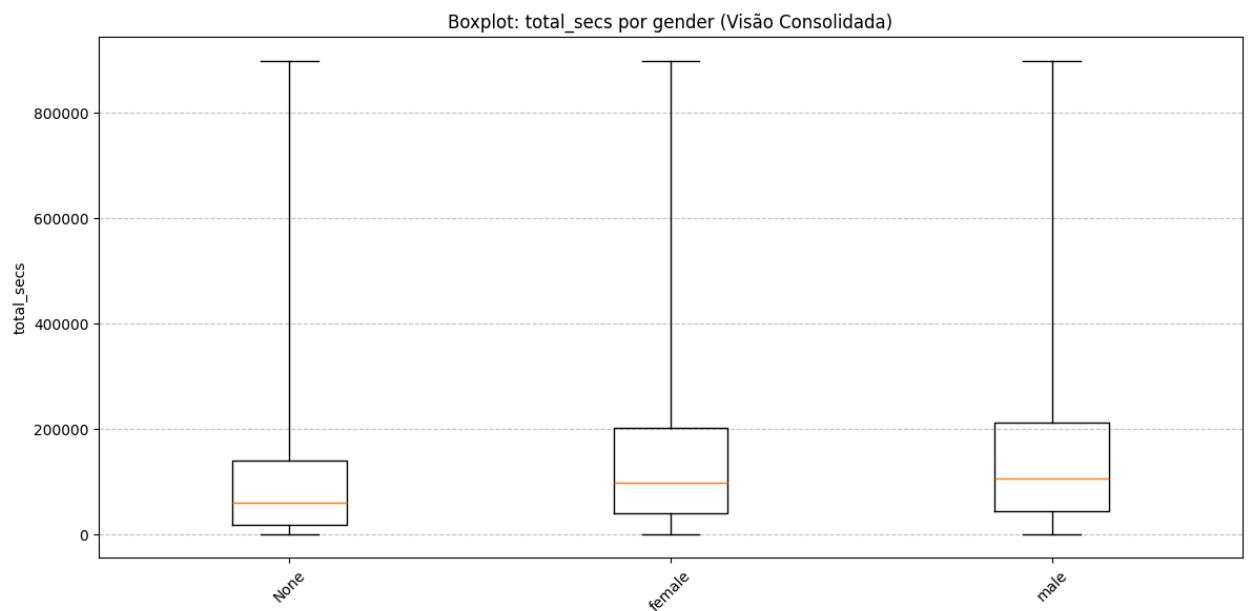
```
In [65]: plot_boxplot(df_atv_members_logs, ["gender"], "num_unq", agrupar_por_safra=False)
```

Processando estatísticas para: gender...



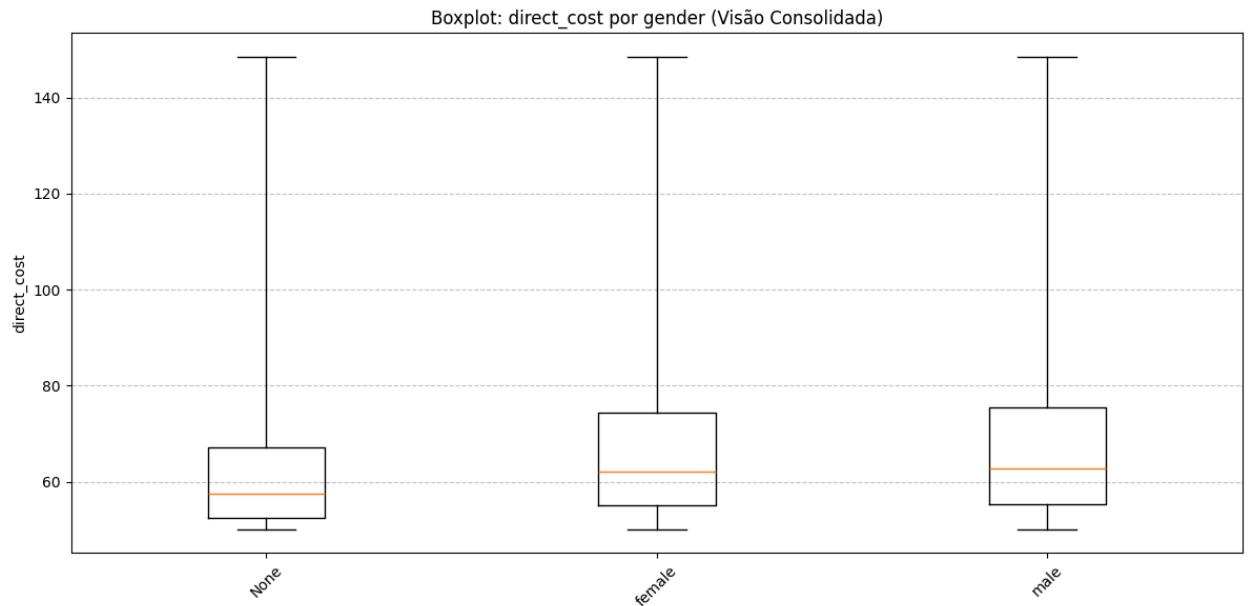
```
In [64]: plot_boxplot(df_atv_members_logs, ["gender"], "total_secs", agrupar_por_safra=False)
```

Processando estatísticas para: gender...



```
In [62]: plot_boxplot(df_atv_members_logs, ["gender"], "direct_cost", agrupar_por_safra=False)
```

Processando estatísticas para: gender...



```
#### bd
```

```
In [28]: calcular_distribuicao(df_atv_members_logs, ["bd"], n_show=25)
```

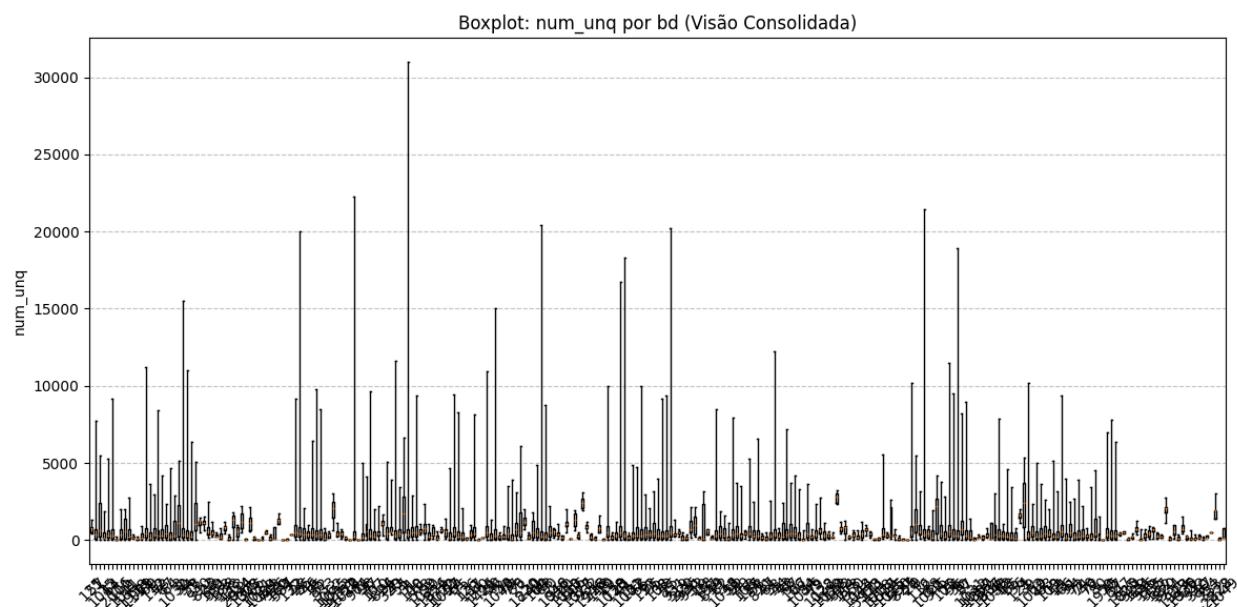
bd	total	pct_total
0	4512345	45.23
27	317388	3.18
26	301659	3.02
25	287969	2.89
24	285138	2.86
28	284453	2.85
29	276414	2.77
23	274432	2.75
22	267905	2.69
30	243160	2.44
21	240832	2.41
20	213352	2.14
31	211683	2.12
32	203737	2.04
33	187725	1.88
34	169207	1.7
35	154330	1.55
19	145388	1.46
36	139937	1.4
37	126112	1.26
18	115591	1.16
38	106141	1.06
39	90791	0.91
40	78019	0.78
17	75544	0.76

only showing top 25 rows

```
Out[28]: DataFrame[bd: string, total: bigint, pct_total: double]
```

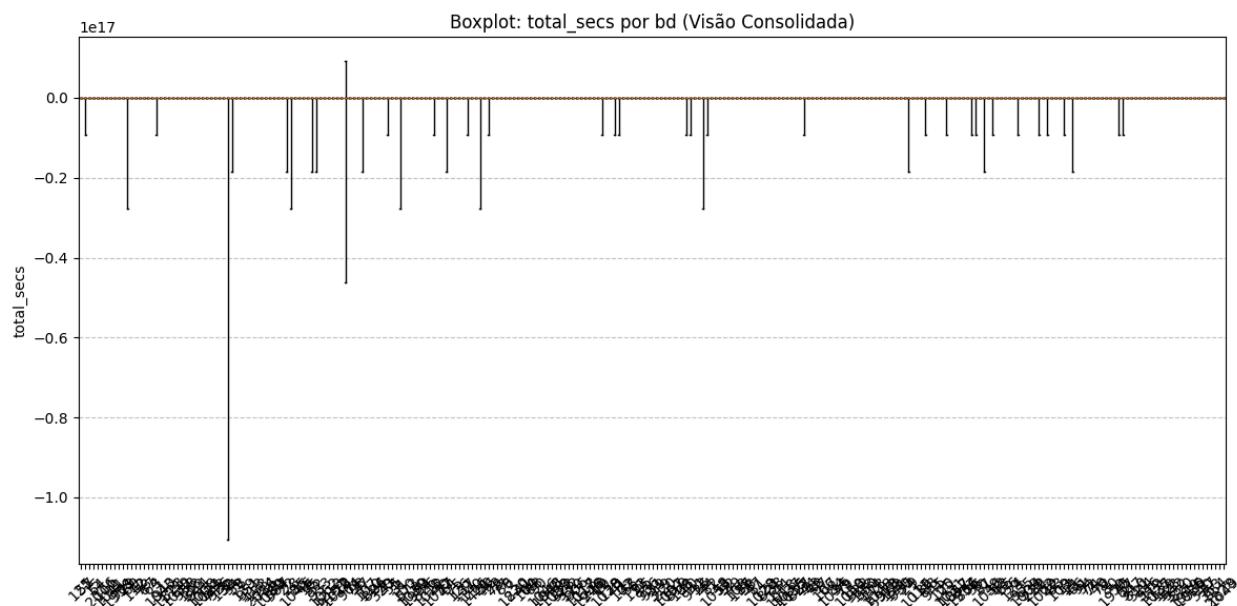
```
In [25]: plot_boxplot(df_atv_members_logs, ["bd"], "num_unq", agrupar_por_safra=False)
```

Processando estatísticas para: bd...



```
In [26]: plot_boxplot(df_atv_members_logs, ["bd"], "total_secs", agrupar_por_safra=False)
```

Processando estatísticas para: bd...

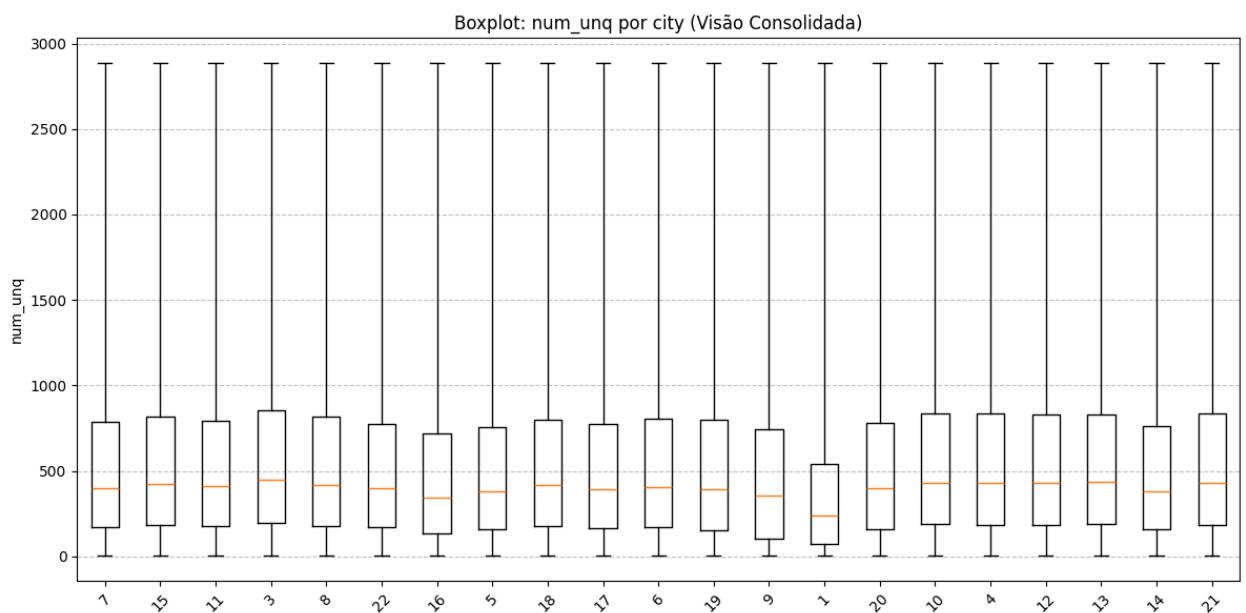


Melhor verificar depois do tratamento de outliers

```
#### city
```

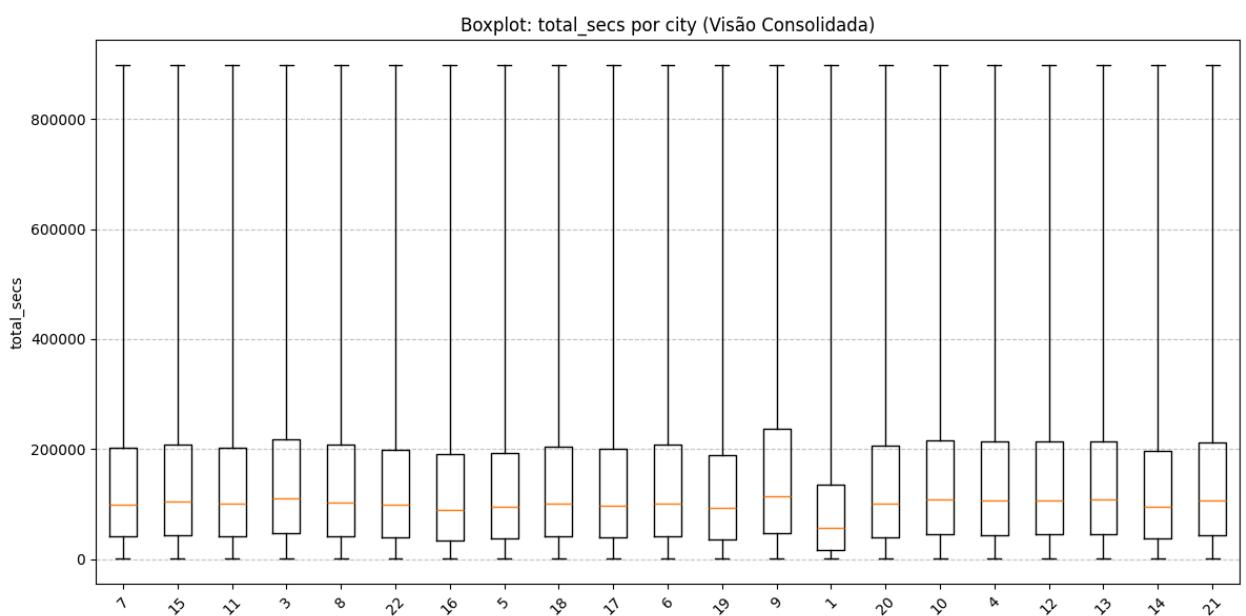
```
In [66]: plot_boxplot(df_atv_members_logs, ["city"], "num_unq", agrupar_por_safra=False)
```

Processando estatísticas para: city...



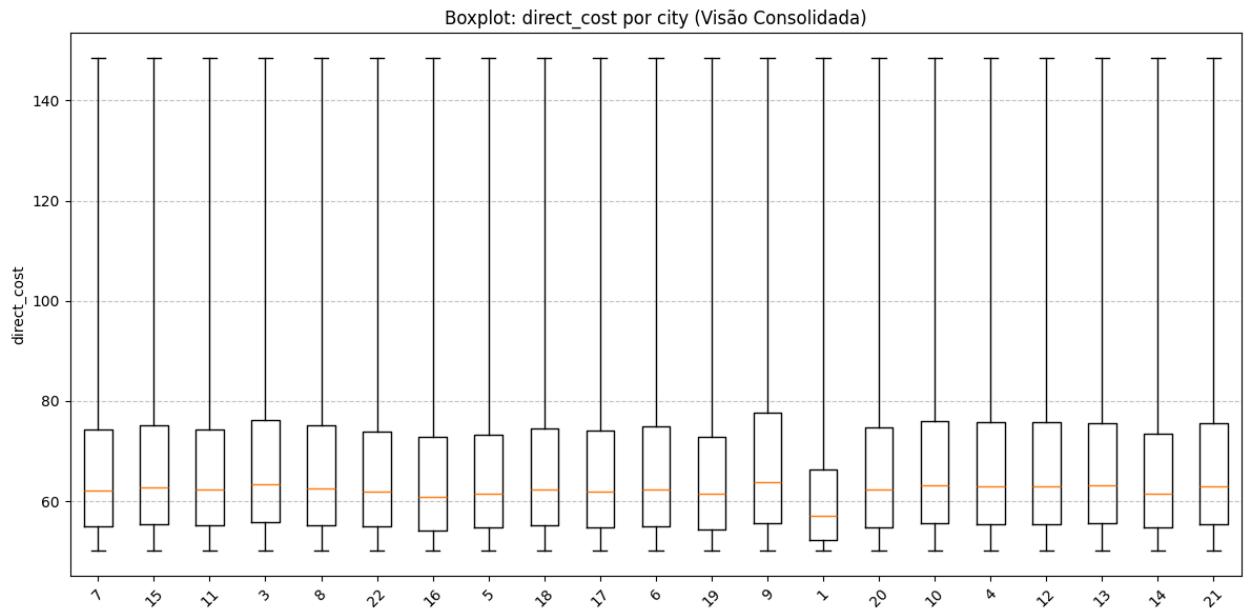
```
In [67]: plot_boxplot(df_atv_members_logs, ["city"], "total_secs", agrupar_por_safra=False)
```

Processando estatísticas para: city...



```
In [68]: plot_boxplot(df_atv_members_logs, ["city"], "direct_cost", agrupar_por_safra=False)
```

Processando estatísticas para: city...



Cidade 1 parece ser bem mais comprimida, talvez faça sentido se tornar uma variável binária de fato

```
In [69]: df_atv_members_logs = df_atv_members_logs.withColumn("flag_city_1", F.when(F.col("city") == 1, 1).otherwise(0))
```

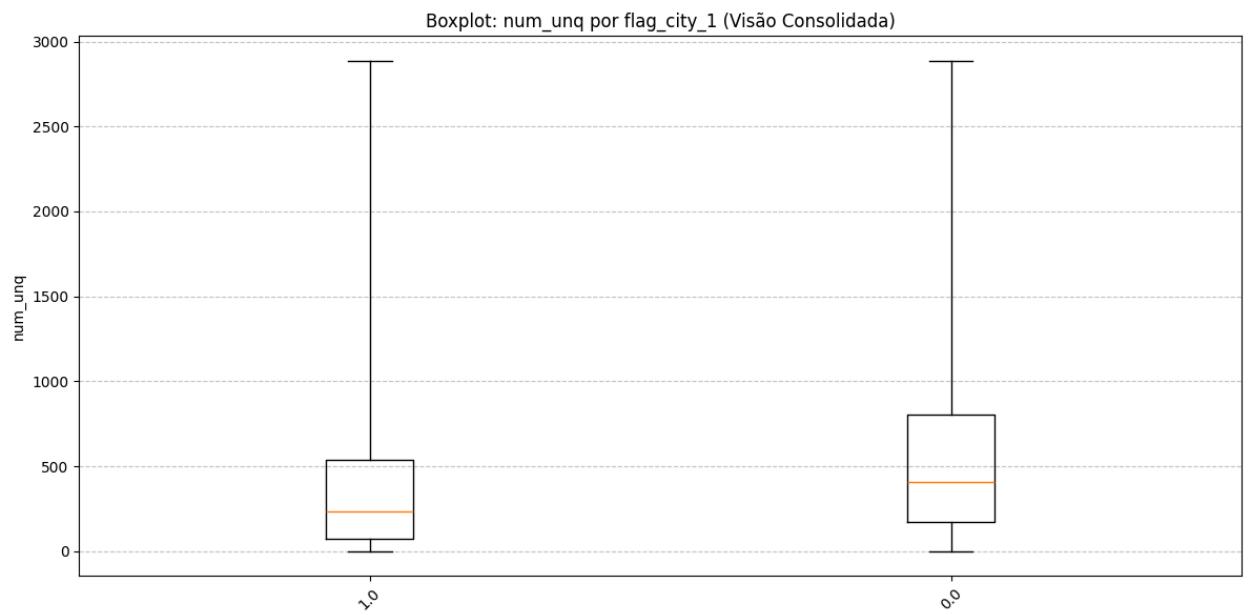
```
In [71]: calcular_distribuicao(df_atv_members_logs, ["flag_city_1"], n_show=25)
```

flag_city_1	total	pct_total
0	5900484	59.15
1	4075396	40.85

Out[71]: DataFrame[flag_city_1: int, total: bigint, pct_total: double]

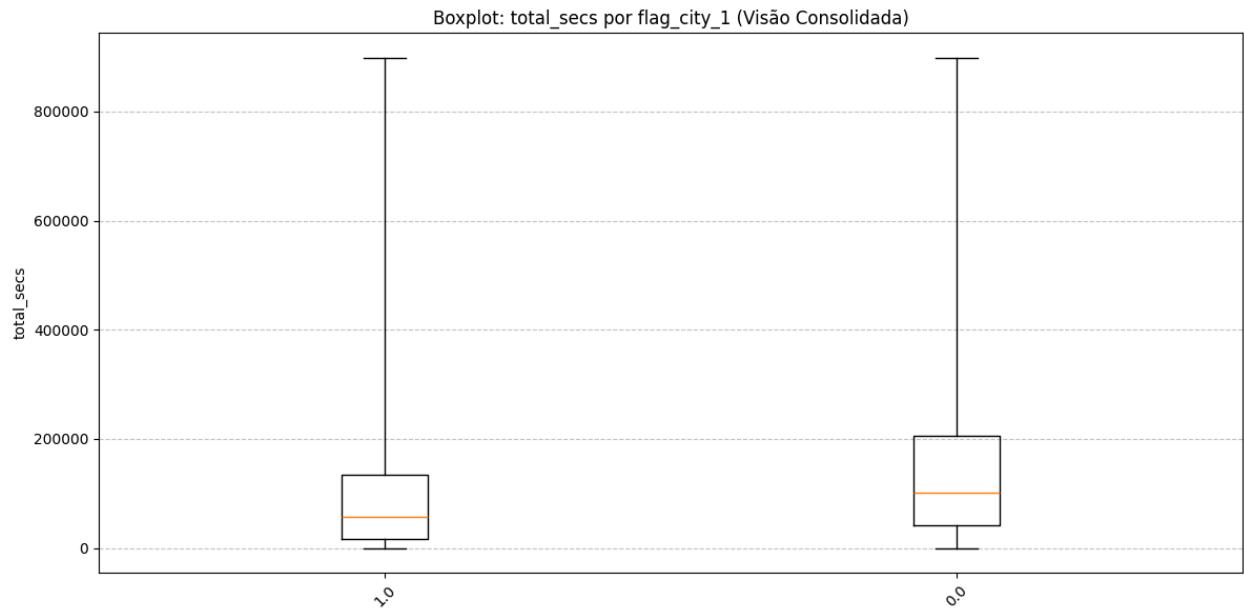
```
In [70]: plot_boxplot(df_atv_members_logs, ["flag_city_1"], "num_unq", agrupar_por_safra=False)
```

Processando estatísticas para: flag_city_1...



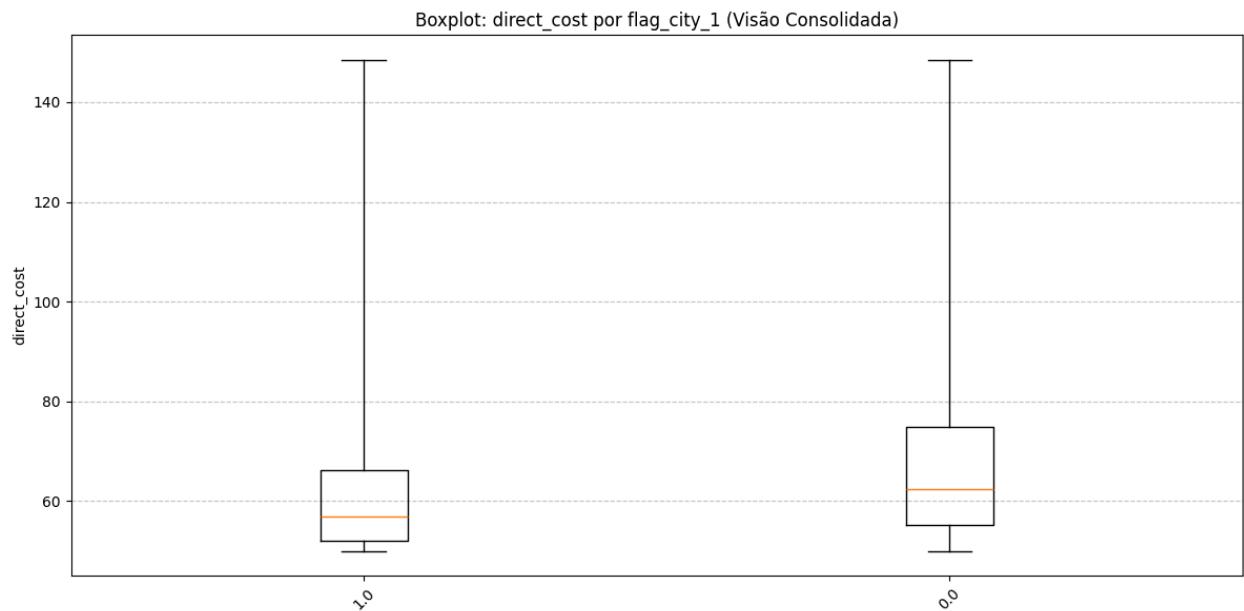
```
In [72]: plot_boxplot(df_atv_members_logs, ["flag_city_1"], "total_secs", agrupar_por_safra=False)
```

Processando estatísticas para: flag_city_1...



```
In [73]: plot_boxplot(df_atv_members_logs, ["flag_city_1"], "direct_cost", agrupar_por_safra=False)
```

Processando estatísticas para: flag_city_1...



registration_time

```
In [44]: df_atv_members_logs = df_atv_members_logs.withColumn("ano_registration", F.year(F.to_date(F.col("registration_init_time").cast("string"), "yyyyMMdd"))))
```

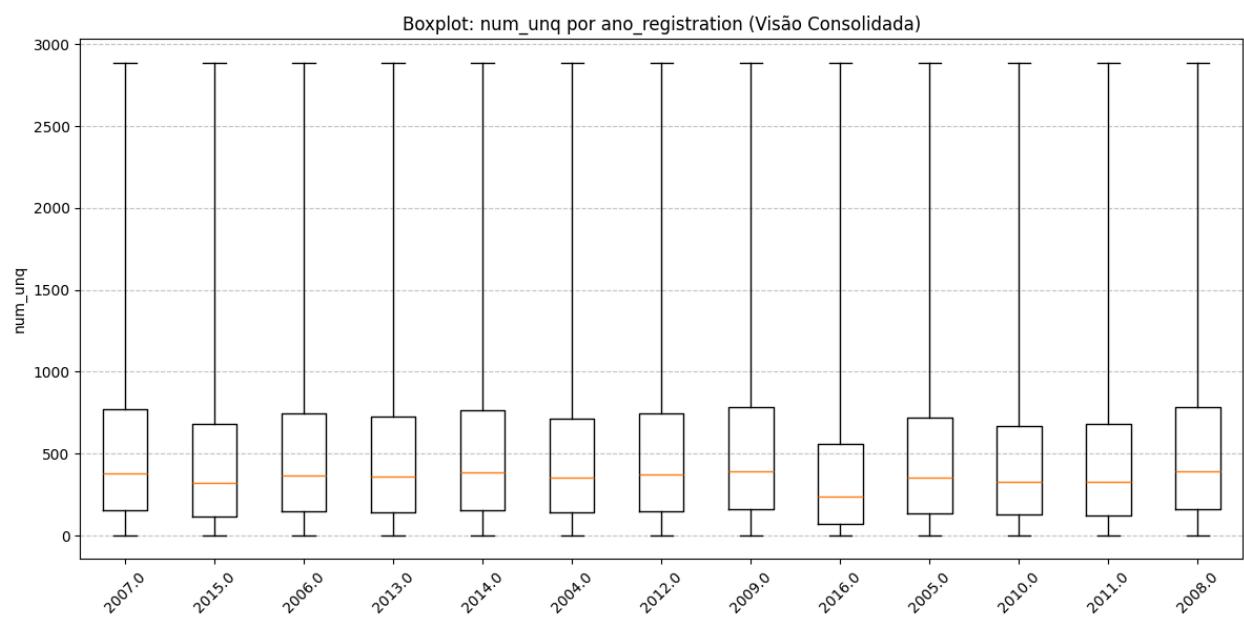
```
In [45]: calcular_distribuicao(df_atv_members_logs, ["ano_registration"], n_show=25)
```

ano_registration	total	pct_total
2015	2054475	20.59
2016	1536902	15.41
2013	1464533	14.68
2014	1191467	11.94
2012	1020387	10.23
2011	813337	8.15
2010	509228	5.1
2007	351636	3.52
2008	245776	2.46
2006	240139	2.41
2009	225910	2.26
2005	203499	2.04
2004	118591	1.19

```
Out[45]: DataFrame[ano_registration: int, total: bigint, pct_total: double]
```

```
In [76]: plot_boxplot(df_atv_members_logs, ["ano_registration"], "num_unq", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration...



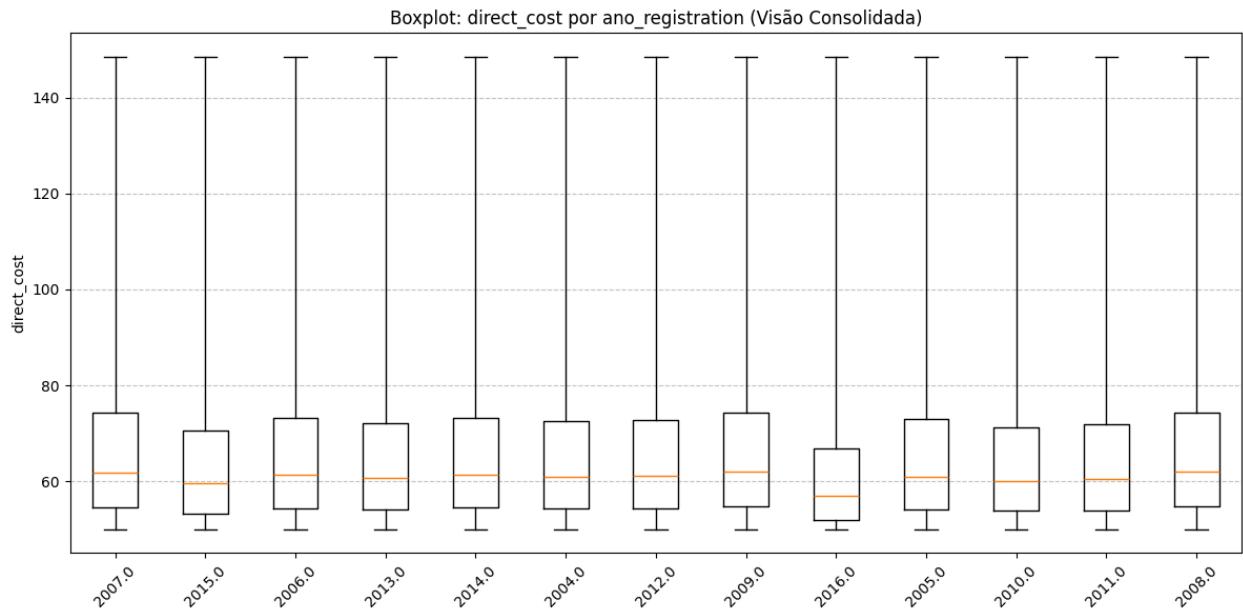
```
In [77]: plot_boxplot(df_atv_members_logs, ["ano_registration"], "total_secs", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration...



```
In [78]: plot_boxplot(df_atv_members_logs, ["ano_registration"], "direct_cost", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration...



```
In [79]: df_atv_members_logs = df_atv_members_logs.withColumn("ano_registration_final",
    F.when(F.col("ano_registration") <= 2011, F.lit("2004-2011"))\
    .when(F.col("ano_registration").isin(2012, 2013), F.lit("2012-2013"))
    .otherwise(F.col("ano_registration")))
```

Agrupamento considerando distribuicao inicial e questao de antiguidade.

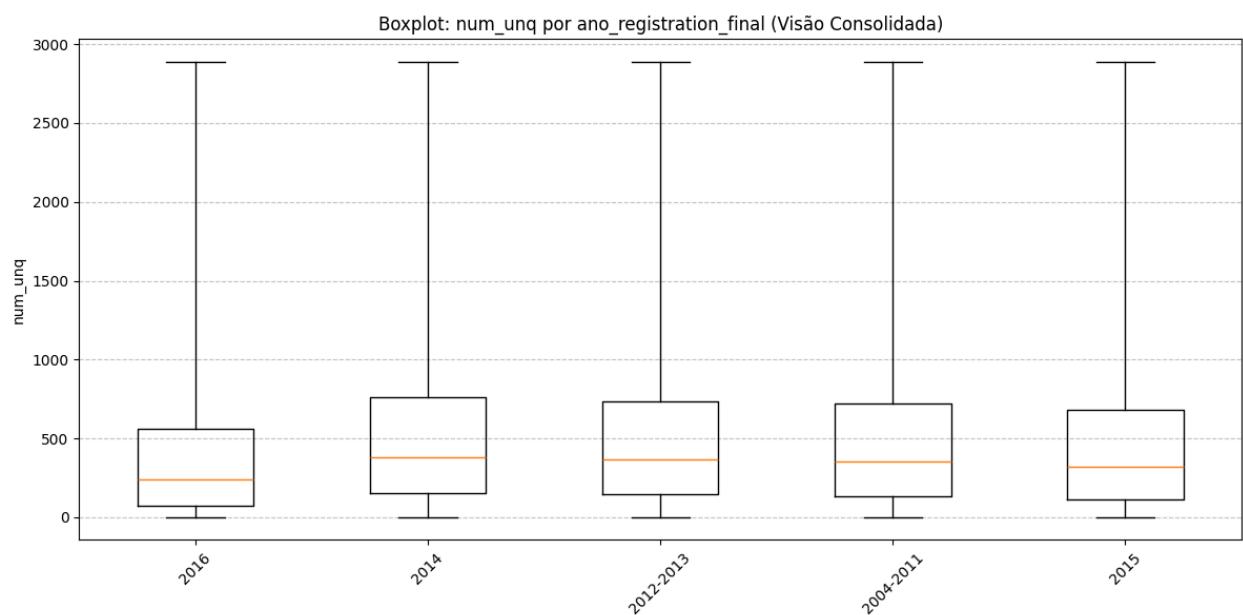
```
In [83]: calcular_distribuicao(df_atv_members_logs, ["ano_registration_final"], n_show=25)
```

ano_registration_final	total	pct_total
2004-2011	2708116	27.15
2012-2013	2484920	24.91
2015	2054475	20.59
2016	1536902	15.41
2014	1191467	11.94

Out[83]: DataFrame[ano_registration_final: string, total: bigint, pct_total: double]

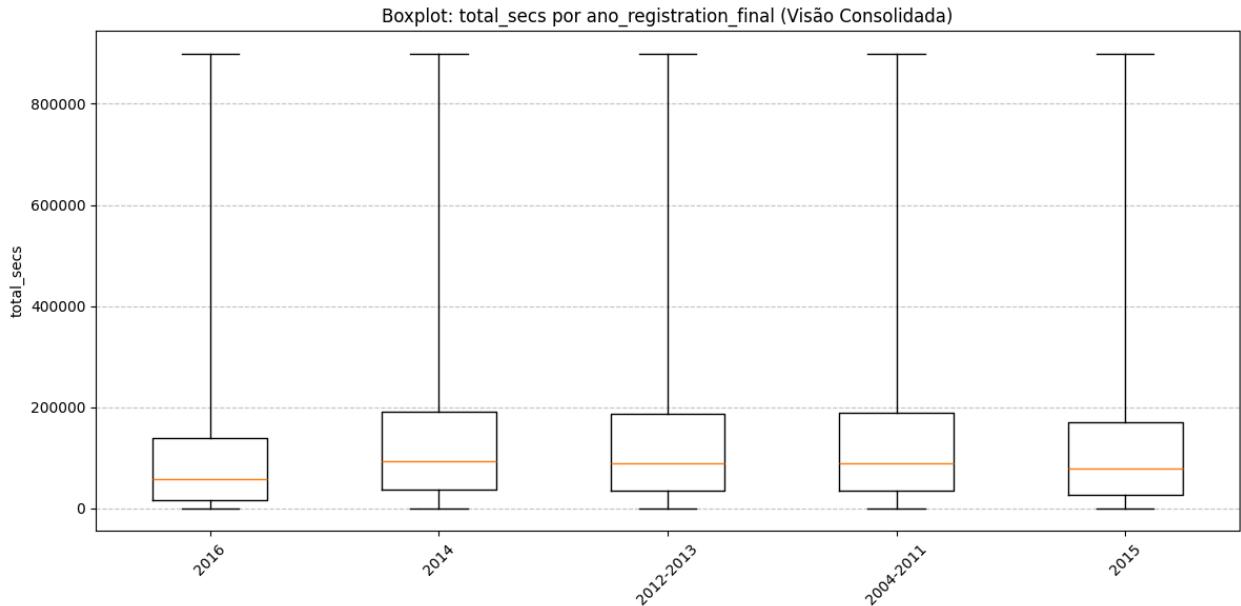
```
In [80]: plot_boxplot(df_atv_members_logs, ["ano_registration_final"], "num_unq", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration_final...



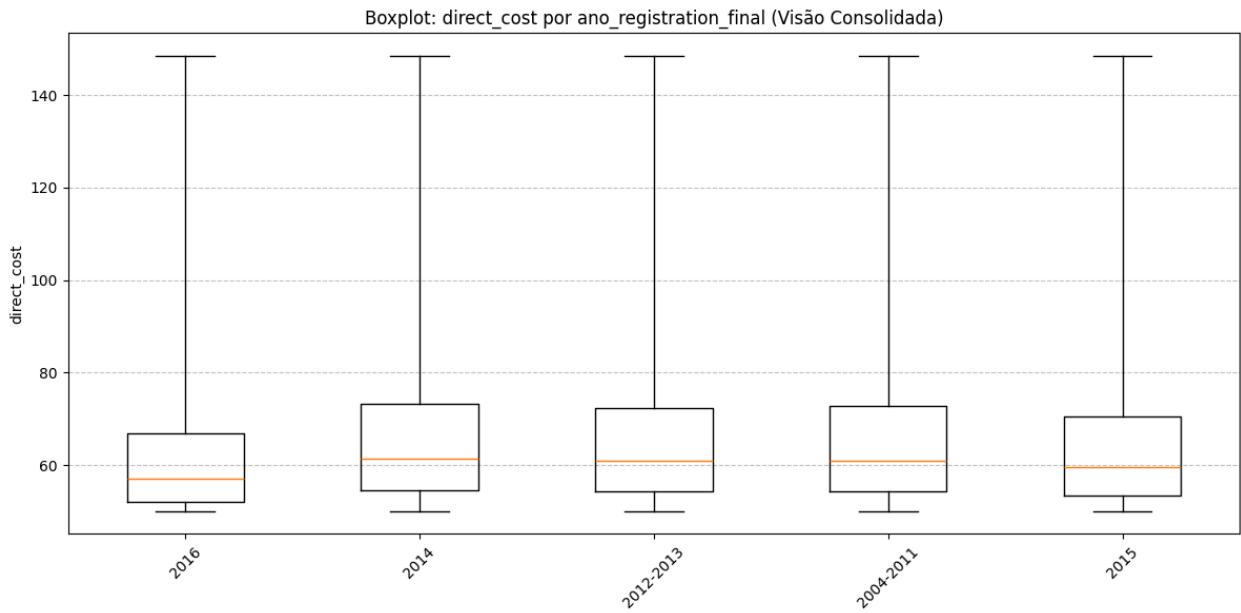
```
In [81]: plot_boxplot(df_atv_members_logs, ["ano_registration_final"], "total_secs", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration_final...



```
In [82]: plot_boxplot(df_atv_members_logs, ["ano_registration_final"], "direct_cost", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration_final...



As estatísticas dos anos são similares... talvez nem compense considerar inserir essa variável. Veremos na etapa de feature selection.

4.4.2. tabela members + transactions

```
In [73]: integridade_entre_bases(df_members, df_transactions, "Membros x Transações")
```

--- Integridade: Membros x Transações ---
Registros na base: 63867246 | Registros correspondentes: 9390483
Taxa de Perda: 85.30%

```
In [27]: integridade_entre_bases(df_members.filter(F.col("is_ativo").isin(1)), df_transactions, "Membros ativos x Transações")
```

--- Integridade: Membros ativos x Transações ---
Registros na base: 11242865 | Registros correspondentes: 9390483
Taxa de Perda: 16.48%

```
In [23]: df_atv_members_transactions = df_members.filter(F.col("is_ativo").isin(1)).join(df_transactions, on=["msno", "safra"], how="inner")
```

```
In [17]: df_atv_members_transactions.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: string (nullable = true)
 |-- registration_init_time: string (nullable = true)
 |-- city: string (nullable = true)
 |-- bd: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- registered_via: string (nullable = true)
 |-- is_ativo: integer (nullable = true)
 |-- payment_method_id: string (nullable = true)
 |-- payment_plan_days: string (nullable = true)
 |-- plan_list_price: string (nullable = true)
 |-- actual_amount_paid: string (nullable = true)
 |-- is_auto_renew: string (nullable = true)
 |-- transaction_date: string (nullable = true)
 |-- membership_expire_date: string (nullable = true)
 |-- is_cancel: string (nullable = true)
```

registered_via

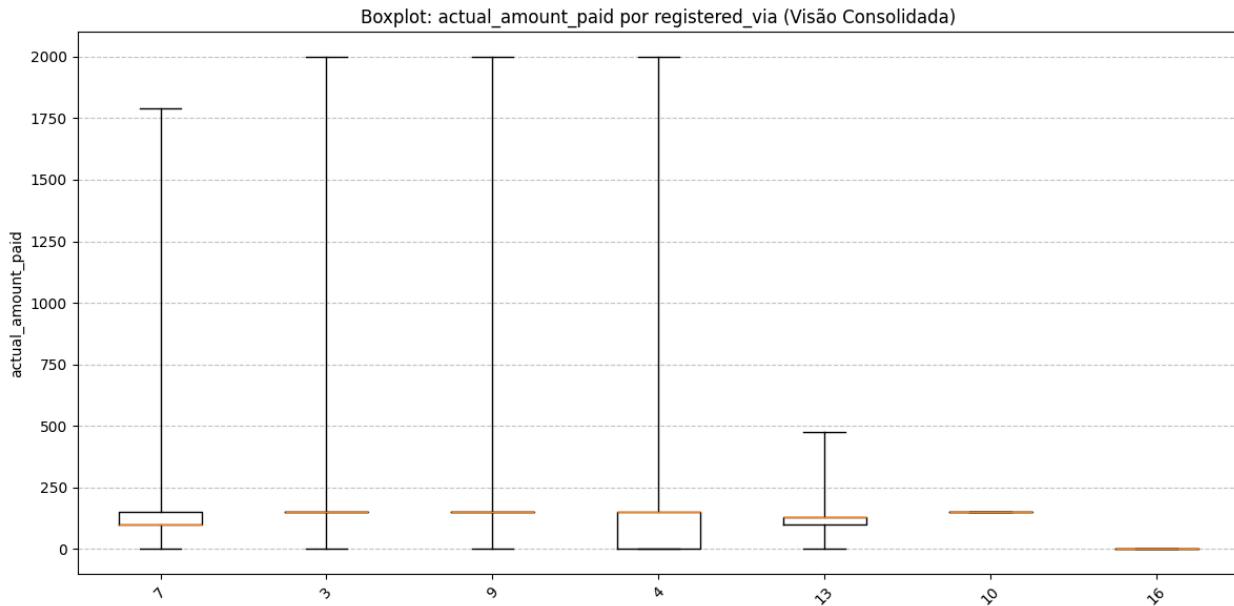
```
In [18]: calcular_distribuicao(df_atv_members_transactions, ["registered_via"], n_show=25)
```

registered_via	total	pct_total
7	5027606	53.54
9	2671514	28.45
3	1171089	12.47
4	513337	5.47
13	6926	0.07
10	10	0.0
16	1	0.0

Out[18]: DataFrame[registered_via: string, total: bigint, pct_total: double]

```
In [19]: plot_boxplot(df_atv_members_transactions, ["registered_via"], "actual_amount_paid", agrupar_por_safra=False)
```

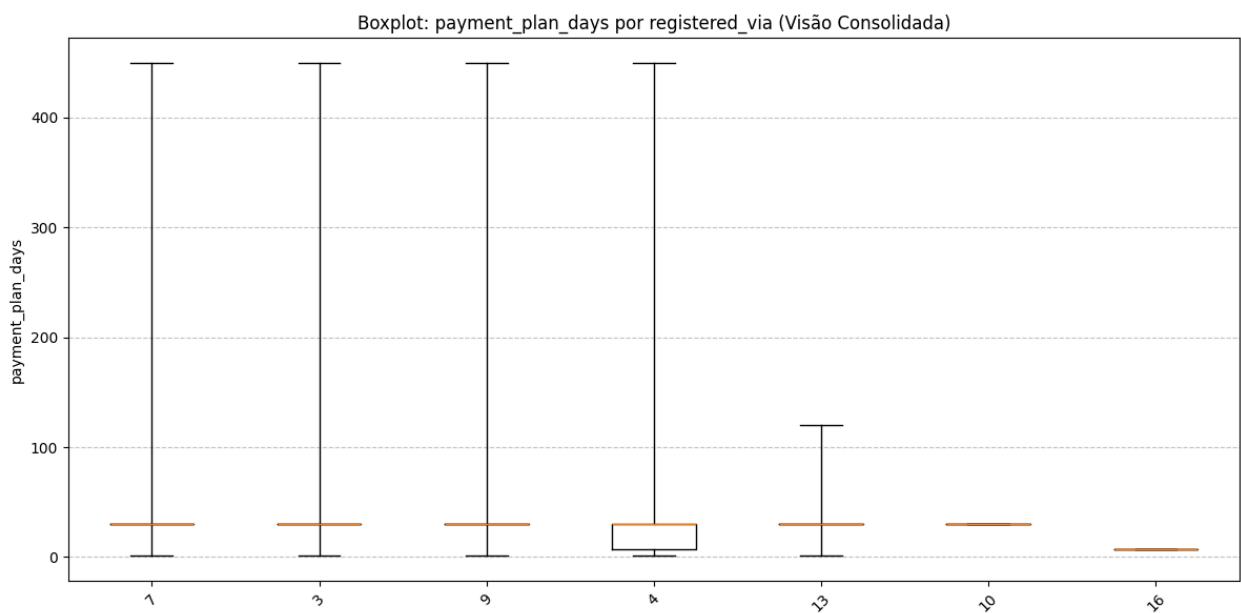
Processando estatísticas para: registered_via...



Medianas muito similares para a segunda, terceira e quarta forma de registro mais frequente.

```
In [20]: plot_boxplot(df_atv_members_transactions, ["registered_via"], "payment_plan_days", agrupar_por_safra=False)
```

Processando estatísticas para: registered_via...



Plots muito similares para as tres mais frequentes vias de registro.

```
#### gender
```

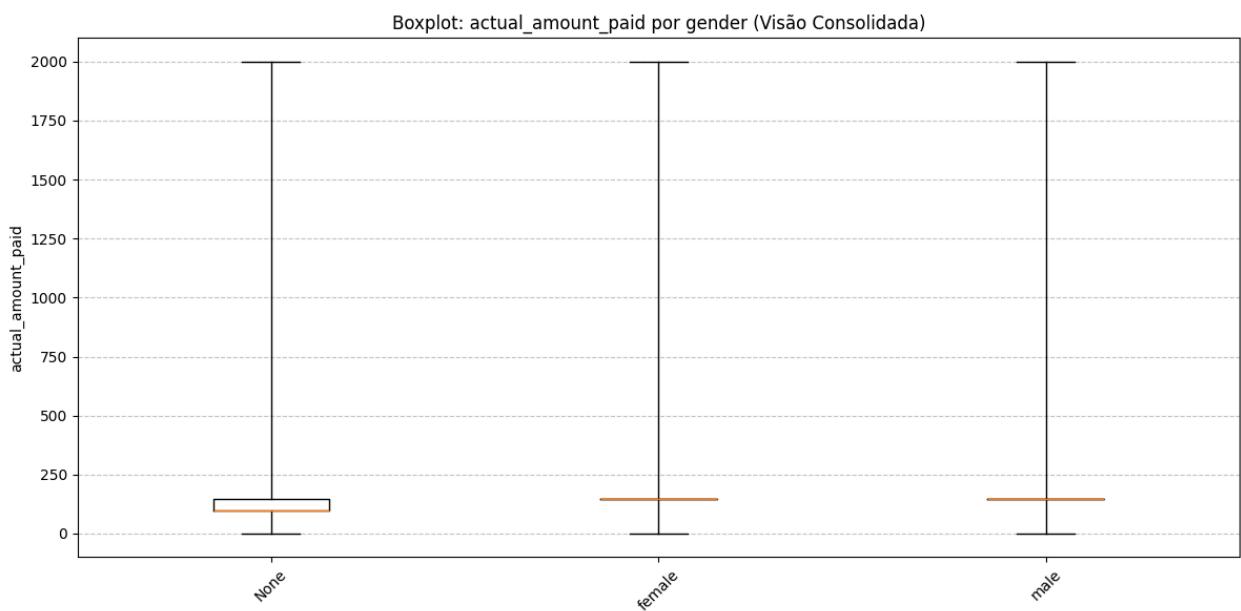
```
In [21]: calcular_distribuicao(df_atv_members_transactions, ["gender"], n_show=25)
```

gender	total	pct_total
NULL	5043795	53.71
male	2302299	24.52
female	2044389	21.77

```
Out[21]: DataFrame[gender: string, total: bigint, pct_total: double]
```

```
In [22]: plot_boxplot(df_atv_members_transactions, ["gender"], "actual_amount_paid", agrupar_por_safra=False)
```

Processando estatísticas para: gender...



```
In [23]: plot_boxplot(df_atv_members_transactions, ["gender"], "payment_plan_days", agrupar_por_safra=False)
```

Processando estatísticas para: gender...



Nao existe correlacao com genero + actual_amount_paid/payment_plan_days

city

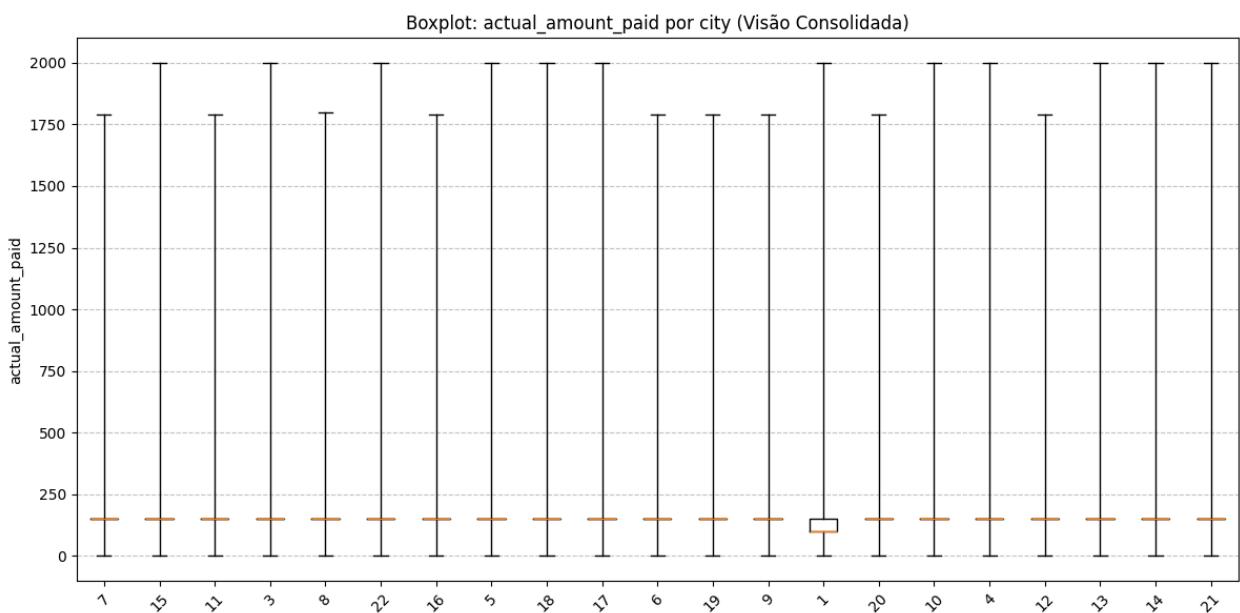
```
In [29]: calcular_distribuicao(df_atv_members_transactions, ["city"], n_show=25)
```

city	total	pct_total
1	4729450	50.36
13	1083693	11.54
5	797083	8.49
4	535788	5.71
15	461228	4.91
22	459465	4.89
6	280901	2.99
14	220515	2.35
12	129680	1.38
9	123634	1.32
11	94952	1.01
18	86843	0.92
8	86282	0.92
10	72664	0.77
17	61047	0.65
21	59808	0.64
3	57723	0.61
7	29135	0.31
16	11059	0.12
20	7894	0.08
19	1639	0.02

```
Out[29]: DataFrame[city: string, total: bigint, pct_total: double]
```

```
In [30]: plot_boxplot(df_atv_members_transactions, ["city"], "actual_amount_paid", agrupar_por_safra=False)
```

Processando estatísticas para: city...



```
In [24]: df_atv_members_transactions = df_atv_members_transactions.withColumn("flag_city_1", F.when(F.col("city") == 1, 1).otherwise(0))
```

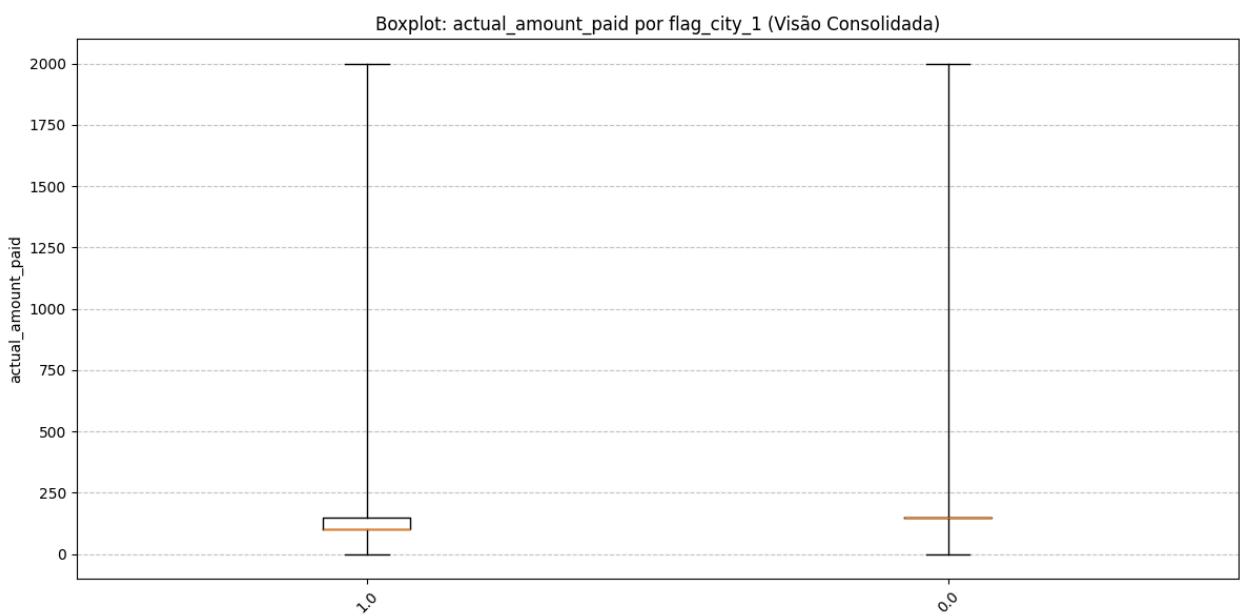
```
In [25]: calcular_distribuicao(df_atv_members_transactions, ["flag_city_1"], n_show=25)
```

flag_city_1	total	pct_total
1	4729450	50.36
0	4661033	49.64

Out[25]: DataFrame[flag_city_1: int, total: bigint, pct_total: double]

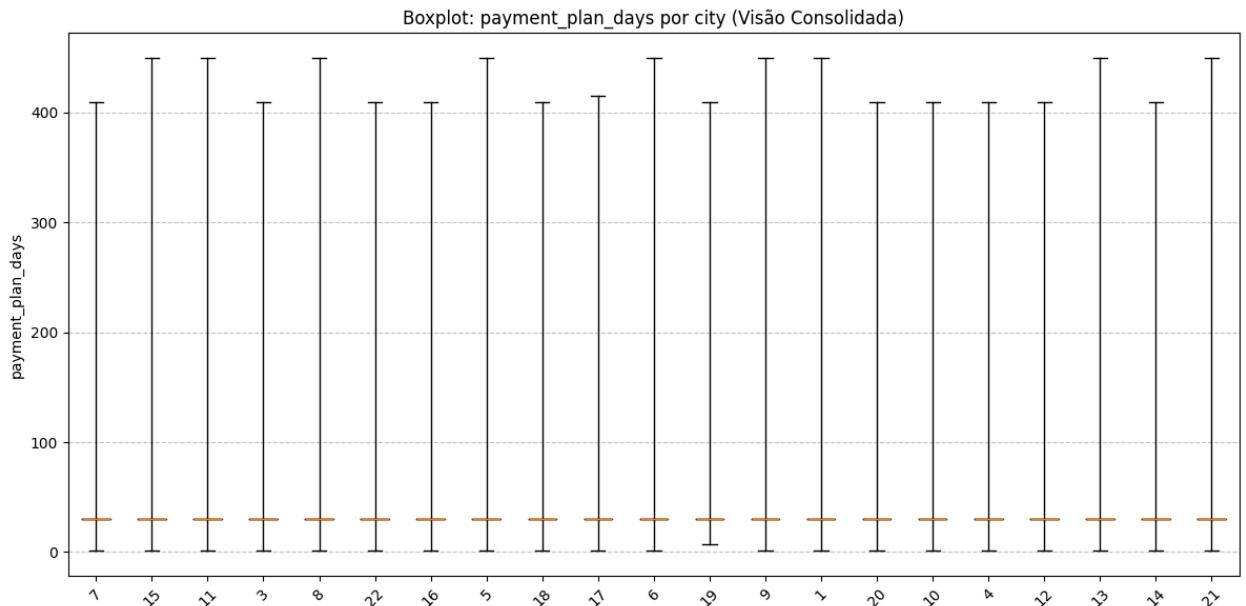
```
In [26]: plot_boxplot(df_atv_members_transactions, ["flag_city_1"], "actual_amount_paid", agrupar_por_safra=False)
```

Processando estatísticas para: flag_city_1...



```
In [31]: plot_boxplot(df_atv_members_transactions, ["city"], "payment_plan_days", agrupar_por_safra=False)
```

Processando estatísticas para: city...



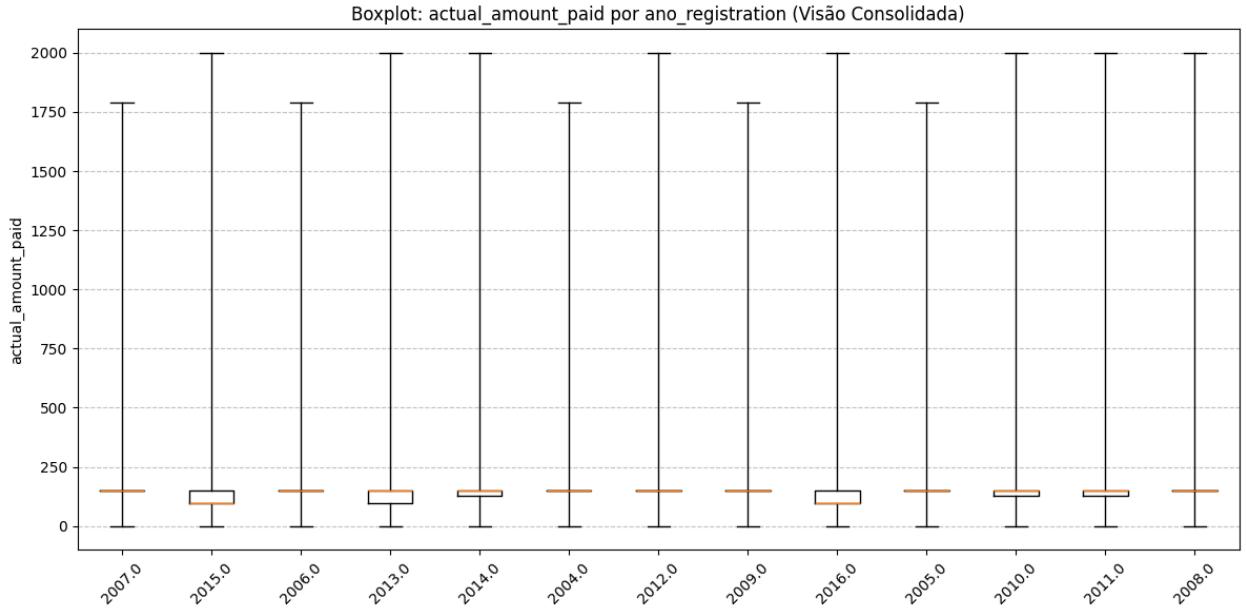
Nao existe correlacao com cidade + actual_amount_paid/payment_plan_days

```
#### registration_init_time
```

```
In [32]: df_atv_members_transactions = df_atv_members_transactions.withColumn("ano_registration", F.year(F.to_date(F.col("registration_init_time").cast("string"), "yyyyMMdd"))))
```

```
In [33]: plot_boxplot(df_atv_members_transactions, ["ano_registration"], "actual_amount_paid", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration...



```
In [34]: plot_boxplot(df_atv_members_transactions, ["ano_registration"], "payment_plan_days", agrupar_por_safra=False)
```

Processando estatísticas para: ano_registration...



Nao existe correlacao com ano de registro + actual_amount_paid/payment_plan_days

4.4.3. tabela logs + transactions

```
In [57]: integridade_entre_bases(df_logs, df_transactions, "Logs x Transações")
```

--- Integridade: Logs x Transações ---
Registros na base: 26758971 | Registros correspondentes: 16160111
Taxa de Perda: 39.61%

```
In [13]: df_logs_transactions = df_logs.join(df_transactions, on=["msno", "safra"], how="full_outer")
```

```
In [16]: df_logs.transactions.show(30, truncate=False)
```

msno	safra	num_25	num_50	num_75	num_985	num_100	num_unq	total_secs
payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	
+++snpr7pmobhLKUgSHTV/mpkqgBT0tQJ0zQj6qKrqc=	201509	375.0	73.0	46.0	43.0	631.0	1041.0	180656.804
30	149	149	1	20150926		20151026	0	41
++tnuKxIMqc8AGBaCielES3l1wnQ9tz30wrI0zTL6Y=	201501	3.0	0.0	0.0	0.0	5.0	5.0	1273.048
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++0+IdHga8fCSioOvpU8K7y4Asw8AveIApVH2r9q9yY=	201609	766.0	109.0	71.0	86.0	821.0	1740.0	249185.2100000008
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++2ev7dAQkJqmR/yiSGZL5cKMBzRZpBEncdzXjLWPY8=	201611	18.0	8.0	8.0	11.0	191.0	134.0	48053.34
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++2wdWRV3Thy9HzYRjtKtx1Ns55oDiDc7arR1guypc=	201510	NULL	NULL	NULL	NULL	NULL	NULL	NULL
30	149	0	1		20151008		20151015	1
++3A6JMzYJeron30GTcDostFxoAl8rTBuB2M8GeVdNu=	201506	27.0	10.0	4.0	2.0	116.0	150.0	29055.414
30	149	119	1		20150611		20150711	0
++3A6JMzYJeron30GTcDostFxoAl8rTBuB2M8GeVdNu=	201509	25.0	7.0	6.0	8.0	505.0	391.0	128267.3740000001
30	149	119	1		20150911		20151011	0
++3Z+w80PnpbHYfrKwqRKN1bF83XEbxjdYUohGdHzg=	201511	90.0	40.0	22.0	18.0	397.0	365.0	105767.1060000001
30	129	129	1		20151103		20151204	0
++4cUL0b9Cfw8cj0A/wfSxQc4k4fcVtWcLqk2U0dpKs=	201702	38.0	18.0	11.0	27.0	2432.0	2403.0	626162.8389999999
30	99	99	1		20170226		20170327	0
++4yteQFM9k0Gjq5fYL02l14u32iEAgTVHXePKB8zcM=	201607	9.0	5.0	2.0	1.0	7.0	18.0	2995.232
7	0	0	0		20160713		20160718	0
++4yteQFM9k0Gjq5fYL02l14u32iEAgTVHXePKB8zcM=	201609	4.0	1.0	0.0	1.0	1.0	4.0	640.198
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++5HPICzApqAq9mYdGB/mdlke0MbubM8yUKL0mrPt4xU=	201504	0.0	2.0	0.0	1.0	0.0	3.0	389.061
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++5HPICzApqAq9mYdGB/mdlke0MbubM8yUKL0mrPt4xU=	201506	NULL	NULL	NULL	NULL	NULL	NULL	35
7	0	0	0		20150610		20150615	0
++5nB0VuUuyj9x1ngqy30KUrmQXWZk05QtB9FYXWTok=	201604	122.0	81.0	22.0	24.0	2444.0	376.0	566058.478
30	180	180	0		20160412		20160512	0
++7IULiyKbNc8jllqhRuyKzjX1J4mPF4tsudFCJfv4k=	201606	91.0	23.0	24.0	33.0	5074.0	3504.0	1188302.5849999997
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++9G1StDXR8UJPes+rZhzhKGJtQqKtZnfws3YDzbZim=	201504	2.0	3.0	0.0	0.0	0.0	4.0	354.695
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++9KNLT3rbAF1G8a2H0qKAdmnWJM2j+z/ssAWVLNT8k=	201603	0.0	0.0	0.0	0.0	1.0	1.0	302.524
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++917+wGOZ96gNp0TDxxHAyd01XYE0ciuqWFTxA6zZI=	201606	243.0	63.0	39.0	36.0	1147.0	1173.0	291746.050000001
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++AH7m/EQ4iKe6wSlf0/xAJx50p+fCeTyF90GoE9Pg=	201609	32.0	8.0	2.0	1.0	69.0	97.0	18282.970999999998
30	149	149	0		20160924		20161024	0
++BBU1qnsavj2eC5j5G66dTda9ntLhRiz+pidG73im5s=	201509	33.0	3.0	2.0	3.0	94.0	94.0	25834.870000000003
30	149	149	1		20150930		20151114	0
++BBU1qnsavj2eC5j5G66dTda9ntLhRiz+pidG73im5s=	201607	70.0	34.0	19.0	22.0	1329.0	1264.0	331740.639
30	149	149	1		20160731		20160914	0
++BW1PjYU50Zi3n3+IDLiu+d1IL1VE/GLx6p64TDs6u=	201504	378.0	59.0	48.0	27.0	526.0	689.0	145620.132
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++CRI8Qf5u04/SCpq8+Z3KAiSwEji91XMzDHa3v+a0=	201505	5.0	0.0	0.0	0.0	0.0	5.0	111.934
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++CSmvB41dS00TeDSMly9vH0M1oQJN0XXyM7sYLkoTc=	201612	NULL	NULL	NULL	NULL	NULL	NULL	NULL
30	99	99	1		20161228		20161228	1
++EJp/B0Rn9pb2Bj6VHj9zCrAIInD0KTXMnvRz100Aj4=	201609	7.0	0.0	0.0	0.0	9.0	12.0	1540.21
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++FA1HTRYiSx0jcr+4SbBqd60c8XS/y046DDVJ3eZ6E=	201508	NULL	NULL	NULL	NULL	NULL	NULL	NULL
30	149	149	1		20150817		20150917	0
++GWCJ/8c4RVSUmn5mrt3VFGLufmYIGVJ/SHLWPACw=	201610	131.0	46.0	42.0	46.0	2231.0	2294.0	541657.87
30	129	129	1		20161027		20161128	0
++I1wsXaVBTTZE3gwz05qBwlcmcSinxWe2pLsJ8hS4=	201605	18.0	8.0	0.0	0.0	138.0	146.0	37708.735
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++IrpeciSQ6NWOp78CLVSLjCJvWbecNHnYzvOrxFape=	201612	112.0	31.0	18.0	17.0	1698.0	1328.0	429325.543
NULL	NULL	NULL		NULL	NULL	NULL	NULL	NULL
++Jgz/7glh40FgjboBC9t1zIrRCogvp8P7wYf2jpsir4=	201603	28.0	17.0	7.0	7.0	134.0	188.0	39185.99
30	149	149	1		20160310		20160409	0

```
s filter(F col("total_secs") isNull()) show(10, truncate=False)
```

```
+---+-----+-----+-----+-----+-----+-----+-----+
|msno|safra|num_25|num_50|num_75|num_985|num_100|num_unq|total_secs|
+---+-----+-----+-----+-----+-----+-----+-----+
```

```
In [18]: df_transactions.filter(F.col("is_auto_renew").isNull()).show(10, truncate=False)
```

msno	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	safra

```
In [30]: df_logs_transactions = (df_logs_transactions
    .withColumn("flag_logs", F.when(F.col("total_secs").isNotNull(), 1).otherwise(0))
    .withColumn("flag_transactions", F.when(F.col("is_auto_renew").isNotNull(), 1).otherwise(0))
)
```

```
In [31]: df_logs_transactions.show(5, truncate=False)
```

msno	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	flag_logs	flag_transactions
++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc=	201509	375.0	73.0	46.0	43.0	631.0	1041.0	180656.804	41	1
30	149	149	1	20150926		20151026		0		1
1										
++tnuKxIMqc8AGBaCielES3l1wnQ9tz30wrI0zTL6Y=	201501	3.0	0.0	0.0	0.0	5.0	5.0	1273.048		NULL
NULL	NULL	NULL	NULL	NULL		NULL		NULL		1
0										
++0+IdHga8fCSio0VpU8K7y4Asw8AveIApVH2r9q9yY=	201609	766.0	109.0	71.0	86.0	821.0	1740.0	249185.2100000008	NULL	
NULL	NULL	NULL	NULL	NULL		NULL		NULL		1
0										
++2ev7dAQkJqmR/yiSGZL5cKMBzRzpBEncdzXjLPy8=	201611	18.0	8.0	8.0	11.0	191.0	134.0	48053.34		NULL
NULL	NULL	NULL	NULL	NULL		NULL		NULL		1
0										
++2wdWRV3Thy9HZyRJtKx1Nsa55oDiDc7arR1guiypc=	201510	NULL	NULL	NULL	NULL	NULL	NULL	NULL	1	41
30	149	0	1	20151008		20151015		1		0
1										
only showing top 5 rows										

```
In [32]: df_logs_transactions.groupBy("flag_logs", "flag_transactions").agg(
    F.count("msno").alias("total_registros")
).orderBy("flag_logs", "flag_transactions").show(3, truncate=False)
```

flag_logs	flag_transactions	total_registros
0	1	4552114
1	0	10598860
1	1	16160111

```
In [33]: calcular_distribuicao(df_logs_transactions, ["flag_logs", "flag_transactions"], agrupar_por_safra=False)
```

flag_logs	flag_transactions	total	pct_total
1	1	16160111	51.61
1	0	10598860	33.85
0	1	4552114	14.54

Out[33]: DataFrame[flag_logs: int, flag_transactions: int, total: bigint, pct_total: double]

In [35]: calcular_distribuicao(df_logs_transactions, ["flag_logs", "flag_transactions"], n_show=100, agrupar_por_safra=True)

safra	flag_logs	flag_transactions	total	pct_safra	pct_total
201501 1	0	483889 46.86	1.55		
201501 1	1	453900 43.95	1.45		
201501 0	1	94892 9.19	0.3		
201502 1	0	479819 46.81	1.53		
201502 1	1	453221 44.21	1.45		
201502 0	1	92082 8.98	0.29		
201503 1	1	521829 49.73	1.67		
201503 1	0	422910 40.3	1.35		
201503 0	1	104659 9.97	0.33		
201504 1	1	471418 45.63	1.51		
201504 1	0	468512 45.35	1.5		
201504 0	1	93164 9.02	0.3		
201505 1	1	477906 46.95	1.53		
201505 1	0	446310 43.85	1.43		
201505 0	1	93646 9.2	0.3		
201506 1	1	610824 56.46	1.95		
201506 1	0	306038 28.29	0.98		
201506 0	1	164913 15.24	0.53		
201507 1	1	545247 54.99	1.74		
201507 1	0	326244 32.9	1.04		
201507 0	1	120033 12.11	0.38		
201508 1	1	570251 54.01	1.82		
201508 1	0	349878 33.14	1.12		
201508 0	1	135724 12.85	0.43		
201509 1	1	570163 54.42	1.82		
201509 1	0	333031 31.79	1.06		
201509 0	1	144447 13.79	0.46		
201510 1	1	523743 44.78	1.67		
201510 1	0	489210 41.82	1.56		
201510 0	1	156722 13.4	0.5		
201511 1	1	624554 50.46	1.99		
201511 1	0	417421 33.72	1.33		
201511 0	1	195791 15.82	0.63		
201512 1	1	647353 51.66	2.07		
201512 1	0	391918 31.28	1.25		
201512 0	1	213754 17.06	0.68		
201601 1	1	647180 50.32	2.07		
201601 1	0	429532 33.39	1.37		
201601 0	1	209536 16.29	0.67		
201602 1	1	598886 48.51	1.91		
201602 1	0	442362 35.83	1.41		
201602 0	1	193414 15.67	0.62		
201603 1	1	593959 48.27	1.9		
201603 1	0	454982 36.98	1.45		
201603 0	1	181510 14.75	0.58		
201604 1	1	595508 48.77	1.9		
201604 1	0	446898 36.6	1.43		
201604 0	1	178661 14.63	0.57		
201605 1	1	605970 49.09	1.94		
201605 1	0	450521 36.49	1.44		
201605 0	1	177986 14.42	0.57		
201606 1	1	620370 49.02	1.98		
201606 1	0	460811 36.41	1.47		
201606 0	1	184359 14.57	0.59		
201607 1	1	721393 55.26	2.3		
201607 1	0	381414 29.22	1.22		
201607 0	1	202639 15.52	0.65		
201608 1	1	748687 56.68	2.39		
201608 1	0	354391 26.83	1.13		
201608 0	1	217763 16.49	0.7		
201609 1	1	759464 56.86	2.43		
201609 1	0	353137 26.44	1.13		
201609 0	1	223176 16.71	0.71		
201610 1	1	784324 56.48	2.5		
201610 1	0	354765 25.55	1.13		
201610 0	1	249574 17.97	0.8		
201611 1	1	810671 55.25	2.59		
201611 1	0	372417 25.38	1.19		
201611 0	1	284270 19.37	0.91		
201612 1	1	757056 56.2	2.42		
201612 1	0	378517 28.1	1.21		
201612 0	1	211491 15.7	0.68		
201701 1	1	772223 57.1	2.47		
201701 1	0	363780 26.9	1.16		
201701 0	1	216353 16.0	0.69		
201702 1	1	674011 50.84	2.15		
201702 1	0	440153 33.2	1.41		
201702 0	1	211555 15.96	0.68		

```
Out[35]: DataFrame[safra: int, flag_logs: int, flag_transactions: int, total: bigint, pct_safra: double, pct_total: double]
```

4.4.4. tabela members + logs + transactions

```
In [27]: df_all = df_members.join(df_logs, on=["msno", "safra"], how="full_outer")\
    .join(df_transactions, on=["msno", "safra"], how="full_outer")
```

```
In [14]: df_all.show(5, truncate=False)
```

msno	safra	registration_init_time	city	bd	gender	registered_via	is_ativo	num_25	num_50	num_75	num_985	num_100	num_unq	total_secs	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel		
+--dz9ZCWE2HB/47pJU82NjXQzQuZDx1Wm50YSk/KK=	201606 20160228	1	0	NULL	4	0	NULL																	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
+++hVY1rZox/33YtvDgmKA2Frg/2qhkz12B9y1Cvh8o=	201608 20140608	1	0	NULL	7	0	NULL																	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
++snpr7pmobhLKUgSHTv/mpkqgbT0tQJ0zQj6qKrqc=	201509 NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
73.0	46.0	43.0	631.0	1041.0	180656.804	41	30																	
1																								
20150926		20151026	0																					
++/gTmVgKUbNFMsTiriZdwV1uZIrLxCUiEWN0fEU6BM=	201611 20151111	1	0	NULL	7	0	NULL																	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
++0+IdHga8fcSi0OvpU8K7y4Asw8AveIApVH2r9q9yY=	201609 20101019	13	32	male	9	0	NULL																	
109.0	71.0	86.0	821.0	1740.0	249185.2100000008	NULL	NULL																	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL																	
only showing top 5 rows																								

```
In [15]: df_members.filter(F.col("is_ativo").isNull()).show(1, truncate=False)
```

msno	safra	registration_init_time	city	bd	gender	registered_via	is_ativo
+--	+--	+--	+--	+--	+--	+--	+--
+--	+--	+--	+--	+--	+--	+--	+--
+--	+--	+--	+--	+--	+--	+--	+--

```
In [16]: df_all = (df_all
    .withColumn("flag_logs", F.when(F.col("total_secs").isNotNull(), 1).otherwise(0))
    .withColumn("flag_transactions", F.when(F.col("is_auto_renew").isNotNull(), 1).otherwise(0))
    .withColumn("flag_members", F.when(F.col("is_ativo").isNotNull(), 1).otherwise(0))
)
```

```
In [20]: calcular_distribuicao(df_all, ["flag_logs", "flag_transactions", "flag_members"], agrupar_por_safra=False)
```

flag_logs	flag_transactions	flag_members	total	pct_total
+-----+	+-----+	+-----+	+-----+	+-----+
0	0	1	49616104	61.31
1	1	1	8242914	10.19
1	1	0	7917197	9.78
1	0	0	5738201	7.09
1	0	1	4860659	6.01
0	1	0	3404545	4.21
0	1	1	1147569	1.42
+-----+	+-----+	+-----+	+-----+	+-----+

```
Out[20]: DataFrame[flag_logs: int, flag_transactions: int, flag_members: int, total: bigint, pct_total: double]
```

```
In [21]: df_all = (df_all
    .withColumn("flag_all", F.when(
        (F.col("flag_logs") == 1) & (F.col("flag_transactions") == 1) & (F.col("flag_members") == 1), 1).otherwise(0)))
```

relacao entre is_ativo e is_cancel

```
In [ ]: calcular_distribuicao(df_members, )
```

```
In [24]: calcular_distribuicao(df_all.filter(F.col("flag_all").isin(1)), ["is_cancel", "is_ativo"], agrupar_por_safra=False)
```

is_cancel	is_ativo	total	pct_total
0	1	8085381	98.09
1	1	157533	1.91

```
Out[24]: DataFrame[is_cancel: string, is_ativo: int, total: bigint, pct_total: double]
```

verificando a quantidade por safra de registros "completos"

```
In [26]: calcular_distribuicao(df_all, ["flag_all"], n_show=60, agrupar_por_safra=True)
```

safra	flag_all	total	pct_safra	pct_total
201501	0	1032681	100.0	1.28
201502	0	1025122	100.0	1.27
201503	0	1049398	100.0	1.3
201504	0	1033094	100.0	1.28
201505	0	1017862	100.0	1.26
201506	0	1081775	100.0	1.34
201507	0	991524	100.0	1.23
201508	0	1055853	100.0	1.3
201509	0	1047641	100.0	1.29
201510	0	1169675	100.0	1.45
201511	0	1237766	100.0	1.53
201512	0	1253025	100.0	1.55
201601	0	3795596	85.43	4.69
201601	1	647110	14.57	0.8
201602	0	4018793	87.03	4.97
201602	1	598819	12.97	0.74
201603	0	4209629	87.64	5.2
201603	1	593901	12.36	0.73
201604	0	4384640	88.04	5.42
201604	1	595461	11.96	0.74
201605	0	4557809	88.27	5.63
201605	1	605933	11.73	0.75
201606	0	4733217	88.41	5.85
201606	1	620331	11.59	0.77
201607	0	4828773	87.0	5.97
201607	1	721355	13.0	0.89
201608	0	4981886	86.94	6.16
201608	1	748647	13.06	0.93
201609	0	5135218	87.12	6.35
201609	1	759425	12.88	0.94
201610	0	5286154	87.08	6.53
201610	1	784288	12.92	0.97
201611	0	5434358	87.02	6.72
201611	1	810612	12.98	1.0
201612	0	5644711	88.17	6.98
201612	1	757032	11.83	0.94
201701	0	1352356	100.0	1.67
201702	0	1325719	100.0	1.64

```
Out[26]: DataFrame[safra: string, flag_all: int, total: bigint, pct_safra: double, pct_total: double]
```

5. Dataprep - Começo camada _Silver_

5.1. Decisão: manter ou não os clientes que não possuem tanto informação de uso (logs) quanto de pagamentos (transactions)?

Justificativa Estratégica: Manutenção de Populações Heterogêneas na Base de Modelagem

Uma decisão crítica neste projeto foi a não-exclusão de usuários que não apresentam registros em uma das tabelas core (logs ou transactions) em uma determinada safra. Embora a tentação analítica seja focar no "usuário ideal" (aquele que paga e usa), a exclusão de perfis incompletos introduziria um viés de seleção severo, comprometendo a utilidade do modelo em produção.

A manutenção desses grupos fundamenta-se em três pilares:

* Captura de Extremos de Margem: Usuários com logs mas sem transações representam o maior risco operacional (custo sem receita), enquanto usuários com transações mas sem logs representam a maior margem líquida (receita sem custo). Ignorá-los resultaria em um modelo incapaz de prever os cenários de maior impacto financeiro para o negócio;

* Poder Preditivo da Mudança de Estado: O valor real de um modelo de propensão e rentabilidade não está em identificar quem paga hoje, mas em prever quem deixará de pagar ou começará a consumir excessivamente no futuro (M+1). Manter o histórico de "silêncio" (ausência de logs ou transações) permite que o modelo aprenda padrões de inatividade que precedem o churn ou a inadimplência;

* Robustez em Produção: Um modelo treinado apenas em usuários ativos falha ao encontrar usuários intermitentes no mundo real. Ao tratar a ausência de dados como uma informação em si (via flags e imputação semântica), garantimos que o modelo seja resiliente a anomalias e variações do ciclo de vida do cliente.

Portanto, em vez de filtrar a base, optamos por enriquecer o espaço de features com interações e tendências temporais, permitindo que os algoritmos de aprendizado (especialmente os baseados em árvores) segmentem essas subpopulações internamente de forma automática e otimizada.

5.2. Agregacoes

5.2.1. Agregação Preventiva de Transações por Cliente-Mês - Contexto

Embora a base histórica não apresente múltiplas transações por cliente no mesmo mês (msno, safra), a implementação de uma camada explícita de agregação é uma decisão de engenharia voltada para robustez em produção.

Em ambientes reais, eventos como _upgrades_, _downgrades_, reprocessamentos, falhas de cobrança e retentativas podem gerar múltiplas linhas transacionais para o mesmo cliente no mesmo período. Sem tratamento adequado, isso pode:

- * Duplicar registros na _spine_ de modelagem;
- * Gerar inconsistências em _joins_;
- * Enviesar _features_ temporais;
- * Quebrar _pipelines_ de inferência.

Por isso, antes do _feature engineering_, optei por consolidar as transações no nível (msno, safra), aplicando regras de agregação específicas para cada dataframe e por tipo de variável, priorizando o último estado observado para atributos categóricos e de plano, e acumulação para valores monetários.

A ideia é que, uma vez em produção, o código base sempre execute uma verificação de duplicatas. Ao perceber, são chamadas as funções de agregação para cada tabela e variável. Caso não haja duplicatas, o processo roda normalmente.

5.2.2. Agregação Preventiva de Transações por Cliente-Mês - Execução

```
In [118]: df_logs_agg = agrregar_logs(df_logs)
          ✓ LOGS: Base já está no nível (msno, safra). Nenhuma agregação necessária.

In [119]: df_transactions_agg = agrregar_transactions(df_transactions)
          ✓ TRANSACTIONS: Base já está no nível (msno, safra). Nenhuma agregação necessária.

In [120]: df_members_agg = agrregar_members(df_members)
          ✓ MEMBERS: Base já está no nível (msno, safra). Nenhuma agregação necessária.
```

5.3. Tratamento de nulos e outliers

5.3.1. Logs

Afim de tratar os outliers encontrados na tabela em questão, aplicaremos a *Winsorização*: técnica que consiste em "limitar" os valores extremos (outliers) inferiores e superiores, substituindo-os pelos valores de um determinado percentual (ex: percentis P1 e P99). Durante a análise exploratória de dados, tomando a variável `num_25` como exemplo, observamos que enquanto a média é de ~95 e o percentil 99,5% está em 975, o valor máximo atinge impressionantes 111.864. Essa discrepância indica uma cauda extremamente longa à direita.

Valores extremos como este podem enviesar modelos que se baseiam em médias e variâncias (como Regressão Linear, PCA ou K-means). A Winsorização mitiga esse efeito sem a perda de linhas que ocorreria em um "trimming" (remoção direta), mantendo o tamanho da amostra para o treinamento. O valor máximo de 111 mil músicas parciais em um período de log provavelmente não reflete o comportamento de um usuário humano, sendo possivelmente fruto de erros de medição, bots ou contas compartilhadas. O "capping" via Winsorização traz esses registros para um limite estatisticamente aceitável.

2. Justificativa de Negócio

* Foco no Usuário Real: Para fins de recomendação ou predição de churn, o negócio está interessado no comportamento do consumidor típico. Usuários com volumes astronômicos (outliers) distorcem os KPIs (indicadores-chave de desempenho), levando a interpretações errôneas sobre o engajamento médio da base.

* Consistência em Ambientes de Produção: Aplicar uma função padronizada de Winsorização (como a `aplicar_winsorizacao` definida no código) permite que o processo de limpeza de dados seja automatizado e replicável em diferentes safras de dados, garantindo que o modelo de ML não "quebre" ao encontrar novos valores extremos no futuro.

* Integridade da Base: Como cerca de 56% dos usuários aparecem apenas uma vez na base de logs, é crucial que os registros desses usuários — que já possuem pouca informação histórica — não sejam simplesmente deletados por serem outliers, mas sim ajustados para não prejudicar a performance preditiva do modelo.

```
In [ ]: df_logs_fix = aplicar_winsorizacao(df_logs_agg, ["num_25", "num_50", "num_75", "num_985", "num_100", "num_unq", "total_secs"])
```

```
Coluna num_25: Limite Inferior=0.0, Limite Superior=746.0
Coluna num_50: Limite Inferior=0.0, Limite Superior=174.0
Coluna num_75: Limite Inferior=0.0, Limite Superior=101.0
Coluna num_985: Limite Inferior=0.0, Limite Superior=122.0
Coluna num_100: Limite Inferior=0.0, Limite Superior=3487.0
Coluna num_unq: Limite Inferior=1.0, Limite Superior=2819.0
Coluna total_secs: Limite Inferior=28.11, Limite Superior=881683.002
```

```
#### Salvar base
```

```
In [25]: df_logs_fix.coalesce(4).write.mode("overwrite").parquet("C:\\\\Users\\\\Gustavo\\\\Downloads\\\\datamaster\\\\dados\\\\silver\\\\df_logs_fix.parquet")
```

5.3.2. Transactions

```
In [ ]: df_transactions_fix = (df_transactions_agg
    .withColumn("payment_plan_days", F.col("payment_plan_days").cast("int"))
    .withColumn("payment_method_id", F.col("payment_method_id").cast("int"))
    .withColumn("plan_list_price", F.col("plan_list_price").cast("float"))
    .withColumn("actual_amount_paid", F.col("actual_amount_paid").cast("float"))
    .withColumn("is_auto_renew", F.col("is_auto_renew").cast("int"))
    .withColumn("transaction_date", F.to_date(F.col("transaction_date"), "yyyyMMdd"))
    .withColumn("membership_expire_date", F.to_date(F.col("membership_expire_date"), "yyyyMMdd"))
    .withColumn("is_cancel", F.col("is_cancel").cast("int"))
)
```

```
In [38]: df_transactions_fix.show(5, truncate=False)
```

msno	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	safra
+++IZseRQiQS9aaSkH6cMYU6bGDcxUiAi/tH67sC5s= 38				410		1788.0	1788.0		0
2015-11-21	2017-01-04	0	201511						
++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc= 41				30		149.0	149.0		1
2015-05-26	2015-06-26	0	201505						
++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc= 41				30		149.0	149.0		1
2015-09-26	2015-10-26	0	201509						
++/9R3sX37CjxbY/AaGvbwr3QkwElKBCtSvVzhCBDDok= 41				30		149.0	149.0		1
2016-06-15	2016-07-15	0	201606						
++/Gw1B9K+X01B3hLTloeUK2QlCa2m+Bj8TrzGf7djI= 40				31		149.0	149.0		1
2015-01-13	2015-02-16	0	201501						

only showing top 5 rows

membership_expire_date

```
In [23]: df_transactions_fix = df_transactions_fix.withColumn("flag_expire_invalido", F.when(F.col("membership_expire_date") < "2015-01-01", 1).otherwise(0))
```

```
In [48]: calcular_distribuicao(df_transactions_fix, ["flag_expire_invalido"])
```

flag_expire_invalido	total	pct_total
0	20705003 99.97	
1	7222	0.03

```
Out[48]: DataFrame[flag_expire_invalido: int, total: bigint, pct_total: double]
```

```
In [26]: df_transactions_fix.coalesce(4).write.mode("overwrite").parquet("C:\\\\Users\\\\Gustavo\\\\Downloads\\\\datamaster\\\\dados\\\\silver\\\\df_transactions_fix.parquet")
```

5.3.3. Members

```
In [ ]: df_members_fix = (df_members_agg
    .withColumn("registration_init_time", F.to_date(F.col("registration_init_time"), "yyyyMMdd"))
    .withColumn("city", F.col("city").cast("int"))
    .withColumn("bd", F.col("bd").cast("int"))
    .withColumn("registered_via", F.col("registered_via").cast("int"))
    .withColumn("is_ativo", F.col("is_ativo").cast("int"))
)
```

```
In [18]: df_members_fix.show(5, truncate=False)
```

msno	safra	registration_init_time	city	bd	gender	registered_via	is_ativo
++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc= 201612 2014-09-27	1	0	NULL	7		1	1
++/AwGzubug3gT6J+0STBGMdWKxaM+UFZT18Tcmq4To= 201607 2015-03-22	1	0	NULL	9		0	0
++/Gw1B9K+X01B3hLTloeUK2QlCa2m+Bj8TrzGf7djI= 201601 2012-12-17	15	32	male	3		1	1
++02XbtviomSxcIBUHMOiJkjRxdtCTSfiVqlLdsr5lo= 201603 2013-11-12	14	21	male	7		0	0
++000Bq04sB/9ZcOS+pajpYL2Hin9jCqnc/8bKzKfUE= 201610 2014-10-21	5	33	male	3		0	0

only showing top 5 rows

bd (idade)

A variável de idade (bd) apresenta valores inválidos e inconsistentes, como zeros e idades irrealas. Esses casos devem ser tratados como informação ausente, e não como outliers estatísticos. Foi criada uma flag indicativa de idade inválida.

```
In [19]: df_members_fix = df_members_fix.withColumn("flag_idade_invalida",
    F.when(
        (F.col("bd") < 16) | # idade minima para assinatura segundo termos de uso
        (F.col("bd") > 80) | # idade maxima razoavel, mediante entendimento do desenvolvedor
        (F.col("bd").isNull()), 1)\.
    .otherwise(0))
```

```
In [38]: calcular_distribuicao(df_members_fix, ["flag_idade_invalida"], n_show=25)
```

flag_idade_invalida	total	pct_total
1	39794138	62.31
0	24073108	37.69

```
Out[38]: DataFrame[flag_idade_invalida: int, total: bigint, pct_total: double]
```

```
In [20]: df_members_fix = df_members_fix.withColumn("idade_clean",
    F.when(F.col("flag_idade_invalida") == 1, None).otherwise(F.col("bd"))))
```

```
In [40]: df_members_fix.select("idade_clean").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	idade_clean
count	24073108
mean	29.733778870597018
stddev	10.126336376192915
min	16
1%	16
5%	17
25%	22
50%	27
75%	35
95%	50
99.5%	65
max	80

```
#### gender
```

```
In [21]: df_members_fix = df_members_fix.withColumn("gender_clean",
    F.when(F.col("gender").isNull(), F.lit("unknown")).otherwise(F.col("gender"))))
```

```
In [42]: calcular_distribuicao(df_members_fix, ["gender_clean"], n_show=25)
```

gender_clean	total	pct_total
unknown	38210177	59.83
male	13075425	20.47
female	12581644	19.7

```
Out[42]: DataFrame[gender_clean: string, total: bigint, pct_total: double]
```

```
#### Salvar base
```

```
In [22]: df_members_fix.coalesce(4).write.mode("overwrite").parquet("C:\\\\Users\\\\Gustavo\\\\Downloads\\\\datamaster\\\\dados\\\\silver\\\\df_members_fix.parquet")
```

6. Definição da _Spine_

```
In [16]: silver_path = r"C:\\Users\\Gustavo\\Downloads\\datamaster\\dados\\silver"
```

```
In [19]: df_members_fix = spark.read.parquet(f"{silver_path}/df_members_fix.parquet")
```

```
In [31]: df_spine = df_members_fix.filter(F.col("is_ativo").isin(1)).select("msno", "safra").distinct()
```

```
In [33]: df_spine.count()
```

```
Out[33]: 11242865
```

```
In [32]: df_spine.show(5, truncate=False)
```

```
+-----+-----+
|msno |safra |
+-----+-----+
|+++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc=|201612|
|++Bks8kE9oclzxZM3hcWs+qzsxuoXFeIE1+7pxKBCQg=|201607|
|++D+fngRaZW1kQC5bvDwcoLYEp6Rn9LY1lxslM0TqM=|201611|
|++DcyRE+ZfLtlKGigvv9dv5EQ4KAhHhiN9LM9X+F1vw=|201612|
|++GVWCJ/8c4RVSUmn5mrt3VFGLUfmYIGVJ/SHLWPACw=|201602|
+-----+-----+
only showing top 5 rows
```

A construção da base de modelagem foi baseada em uma spine temporal explícita, definida por cliente ativo e safra, garantindo que a ausência de eventos em logs ou transações não implicasse exclusão de observações.

Clientes ativos foram identificados a partir da base de members.

```
In [34]: df_spine.write.mode("overwrite").partitionBy("safra").parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_spine")
```

7. Construção da variável de margem líquida mensal + _target_

7.1. Carregando e unindo bases

```
In [19]: silver_path = r"C:\Users\Gustavo\Downloads\datamaster\dados\silver"
```

```
In [20]: df_spine = spark.read.parquet(f"{silver_path}/df_spine")
```

```
In [21]: df_members_fix = spark.read.parquet(f"{silver_path}/df_members_fix.parquet")
```

```
In [22]: df_logs_fix = spark.read.parquet(f"{silver_path}/df_logs_fix.parquet")
```

```
In [23]: df_transactions_fix = spark.read.parquet(f"{silver_path}/df_transactions_fix.parquet")
```

```
In [24]: df_base = (df_spine
    .join(df_members_fix.filter(F.col("is_ativo").isin(1)), on=["msno", "safra"], how="left")
    .join(df_logs_fix, on=["msno", "safra"], how="left")
    .join(df_transactions_fix, on=["msno", "safra"], how="left")
)
```

```
In [50]: df_base.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: string (nullable = true)
 |-- registration_init_time: date (nullable = true)
 |-- city: integer (nullable = true)
 |-- bd: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- registered_via: integer (nullable = true)
 |-- is_ativo: integer (nullable = true)
 |-- flag_idade_invalida: integer (nullable = true)
 |-- idade_clean: integer (nullable = true)
 |-- gender_clean: string (nullable = true)
 |-- num_25: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- num_100: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_secs: double (nullable = true)
 |-- payment_method_id: integer (nullable = true)
 |-- payment_plan_days: integer (nullable = true)
 |-- plan_list_price: float (nullable = true)
 |-- actual_amount_paid: float (nullable = true)
 |-- is_auto_renew: integer (nullable = true)
 |-- transaction_date: date (nullable = true)
 |-- membership_expire_date: date (nullable = true)
 |-- is_cancel: integer (nullable = true)
 |-- flag_expire_invalido: integer (nullable = true)
 |-- margem_liquida_mensal: double (nullable = true)
 |-- target: double (nullable = true)
```

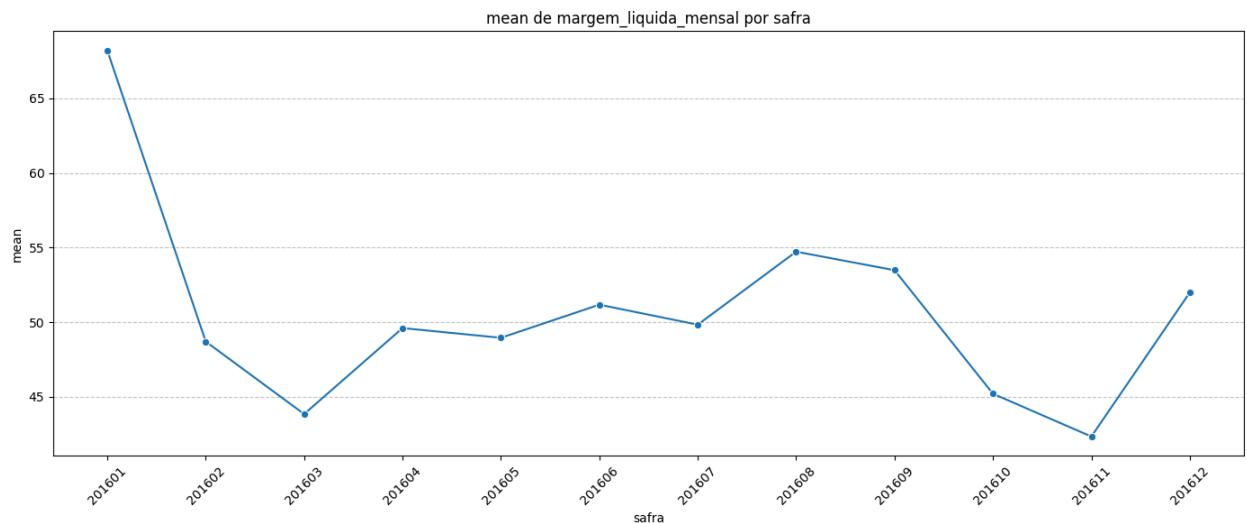
7.2. Margem líquida mensal

```
In [25]: df_base = df_base.withColumn("margem_liquida_mensal",
    F.when((F.col("actual_amount_paid").isNull() & (F.col("num_unq").isNull())), F.lit(-50))\
    .when(F.col("actual_amount_paid").isNull() & (F.col("num_unq").isNotNull()), (0 - (50 + (0.0051 * F.col("num_unq")) + (0.0001 * F.col("total_secs")))))\
    .when((F.col("actual_amount_paid").isNotNull() & (F.col("num_unq").isNull())), F.col("actual_amount_paid") - 50)\\
    .otherwise(F.col("actual_amount_paid") - (50 + (0.0051 * F.col("num_unq")) + (0.0001 * F.col("total_secs")))))
```

```
In [42]: df_base.select("margem_liquida_mensal").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

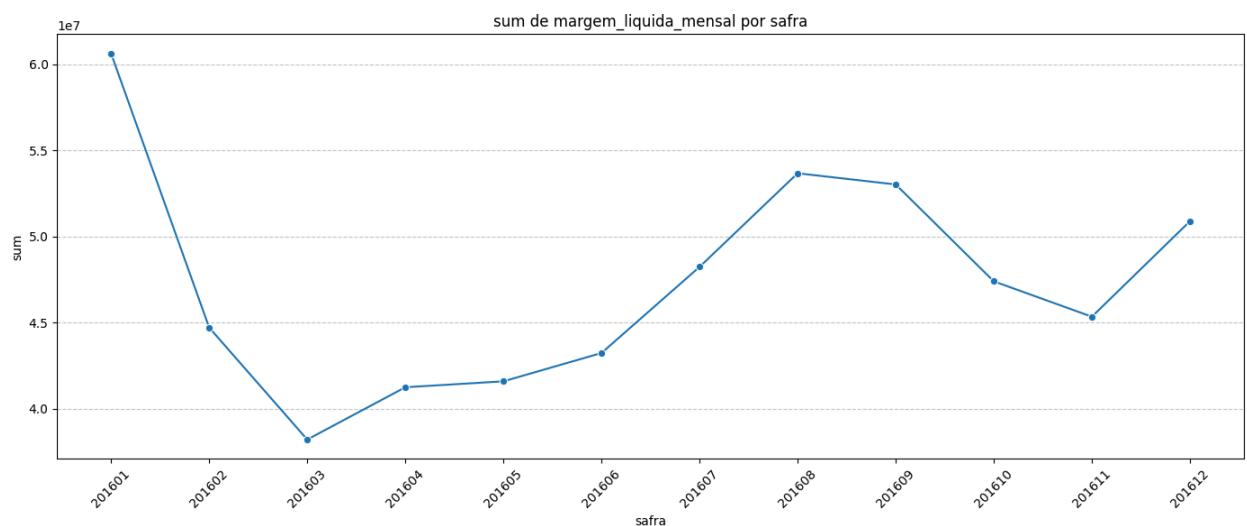
summary margem_liquida_mensal
count 11242865
mean 50.5382748232691
stddev 101.38291623295063
min -152.5452002
1% -106.9493839
5% -71.24760119999999
25% 29.82013469999997
50% 59.6337951
75% 90.9997278
95% 99.0
99.5% 807.8365229999999
max 1950.0

```
In [44]: plot_tendencia_temporal(df_base, col_valor="margem_liquida_mensal")
```



Nos melhores meses, um cliente ativo gera, em média, cerca de 50 reais de margem líquida naquele mês

```
In [45]: plot_tendencia_temporal(df_base, col_valor="margem_liquida_mensal", aggregation="sum")
```



`margem_liquida_mensal` representa o resultado econômico daquele mês, sendo consequência direta de comportamento, plano, uso e pagamento do cliente. A partir desse princípio, e de que existe ao mesmo tempo que as demais variáveis, entende-se que é boa para EDA, ou seja, permite tirar insights de sua relação com outras variáveis. Se usarmos a target para o EDA, cairíamos em leakage, prejudicando o pipeline de modelagem, dado que as conclusões obtidas seriam mais relacionadas a algo "suposto", mas não concreto.

7.3. Target (Margem Líquida M+1)

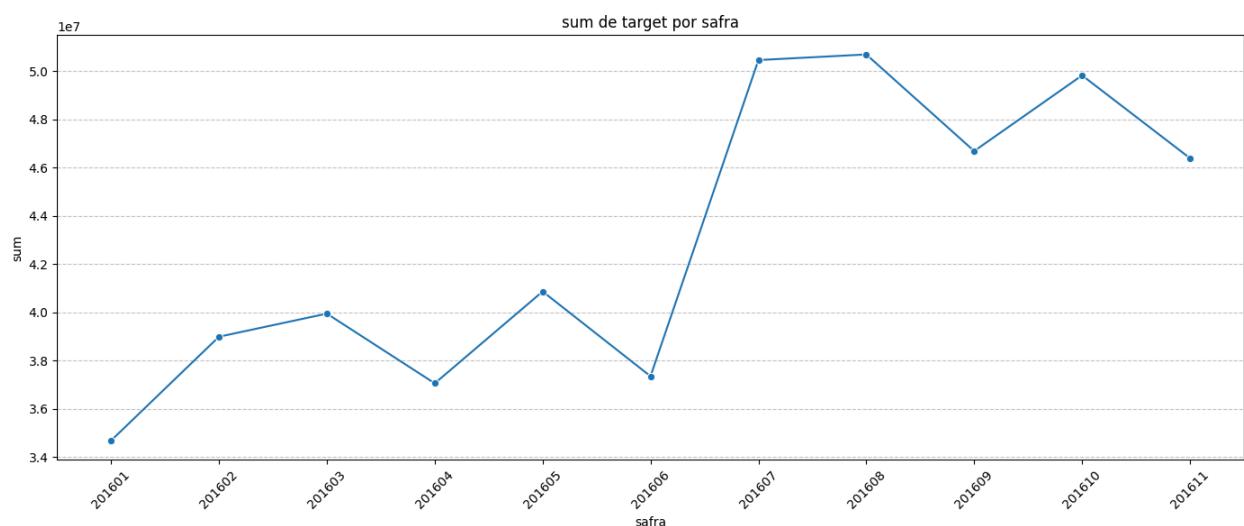
```
In [26]: w = Window.partitionBy("msno").orderBy("safra")
# Target: margem do mês seguinte
df_base = df_base.withColumn("target", F.lead("margem_liquida_mensal", 1).over(w))
```

```
In [48]: df_base.select("target").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	target
count	9678866
mean	48.86269394980939
stddev	81.7297910785321
min	-152.5452002
1%	-108.23865709999998
5%	-72.14227840000001
25%	32.62624239999995
50%	60.42658570000004
75%	90.8602606
95%	99.0
99.5%	461.2534063
max	1950.0

Plotando o lucro do mês seguinte, atribuído ao mês atual (grafico diferente dos de cima pelo shift temporal):

```
In [47]: plot_tendencia_temporal(df_base, col_valor="target", aggregation="sum")
```



7.3.1. Correlação da margem líquida (mês "n") com a target (margem líquida n+1) - Possibilidade para _feature engineering_

```
In [96]: # grau de correlacao de margem com target
df_base.select("target", "margem_liquida_mensal").corr("target", "margem_liquida_mensal")
```

```
Out[96]: 0.1513006611528001
```

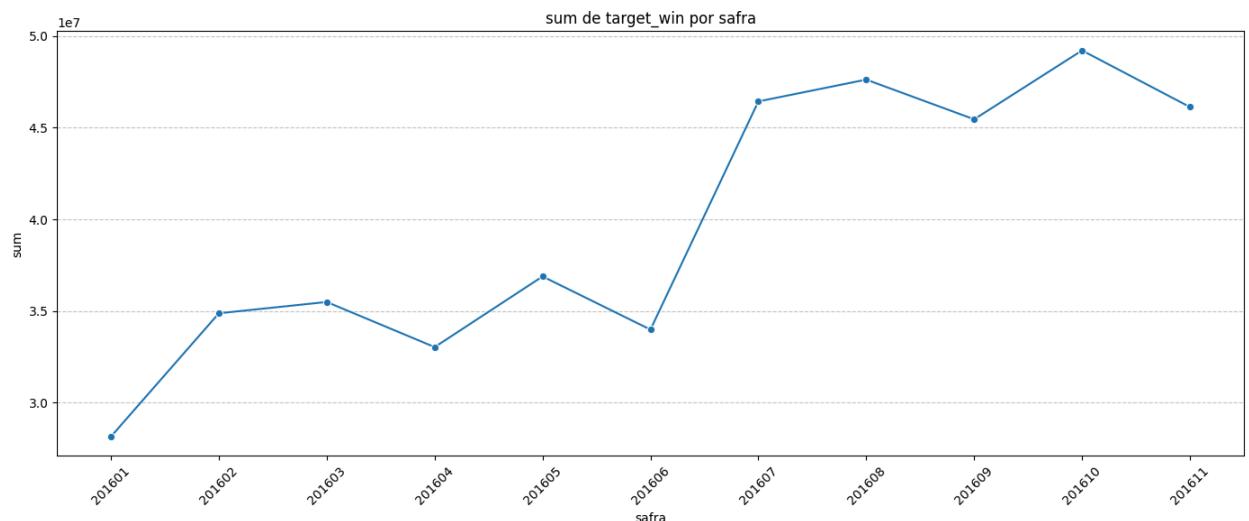
7.3.2. Tratamento de target - criação de target com _winsorization_

Para fins de ajuste linear (trazer maior confiança nos agrupamentos a partir da media, não somente da mediana), vou winsorizar a target e ver o comportamento em diferentes variáveis.

```
In [27]: df_base = aplicar_winsorizacao(df_base, ["target"])
```

```
Coluna target: Limite Inferior=-108.96492529999998, Limite Superior=128.953521
```

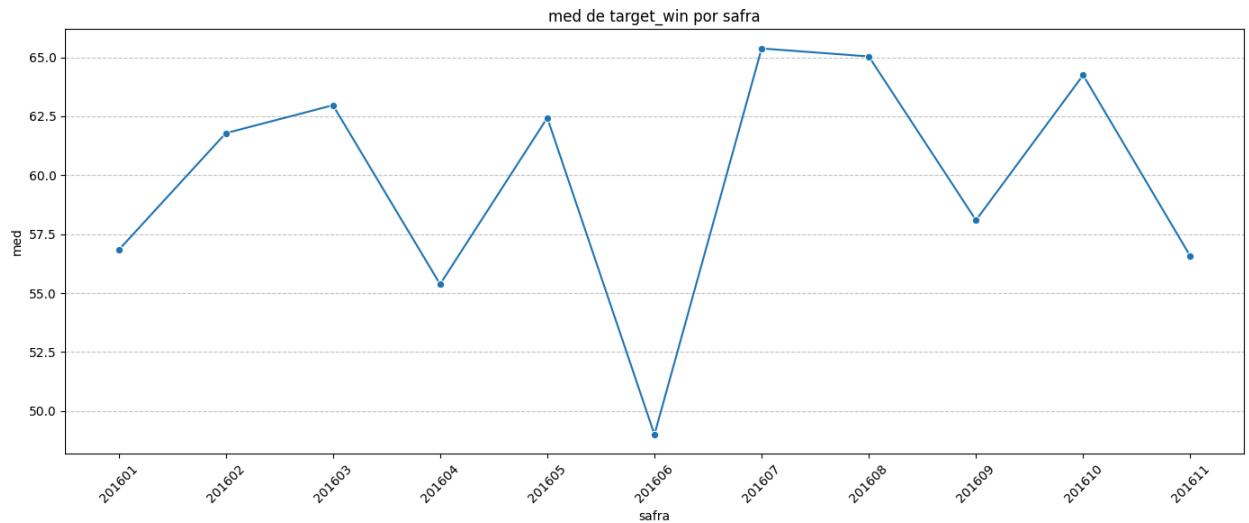
```
In [26]: plot_tendencia_temporal(df_base, col_valor="target_win", aggregation="sum")
```



```
In [27]: plot_tendencia_temporal(df_base, col_valor="target_win", aggregation="mean")
```



```
In [31]: plot_tendencia_temporal(df_base, col_valor="target_win", aggregation="med")
```



8. Relação das tabelas com _target_ - ideias principais para _feature engineering_

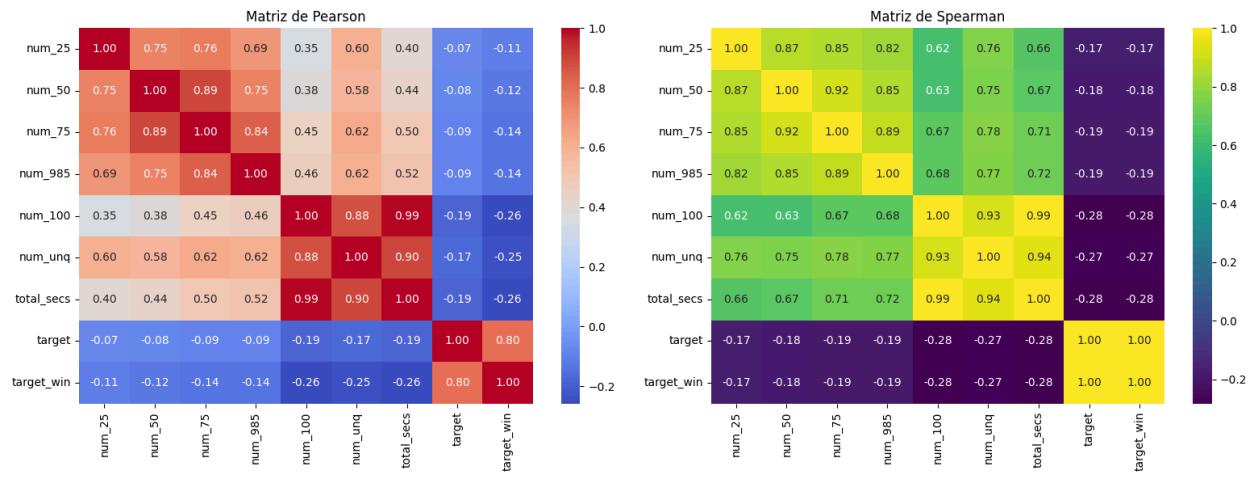
8.1. Logs

```
In [28]: logs_vars = [
    "num_25",
    "num_50",
    "num_75",
    "num_985",
    "num_100",
    "num_unq",
    "total_secs",
]

df_base_logs = df_base.select("msno", "safra", "target", "target_win", *logs_vars)
```

8.1.1. Correlacao geral

```
In [58]: correlation_matrix(df_base_logs, logs_vars + ["target", "target_win"])
```



```
(  num_25      num_50      num_75      num_985     num_100     num_unq  \
num_25  1.000000  0.753741  0.759602  0.693400  0.354383  0.604572
num_50  0.753741  1.000000  0.894074  0.745172  0.384573  0.578439
num_75  0.759602  0.894074  1.000000  0.842939  0.448068  0.622874
num_985 0.693400  0.745172  0.842939  1.000000  0.461307  0.615890
num_100 0.354383  0.384573  0.448068  0.461307  1.000000  0.881767
num_unq 0.604572  0.578439  0.622874  0.615890  0.881767  1.000000
total_secs 0.399223  0.437446  0.501727  0.517786  0.993097  0.898821
target  -0.069657 -0.079140 -0.089230 -0.089948 -0.185600 -0.173459
target_win -0.110813 -0.124994 -0.137714 -0.136597 -0.256659 -0.246365

                           total_secs   target  target_win
num_25  0.399223 -0.069657 -0.110813
num_50  0.437446 -0.079140 -0.124994
num_75  0.501727 -0.089230 -0.137714
num_985 0.517786 -0.089948 -0.136597
num_100 0.993097 -0.185600 -0.256659
num_unq 0.898821 -0.173459 -0.246365
total_secs 1.000000 -0.186112 -0.258363
target  -0.186112  1.000000  0.795024
target_win -0.258363  0.795024  1.000000 ,
                           num_25      num_50      num_75      num_985     num_100     num_unq  \
num_25  1.000000  0.869171  0.850199  0.818937  0.624961  0.762015
num_50  0.869171  1.000000  0.917541  0.850349  0.628252  0.749846
num_75  0.850199  0.917541  1.000000  0.886688  0.674382  0.775586
num_985 0.818937  0.850349  0.886688  1.000000  0.682427  0.767588
num_100 0.624961  0.628252  0.674382  0.682427  1.000000  0.928506
num_unq 0.762015  0.749846  0.775586  0.767588  0.928506  1.000000
total_secs 0.660181  0.670780  0.714959  0.723459  0.994513  0.942090
target  -0.174354 -0.182878 -0.194895 -0.192701 -0.282175 -0.274929
target_win -0.174402 -0.182920 -0.194936 -0.192737 -0.282201 -0.274956

                           total_secs   target  target_win
num_25  0.660181 -0.174354 -0.174402
num_50  0.670780 -0.182878 -0.182920
num_75  0.714959 -0.194895 -0.194936
num_985 0.723459 -0.192701 -0.192737
num_100 0.994513 -0.282175 -0.282201
num_unq 0.942090 -0.274929 -0.274956
total_secs 1.000000 -0.283551 -0.283578
target  -0.283551  1.000000  0.999999
target_win -0.283578  0.999999  1.000000 )
```

8.1.2. flag_has_logs

Presente ou nao na base de logs.

```
In [29]: df_base_logs = df_base_logs.withColumn("flag_has_logs", F.when(F.col("total_secs").isNull(), 0).otherwise(1))
```

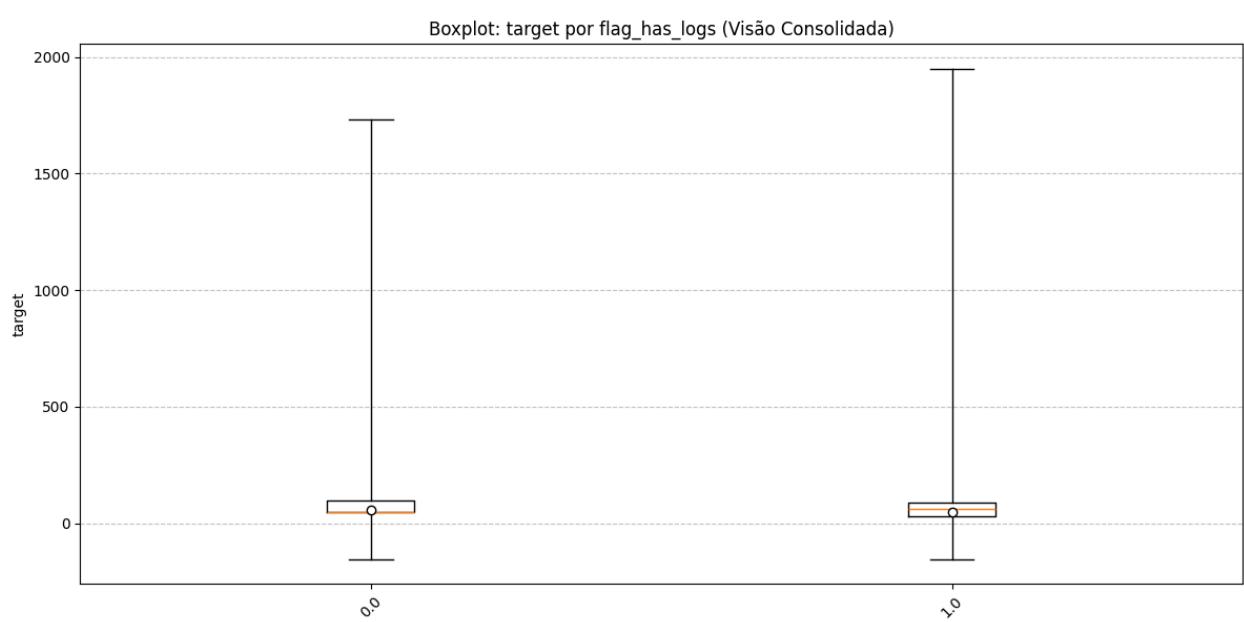
```
In [115]: calcular_distribuicao(df_base_logs, ["flag_has_logs"])
```

flag_has_logs	total	pct_total
1	9975880	88.73
0	1266985	11.27

```
Out[115]: DataFrame[flag_has_logs: int, total: bigint, pct_total: double]
```

```
In [147]: plot_boxplot(df_base_logs, ["flag_has_logs"], "target", table=True)
```

```
[147]: Processando estatísticas para: flag_has_logs...
```

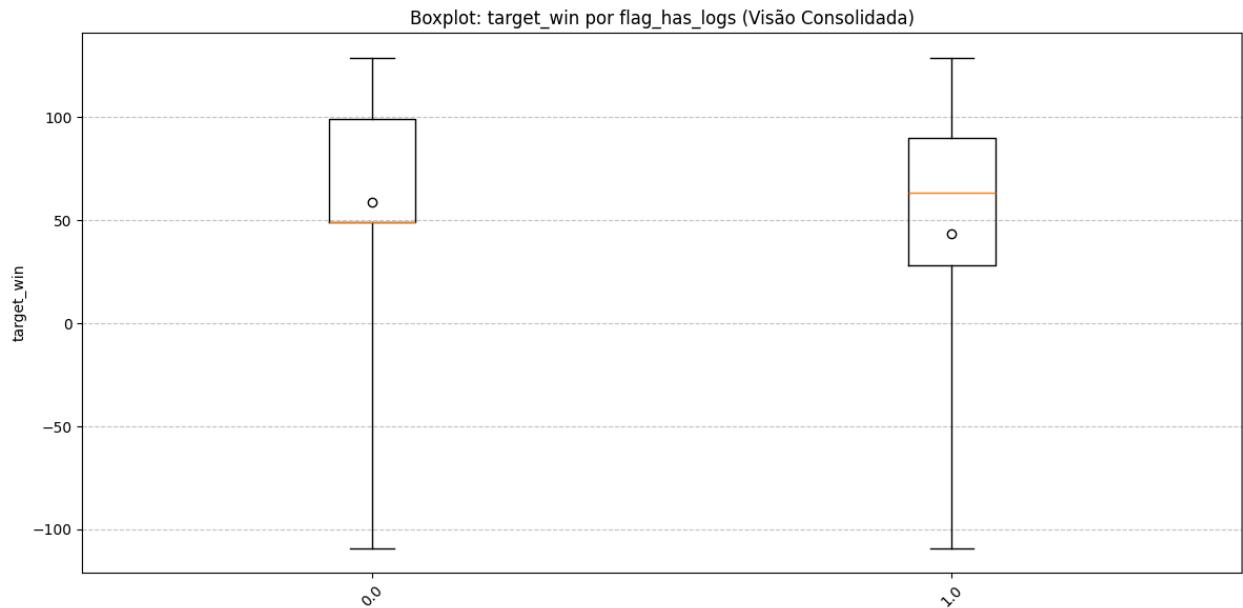


```
--- Estatísticas: flag_has_logs (Visão Consolidada) ---
```

	flag_has_logs	min	q1	med	mean	q3	max
1	0	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386
0	1	-152.5452	28.205901	63.178193	47.690904	90.054096	1950.000000

```
In [148]: plot_boxplot(df_base_logs, ["flag_has_logs"], "target_win", table=True)
```

Processando estatísticas para: flag_has_logs...



--- Estatísticas: flag_has_logs (Visão Consolidada) ---

	flag_has_logs	min	q1	med	mean	q3	max
1	0	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
0	1	-108.964925	28.205901	63.178193	43.633896	90.054096	128.953521

=====

8.1.3. total_plays

```
In [30]: df_base_logs = df_base_logs.withColumn("total_plays", F.col("num_25") + F.col("num_50") + F.col("num_75") + F.col("num_985") + F.col("num_100"))
```

Descriptivas + correlacão com target

```
In [28]: df_base_logs.select("total_plays").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+-----+
|summary|      total_plays|
+-----+-----+
|  count|    9975880|
|  mean|685.8388238431096|
| stddev| 748.163833373172|
|  min|      1.0|
|  1%|      2.0|
|  5%|     18.0|
| 25%|    164.0|
| 50%|    443.0|
| 75%|    932.0|
| 95%|   2242.0|
| 99.5%|  3780.0|
|  max|   4630.0|
+-----+-----+
```

```
In [29]: df_base_logs.select("total_plays", "target").corr("total_plays", "target")
```

Out[29]: -0.15308296170078672

```
In [46]: df_base_logs.select("total_plays", "target_win").corr("total_plays", "target_win")
```

Out[46]: -0.21555169062632873

Correlacao negativa e relativamente baixa: quanto mais musicas escutadas (completas ou nao), menor valor de target

Agrupamento por decis

```
In [32]: df_base_logs.select("total_plays").summary("count", "mean", "stddev", "min", "10%", "20%", "30%", "40%", "50%", "60%", "70%", "80%", "90%", "max").show()
```

summary	total_plays
count	9975880
mean	685.8388238431096
stddev	748.163833373172
min	1.0
10%	46.0
20%	120.0
30%	211.0
40%	317.0
50%	443.0
60%	598.0
70%	801.0
80%	1097.0
90%	1638.0
max	4630.0

```
In [ ]: df_base_logs = df_base_logs.withColumn("total_plays_group",  
    F.when(F.col("flag_has_logs").isIn(0), "unknown")\br/>    .when(F.col("total_plays").between(1, 46), "p10")\br/>    .when(F.col("total_plays").between(47, 120), "p20")\br/>    .when(F.col("total_plays").between(121, 211), "p30")\br/>    .when(F.col("total_plays").between(212, 317), "p40")\br/>    .when(F.col("total_plays").between(318, 443), "p50")\br/>    .when(F.col("total_plays").between(444, 598), "p60")\br/>    .when(F.col("total_plays").between(599, 801), "p70")\br/>    .when(F.col("total_plays").between(802, 1097), "p80")\br/>    .when(F.col("total_plays").between(1098, 1638), "p90")\br/>    .otherwise("p90+"))
```

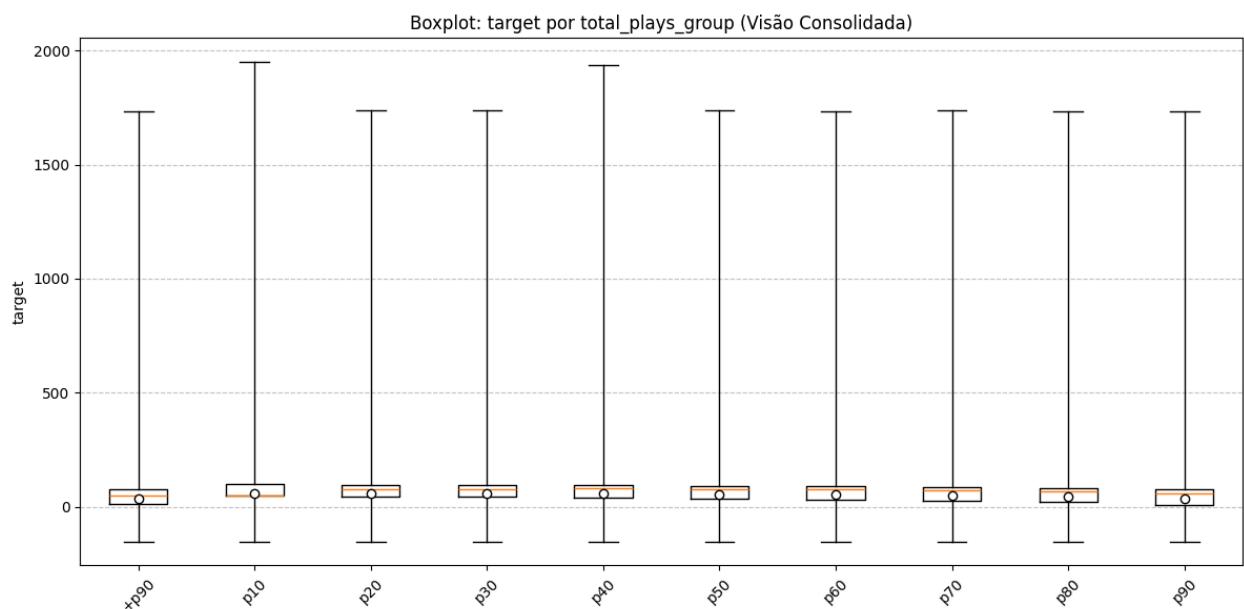
```
In [34]: calcular_distribuicao(df_base_logs, ["total_plays_group"])
```

total_plays_group	total	pct_total
+p90	2263654	20.13
p10	1006877	8.96
p30	1003579	8.93
p60	997308	8.87
p90	997296	8.87
p80	996967	8.87
p50	996268	8.86
p70	995794	8.86
p40	994730	8.85
p20	990392	8.81

Out[34]: DataFrame[total_plays_group: string, total: bigint, pct_total: double]

```
In [35]: plot_boxplot(df_base_logs, ["total_plays_group"], "target", table=True)
```

Processando estatísticas para: total_plays_group...



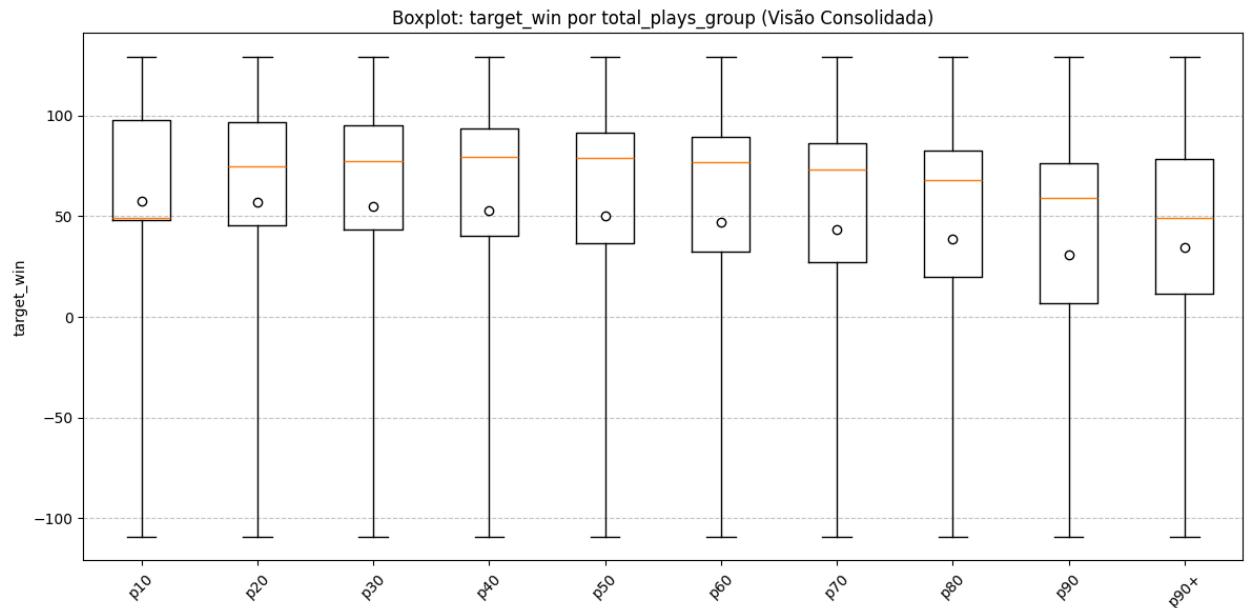
--- Estatísticas: total_plays_group (Visão Consolidada) ---

	total_plays_group	min	q1	med	mean	q3	max
1	+p90	-152.5452	11.743867	49.000000	36.270471	78.098894	1733.833386
4	p10	-152.5452	47.816344	49.000000	58.723051	97.750115	1950.000000
3	p20	-152.5452	45.577824	74.819067	59.336415	96.672926	1737.241320
2	p30	-152.5452	43.116862	77.519916	58.474130	95.242394	1737.711061
8	p40	-152.5452	40.219816	79.451847	56.856944	93.559796	1936.799455
9	p50	-152.5452	36.710126	78.819060	54.406560	91.577108	1736.987900
6	p60	-152.5452	32.543869	76.582929	51.756862	89.256250	1735.253213
5	p70	-152.5452	27.326239	73.081341	48.392427	86.318137	1736.541069
7	p80	-152.5452	19.899592	68.034976	43.812993	82.433442	1734.140378
0	p90	-152.5452	6.999107	59.234548	36.448164	76.231174	1735.146209

=====

```
In [49]: plot_boxplot(df_base_logs, ["total_plays_group"], "target_win", table=True)
```

Processando estatísticas para: total_plays_group...



--- Estatísticas: total_plays_group (Visão Consolidada) ---

	total_plays_group	min	q1	med	mean	q3	max
4	p10	-108.964925	47.816344	49.000000	57.232945	97.750115	128.953521
3	p20	-108.964925	45.577824	74.819067	56.688432	96.672926	128.953521
2	p30	-108.964925	43.116862	77.519916	55.083761	95.242394	128.953521
8	p40	-108.964925	40.219816	79.451847	52.876757	93.559796	128.953521
9	p50	-108.964925	36.710126	78.819060	50.039286	91.577108	128.953521
6	p60	-108.964925	32.543869	76.582929	46.983059	89.256250	128.953521
5	p70	-108.964925	27.326239	73.081341	43.285811	86.318137	128.953521
7	p80	-108.964925	19.899592	68.034976	38.445000	82.433442	128.953521
0	p90	-108.964925	6.999107	59.234548	30.786098	76.231174	128.953521
1	p90+	-108.964925	11.743867	49.000000	34.450294	78.098894	128.953521

=====

Agrupamento por quartis

```
In [36]: df_base_logs.select("total_plays").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "max").show()
```

```
+-----+-----+
|summary|      total_plays|
+-----+-----+
|  count|      9975880|
|  mean|685.8388238431096|
| stddev| 748.163833373172|
|  min|       1.0|
| 25%|     164.0|
| 50%|     443.0|
| 75%|     932.0|
|  max|    4630.0|
+-----+-----+
```

```
In [ ]: df_base_logs = df_base_logs.withColumn("total_plays_group",
    F.when(F.col("flag_has_logs").isin(0), "unknown")\
    .when(F.col("total_plays").between(1, 164), "p25")\
    .when(F.col("total_plays").between(165, 443), "p50")\
    .when(F.col("total_plays").between(444, 932), "p75")\
    .otherwise("p75+")
)

# Analise pelos valores brutos: pode ocasionar problemas na performance futura do modelo, com a mudanca da distribuicao dos
valores da var
```

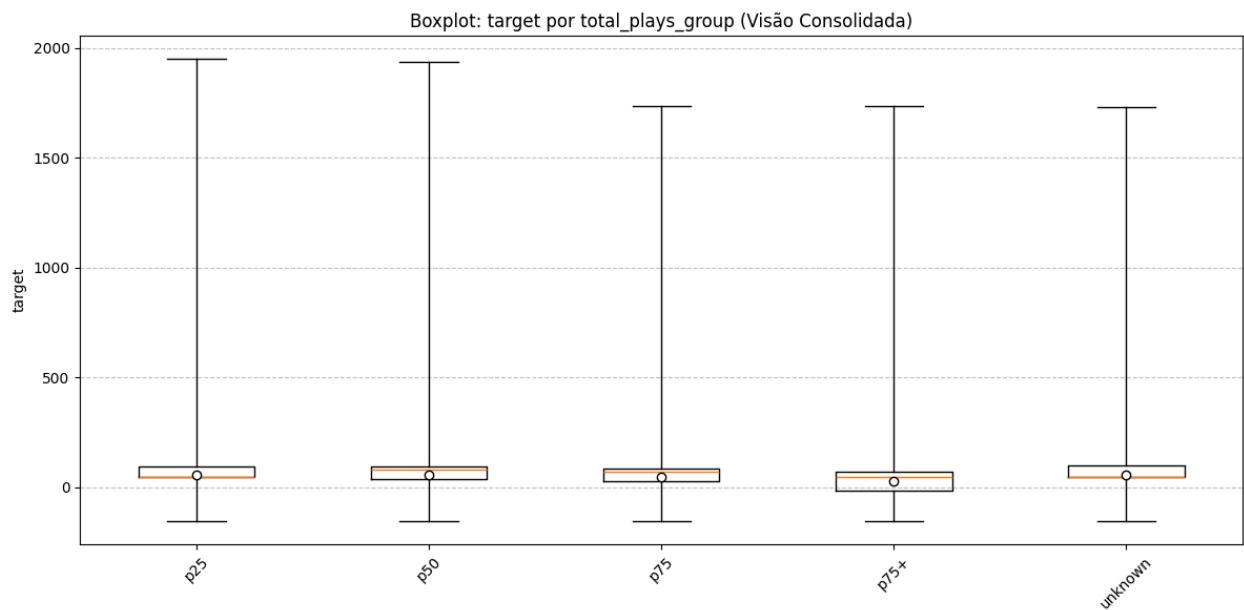
```
In [117]: calcular_distribuicao(df_base_logs, ["total_plays_group"])
```

total_plays_group	total	pct_total
p25	2502893	22.26
p75+	2493687	22.18
p75	2490347	22.15
p50	2488953	22.14
unknown	1266985	11.27

Out[117]: DataFrame[total_plays_group: string, total: bigint, pct_total: double]

```
In [62]: plot_boxplot(df_base_logs, ["total_plays_group"], "target", table=True)
```

Processando estatísticas para: total_plays_group...



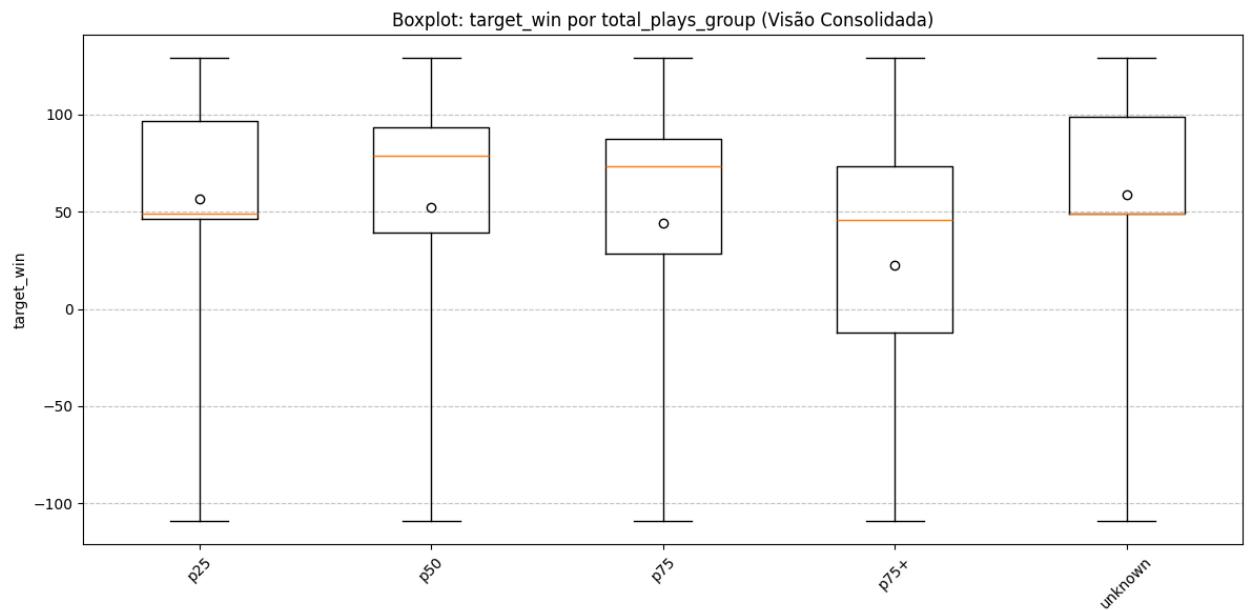
--- Estatísticas: total_plays_group (Visão Consolidada) ---

	total_plays_group	min	q1	med	mean	q3	max
0	p25	-152.5452	46.211867	49.000000	59.002846	96.806022	1950.000000
4	p50	-152.5452	39.313317	78.853285	56.113354	93.126420	1936.799455
2	p75	-152.5452	28.287261	73.315000	49.063618	87.193322	1736.541069
3	p75+	-152.5452	-12.338117	45.704088	27.411001	73.146944	1735.146209
1	unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386

=====

```
In [63]: plot_boxplot(df_base_logs, ["total_plays_group"], "target_win", table=True)
```

Processando estatísticas para: total_plays_group...



--- Estatísticas: total_plays_group (Visão Consolidada) ---

	total_plays_group	min	q1	med	mean	q3	max
0	p25	-108.964925	46.211867	49.000000	56.665533	96.806022	128.953521
4	p50	-108.964925	39.313317	78.853285	52.072322	93.126420	128.953521
2	p75	-108.964925	28.287261	73.315000	44.067086	87.193322	128.953521
3	p75+	-108.964925	-12.338117	45.704088	22.662635	73.146944	128.953521
1	unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521

=====

A categoria unknown foi mantida como regime por representar ausencia de consumo, associada a baixo custo operacional e margens sistematicamente superiores as faixas de alto uso. Entendo que, por esta interpretacao, nao seria apropriado trata-la como dado imputavel (apesar na similaridade com as estatísticas descritivas de p25).

```
In [32]: df_base_logs = segment_by_percentile(df_base_logs, "total_plays", "logs")
```

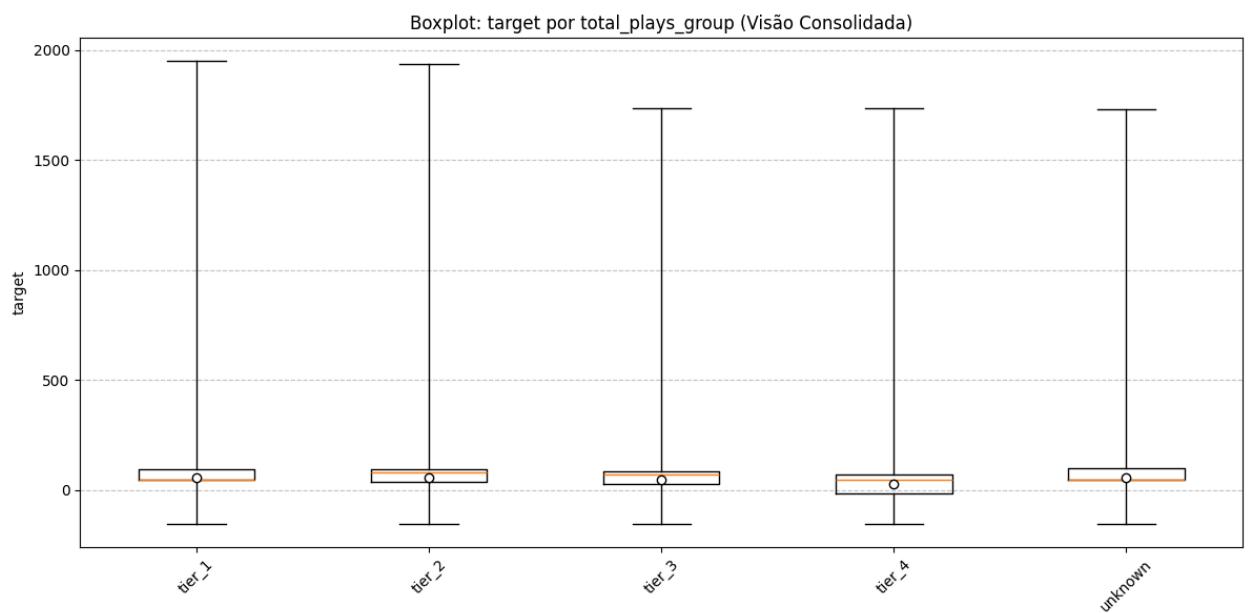
```
In [33]: calcular_distribuicao(df_base_logs, ["total_plays_group"])
```

total_plays_group	total	pct_total
tier_1	2502893	22.26
tier_4	2493687	22.18
tier_3	2490347	22.15
tier_2	2488953	22.14
unknown	1266985	11.27

```
Out[33]: DataFrame[total_plays_group: string, total: bigint, pct_total: double]
```

```
In [39]: plot_boxplot(df_base_logs, ["total_plays_group"], "target", table=True)
```

Processando estatísticas para: total_plays_group...



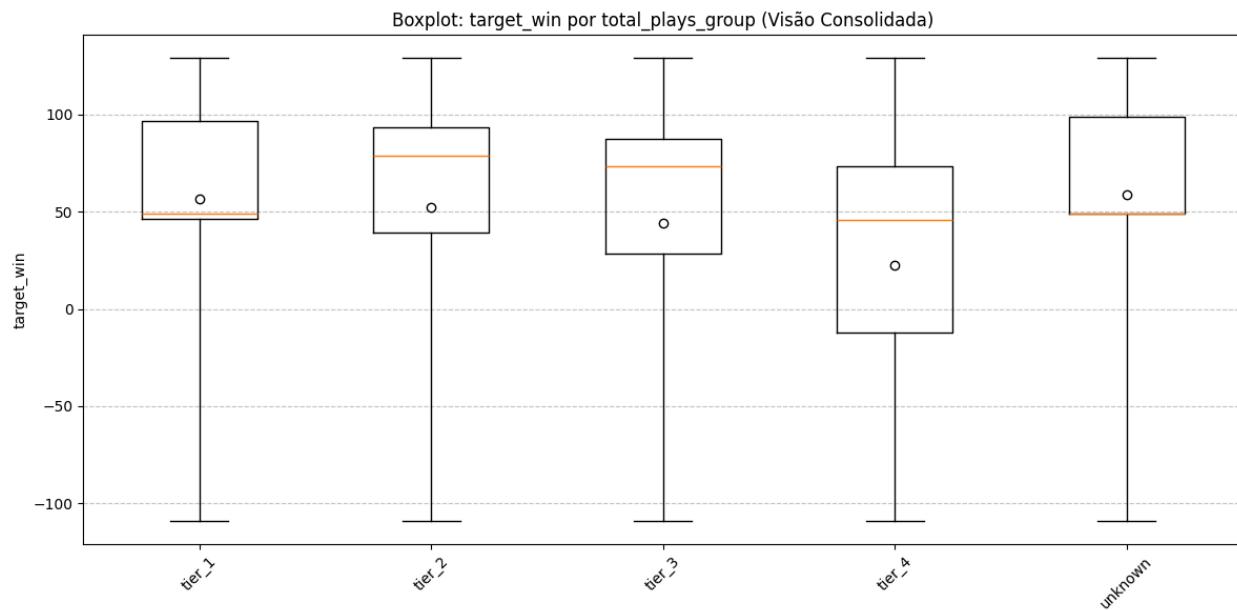
--- Estatísticas: total_plays_group (Visão Consolidada) ---

	total_plays_group	min	q1	med	mean	q3	max
0	tier_1	-152.5452	46.211867	49.000000	59.002846	96.806022	1950.000000
2	tier_2	-152.5452	39.313317	78.853285	56.113354	93.126420	1936.799455
3	tier_3	-152.5452	28.287261	73.315000	49.063618	87.193322	1736.541069
4	tier_4	-152.5452	-12.338117	45.704088	27.411001	73.146944	1735.146209
1	unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386

=====

```
In [40]: plot_boxplot(df_base_logs, ["total_plays_group"], "target_win", table=True)
```

Processando estatísticas para: total_plays_group...



--- Estatísticas: total_plays_group (Visão Consolidada) ---

	total_plays_group	min	q1	med	mean	q3	max
0	tier_1	-108.964925	46.211867	49.000000	56.665533	96.806022	128.953521
2	tier_2	-108.964925	39.313317	78.853285	52.072322	93.126420	128.953521
3	tier_3	-108.964925	28.287261	73.315000	44.067086	87.193322	128.953521
4	tier_4	-108.964925	-12.338117	45.704088	22.662635	73.146944	128.953521
1	unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521

=====

Conclusao

Apesar do agrupamento por decis trazer mais granularidade, vimos a partir das estatísticas que o padrao se repete boas vezes, nao adicionando novas informacoes. Os quartis capturam melhor, sendo mais estaveis (o que implica em menor risco de overfitting e maior facilidade em regularizacao - pensando numa solucao linear com elastic net, por exemplo)

```
In [43]: df_base_logs = segment_by_percentile(df_base_logs, "total_plays", "logs")

map_total_plays = {
    "tier_1": "01_casual_listener",
    "tier_2": "02_regular_listener",
    "tier_3": "03_frequent_listener",
    "tier_4": "04_power_user"
}
```

8.1.4. completed_songs_rate

```
In [40]: # Cálculo da taxa de completude com tratamento de divisão por zero
df_base_logs = df_base_logs.withColumn("completed_songs_rate",
    F.when(F.col("total_plays") > 0, F.col("num_100") / F.col("total_plays")).otherwise(0.0))
```

Descritivas + correlacao com target

```
In [53]: df_base_logs.select("completed_songs_rate").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary completed_songs_rate 	
count	11242865
mean	0.6103863543898564
stddev	0.29662970789050547
min	0.0
1%	0.0
5%	0.0
25%	0.4772313296903461
50%	0.6943521594684385
75%	0.8355555555555556
95%	0.9528619528619529
99.5%	1.0
max	1.0

```
In [55]: df_base_logs.select("completed_songs_rate", "target").corr("completed_songs_rate", "target")
```

```
Out[55]: -0.043266520000030825
```

```
In [56]: df_base_logs.select("completed_songs_rate", "target_win").corr("completed_songs_rate", "target_win")
```

```
Out[56]: -0.07149880745346689
```

Elastic Net pode aprender coeficiente pequeno ou zerar via L1, nao atrapalhando no modelo.

Agrupamento por quartis

```
In [67]: df_base_logs.select("completed_songs_rate").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "max").show()
```

summary completed_songs_rate 	
count	11242865
mean	0.6103863543898564
stddev	0.29662970789050547
min	0.0
25%	0.4772313296903461
50%	0.6943521594684385
75%	0.8355555555555556
max	1.0

```
In [137]: df_base_logs = df_base_logs.withColumn("completed_songs_rate_group", F.when(F.col("flag_has_logs").isin(0), "unknown")\ .when((F.col("completed_songs_rate") > 0.00) & (F.col("completed_songs_rate") <= 0.477), "p25")\ .when((F.col("completed_songs_rate") > 0.477) & (F.col("completed_songs_rate") <= 0.694), "p50")\ .when((F.col("completed_songs_rate") > 0.694) & (F.col("completed_songs_rate") <= 0.835), "p75")\ .otherwise("p75+"))
```

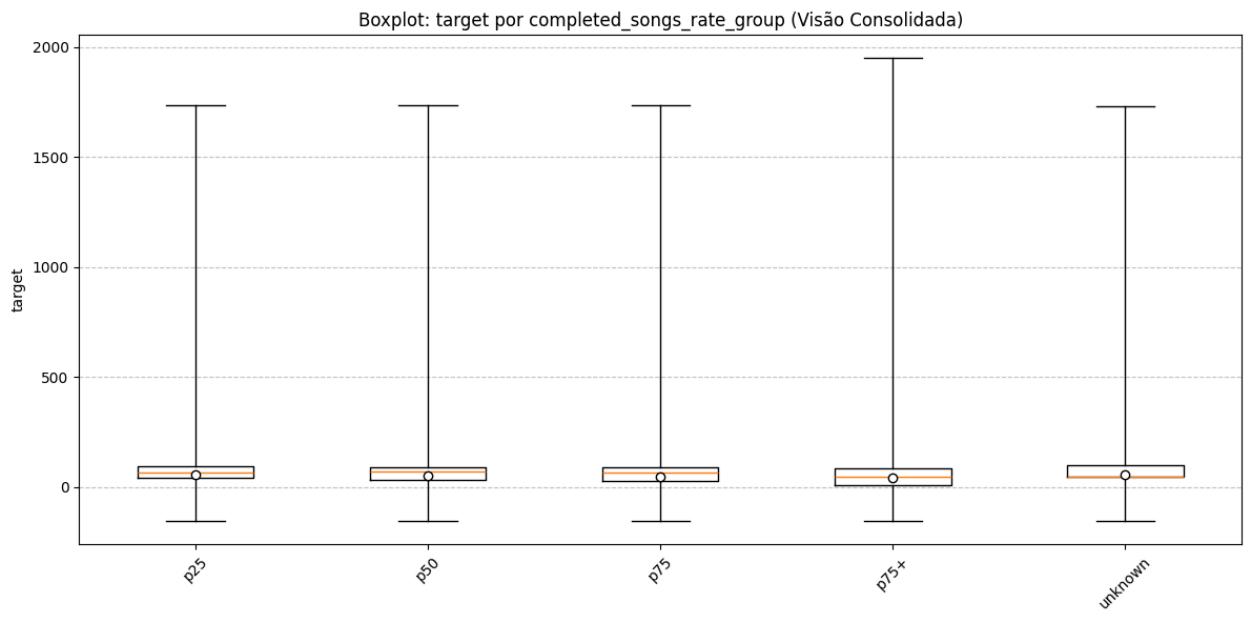
```
In [138]: calcular_distribuicao(df_base_logs, ["completed_songs_rate_group"])
```

completed_songs_rate_group	total	pct_total
p75+	2964854	26.37
p50	2806207	24.96
p75	2805589	24.95
p25	1399230	12.45
unknown	1266985	11.27

```
Out[138]: DataFrame[completed_songs_rate_group: string, total: bigint, pct_total: double]
```

```
In [139]: plot_boxplot(df_base_logs, ["completed_songs_rate_group"], "target", table=True)
```

Processando estatísticas para: completed_songs_rate_group...



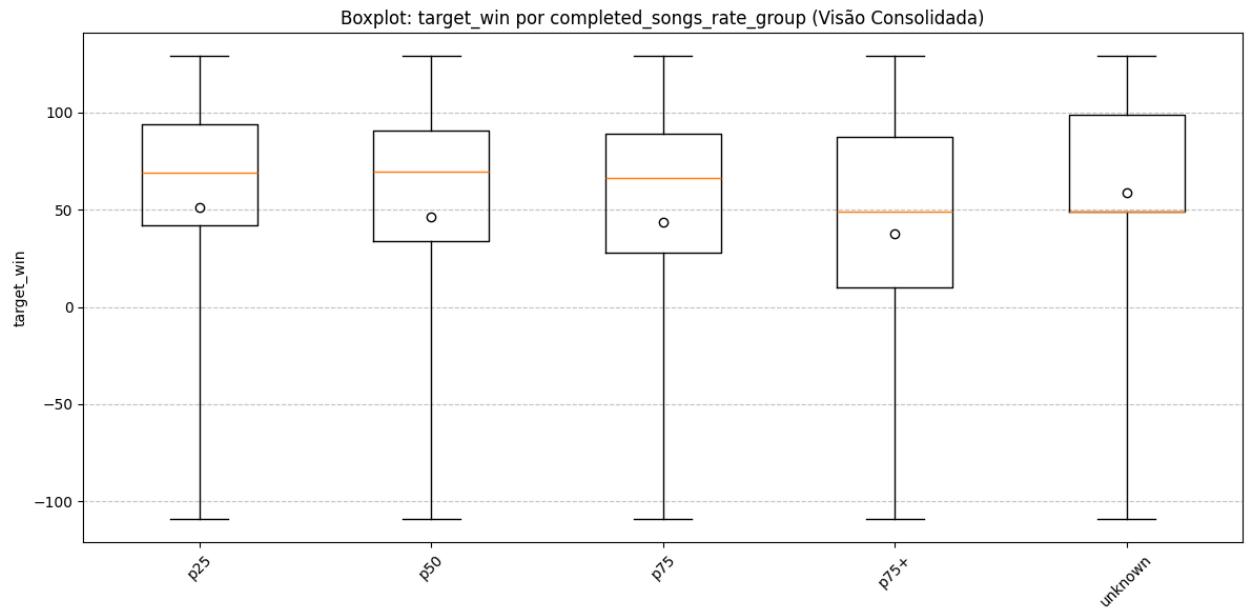
--- Estatísticas: completed_songs_rate_group (Visão Consolidada) ---

	completed_songs_rate_group	min	q1	med	mean	q3	max
0	p25	-152.5452	41.910833	68.849401	55.356015	93.868893	1737.947047
4	p50	-152.5452	33.850169	69.618255	51.131367	90.748839	1737.859653
2	p75	-152.5452	27.796545	66.216070	47.810112	89.305776	1737.929378
3	p75+	-152.5452	10.055245	49.000000	40.749212	87.412344	1950.000000
1	unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386

=====

```
In [140]: plot_boxplot(df_base_logs, ["completed_songs_rate_group"], "target_win", table=True)
```

Processando estatísticas para: completed_songs_rate_group...



--- Estatísticas: completed_songs_rate_group (Visão Consolidada) ---

	completed_songs_rate_group	min	q1	med	mean	q3	max
0	p25	-108.964925	41.910833	68.849401	51.348515	93.868893	128.953521
4	p50	-108.964925	33.850169	69.618255	46.536379	90.748839	128.953521
2	p75	-108.964925	27.796545	66.216070	43.424178	89.305776	128.953521
3	p75+	-108.964925	10.055245	49.000000	37.496112	87.412344	128.953521
1	unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521

=====

Conclusao

completed_songs_rate transformada em grupos por quartis, complementa o volume de uso (total_plays) ao capturar profundidade de consumo. Dois usuarios podem ter o mesmo numero de musicas escutadas (repetidas ou nao) mas taxas de completude diferentes. Observa-se uma relação negativa com a target, indicando que usuarios com maior taxa de conclusao de musicas tendem a gerar maior custo marginal, reduzindo a margem futura. O comportamento permanece o mesmo depois de winsorizar, sugerindo sinal estrutural e nao efeito de outliers. Sera levada para feature engineering (mesmo que possa ser zerada no elastic net)

```
In [41]: # Agrupamento por quartis
df_base_logs = df_base_logs.withColumn("completed_songs_rate_group",
    F.when(F.col("flag_has_logs").isin(0), "00_unknown")
    .when((F.col("completed_songs_rate") > 0.00) & (F.col("completed_songs_rate") <= 0.477), "01_bouncer")
    .when((F.col("completed_songs_rate") > 0.477) & (F.col("completed_songs_rate") <= 0.694), "02_skipping_listener")
    .when((F.col("completed_songs_rate") > 0.694) & (F.col("completed_songs_rate") <= 0.835), "03_engaged_listener")
    .otherwise("04_completionist"))
```

8.1.5. avg_secs_per_play

```
In [59]: df_base_logs = df_base_logs.withColumn("avg_secs_per_play", F.when(F.col("total_plays") > 0, F.col("total_secs") /
F.col("total_plays")).otherwise(0.0))
```

Descritivas + correlacao com target

```
In [64]: df_base_logs.select("avg_secs_per_play").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	avg_secs_per_play
count	11242865
mean	167.91906343205807
stddev	74.89269521624702
min	0.0
1%	0.0
5%	0.0
25%	145.6058401486989
50%	189.2933519736842
75%	217.4252882527147
95%	245.43684210526317
99.5%	280.24546994535524
max	4925.603363128492

```
In [65]: df_base_logs.select("avg_secs_per_play", "target").corr("avg_secs_per_play", "target")
```

```
Out[65]: -0.035236909585071315
```

```
In [66]: df_base_logs.select("avg_secs_per_play", "target_win").corr("avg_secs_per_play", "target_win")
```

```
Out[66]: -0.061425079113587744
```

`avg_secs_per_play` apresenta correlação muito fraca com o `target`, mesmo após winsorizar. Além disso, seu significado econômico é curioso, pois a estrutura de custos se mostrou mais sensível para a quantidade de reproduções do que para a duração média das músicas (mas o `total_secs` sendo parte da fórmula de custo, que compõe a margem líquida?). Para uma solução linear, a variável continua não acrescenta muito valor.

Conclusão

- * Para soluções lineares, a variável seria um empecilho, caso inserida, pois poderia instabilizar coeficientes (pensando em regularização de elastic net, isso se não zera-lo);
- * Para algoritmos de boosting, a variável não cria bons thresholds, o que permite concluir que o ganho marginal seria próximo de zero;
- * Para algoritmos de bagging (random forest, por exemplo), `avg_secs_per_play` não cria bons cortes e possivelmente aumentaria a variação.

Para este caso, mesmo considerando a construção de algum agrupamento, a relação com o `target` possivelmente continuaria fraca, dado que não separa regimes de negócios. Descartar sua construção.

8.1.5. avg_secs_per_unq

Profundidade média por música. Separa usuários que repetem músicas com baixa frequência, ajudando a definir usuário que mais explora a plataforma. Não capturada por `total_secs` e `num_unq` isoladamente

```
In [ ]: df_base_logs = df_base_logs.withColumn("avg_secs_per_unq", F.when(F.col("num_unq") > 0, F.col("total_secs") / F.col("num_unq")).otherwise(0.0))
```

Descriptivas + correlação com target

```
In [82]: df_base_logs.select("avg_secs_per_unq").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	avg_secs_per_unq
count	11242865
mean	253.0199987430327
stddev	417.6774147961091
min	0.0
1%	0.0
5%	0.0
25%	173.9653440538519
50%	230.14118142235125
75%	276.917049382716
95%	468.9416877133106
99.5%	1830.37644
max	157112.34133333334

Distribuicao assimetrica a direita

```
In [83]: df_base_logs.select("avg_secs_per_unq", "target").corr("avg_secs_per_unq", "target")
```

```
Out[83]: -0.017425877847683863
```

```
In [84]: df_base_logs.select("avg_secs_per_unq", "target_win").corr("avg_secs_per_unq", "target_win")
```

```
Out[84]: -0.024362072685376972
```

Nao existe relacao linear valida, o que torna uma variavel ruim para elastic net. Porem, da pra tratar para demais casos.

```
In [ ]: # Winsorizar pra cima, no p995
df_base_logs = df_base_logs.withColumn("avg_secs_per_unq_cap",
    F.when(F.col("avg_secs_per_unq") > 1830, 1830).otherwise(F.col("avg_secs_per_unq")))

# Aplicar transformação logarítmica - considerar levar esta como o valor continuo no modelo linear
df_base_logs = df_base_logs.withColumn("log_avg_secs_per_unq", F.log1p(F.col("avg_secs_per_unq")))
```

Agrupamento por quartis

```
In [87]: df_base_logs.select("avg_secs_per_unq_cap").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "max").show()
```

```
+-----+-----+
|summary|avg_secs_per_unq_cap|
+-----+-----+
| count| 11242865|
| mean | 240.71645049590418|
| stddev| 195.6380292116396|
| min  | 0.0|
| 25% | 173.9653440538519|
| 50% | 230.14118142235125|
| 75% | 276.917049382716|
| max  | 1830.0|
+-----+-----+
```

```
In [130]: df_base_logs = df_base_logs.withColumn("avg_secs_per_unq_cap_group",
    F.when(F.col("flag_has_logs").isin(0), "unknown")\
    .when((F.col("avg_secs_per_unq_cap") > 0) & (F.col("avg_secs_per_unq_cap") <= 173.965), "p25")\
    .when((F.col("avg_secs_per_unq_cap") > 173.965) & (F.col("avg_secs_per_unq_cap") <= 230.141), "p50")\
    .when((F.col("avg_secs_per_unq_cap") > 230.141) & (F.col("avg_secs_per_unq_cap") <= 276.917), "p75")\
    .otherwise("p75+"))
```

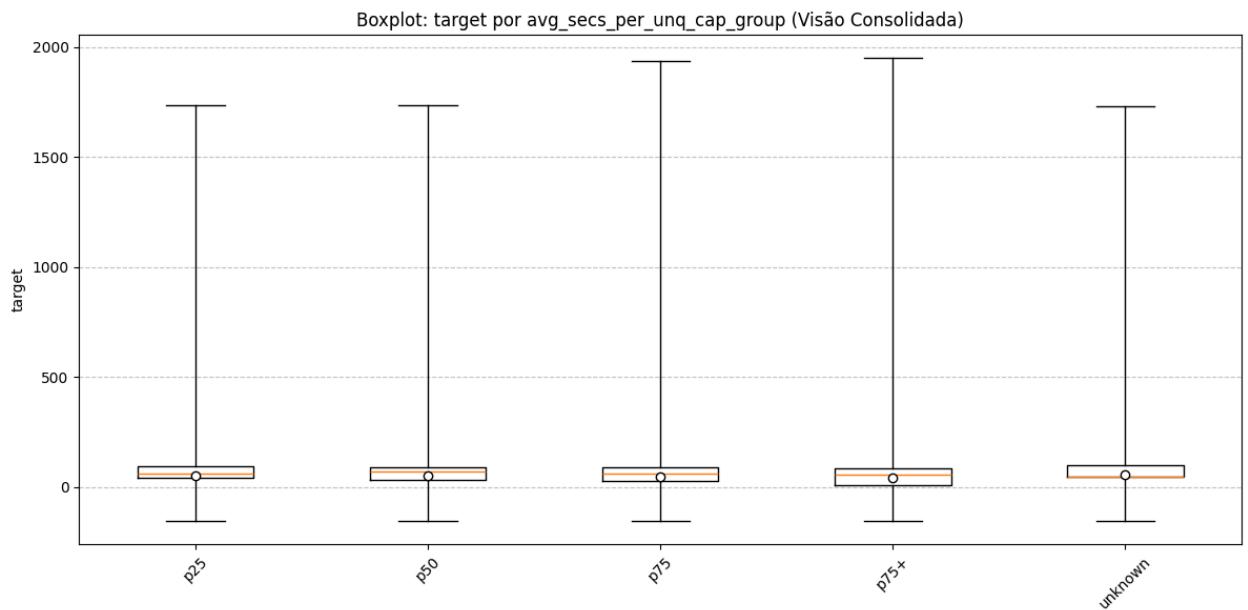
```
In [131]: calcular_distribuicao(df_base_logs, ["avg_secs_per_unq_cap_group"])
```

```
+-----+-----+-----+
|avg_secs_per_unq_cap_group|total |pct_total|
+-----+-----+-----+
|p75+                   |2811090|25.0   |
|p75                   |2810804|25.0   |
|p50                   |2810614|25.0   |
|p25                   |1543372|13.73  |
|unknown                |1266985|11.27  |
+-----+-----+-----+
```

```
Out[131]: DataFrame[avg_secs_per_unq_cap_group: string, total: bigint, pct_total: double]
```

```
In [90]: plot_boxplot(df_base_logs, ["avg_secs_per_unq_cap_group"], "target", table=True)
```

Processando estatísticas para: avg_secs_per_unq_cap_group...



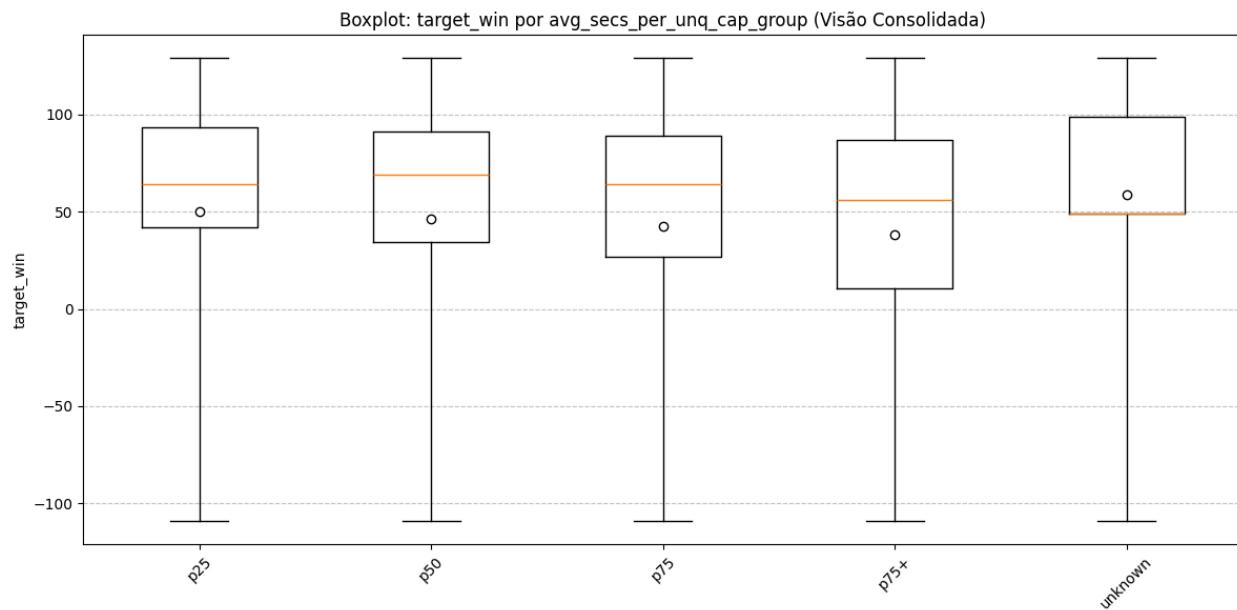
--- Estatísticas: avg_secs_per_unq_cap_group (Visão Consolidada) ---

	avg_secs_per_unq_cap_group	min	q1	med	mean	q3	max
0	p25	-152.5452	41.861596	63.927131	54.274849	93.500876	1737.859653
4	p50	-152.5452	34.417364	69.270302	51.039135	91.240601	1737.484560
2	p75	-152.5452	26.852644	64.044408	46.968586	89.190420	1936.799455
3	p75+	-152.5452	10.548187	56.028176	41.566656	87.156328	1950.000000
1	unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386

=====

```
In [91]: plot_boxplot(df_base_logs, ["avg_secs_per_unq_cap_group"], "target_win", table=True)
```

Processando estatísticas para: avg_secs_per_unq_cap_group...



--- Estatísticas: avg_secs_per_unq_cap_group (Visão Consolidada) ---

	avg_secs_per_unq_cap_group	min	q1	med	mean	q3	max
0	p25	-108.964925	41.861596	63.927131	50.304066	93.500876	128.953521
4	p50	-108.964925	34.417364	69.270302	46.489987	91.240601	128.953521
2	p75	-108.964925	26.852644	64.044408	42.727295	89.190420	128.953521
3	p75+	-108.964925	10.548187	56.028176	38.141249	87.156328	128.953521
1	unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521

=====

Conclusao

avg_secs_per_unq , depois do cap e do agrupamento, apresentou relacao negativa clara com a target, refletindo maior custo medio de consumo por item. Apesar da baixa correlacao linear global, seu efeito emerge de forma consistente em regimes comportamentais, sendo informativa para modelos lineares regularizados e essencial para modelos nao lineares. Levar para feature engineering.

8.1.6. plays_per_unq

Plays por musica (intensidade). RF e LightGBM capturam bons thresholds e apos transformar da pra usar no elastic net

```
In [34]: df_base_logs = df_base_logs.withColumn("plays_per_unq", F.when(F.col("num_unq") > 0, F.col("num_100") / F.col("num_unq")).otherwise(0.0))
```

Descritivas + correlacao com target

```
In [93]: df_base_logs.select("plays_per_unq").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	plays_per_unq
count	11242865
mean	0.9334199641487592
stddev	1.7138072687322223
min	0.0
1%	0.0
5%	0.0
25%	0.5719512195121951
50%	0.8518518518518519
75%	1.0454545454545454
95%	1.7857142857142858
99.5%	7.145772594752186
max	871.75

Distribuição bem assimétrica a direita

```
In [94]: df_base_logs.select("plays_per_unq", "target").corr("plays_per_unq", "target")
```

```
Out[94]: -0.01866538221161783
```

```
In [95]: df_base_logs.select("plays_per_unq", "target_win").corr("plays_per_unq", "target_win")
```

```
Out[95]: -0.025806740358633715
```

Não existe relação linear válida, o que torna uma variável ruim para elastic net. Porem, da pra tratar para demais casos.

```
In [35]: # Winsorizar pra cima, no p995
df_base_logs = df_base_logs.withColumn("plays_per_unq_cap",
    F.when(F.col("plays_per_unq") > 7.14, 7.14).otherwise(F.col("plays_per_unq")))
```

Agrupamento por quartis

```
In [97]: df_base_logs.select("plays_per_unq_cap").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "max").show()
```

summary	plays_per_unq_cap
count	11242865
mean	0.8834730591295731
stddev	0.7726093126137725
min	0.0
25%	0.5719512195121951
50%	0.8518518518518519
75%	1.0454545454545454
max	7.14

```
In [118]: df_base_logs = df_base_logs.withColumn("plays_per_unq_cap_group",
    F.when(F.col("flag_has_logs").isin(0), "unknown")\
    .when((F.col("plays_per_unq_cap") >= 0) & (F.col("plays_per_unq_cap") <= 0.5719), "p25")\
    .when((F.col("plays_per_unq_cap") > 0.5719) & (F.col("plays_per_unq_cap") <= 0.8518), "p50")\
    .when((F.col("plays_per_unq_cap") > 0.8518) & (F.col("plays_per_unq_cap") <= 1.045), "p75")\
    .otherwise("p75+"))
```

```
In [119]: calcular_distribuicao(df_base_logs, ["plays_per_unq_cap_group"])

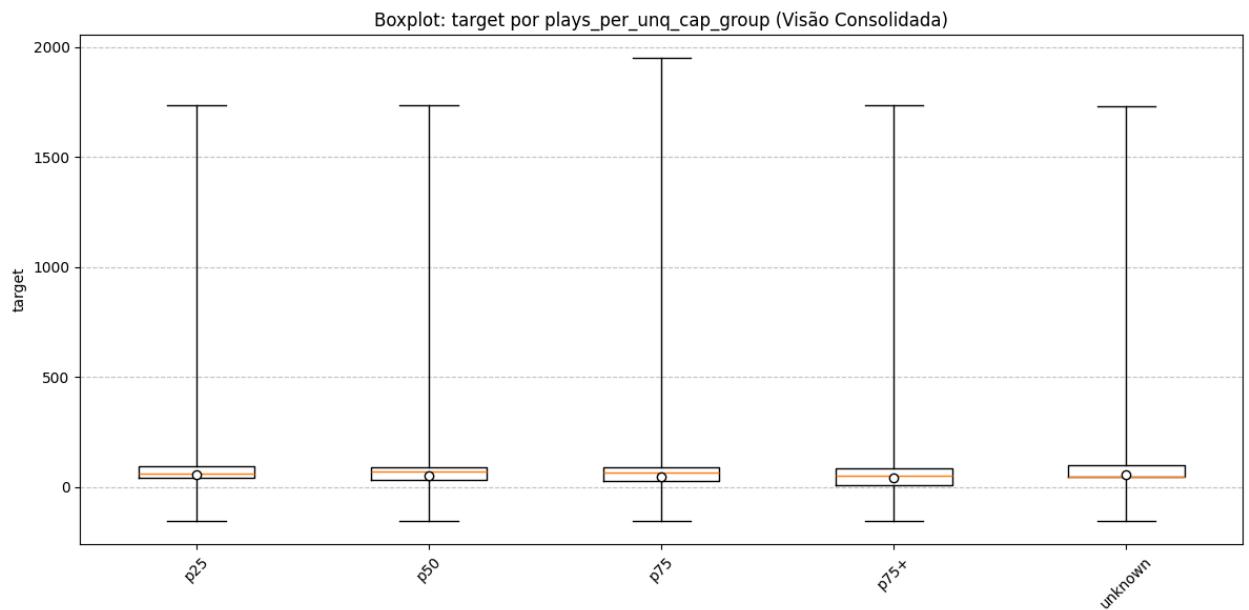
[119]:
```

plays_per_unq_cap_group	total	pct_total
p75+	281539	25.04
p50	2809320	24.99
p75	2808014	24.98
p25	1543207	13.73
unknown	1266985	11.27

```
Out[119]: DataFrame[plays_per_unq_cap_group: string, total: bigint, pct_total: double]
```

```
In [120]: plot_boxplot(df_base_logs, ["plays_per_unq_cap_group"], "target", table=True)
```

Processando estatísticas para: plays_per_unq_cap_group...



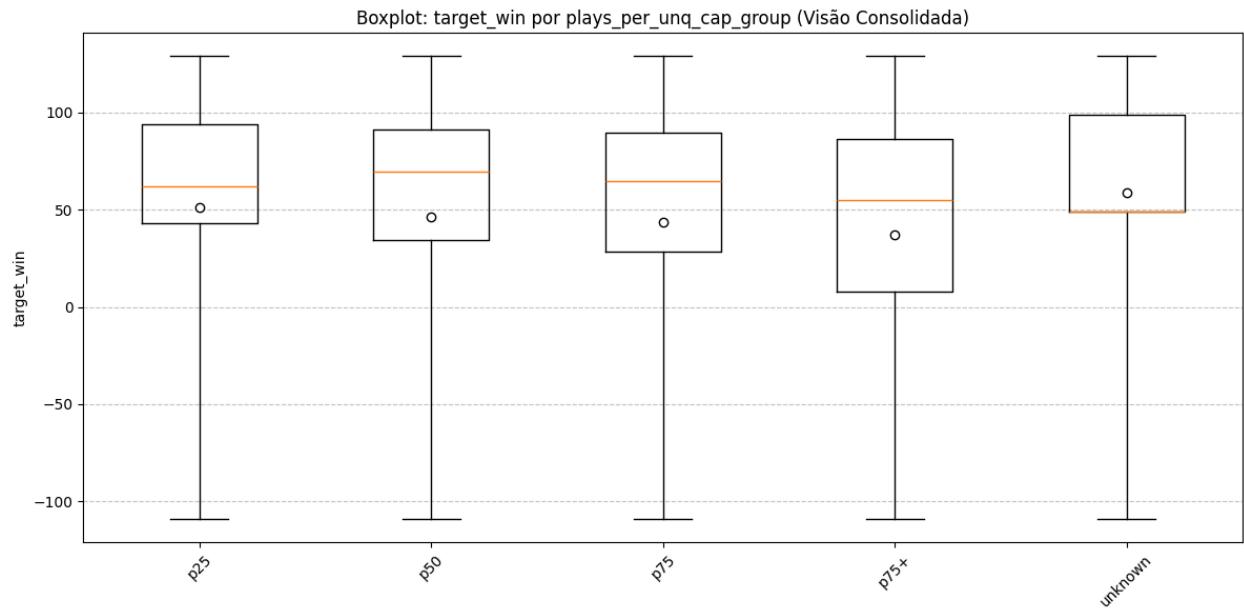
--- Estatísticas: plays_per_unq_cap_group (Visão Consolidada) ---

	plays_per_unq_cap_group	min	q1	med	mean	q3	max
0	p25	-152.5452	42.797811	62.055705	54.864552	93.983821	1737.859653
4	p50	-152.5452	34.198184	69.836618	51.114125	91.116295	1737.947047
2	p75	-152.5452	28.504832	64.806716	47.640133	89.518001	1950.000000
3	p75+	-152.5452	7.980188	54.715091	40.524290	86.528988	1737.041373
1	unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386

=====

```
In [121]: plot_boxplot(df_base_logs, ["plays_per_unq_cap_group"], "target_win", table=True)
```

Processando estatísticas para: plays_per_unq_cap_group...



--- Estatísticas: plays_per_unq_cap_group (Visão Consolidada) ---

	plays_per_unq_cap_group	min	q1	med	mean	q3	max
0	p25	-108.964925	42.797811	62.055705	51.023214	93.983821	128.953521
4	p50	-108.964925	34.198184	69.836618	46.493361	91.116295	128.953521
2	p75	-108.964925	28.504832	64.806716	43.437247	89.518001	128.953521
3	p75+	-108.964925	7.980188	54.715091	37.063461	86.528988	128.953521
1	unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521

=====

O agrupamento por quartil não me parece fazer sentido nesse caso. Quero que a variável me diga se o usuário tende a repetir ou explorar músicas, e não a faixa de distribuição que ele cai pra cada um dos subcasos. Vamos tentar outra abordagem.

Agrupando semanticamente - ideal

```
In [36]: df_base_logs = df_base_logs.withColumn("plays_per_unq_behavior",
    F.when(F.col("flag_has_logs").isIn(0), "00_unknown")
    .when((F.col("plays_per_unq_cap") >= 0) & (F.col("plays_per_unq_cap") < 1.1), "01_explorer")
    .when((F.col("plays_per_unq_cap") >= 1.1) & (F.col("plays_per_unq_cap") < 1.5), "02_light_repeat")
    .when((F.col("plays_per_unq_cap") >= 1.5) & (F.col("plays_per_unq_cap") < 3.0), "03_repeat")
    .otherwise("04_heavy_repeat"))
```

```
In [123]: calcular_distribuicao(df_base_logs, ["plays_per_unq_behavior"])

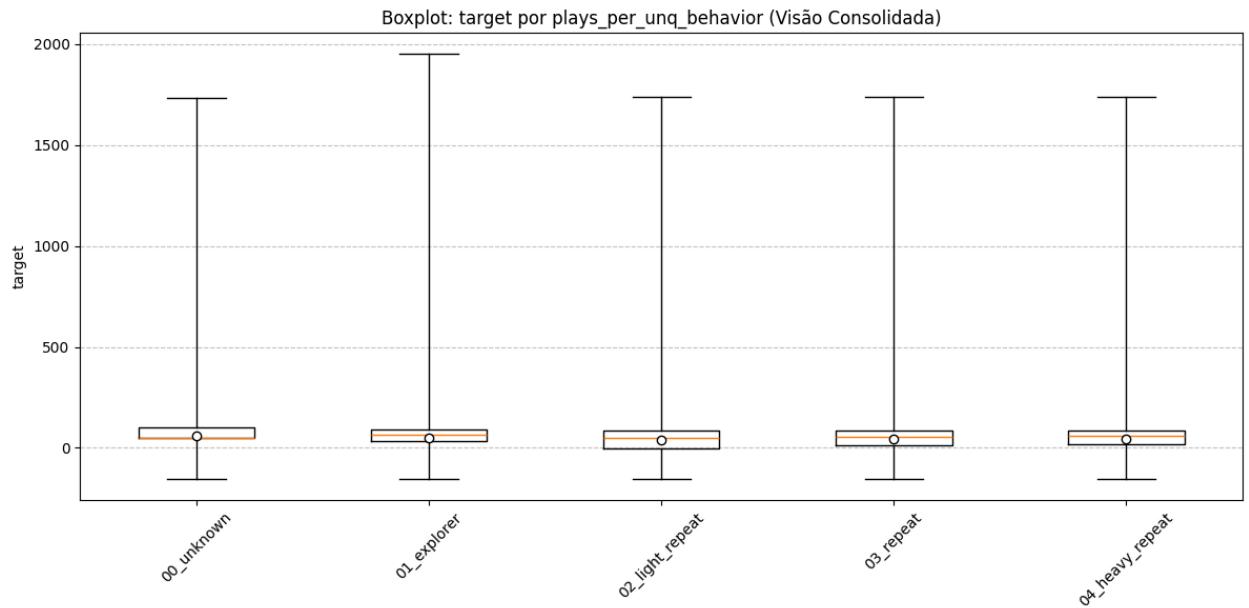
[123]:
```

plays_per_unq_behavior	total	pct_total
01_explorer	7634463	67.9
02_light_repeat	1470819	13.08
00_unknown	1266985	11.27
03_repeat	665352	5.92
04_heavy_repeat	205246	1.83

```
Out[123]: DataFrame[plays_per_unq_behavior: string, total: bigint, pct_total: double]
```

```
In [124]: plot_boxplot(df_base_logs, ["plays_per_unq_behavior"], "target", table=True)
```

Processando estatísticas para: plays_per_unq_behavior...



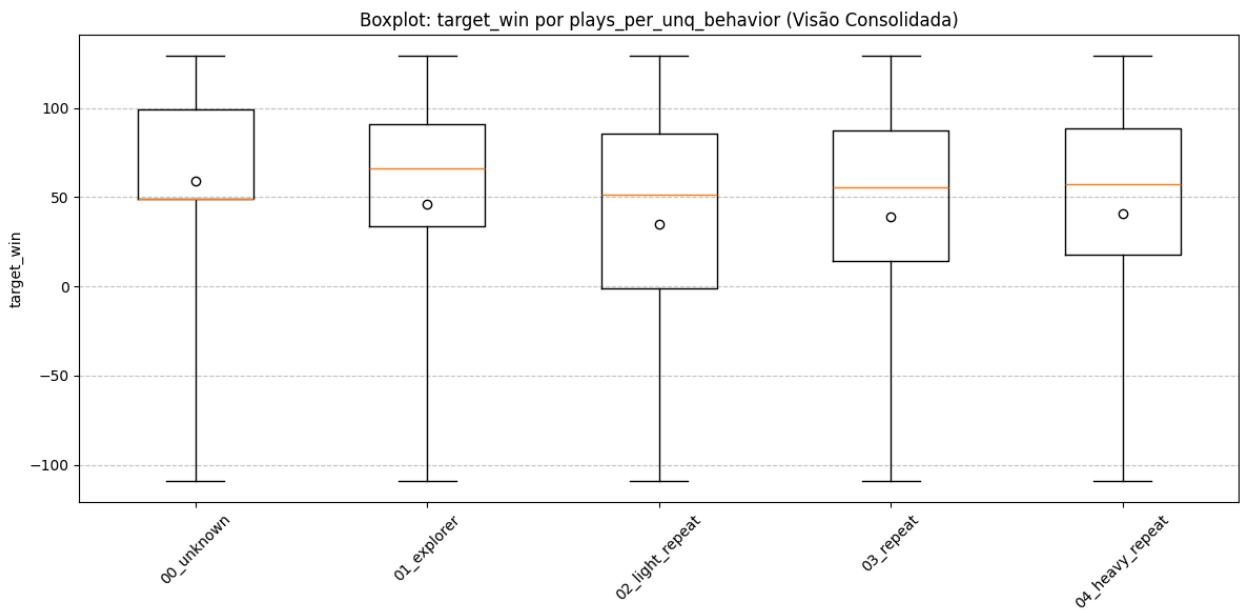
--- Estatísticas: plays_per_unq_behavior (Visão Consolidada) ---

	plays_per_unq_behavior	min	q1	med	mean	q3	max
4	00_unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386
1	01_explorer	-152.5452	33.464282	66.396955	50.099908	90.921769	1950.000000
2	02_light_repeat	-152.5452	-1.299418	51.556471	38.343983	85.778611	1736.835211
3	03_repeat	-152.5452	14.007144	55.626256	42.166822	87.161385	1736.374073
0	04_heavy_repeat	-152.5452	17.997014	57.453219	43.509551	88.354067	1737.041373

=====

```
In [125]: plot_boxplot(df_base_logs, ["plays_per_unq_behavior"], "target_win", table=True)
```

Processando estatísticas para: plays_per_unq_behavior...



--- Estatísticas: plays_per_unq_behavior (Visão Consolidada) ---

	plays_per_unq_behavior	min	q1	med	mean	q3	max
4	00_unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
1	01_explorer	-108.964925	33.464282	66.396955	45.817970	90.921769	128.953521
2	02_light_repeat	-108.964925	-1.299418	51.556471	34.983622	85.778611	128.953521
3	03_repeat	-108.964925	14.007144	55.626256	38.707490	87.161385	128.953521
0	04_heavy_repeat	-108.964925	17.997014	57.453219	40.866520	88.354067	128.953521

=====

Conclusao

O agrupamento por regime comportamental de repeticao se mostra mais adequado do que quartis porque plays_per_unq nao representa intensidade continua, mas sim padroes discretos de consumo. Quartis fragmentam artificialmente a populacao e misturam comportamentos distintos dentro da mesma classe, reduzindo interpretabilidade e poder explicativo. plays_per_unq_behavior captura regimes de comportamento que nao sao explicados por volume ou tempo total de consumo, oferecendo ganho de informacao complementar e interpretavel para todos os modelos considerados no projeto. Levar para feature engineering.

8.1.7. plays_behavior_vs_volume

Cruzamento entre plays_per_unq_behavior e total_plays_group . O objetivo: separar usuarios com mesmo volume total, mas comportamentos de repeticao diferentes.

Ex.: baixo volume + explorer, alto volume + heavy_repeat

```
In [42]: df_base_logs = df_base_logs.withColumn("plays_behavior_vs_volume", F.concat_ws("_", F.col("plays_per_unq_behavior"), F.col("total_plays_group")))
```

```
In [134]: calcular_distribuicao(df_base_logs, ["plays_behavior_vs_volume"], n_show=25)
```

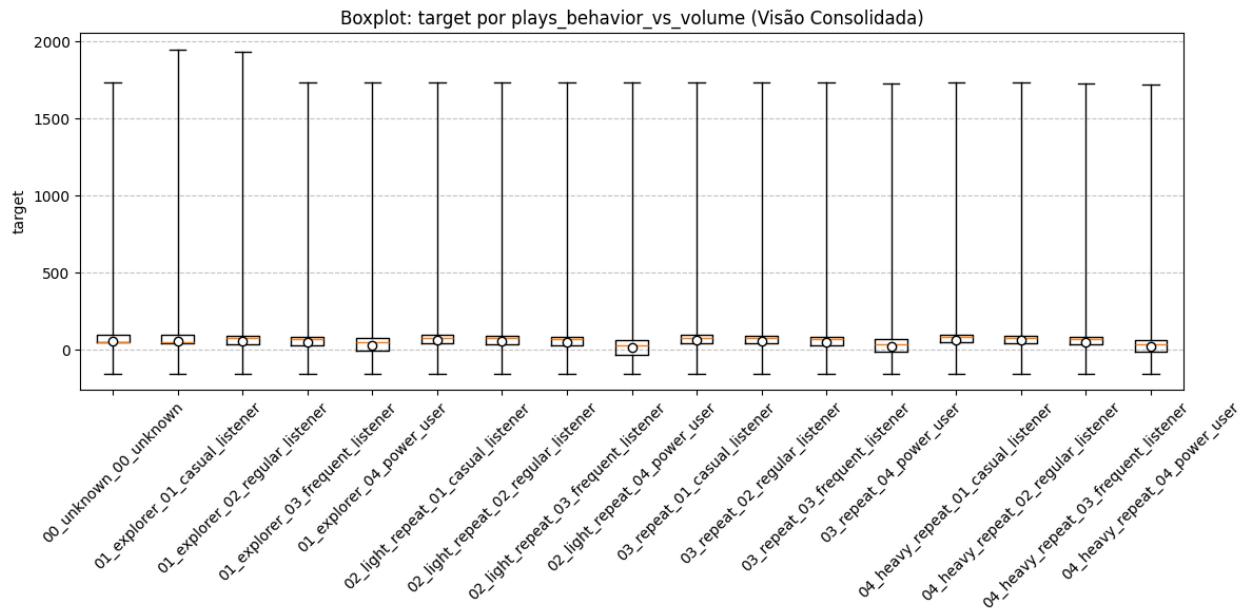
plays_behavior_vs_volume	total	pct_total
01_explorer_01_casual_listener	2179718 19.39	
01_explorer_02_regular_listener	2015185 17.92	
01_explorer_03_frequent_listener	1906038 16.95	
01_explorer_04_power_user	1533522 13.64	
00_unknown_00_unknown	1266985 11.27	
02_light_repeat_04_power_user	603188 5.37	
02_light_repeat_03_frequent_listener	387566 3.45	
02_light_repeat_02_regular_listener	304258 2.71	
03_repeat_04_power_user	271306 2.41	
02_light_repeat_01_casual_listener	175807 1.56	
03_repeat_03_frequent_listener	158865 1.41	
03_repeat_02_regular_listener	131869 1.17	
03_repeat_01_casual_listener	103312 0.92	
04_heavy_repeat_04_power_user	85671 0.76	
04_heavy_repeat_01_casual_listener	44056 0.39	
04_heavy_repeat_03_frequent_listener	37878 0.34	
04_heavy_repeat_02_regular_listener	37641 0.33	

```
Out[134]: DataFrame[plays_behavior_vs_volume: string, total: bigint, pct_total: double]
```

In plot_boxplot(df_base_logs, ["plays_behavior_vs_volume"], "target", table=True)

[135]:

Processando estatísticas para: plays_behavior_vs_volume...



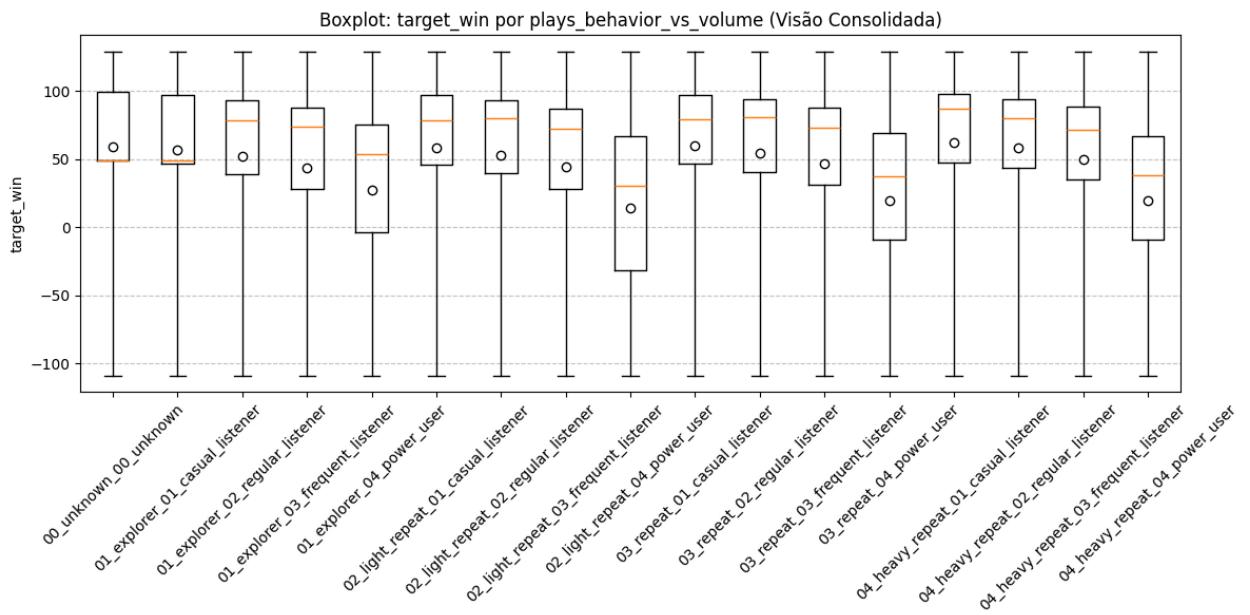
--- Estatísticas: plays_behavior_vs_volume (Visão Consolidada) ---

	plays_behavior_vs_volume	min	q1	med	mean	q3	max
6	00_unknown_00_unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386
11	01_explorer_01_casual_listener	-152.5452	46.214381	49.000000	58.630713	96.755751	1950.000000
0	01_explorer_02_regular_listener	-152.5452	39.169019	78.465469	55.869866	93.099059	1936.799455
2	01_explorer_03_frequent_listener	-152.5452	27.889035	73.665545	48.971521	87.266072	1736.541069
3	01_explorer_04_power_user	-152.5452	-4.038993	53.591662	32.711685	75.355310	1735.146209
7	02_light_repeat_01_casual_listener	-152.5452	45.752592	78.495220	60.927141	96.858051	1736.835211
12	02_light_repeat_02_regular_listener	-152.5452	39.286901	79.945016	56.538343	93.022058	1735.032432
5	02_light_repeat_03_frequent_listener	-152.5452	28.254581	72.061216	48.501377	86.614173	1734.140378
10	02_light_repeat_04_power_user	-152.5452	-31.820184	30.053257	16.579230	67.057576	1732.840396
16	03_repeat_01_casual_listener	-152.5452	46.390330	78.928849	61.471305	97.268736	1734.154509
14	03_repeat_02_regular_listener	-152.5452	40.442198	80.878012	57.599496	93.593744	1736.374073
13	03_repeat_03_frequent_listener	-152.5452	30.754320	73.040431	50.698812	87.455888	1734.872871
9	03_repeat_04_power_user	-152.5452	-9.279213	37.507139	23.025309	69.352397	1731.871480
15	04_heavy_repeat_01_casual_listener	-152.5452	47.444377	87.161739	63.977910	97.629844	1737.041373
4	04_heavy_repeat_02_regular_listener	-152.5452	43.205639	80.242414	60.700417	93.894581	1736.048807
1	04_heavy_repeat_03_frequent_listener	-152.5452	34.620558	71.019404	52.716810	88.030601	1726.230960
8	04_heavy_repeat_04_power_user	-152.5452	-8.875781	37.820981	22.194435	66.747661	1723.032881

=====

```
In [136]: plot_boxplot(df_base_logs, ["plays_behavior_vs_volume"], "target_win", table=True)
```

Processando estatísticas para: plays_behavior_vs_volume...



--- Estatísticas: plays_behavior_vs_volume (Visão Consolidada) ---

	plays_behavior_vs_volume	min	q1	med	mean	q3	max
6	00_unknown_00_unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
11	01_explorer_01_casual_listener	-108.964925	46.214381	49.000000	56.269536	96.755751	128.953521
0	01_explorer_02_regular_listener	-108.964925	39.169019	78.465469	51.668398	93.099059	128.953521
2	01_explorer_03_frequent_listener	-108.964925	27.889035	73.665545	43.739824	87.266072	128.953521
3	01_explorer_04_power_user	-108.964925	-4.038993	53.591662	26.960411	75.355310	128.953521
7	02_light_repeat_01_casual_listener	-108.964925	45.752592	78.495220	58.476126	96.858051	128.953521
12	02_light_repeat_02_regular_listener	-108.964925	39.286901	79.945016	53.030498	93.022058	128.953521
5	02_light_repeat_03_frequent_listener	-108.964925	28.254581	72.061216	44.200633	86.614173	128.953521
10	02_light_repeat_04_power_user	-108.964925	-31.820184	30.053257	13.641503	67.057576	128.953521
16	03_repeat_01_casual_listener	-108.964925	46.390330	78.928849	59.570182	97.268736	128.953521
14	03_repeat_02_regular_listener	-108.964925	40.442198	80.878012	54.338200	93.593744	128.953521
13	03_repeat_03_frequent_listener	-108.964925	30.754320	73.040431	46.400770	87.455888	128.953521
9	03_repeat_04_power_user	-108.964925	-9.279213	37.507139	19.406706	69.352397	128.953521
15	04_heavy_repeat_01_casual_listener	-108.964925	47.444377	87.161739	62.276180	97.629844	128.953521
4	04_heavy_repeat_02_regular_listener	-108.964925	43.205639	80.242414	58.312439	93.894581	128.953521
1	04_heavy_repeat_03_frequent_listener	-108.964925	34.620558	71.019404	49.644019	88.030601	128.953521
8	04_heavy_repeat_04_power_user	-108.964925	-8.875781	37.820981	19.177723	66.747661	128.953521

=====

À primeira vista, a variável `plays_behavior_vs_volume` pode parecer over engineering, por ser uma junção direta de duas variáveis já existentes (`plays_per_unq_behavior` e `volume / perfil de consumo`). No entanto, as variáveis originais, isoladamente, não capturam o mesmo fenômeno que o cruzamento revela. Embora os valores medios entre algumas classes sejam próximos, a variável cruzada explicita uma interação não linear que não é aditiva não é monotonica e não seria bem representada por coeficientes lineares separados. Para modelos bagging e boosting, se mostra bem construída, mas exige adaptações para modelos lineares.

Mantendo a variável como esta, permite acelerar o aprendizado ao explicitar a interação logo no primeiro nível da árvore, ou seja, reduz a profundidade necessária para capturar o padrão, o que significa menos splits = menor latência de inferência. O que podemos aplicar, diante da distribuição da variável, seria colapsar categorias raras (menos de 1% da base), para melhor performance da RF.

```
In [44]: # Colapsar categorias raras (< 1% da base) para RF/QRF
# LightGBM lida bem com raras, mas RF pode sofrer em bootstrap

total_logs = df_base_logs.count()

threshold = 0.01 # 1%
freq_table = df_base_logs.groupBy("plays_behavior_vs_volume").count()
freq_table = freq_table.withColumn("pct", F.col("count") / total_logs)

rare_categories = (freq_table
    .filter(F.col("pct") < threshold)
    .select("plays_behavior_vs_volume")
    .rdd.flatMap(lambda x: x).collect())

df_base_logs = df_base_logs.withColumn("plays_behavior_vs_volume_collapsed",
    F.when(F.col("plays_behavior_vs_volume").isin(rare_categories), "99_other")
    .otherwise(F.col("plays_behavior_vs_volume")))
```

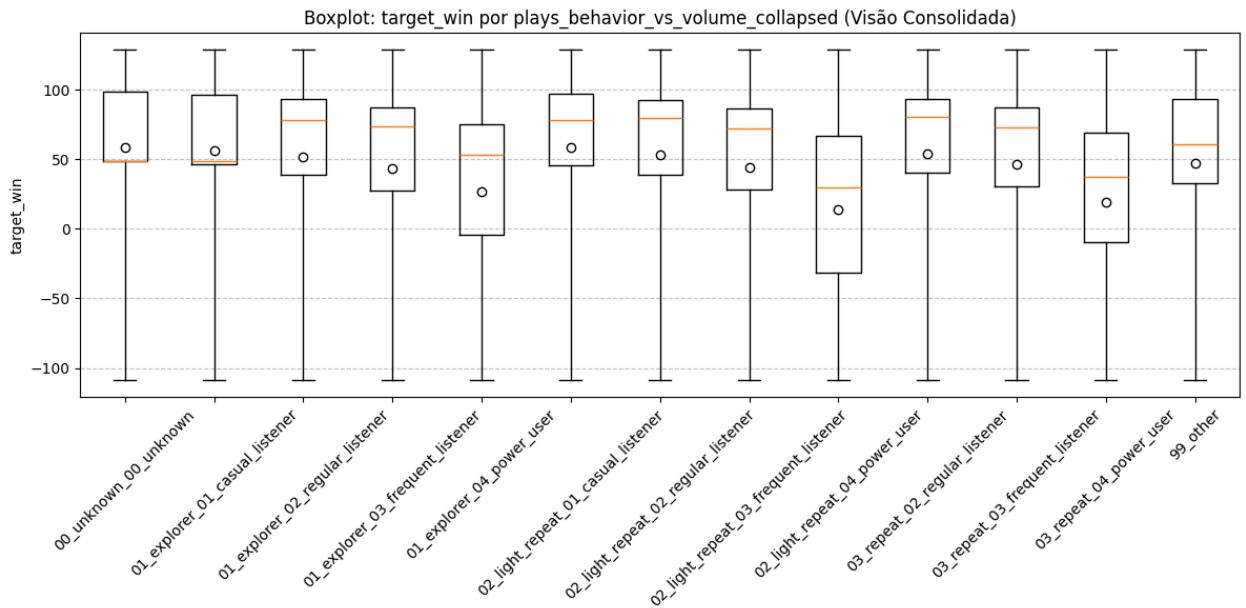
```
In [45]: calcular_distribuicao(df_base_logs, ["plays_behavior_vs_volume_collapsed"], n_show=25)
```

plays_behavior_vs_volume_collapsed	total	pct_total
01_explorer_01_casual_listener	2179718 19.39	
01_explorer_02_regular_listener	2015185 17.92	
01_explorer_03_frequent_listener	1906038 16.95	
01_explorer_04_power_user	1533522 13.64	
00_unknown_00_unknown	1266985 11.27	
02_light_repeat_04_power_user	603188 5.37	
02_light_repeat_03_frequent_listener	387566 3.45	
99_other	308558 2.74	
02_light_repeat_02_regular_listener	304258 2.71	
03_repeat_04_power_user	271306 2.41	
02_light_repeat_01_casual_listener	175807 1.56	
03_repeat_03_frequent_listener	158865 1.41	
03_repeat_02_regular_listener	131869 1.17	

Out[45]: DataFrame[plays_behavior_vs_volume_collapsed: string, total: bigint, pct_total: double]

```
In [52]: plot_boxplot(df_base_logs, ["plays_behavior_vs_volume_collapsed"], "target_win", table=True)
```

Processando estatísticas para: plays_behavior_vs_volumeCollapsed...



--- Estatísticas: plays_behavior_vs_volume_collapsed (Visão Consolidada) ---

	plays_behavior_vs_volume_collapsed	min	q1	med	mean	q3	max
4	00_unknown_00_unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
9	01_explorer_01_casual_listener	-108.964925	46.214381	49.000000	56.269536	96.755751	128.953521
0	01_explorer_02_regular_listener	-108.964925	39.169019	78.465469	51.668398	93.099059	128.953521
1	01_explorer_03_frequent_listener	-108.964925	27.889035	73.665545	43.739824	87.266072	128.953521
2	01_explorer_04_power_user	-108.964925	-4.038993	53.591662	26.960411	75.355310	128.953521
6	02_light_repeat_01_casual_listener	-108.964925	45.752592	78.495220	58.476126	96.858051	128.953521
10	02_light_repeat_02_regular_listener	-108.964925	39.286901	79.945016	53.030498	93.022058	128.953521
3	02_light_repeat_03_frequent_listener	-108.964925	28.254581	72.061216	44.200633	86.614173	128.953521
8	02_light_repeat_04_power_user	-108.964925	-31.820184	30.053257	13.641503	67.057576	128.953521
12	03_repeat_02_regular_listener	-108.964925	40.442198	80.878012	54.338200	93.593744	128.953521
11	03_repeat_03_frequent_listener	-108.964925	30.754320	73.040431	46.400770	87.455888	128.953521
7	03_repeat_04_power_user	-108.964925	-9.279213	37.507139	19.406706	69.352397	128.953521
5	99_other	-108.964925	32.961968	60.507344	46.984186	93.661620	128.953521

=====

Pra Elastic Net, vou optar pelo One Hot Encoding das duas variaveis raw, dado que algum outro tratamento seria trazer complexidade que pode se mostrar desnecessaria.

8.1.8. plays_behavior_vs_completion

Cruzamento entre plays_per_unq_behaviour e completed_songs_rate_group . O objetivo: diferenciar exploracao superficial de exploracao "qualificada".

```
In [49]: df_base_logs = df_base_logs.withColumn("plays_behavior_vs_completion", F.concat_ws("_", F.col("plays_per_unq_behavior"), F.col("completed_songs_rate_group")))
```

```
In [149]: calcular_distribuicao(df_base_logs, ["plays_behavior_vs_completion"], n_show=25)
```

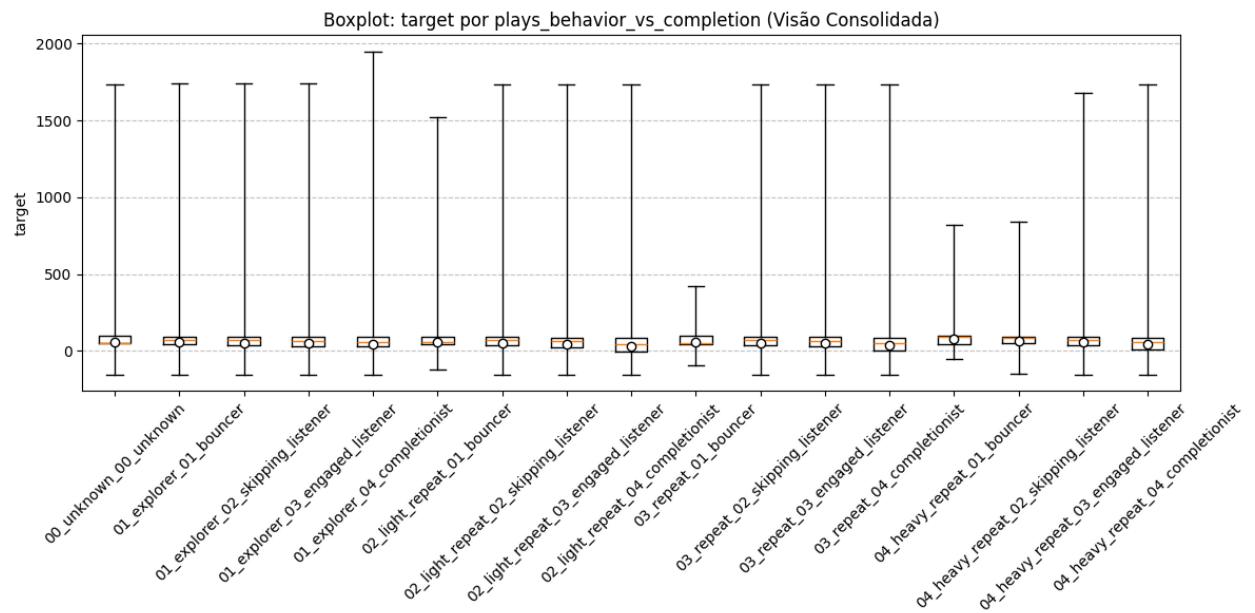
plays_behavior_vs_completion	total	pct_total
01_explorer_02_skipping_listener	2582106 22.97	
01_explorer_03_engaged_listener	2085466 18.55	
01_explorer_04_completionist	1572874 13.99	
01_explorer_01_bouncer	1394017 12.4	
00_unknown_00_unknown	1266985 11.27	
02_light_repeat_04_completionist	831674 7.4	
02_light_repeat_03_engaged_listener	474983 4.22	
03_repeat_04_completionist	393622 3.5	
03_repeat_03_engaged_listener	209330 1.86	
04_heavy_repeat_04_completionist	166684 1.48	
02_light_repeat_02_skipping_listener	160246 1.43	
03_repeat_02_skipping_listener	61204 0.54	
04_heavy_repeat_03_engaged_listener	35810 0.32	
02_light_repeat_01_bouncer	3916 0.03	
04_heavy_repeat_02_skipping_listener	2651 0.02	
03_repeat_01_bouncer	1196 0.01	
04_heavy_repeat_01_bouncer	101 0.0	

```
Out[149]: DataFrame[plays_behavior_vs_completion: string, total: bigint, pct_total: double]
```

In [145]:

```
plot_boxplot(df_base_logs, ["plays_behavior_vs_completion"], "target", table=True)
```

Processando estatísticas para: plays_behavior_vs_completion...



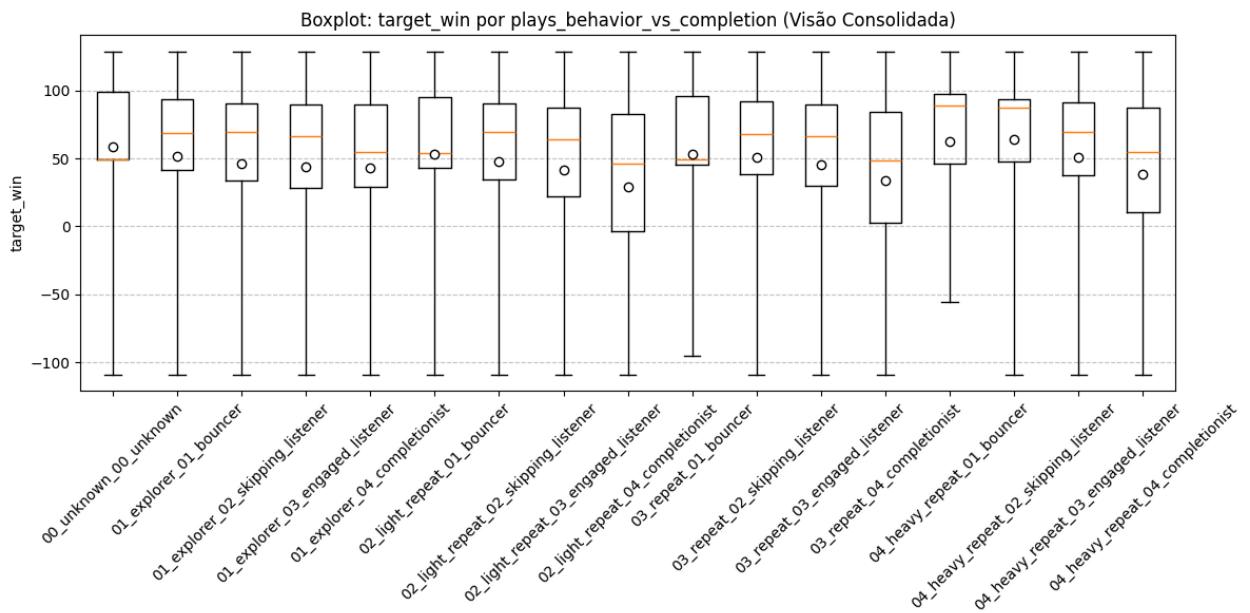
--- Estatísticas: plays_behavior_vs_completion (Visão Consolidada) ---

	plays_behavior_vs_completion	min	q1	med	mean	q3	max
9	00_unknown_00_unknown	-152.545200	49.000000	49.000000	59.177692	99.000000	1733.833386
14	01_explorer_01_bouncer	-152.545200	41.904914	68.881134	55.355141	93.863099	1737.947047
0	01_explorer_02_skipping_listener	-152.545200	33.623792	69.655266	51.003020	90.729018	1737.859653
1	01_explorer_03_engaged_listener	-152.545200	28.703267	66.668945	48.053611	89.598966	1737.929378
15	01_explorer_04_completionist	-152.545200	29.448625	54.522620	46.744548	90.084392	1950.000000
13	02_light_repeat_01_bouncer	-121.131346	42.927304	53.893583	55.613547	95.061518	1518.763682
7	02_light_repeat_02_skipping_listener	-152.545200	34.942259	69.217471	51.981705	90.563214	1735.032432
12	02_light_repeat_03_engaged_listener	-152.545200	21.829411	64.133545	45.666006	87.607069	1733.827442
10	02_light_repeat_04_completionist	-152.545200	-3.545200	46.113688	31.571402	82.814711	1736.835211
8	03_repeat_01_bouncer	-95.561743	45.556342	49.000000	53.887820	95.794939	423.247290
6	03_repeat_02_skipping_listener	-152.545200	38.638552	68.285445	53.886423	92.020910	1736.374073
3	03_repeat_03_engaged_listener	-152.545200	29.738476	66.357642	49.209238	89.616214	1734.314575
11	03_repeat_04_completionist	-152.545200	2.603474	48.940250	36.721866	84.374551	1735.612685
4	04_heavy_repeat_01_bouncer	-55.654617	45.896249	88.984306	79.307562	97.682831	817.826907
16	04_heavy_repeat_02_skipping_listener	-145.961100	48.162212	87.234535	66.521618	93.519566	843.339009
2	04_heavy_repeat_03_engaged_listener	-152.545200	37.799963	69.285528	54.072363	91.556933	1680.134612
5	04_heavy_repeat_04_completionist	-152.545200	10.632800	54.615218	40.912316	87.082652	1737.041373

=====

```
In [146]: plot_boxplot(df_base_logs, ["plays_behavior_vs_completion"], "target_win", table=True)
```

Processando estatísticas para: plays_behavior_vs_completion...



--- Estatísticas: plays_behavior_vs_completion (Visão Consolidada) ---

	plays_behavior_vs_completion	min	q1	med	mean	q3	max
9	00_unknown_00_unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
14	01_explorer_01_bouncer	-108.964925	41.904914	68.881134	51.342202	93.863099	128.953521
0	01_explorer_02_skipping_listener	-108.964925	33.623792	69.655266	46.338807	90.729018	128.953521
1	01_explorer_03_engaged_listener	-108.964925	28.703267	66.668945	43.554958	89.598966	128.953521
15	01_explorer_04_completionist	-108.964925	29.448625	54.522620	43.159527	90.084392	128.953521
13	02_light_repeat_01_bouncer	-108.964925	42.927304	53.893583	53.164761	95.061518	128.953521
7	02_light_repeat_02_skipping_listener	-108.964925	34.942259	69.217471	48.001702	90.563214	128.953521
12	02_light_repeat_03_engaged_listener	-108.964925	21.829411	64.133545	41.431446	87.607069	128.953521
10	02_light_repeat_04_completionist	-108.964925	-3.545200	46.113688	28.818655	82.814711	128.953521
8	03_repeat_01_bouncer	-95.561743	45.556342	49.000000	53.179968	95.794939	128.953521
6	03_repeat_02_skipping_listener	-108.964925	38.638552	68.285445	50.609403	92.020910	128.953521
3	03_repeat_03_engaged_listener	-108.964925	29.738476	66.357642	45.354010	89.616214	128.953521
11	03_repeat_04_completionist	-108.964925	2.603474	48.940250	33.436580	84.374551	128.953521
4	04_heavy_repeat_01_bouncer	-55.654617	45.896249	88.984306	62.549088	97.682831	128.953521
16	04_heavy_repeat_02_skipping_listener	-108.964925	48.162212	87.234535	64.108226	93.519566	128.953521
2	04_heavy_repeat_03_engaged_listener	-108.964925	37.799963	69.285528	51.258640	91.556933	128.953521
5	04_heavy_repeat_04_completionist	-108.964925	10.632800	54.615218	38.308231	87.082652	128.953521

=====

Observações-chave:

- > Inversão de Monotonidade:
 - explorer_bouncer: 55.4 (média)
 - explorer_skipping: 51.0
 - explorer_engaged: 48.1
 - explorer_completionist: 46.7: MENOR margem

Contra-intuitivo: Quem completa músicas tem margem menor que quem pula. Explicação provável: Completionists ouvem mais segundos, implicando maior custo variável (formula dada).

> Heavy Repeat + Bouncer = Anomalia:

- heavy_repeat_bouncer: 79.3 (média) com apenas 101 casos. Margem altíssima, mas estatisticamente instável. Pode ser ruído ou comportamento de teste/bot.

> Padrão Geral:

- bouncer → margens altas (baixo consumo de segundos)
- completionist → margens médias/baixas (alto consumo)
- Mas a interação com plays_behavior não é aditiva

A variável `completed_songs_rate_group` apresenta relação não-monotônica com a target devido à estrutura de custo. Usuários 'completionists' têm maior consumo de segundos (t), elevando o custo variável. Modelos lineares (Elastic Net) não capturam essa inversão sem transformações adicionais.

Vamos tratar a variável também, considerando o que foi aplicado na variável acima (união das classes que representam menos de 1% da base para LGBM e RF, OHE para Elastic Net):

```
In [50]: freq_table = df_base_logs.groupBy("plays_behavior_vs_completion").count()
freq_table = freq_table.withColumn("pct", F.col("count") / total_logs)
```

```
rare_categories = (freq_table
    .filter(F.col("pct") < threshold)
    .select("plays_behavior_vs_completion")
    .rdd.flatMap(lambda x: x).collect())
```

```
df_base_logs = df_base_logs.withColumn("plays_behavior_vs_completion_collapsed",
    F.when(F.col("plays_behavior_vs_completion").isin(rare_categories), "99_other")
    .otherwise(F.col("plays_behavior_vs_completion")))
```

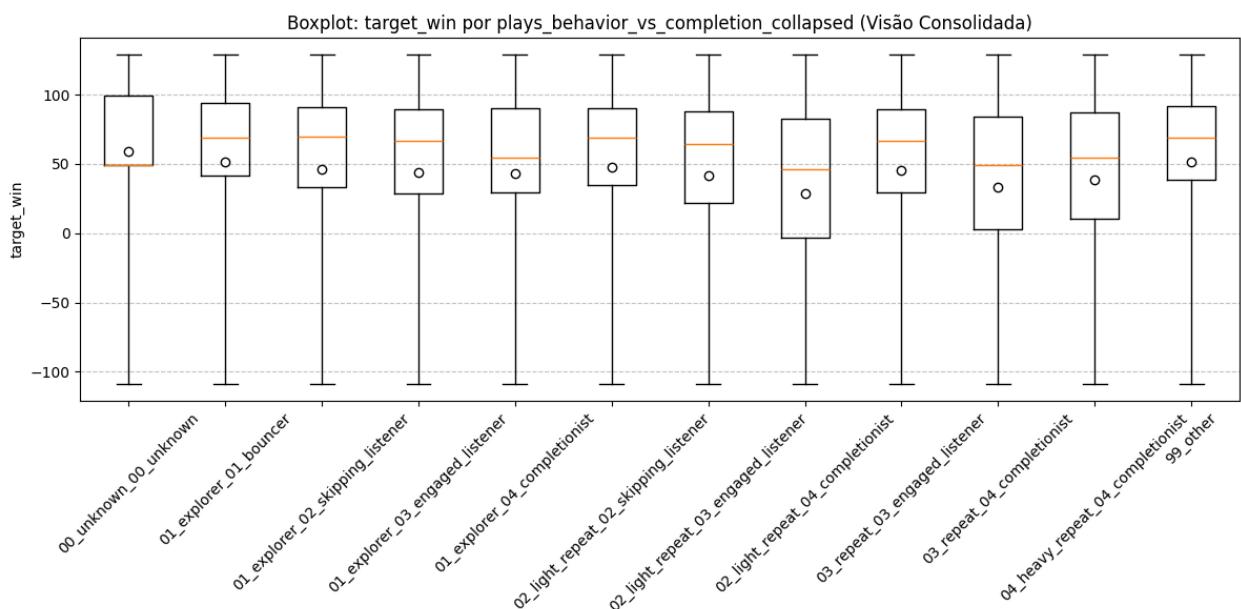
```
In [53]: calcular_distribuicao(df_base_logs, ["plays_behavior_vs_completion_collapsed"], n_show=25)
```

	total	pct_total
01_explorer_02_skipping_listener	2582106 22.97	
01_explorer_03_engaged_listener	2085466 18.55	
01_explorer_04_completionist	1572874 13.99	
01_explorer_01_bouncer	1394017 12.4	
00_unknown_00_unknown	1266985 11.27	
02_light_repeat_04_completionist	831674 7.4	
02_light_repeat_03_engaged_listener	474983 4.22	
03_repeat_04_completionist	393622 3.5	
03_repeat_03_engaged_listener	209330 1.86	
04_heavy_repeat_04_completionist	166684 1.48	
02_light_repeat_02_skipping_listener	160246 1.43	
99_other	104878 0.93	

```
Out[53]: DataFrame[plays_behavior_vs_completion_collapsed: string, total: bigint, pct_total: double]
```

```
In [54]: plot_boxplot(df_base_logs, ["plays_behavior_vs_completion_collapsed"], "target_win", table=True)
```

Processando estatísticas para: plays_behavior_vs_completion_collapsed...



--- Estatísticas: plays_behavior_vs_completion_collapsed (Visão Consolidada) ---

	plays_behavior_vs_completion_collapsed	min	q1	med	mean	q3	max
5	00_unknown_00_unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
10	01_explorer_01_bouncer	-108.964925	41.904914	68.881134	51.342202	93.863099	128.953521
0	01_explorer_02_skipping_listener	-108.964925	33.623792	69.655266	46.338807	90.729018	128.953521
1	01_explorer_03_engaged_listener	-108.964925	28.703267	66.668945	43.554958	89.598966	128.953521
11	01_explorer_04_completionist	-108.964925	29.448625	54.522620	43.159527	90.084392	128.953521
4	02_light_repeat_02_skipping_listener	-108.964925	34.942259	69.217471	48.001702	90.563214	128.953521
9	02_light_repeat_03_engaged_listener	-108.964925	21.829411	64.133545	41.431446	87.607069	128.953521
7	02_light_repeat_04_completionist	-108.964925	-3.545200	46.113688	28.818655	82.814711	128.953521
2	03_repeat_03_engaged_listener	-108.964925	29.738476	66.357642	45.354010	89.616214	128.953521
8	03_repeat_04_completionist	-108.964925	2.603474	48.940250	33.436580	84.374551	128.953521
3	04_heavy_repeat_04_completionist	-108.964925	10.632800	54.615218	38.308231	87.082652	128.953521
6	99_other	-108.964925	38.853453	69.004259	51.279754	92.095744	128.953521

=====

8.1.9. catalog_exploration_ratio

Índice de exploração do catálogo. Se num_unq = 100 e num_100 = 10, significa que o usuário explorou 100 músicas diferentes, mas só 10 delas ele ouviu até o fim, apresentando risco de custo: o usuário é o "pesadelo" da lucratividade. Ele gera custo de únicas em 100 músicas, mas só gera retenção/engajamento real em 10.

```
In [46]: df_base_logs = df_base_logs.withColumn("catalog_exploration_ratio", F.when(F.col("num_100") > 0, F.col("num_unq") / F.col("num_100")).otherwise(0.0))
```

Descritivas + correlação com target

```
In [55]: df_base_logs.select("catalog_exploration_ratio").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary catalog_exploration_ratio	
count 11242865	
mean 1.2416204554900776	
stddev 2.733478915236974	
min 0.0	
1% 0.0	
5% 0.0	
25% 0.7872340425531915	
50% 1.0569620253164558	
75% 1.352189781021898	
95% 2.5555555555555554	
99.5% 9.666666666666666	
max 1409.5	

A média é 1.24, mas o Max de 1409 indica um outlier extremo ou um bot de exploração. O 75º percentil (1.35) mostra que a maioria é estável.

```
In [56]: df_base_logs.select("catalog_exploration_ratio", "target").corr("catalog_exploration_ratio", "target")
```

```
Out[56]: 0.007390192064511299
```

```
In [57]: df_base_logs.select("catalog_exploration_ratio", "target_win").corr("catalog_exploration_ratio", "target_win")
```

```
Out[57]: 0.007591387615717569
```

Conclusao

Embora a correlação linear seja baixa (0.007), ela captura o custo de aquisição de catálogo. Um usuário com ratio alto consome muitos num_unq (carros) para poucos total_secs (baratos), o que achata a margem de forma não linear. Vamos somente winsorizar para tratar o outlier encontrado.

```
In [ ]: # Calcular o p99.5 de catalog_exploration_ratio
p995_value = df_base_logs.approxQuantile("catalog_exploration_ratio", [0.995], 0.001)[0]

# # Winsorizar no p99.5
df_base_logs = df_base_logs.withColumn("catalog_exploration_ratio_cap",
    F.when(F.col("catalog_exploration_ratio") > p995_value, p995_value)
    .otherwise(F.col("catalog_exploration_ratio")))
```

8.1.10. early_drop_rate

Mede a proporção de rejeição imediata. Valor alto = usuário abandona músicas cedo

```
In [44]: df_base_logs = df_base_logs.withColumn("early_drop_rate",
    F.when(F.col("flag_has_logs").isin(1), ((F.col("num_25") + F.col("num_50")) / F.col("total_plays"))).otherwise(0.0))
```

Descritivas + correlacao com target

```
In [74]: df_base_logs.select("early_drop_rate").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary early_drop_rate	
count 11242865	
mean 0.2187835716667449	
stddev 0.1924394433941058	
min 0.0	
1% 0.0	
5% 0.0	
25% 0.07142857142857142	
50% 0.1795774647887324	
75% 0.31841216216216217	
95% 0.5870967741935483	
99.5% 1.0	
max 1.0	

- * Média 0.2188: Usuário típico abandona ~22% das músicas antes de 50%
- * Mediana: 0.1796: Metade dos usuários abandona < 18%
- * Distribuição bem espalhada de 0 a 1

```
In [75]: df_base_logs.select("early_drop_rate", "target").corr("early_drop_rate", "target")
```

```
Out[75]: 0.027585258493752995
```

```
In [76]: df_base_logs.select("early_drop_rate", "target_win").corr("early_drop_rate", "target_win")
```

```
Out[76]: 0.029375331701637936
```

Agrupamento por buckets semanticos

Entendo que a distribuição por quartis não se enquadraria tão bem aqui, então me basearei em benchmarks de mercado de plataformas similares.

```
In [46]: df_base_logs = df_base_logs.withColumn(
    "early_drop_rate_group",
    F.when(F.col("flag_has_logs").isin(0), "00_unknown")
    .when(F.col("early_drop_rate") == 0.0, "01_completionist") # Ouvinte Passivo/Fiel: Ouviu 100% do que começo.
    .when(F.col("early_drop_rate") <= 0.20, "02_engaged") # Ouvinte Engajado: Pula até 1 em cada 5 músicas.
    .when(F.col("early_drop_rate") <= 0.5, "03_explorer") # Explorador: Pula entre 20% e 50%. Está ativamente buscando novas
    músicas ou pulando o que não conhece.
    .otherwise("04_skipping_heavy")) # Skipper Crônico: Pula mais da metade do que começo
```

```
In [49]: calcular_distribuicao(df_base_logs, ["early_drop_rate_group"], n_show=5)
```

early_drop_rate_group	total	pct_total
03_mid_drop	4019658 35.75	
02_low_drop	3461894 30.79	
04_high_drop	2344556 20.85	
00_unknown	1266985 11.27	
01_no_drops	149772 1.33	

```
Out[49]: DataFrame[early_drop_rate_group: string, total: bigint, pct_total: double]
```

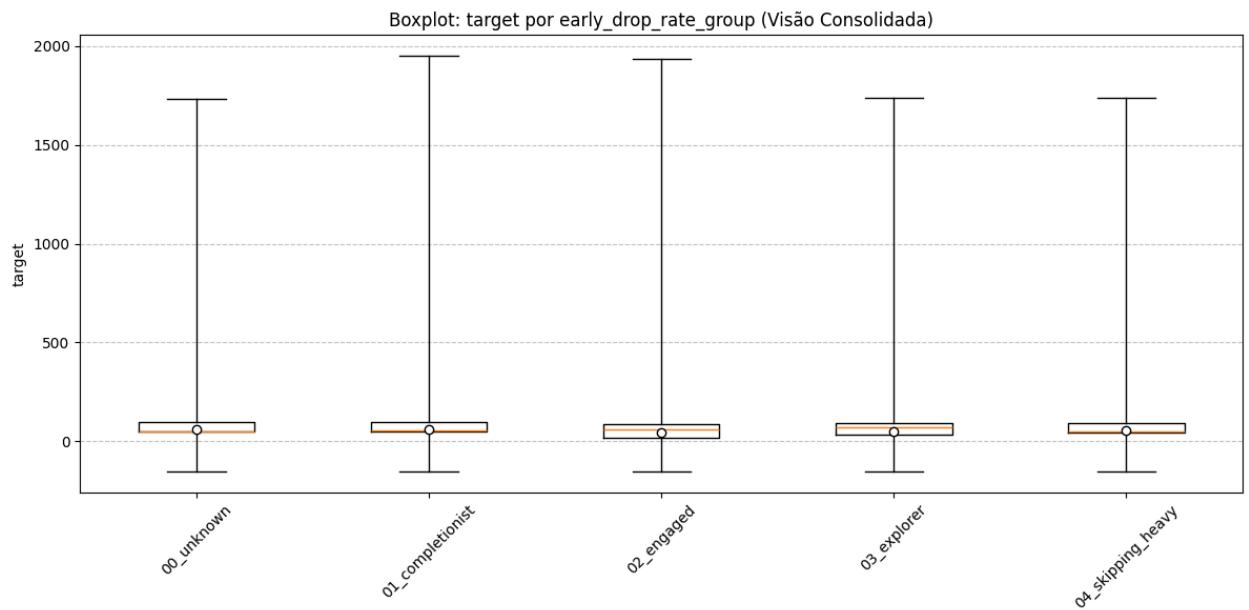
```
In [47]: calcular_distribuicao(df_base_logs, ["early_drop_rate_group"], n_show=5)
```

early_drop_rate_group	total	pct_total
02_engaged	4713736 41.93	
03_explorer	4179512 37.17	
00_unknown	1266985 11.27	
04_skipping_heavy	932860 8.3	
01_completionist	149772 1.33	

```
Out[47]: DataFrame[early_drop_rate_group: string, total: bigint, pct_total: double]
```

```
In [ ]: plot_boxplot(df_base_logs, ["early_drop_rate_group"], "target", table=True)
```

Processando estatísticas para: early_drop_rate_group...



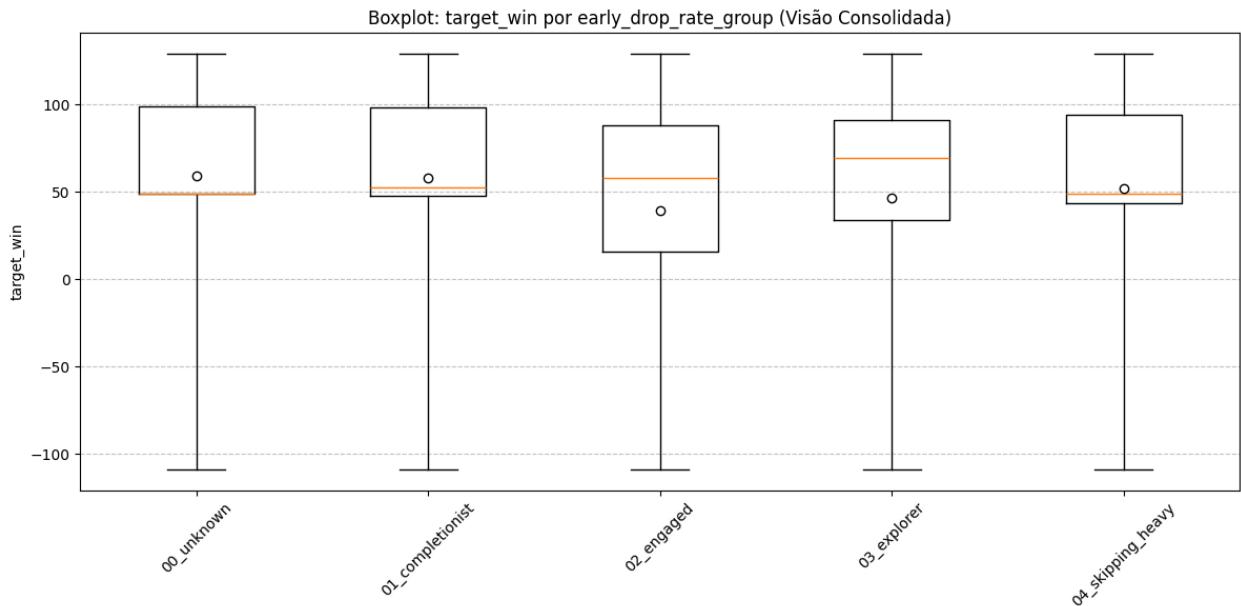
--- Estatísticas: early_drop_rate_group (Visão Consolidada) ---

	early_drop_rate_group	min	q1	med	mean	q3	max
3	00_unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386
4	01_completionist	-152.5452	47.909256	52.531569	59.778722	98.079942	1950.000000
0	02_engaged	-152.5452	15.447453	57.945166	43.065701	88.114122	1936.799455
1	03_explorer	-152.5452	33.682963	69.070284	50.960407	90.766410	1737.947047
2	04_skipping_heavy	-152.5452	43.751409	49.000000	55.236123	94.193216	1737.640612

=====

```
In [ ]: plot_boxplot(df_base_logs, ["early_drop_rate_group"], "target_win", table=True)
```

Processando estatísticas para: early_drop_rate_group...



--- Estatísticas: early_drop_rate_group (Visão Consolidada) ---

	early_drop_rate_group	min	q1	med	mean	q3	max
3	00_unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
4	01_completionist	-108.964925	47.909256	52.531569	58.109204	98.079942	128.953521
0	02_engaged	-108.964925	15.447453	57.945166	39.299785	88.114122	128.953521
1	03_explorer	-108.964925	33.682963	69.070284	46.388026	90.766410	128.953521
2	04_skipping_heavy	-108.964925	43.751409	49.000000	51.688403	94.193216	128.953521

=====

Conforme o early_drop_rate aumenta, a margem melhora. Isso ocorre porque o custo de streaming é altamente sensível ao tempo total escutado (total_secs). O usuário 'skipper' gera muitas transações de início de música, mas como ele abandona a maioria rapidamente, o custo acumulado de tempo é baixo, preservando a margem líquida.

Conclusao

Levar a variavel para feature engineering, considerando que traz uma diferenciacao muito boa entre quem paga mais e menos atraves da atividade (musicas terminadas entre 25% e 50% de sua duracao)

8.1.11. completion_efficiency

Engajamento profundo. Proporção de músicas completadas entre as que chegaram perto do final

```
In [50]: df_base_logs = df_base_logs.withColumn("completion_efficiency",
    F.when(F.col("flag_has_logs").isin(1), (F.col("num_100") / (F.col("num_75") + F.col("num_985") +
    F.col("num_100")))).otherwise(0.0))
```

Descritivas + correlacao com target

```
In [87]: df_base_logs.select("completion_efficiency").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary completion_efficiency
+-----+-----+
count 11162609
mean 0.7935542094731889
stddev 0.3082002886413479
min 0.0
1% 0.0
5% 0.0
25% 0.830379746835443
50% 0.9187935034802784
75% 0.9606060606060606
95% 0.9933692498963945
99.5% 1.0
max 1.0
+-----+-----+

Pela distribuicao, aqui talvez faça mais sentido trabalhar com uma flag do que com agrupamento para mais de uma categoria

```
In [88]: df_base_logs.select("completion_efficiency", "target").corr("completion_efficiency", "target")
```

```
Out[88]: -0.026618318843222976
```

```
In [89]: df_base_logs.select("completion_efficiency", "target_win").corr("completion_efficiency", "target_win")
```

```
Out[89]: -0.054077500832352896
```

flag_near_completion

```
In [51]: df_base_logs = df_base_logs.withColumn("flag_near_completion", F.when(F.col("completion_efficiency") > 0.0, 1).otherwise(0))
```

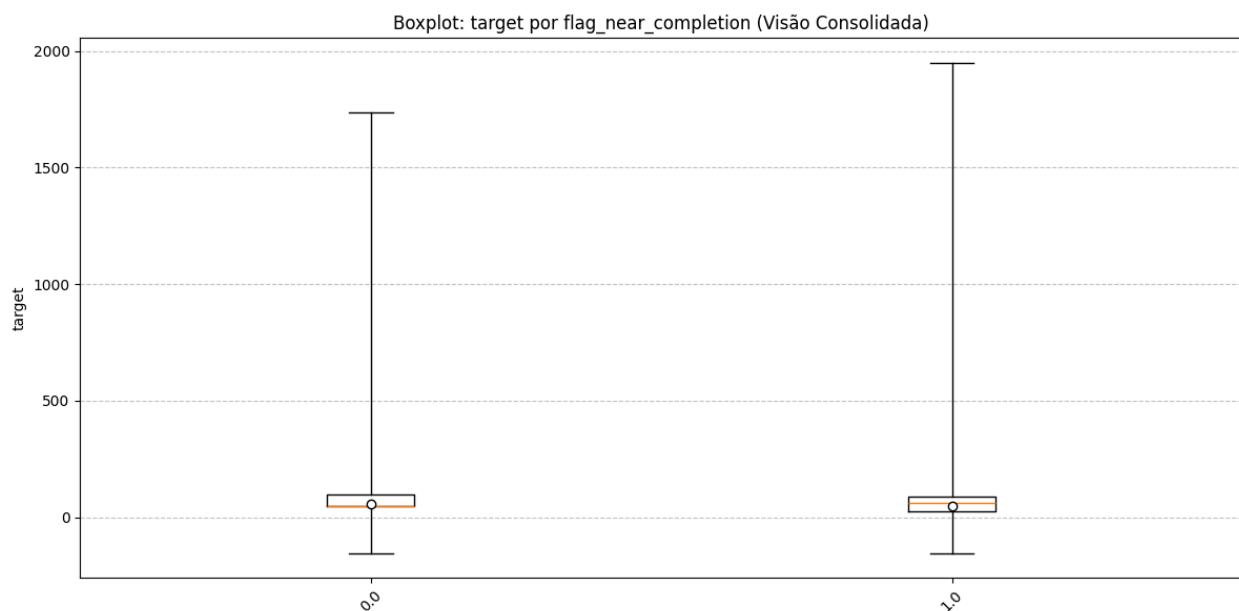
```
In [52]: calcular_distribuicao(df_base_logs, ["flag_near_completion"], n_show=2)
```

flag_near_completion total pct_total
+-----+-----+-----+
1 9833018 87.46
0 1409847 12.54
+-----+-----+-----+

```
Out[52]: DataFrame[flag_near_completion: int, total: bigint, pct_total: double]
```

```
In [53]: plot_boxplot(df_base_logs, ["flag_near_completion"], "target", table=True)
```

Processando estatísticas para: flag_near_completion...



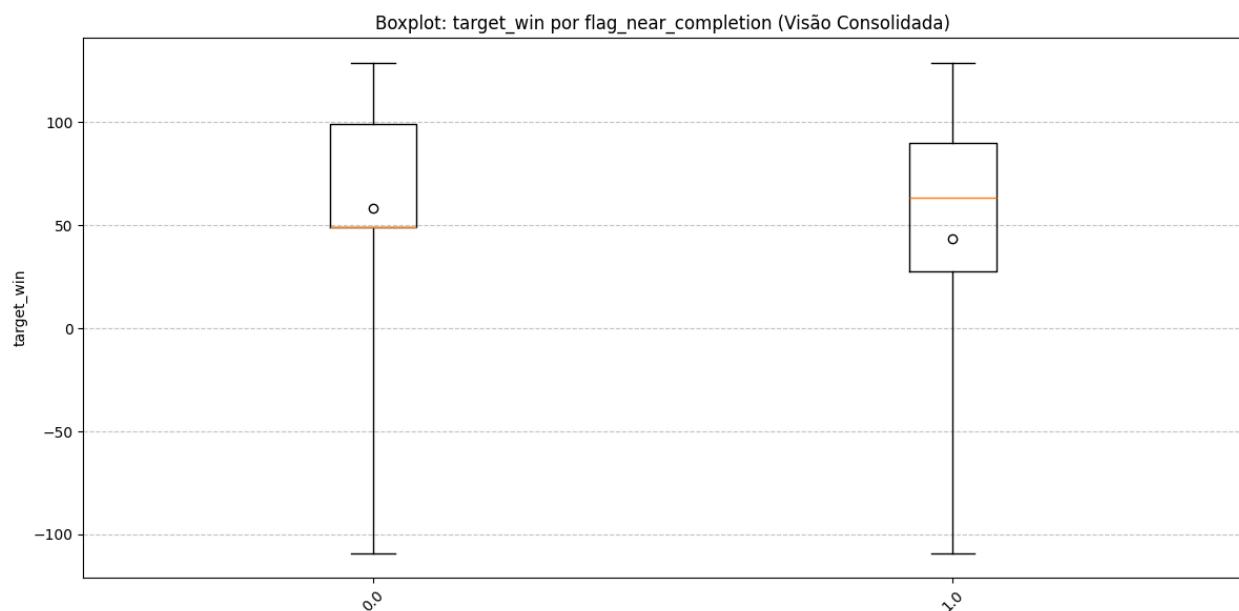
--- Estatísticas: flag_near_completion (Visão Consolidada) ---

	flag_near_completion	min	q1	med	mean	q3	max
1	0	-152.5452	49.000000	49.000000	58.720001	99.000000	1736.522159
0	1	-152.5452	27.704808	63.644117	47.615418	90.030962	1950.000000

=====

```
In [54]: plot_boxplot(df_base_logs, ["flag_near_completion"], "target_win", table=True)
```

Processando estatísticas para: flag_near_completion...



--- Estatísticas: flag_near_completion (Visão Consolidada) ---

	flag_near_completion	min	q1	med	mean	q3	max
1	0	-108.964925	49.000000	49.000000	58.312417	99.000000	128.953521
0	1	-108.964925	27.704808	63.644117	43.522534	90.030962	128.953521

=====

Nao me pareceu boa. Nao agraga em poder preditivo (distribuicao "desequilibrada) ou com informacao nova: por capturar os casos que nao existem registros de logs, se torna extremamente semelhante a `flag_has_logs`. Desconsiderar a flag.

Agrupamento semantic (talvez nao tao bom)

```
In [55]: df_base_logs = df_base_logs.withColumn("completion_efficiency_group", F.when(F.col("flag_has_logs") == 0, "00_unknown") .when(F.col("completion_efficiency") == 0, "01_never_complete") .when(F.col("completion_efficiency") <= 0.5, "02_partial_complete") .when(F.col("completion_efficiency") <= 0.8, "03_mostly_complete") .otherwise("04_full_complete"))
```

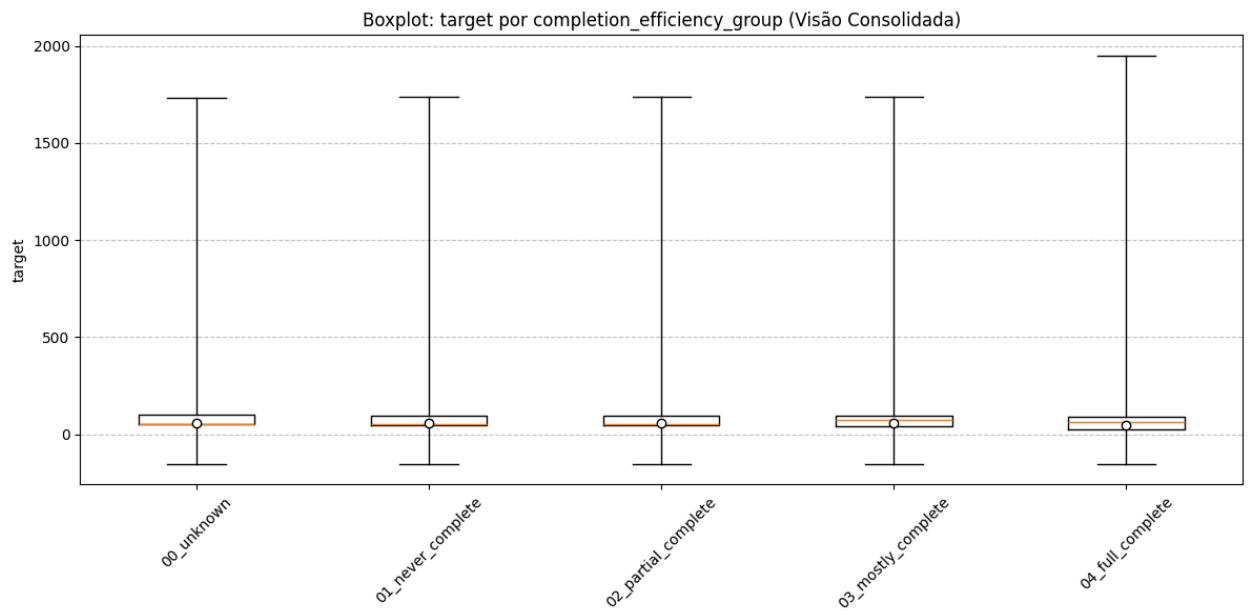
```
In [59]: calcular_distribuicao(df_base_logs, ["completion_efficiency_group"], n_show=5)
```

```
+-----+-----+-----+
|completion_efficiency_group|total |pct_total|
+-----+-----+-----+
|04_full_complete          |8818471|78.44   |
|00_unknown                 |1266985|11.27   |
|03_mostly_complete         |934663 |8.31    |
|02_partial_complete        |160140  |1.42    |
|01_never_complete          |62606   |0.56    |
+-----+-----+-----+
```

Out[59]: DataFrame[completion_efficiency_group: string, total: bigint, pct_total: double]

```
In [57]: plot_boxplot(df_base_logs, ["completion_efficiency_group"], "target", table=True)
```

Processando estatísticas para: completion_efficiency_group...



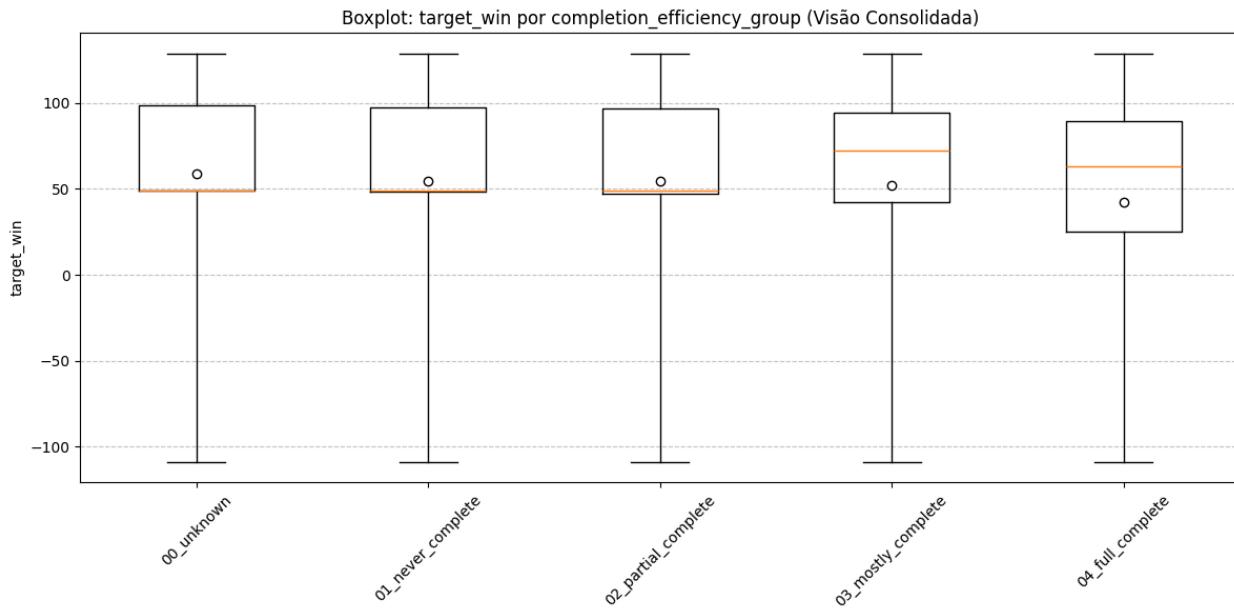
--- Estatísticas: completion_efficiency_group (Visão Consolidada) ---

	completion_efficiency_group	min	q1	med	mean	q3	max
3	00_unknown	-152.5452	49.000000	49.000000	59.177692	99.000000	1733.833386
2	01_never_complete	-152.5452	48.423280	49.000000	55.701827	97.665799	1736.522159
4	02_partial_complete	-152.5452	47.061359	49.000000	57.001885	97.025667	1736.130944
0	03_mostly_complete	-152.5452	42.430658	72.575104	56.269346	94.386882	1737.640612
1	04_full_complete	-152.5452	25.318736	62.944739	46.596456	89.344136	1950.000000

=====

```
In [58]: plot_boxplot(df_base_logs, ["completion_efficiency_group"], "target_win", table=True)
```

Processando estatísticas para: completion_efficiency_group...



--- Estatísticas: completion_efficiency_group (Visão Consolidada) ---

	completion_efficiency_group	min	q1	med	mean	q3	max
3	00_unknown	-108.964925	49.000000	49.000000	58.826712	99.000000	128.953521
2	01_never_complete	-108.964925	48.423280	49.000000	54.449724	97.665799	128.953521
4	02_partial_complete	-108.964925	47.061359	49.000000	54.646151	97.025667	128.953521
0	03_mostly_complete	-108.964925	42.430658	72.575104	52.399756	94.386882	128.953521
1	04_full_complete	-108.964925	25.318736	62.944739	42.475046	89.344136	128.953521

=====

Variável altamente concentrada, e os valores nos permitem concluir relação notória com a flag. Os três primeiros grupos possuem valores muito próximos para a `flag_near_completion == 0`, e os demais `flag_near_completion == 1`. Até existe uma relação clara: quanto maior a eficiência de conclusão, menor a margem. Porem, a separação só acontece no extremo (`full_complete`, demais), evidenciando que os grupos intermediários não se diferenciam. Nem a flag nem a versão agrupada adicionam poder preditivo real.

Conclusao

A taxa de eficiência de conclusão foi inicialmente discretizada para análise exploratória. Entretanto, observou-se forte concentração no grupo de conclusão total (>78%) e baixa separação de margem nos demais grupos. Dado isso, a variável foi mantida em sua forma contínua, e eventualmente será sujeita a transformações temporais (média, razão e estabilidade), buscando ampliar o poder explicativo e interpretabilidade.

Se o usuário dá play em muitas músicas únicas mas a grande maioria cai no bin `num_25` (menos de 25% da música), ele está procurando algo que não acha.

```
In [82]: df_base_logs = df_base_logs.withColumn("frustration_index", F.when(F.col("flag_has_logs").isin(1), (F.col("num_25") / F.col("num_unq"))).otherwise(0.0))
```

Descritivas + correlação com target

```
In [83]: df_base_logs.select("frustration_index").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+-----+
|summary| frustration_index|
+-----+-----+
| count| 11242865|
| mean| 0.2306981666446601|
| stddev| 0.251566060885176|
| min| 0.0|
| 1%| 0.0|
| 5%| 0.0|
| 25%| 0.06716417910447761|
| 50%| 0.18058690744920994|
| 75%| 0.32947976878612717|
| 95%| 0.6226993865030674|
| 99.5%| 1.1443298969072164|
| max| 90.33333333333333|
+-----+
```

```
In [84]: df_base_logs.select("frustration_index", "target").corr("frustration_index", "target")
```

```
Out[84]: 0.01750115038300395
```

```
In [85]: df_base_logs.select("frustration_index", "target_win").corr("frustration_index", "target_win")
```

```
Out[85]: 0.019564472496029384
```

8.1.12. long_listening_flag

Demarcar usuarios que escutam por muitas horas (10h, no caso)

```
In [62]: df_base_logs = df_base_logs.withColumn("long_listening_flag", F.when(F.col("total_secs") >= 36000, 1).otherwise(0)) # 10h/mês
```

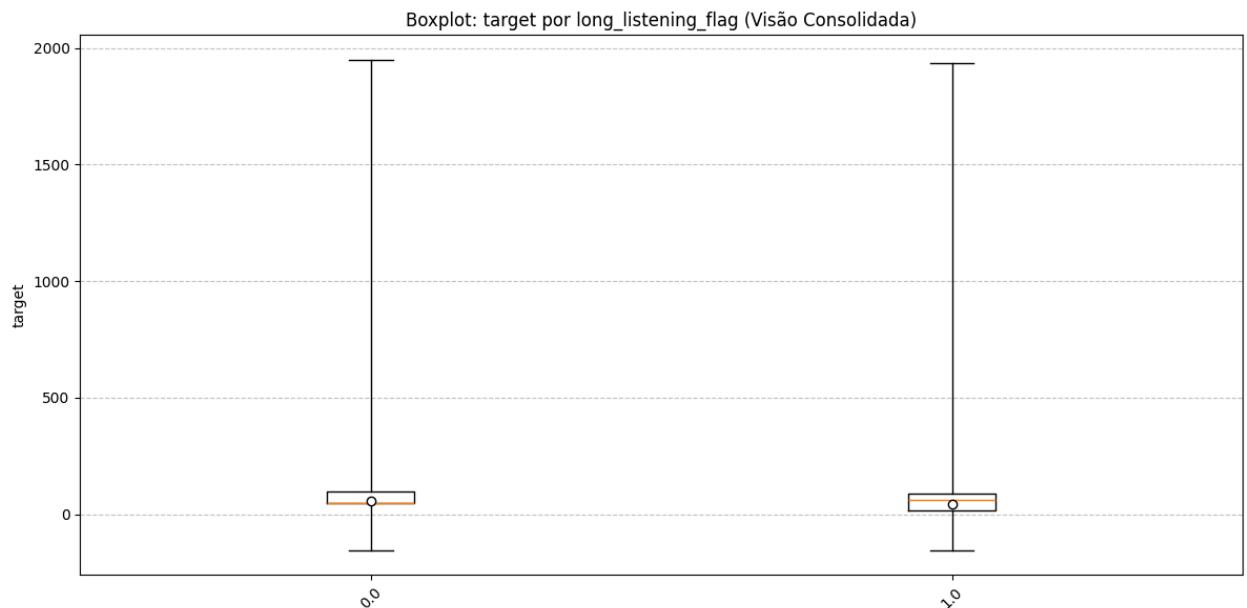
```
In [63]: calcular_distribuicao(df_base_logs, ["long_listening_flag"], n_show=2)
```

```
+-----+-----+
|long_listening_flag|total |pct_total|
+-----+-----+
|1                |7107380|63.22   |
|0                |4135485|36.78   |
+-----+-----+
```

```
Out[63]: DataFrame[long_listening_flag: int, total: bigint, pct_total: double]
```

```
In [64]: plot_boxplot(df_base_logs, ["long_listening_flag"], "target", table=True)
```

Processando estatísticas para: long_listening_flag...



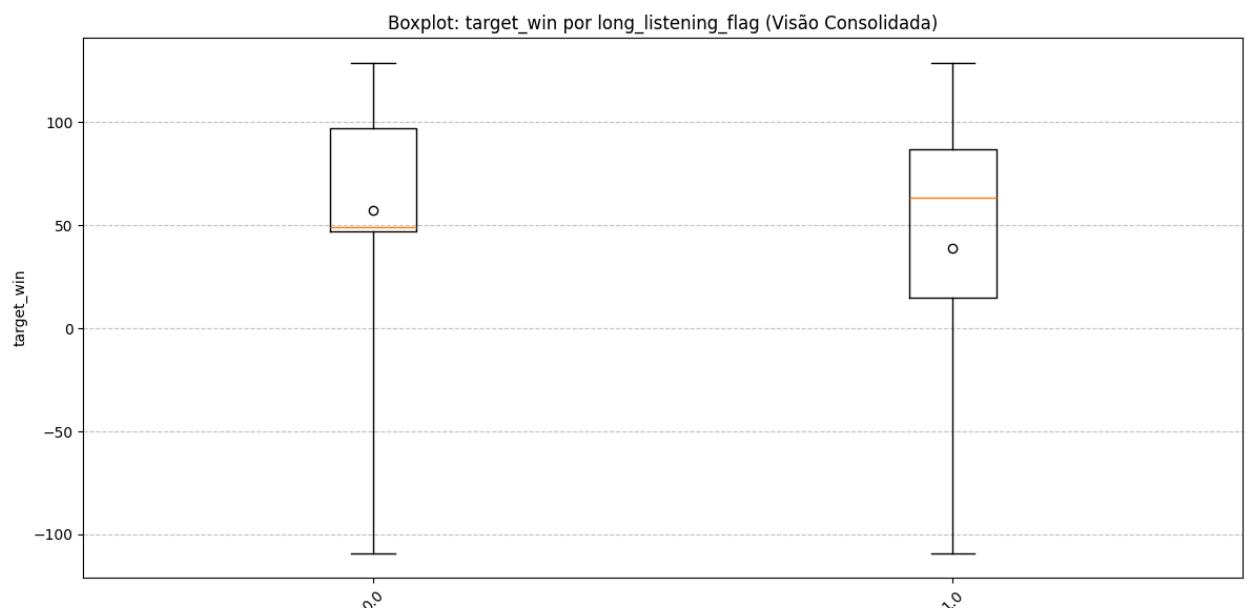
--- Estatísticas: long_listening_flag (Visão Consolidada) ---

	long_listening_flag	min	q1	med	mean	q3	max
1	0	-152.5452	47.113233	49.000000	58.945791	97.288186	1950.000000
0	1	-152.5452	14.945088	63.166128	43.450361	86.763500	1936.799455

=====

```
In [65]: plot_boxplot(df_base_logs, ["long_listening_flag"], "target_win", table=True)
```

Processando estatísticas para: long_listening_flag...



--- Estatísticas: long_listening_flag (Visão Consolidada) ---

	long_listening_flag	min	q1	med	mean	q3	max
1	0	-108.964925	47.113233	49.000000	57.039820	97.288186	128.953521
0	1	-108.964925	14.945088	63.166128	38.819726	86.763500	128.953521

=====

```
#### Conclusao
```

Tambem se mostra muito proxima da `flag_has_logs`. Nao vou levar para feature engineering.

8.1.13. flag_shallow_user

Consumo superficial. Publico que teoricamente gera pouco custo: paga mas nao usa

```
In [ ]: df_base_logs = df_base_logs.withColumn("flag_shallow_user", F.when((F.col("num_25") > 0) & (F.col("num_100") == 0), 1).otherwise(0))
```

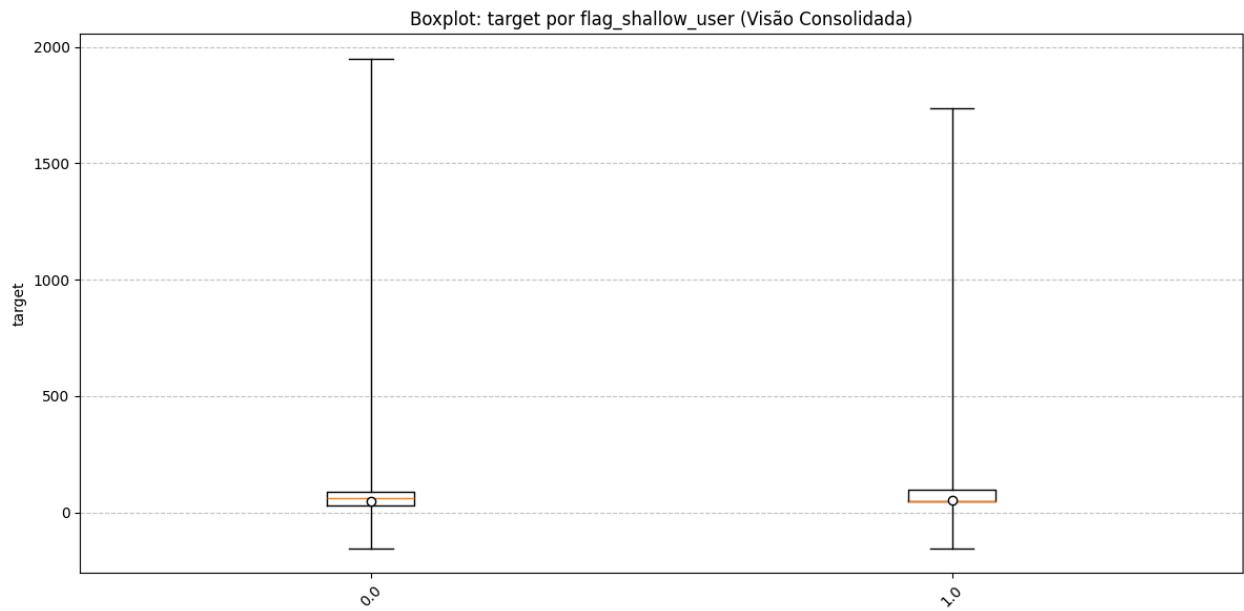
```
In [ ]: calcular_distribuicao(df_base_logs, ["flag_shallow_user"], n_show=2)
```

```
+-----+-----+
|flag_shallow_user|total    |pct_total|
+-----+-----+
|0          |11124479|98.95   |
|1          |118386   |1.05     |
+-----+-----+
```

DataFrame[flag_shallow_user: int, total: bigint, pct_total: double]

```
In [ ]: plot_boxplot(df_base_logs, ["flag_shallow_user"], "target", table=True)
```

Processando estatisticas para: flag_shallow_user...



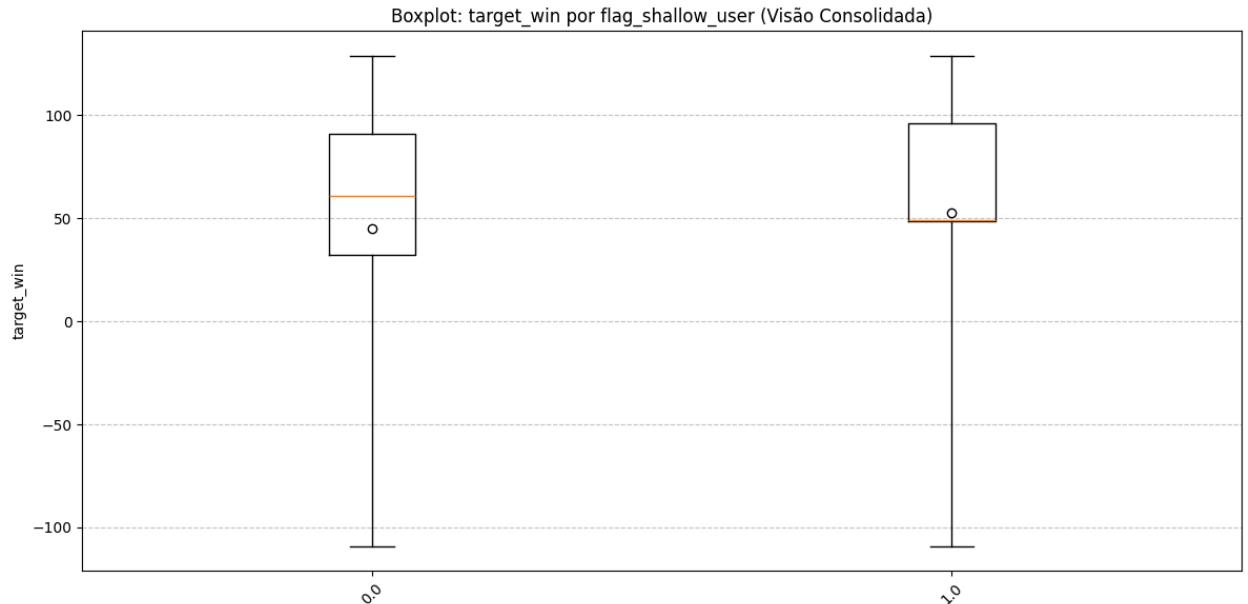
--- Estatisticas: flag_shallow_user (Visão Consolidada) ---

	flag_shallow_user	min	q1	med	mean	q3	max
1	0	-152.5452	32.332491	60.918663	48.820812	90.846064	1950.000000
0	1	-152.5452	48.509472	49.000000	53.715153	96.299630	1736.522159

=====

```
In [ ]: plot_boxplot(df_base_logs, ["flag_shallow_user"], "target_win", table=True)
```

Processando estatísticas para: flag_shallow_user...



--- Estatísticas: flag_shallow_user (Visão Consolidada) ---

	flag_shallow_user	min	q1	med	mean	q3	max
1	0	-108.964925	32.332491	60.918663	45.118069	90.846064	128.953521
0	1	-108.964925	48.509472	49.000000	52.792964	96.299630	128.953521

=====

Conclusao

Shallow users pagam assinatura mas geram baixo custo variável (pouco tempo de escuta). Importante levar pela diferença entre os valores descritivos.

8.2. Transactions

```
In [121]: transactions_vars = [
    "payment_method_id",
    "payment_plan_days",
    "plan_list_price",
    "actual_amount_paid",
    "is_auto_renew",
    "transaction_date",
    "membership_expire_date",
    "is_cancel",
    "flag_expire_invalido",
]
df_base_trans = df_base.select("msno", "safra", "target", "target_win", *transactions_vars)
```

8.2.1. flag_has_transactions

```
In [138]: df_base_trans = df_base_trans.withColumn("flag_has_transactions", F.when(F.col("payment_method_id").isNull(), 0).otherwise(1))
```

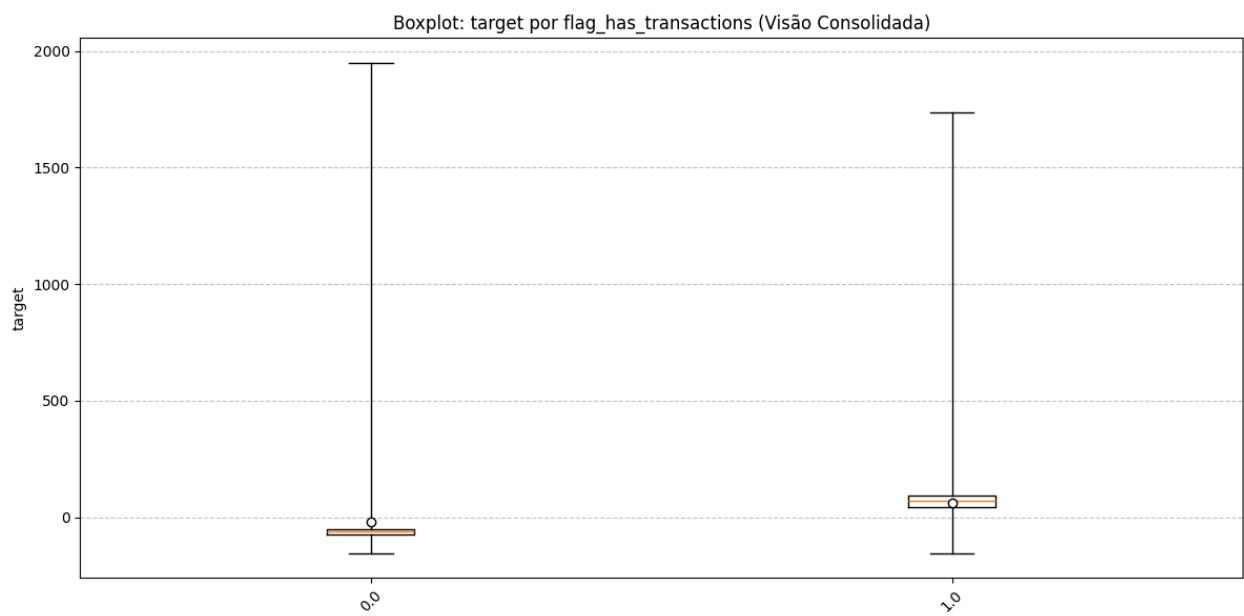
```
In [110]: calcular_distribuicao(df_base_trans, ["flag_has_transactions"], n_show=2)
```

flag_has_transactions	total	pct_total
1	9390483	83.52
0	1852382	16.48

```
Out[110]: DataFrame[flag_has_transactions: int, total: bigint, pct_total: double]
```

```
In [111]: plot_boxplot(df_base_trans, ["flag_has_transactions"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_has_transactions...



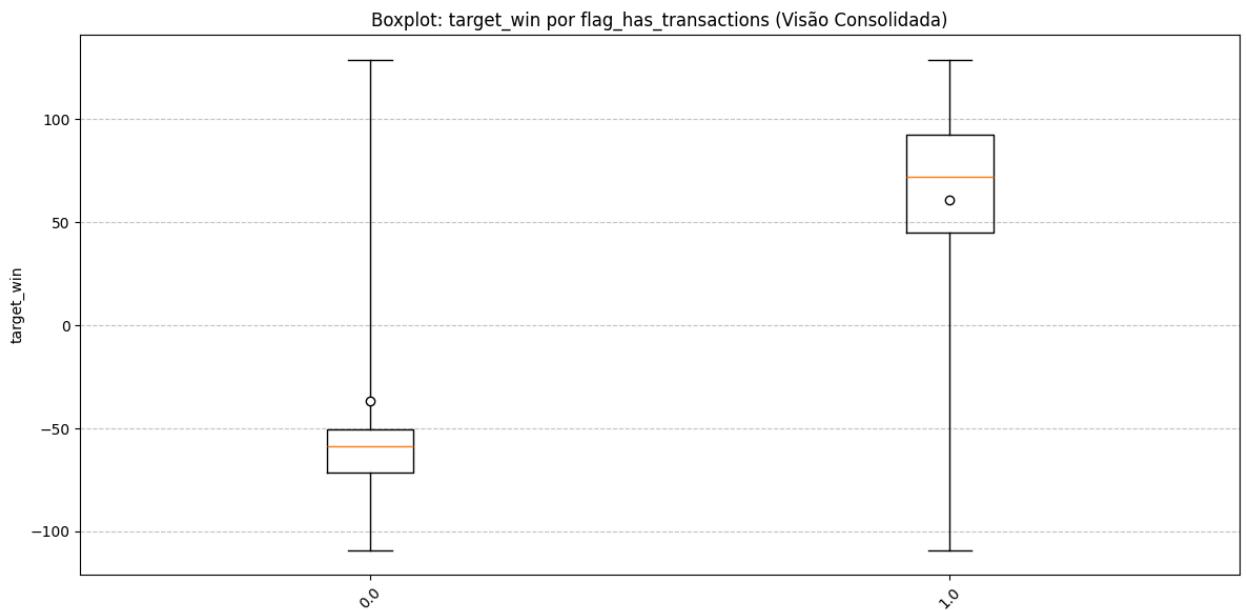
--- Estatísticas: flag_has_transactions (Visão Consolidada) ---

	flag_has_transactions	min	q1	med	mean	q3	max
1	0	-152.5452	-71.491939	-58.890695	-21.018945	-50.309033	1950.000000
0	1	-152.5452	45.088921	71.995375	62.083651	92.325581	1737.859653

=====

```
In [109]: plot_boxplot(df_base_trans, ["flag_has_transactions"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_has_transactions...



--- Estatísticas: flag_has_transactions (Visão Consolidada) ---

	flag_has_transactions	min	q1	med	mean	q3	max
1	0	-108.964925	-71.491939	-58.890695	-36.612835	-50.309033	128.953521
0	1	-108.964925	45.088921	71.995375	60.658898	92.325581	128.953521

=====

Naturalmente, usuarios que nao possuem registros de transacoes nao possuem pagamento registrado, o que implica em valores negativos de margem liquida pelo custo padrao de 50 reais e pelos possiveis acrescimos neste valor nos casos em que existe registros de uso (logs).

8.2.2. transaction_date

Quando os pagamentos sao feitos? Pelo que foi visto no EDA inicial, nao existe dia fixo. Sendo assim, nao aparenta ser uma boa ideia considerar o uso de actual_amount_paid para o modelo prever resultados. Tambem existe essa complicacao para a tabela de logs, que registra a utilizacao do usuario na plataforma.

```
In [102]: df_base_trans = df_base_trans.withColumn("payment_day", F.dayofmonth(F.to_date(F.col("transaction_date"), "yyyyMMdd")))
```

```
In [1]: df_base_trans.filter(F.col("payment_day").isNull()).show(5, truncate=False)
```

```
+-----+-----+-----+-----+-----+
|msno |safra | | | | | |
|margem_liquida_mensal|payment_method_id|payment_plan_days|plan_list_price|actual_amount_paid|is_auto_renew|transaction_date|
|membership_expire_date|is_cancel|flag_plano_mensal|payment_day|
+-----+-----+-----+-----+-----+
|++917+WG0Z96gNp0TDXxAy01XYE0CiugWFTxA6zZI=|201606|-85.15690500000001 |NULL |NULL |NULL | |
|NULL |NULL |NULL |NULL |0 |NULL |NULL |
|++I1wsXaVBTTZE3gwzQ5qBWLcMMCxSInWe2pLsJ8hS4=|201605|-54.5154735 |NULL |NULL |NULL |NULL |
|NULL |NULL |NULL |NULL |0 |NULL |NULL |
|++IrpeciSQ6NWOp78CLvSLjcJVwBecNHnYzvOrxFAPE=|201612|-99.7053543000001 |NULL |NULL |NULL |
|NULL |NULL |NULL |NULL |0 |NULL |NULL |
|++UEvwqAY2F9VpILLHeicRU4D6FzbSDiaDrFkPrhqq8=|201605|-70.5831001000001 |NULL |NULL |NULL |
|NULL |NULL |NULL |NULL |0 |NULL |NULL |
|++aVm+4v87MXzbhiExoS70szzRxcfn8aKa1ISGUR9I=|201606|-84.0426279000001 |NULL |NULL |NULL |
|NULL |NULL |NULL |NULL |0 |NULL |NULL |
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
In [ ]: calcular_distribuicao(df_base_trans, ["payment_day"], n_show=31)
```

payment_day	total	pct_total
NULL	1852382	16.48
31	1092679	9.72
30	596721	5.31
9	287773	2.56
6	287075	2.55
4	286865	2.55
16	283210	2.52
11	283116	2.52
21	282850	2.52
5	282462	2.51
20	281859	2.51
19	281180	2.5
7	277675	2.47
26	277434	2.47
25	275923	2.45
17	274890	2.45
15	274216	2.44
10	273710	2.43
13	273598	2.43
22	272501	2.42
12	272173	2.42
8	271910	2.42
3	271356	2.41
18	270381	2.4
27	267152	2.38
14	261385	2.32
23	260645	2.32
24	260475	2.32
2	260143	2.31
1	259984	2.31
29	165653	1.47

only showing top 31 rows

```
DataFrame[payment_day: int, total: bigint, pct_total: double]
```

8.2.3. payment_plan_days + actual_amount_paid

8.2.3.1. flags

Cobrança Válida - valid_fee

```
In [ ]: df_base_trans = (df_base_trans
[122]:     .withColumn("flag_valid_fee", # Flag: cobrança válida (plano real)
      F.when((F.col("payment_plan_days") > 0) & (F.col("plan_list_price") > 0), 1).otherwise(0)))
```

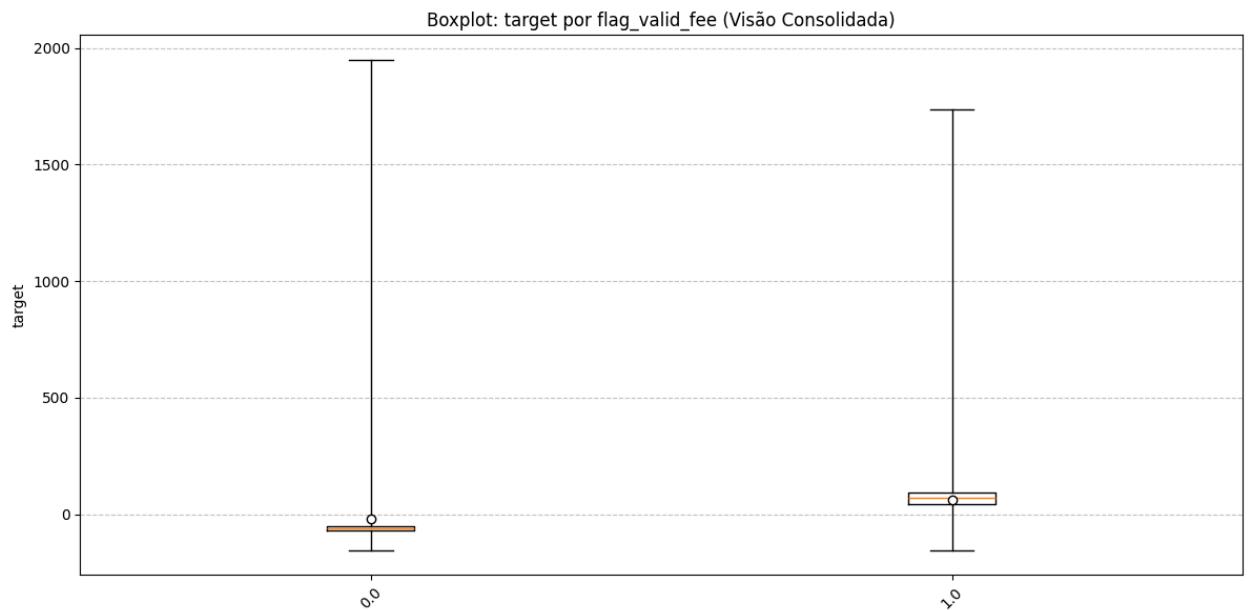
```
In [ ]: calcular_distribuicao(df_base_trans, ["flag_valid_fee"])
```

flag_valid_fee	total	pct_total
1	9081504	80.78
0	2161361	19.22

```
DataFrame[flag_valid_fee: int, total: bigint, pct_total: double]
```

```
In [104]: plot_boxplot(df_base_trans, ["flag_valid_fee"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_valid_fee...



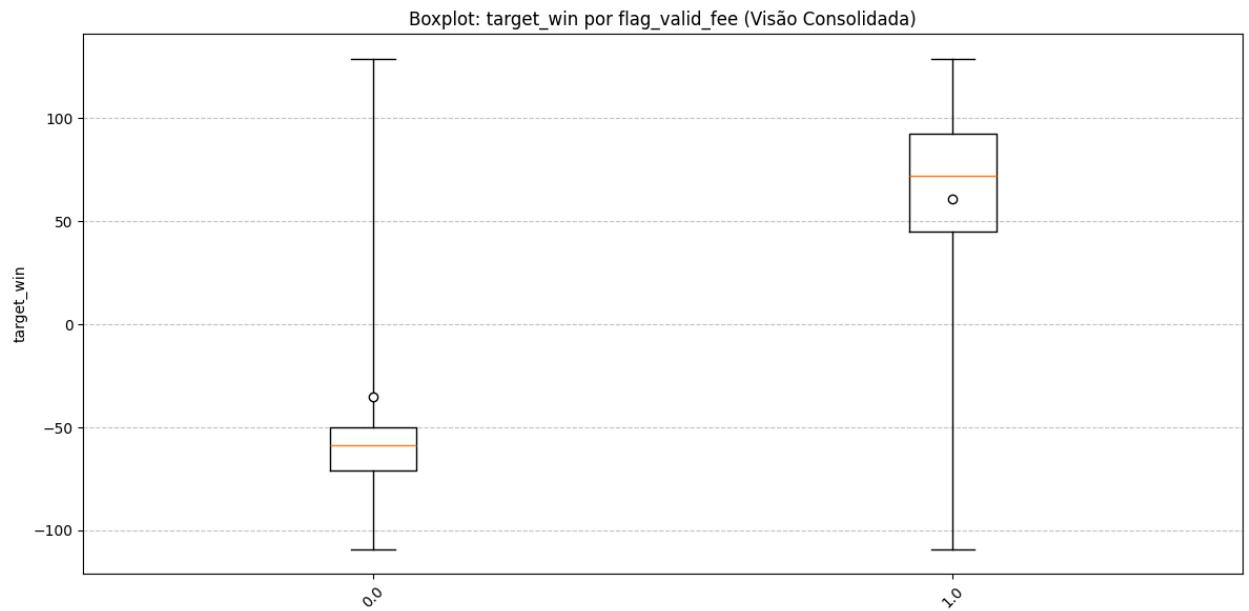
--- Estatísticas: flag_valid_fee (Visão Consolidada) ---

	flag_valid_fee	min	q1	med	mean	q3	max
1	0	-152.5452	-71.081910	-58.479486	-19.786860	-50.033419	1950.000000
0	1	-152.5452	45.212528	72.097178	62.316011	92.335408	1737.859653

=====

```
In [123]: plot_boxplot(df_base_trans, ["flag_valid_fee"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_valid_fee...



--- Estatísticas: flag_valid_fee (Visão Consolidada) ---

	flag_valid_fee	min	q1	med	mean	q3	max
1	0	-108.964925	-71.081910	-58.479486	-35.254039	-50.033419	128.953521
0	1	-108.964925	45.212528	72.097178	60.947213	92.335408	128.953521

=====

- * Separação forte
- * Diferença estrutural clara entre regimes
- * Alta estabilidade estatística (volume alto)

Sem Cobrança - no_flee

```
In [ ]: df_base_trans = (df_base_trans
    .withColumn("flag_no_fee", # Flag: sem cobrança (estado administrativo)
        F.when(F.col("payment_plan_days") == 0, 1).otherwise(0)))
```

```
In [ ]: calcular_distribucao(df_base_trans, ["flag_no_fee"])
```

flag_no_fee	total	pct_total
0	11242865	100.0

DataFrame[flag_no_fee: int, total: bigint, pct_total: double]

```
In [ ]: df_base_trans.filter(F.col("payment_plan_days").isin(0)).show(5, truncate=False)
```

msno	safra	margem_liquida_mensal	payment_method_id	payment_plan_days	plan_list_price	actual_amount_paid	is_auto_renew	transaction_date	membership_expire_date	is_cancel	flag_valid_fee	flag_no_fee	flag_exemption	flag_no_payment	flag_payment_without_charge

Descartar.

Isenção - exemption

```
In [ ]: df_base_trans = (df_base_trans
    .withColumn("flag_exemption", # Flag: isenção (nem cobrou nem pagou, sem cancelamento)
        F.when((F.col("actual_amount_paid") == 0) & (F.col("plan_list_price") == 0) & (F.col("is_cancel") == 0), 1).otherwise(0)))
```

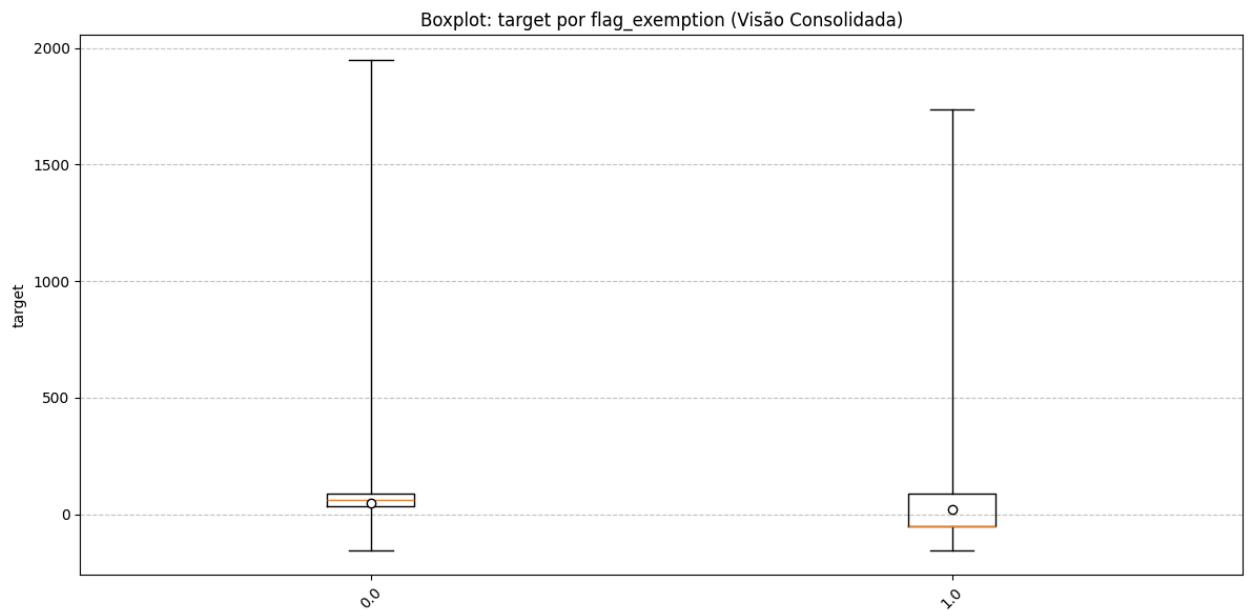
```
In [ ]: calcular_distribucao(df_base_trans, ["flag_exemption"])
```

flag_exemption	total	pct_total
0	10933887	97.25
1	308978	2.75

DataFrame[flag_exemption: int, total: bigint, pct_total: double]

```
In [125]: plot_boxplot(df_base_trans, ["flag_exemption"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_exemption...



--- Estatísticas: flag_exemption (Visão Consolidada) ---

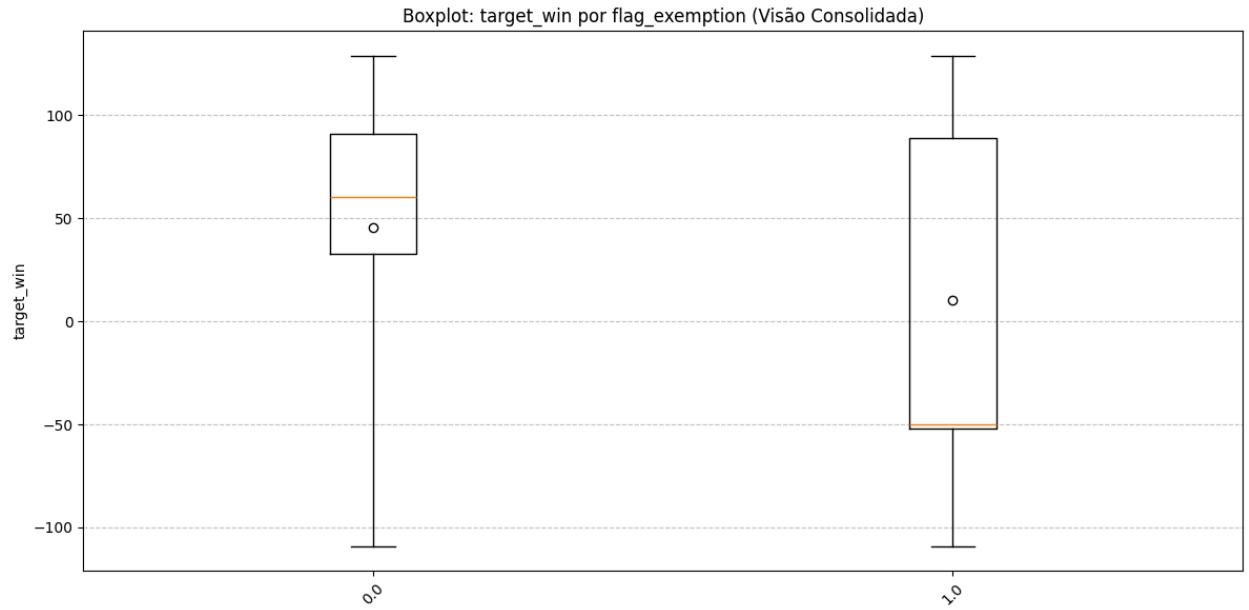
	flag_exemption	min	q1	med	mean	q3	max
1	0	-152.5452	32.992890	60.572939	48.994570	90.865147	1950.000000
0	1	-152.5452	-52.148538	-50.000000	21.331289	89.127693	1735.412335

=====

```
In [126]: plot_boxplot(df_base_trans, ["flag_exemption"], "target_win", agrupar_por_safra=False, table=True)
```

[126]:

Processando estatísticas para: flag_exemption...



--- Estatísticas: flag_exemption (Visão Consolidada) ---

	flag_exemption	min	q1	med	mean	q3	max
1	0	-108.964925	32.992890	60.572939	45.351834	90.865147	128.953521
0	1	-108.964925	-52.148538	-50.000000	10.092382	89.127693	128.953521

=====

Pelas medianas:

- * Apesar do volume pequeno (2,75%), o comportamento é coerente e distinto
- * Representa um estado operacional específico, não ruído

Cobrança sem pagamento - no_payment

```
In [ ]: df_base_trans = (df_base_trans
    .withColumn("flag_no_payment", # Flag: cobrança sem pagamento (possível falha / desconto / churn latente)
        F.when((F.col("actual_amount_paid") <= 0) & (F.col("plan_list_price") > 0), 1).otherwise(0)))
```

```
In [ ]: calcular_distribuicao(df_base_trans, ["flag_no_payment"])
```

flag_no_payment	total	pct_total
0	11187843	99.51
1	55022	0.49

```
DataFrame[flag_no_payment: int, total: bigint, pct_total: double]
```

- * Volume muito baixo (0.49%)
- * Pode confundir com outros regimes (cancelamento, atraso)

Descartar

Pagamento sem cobrança - payment_without_charge

```
In [ ]: df_base_trans = (df_base_trans
    .withColumn("flag_payment_without_charge", # Flag: pagamento sem cobrança (ajuste operacional)
        F.when((F.col("actual_amount_paid") > 0) & (F.col("plan_list_price") == 0), 1).otherwise(0)))
```

```
In [ ]: calcular_distribuicao(df_base_trans, ["flag_payment_without_charge"])
```

flag_payment_without_charge	total	pct_total
0	11242864	100.0
1	1	0.0

```
DataFrame[flag_payment_without_charge: int, total: bigint, pct_total: double]
```

Descartar

8.2.3.2. pagamento mensal

Grupo plano

```
In [127]: df_base_trans = df_base_trans.withColumn("grupo_plano_final",
    F.when(F.col("flag_valid_fee") == 0, "00_Sem_Cobranca")
    .when(F.col("payment_plan_days") < 30, "01_Curto")
    .when(F.col("payment_plan_days").isin(30, 31), "02_Mensal")
    .otherwise("03_Longo"))
```

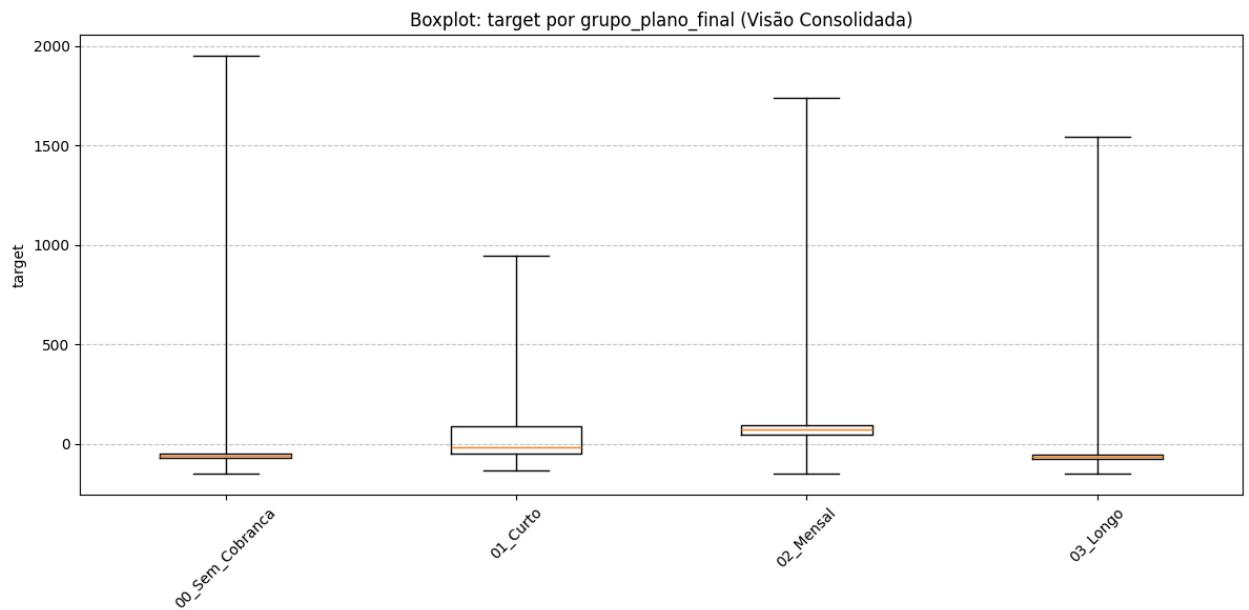
```
In [ ]: calcular_distribuicao(df_base_trans, ["grupo_plano_final"])
```

grupo_plano_final	total	pct_total
02_Mensal	8952209	79.63
00_Sem_Cobranca	2161361	19.22
03_Longo	126477	1.12
01_Curto	2818	0.03

```
DataFrame[grupo_plano_final: string, total: bigint, pct_total: double]
```

```
In [ ]: plot_boxplot(df_base_trans, ["grupo_plano_final"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: grupo_plano_final...



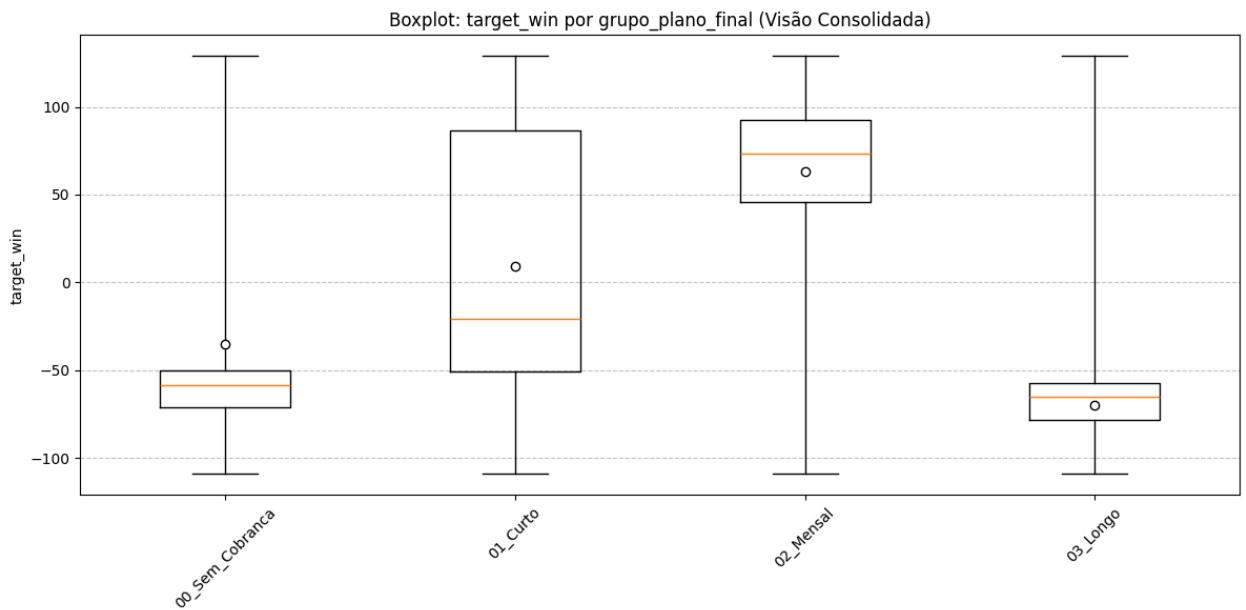
--- Estatísticas: grupo_plano_final (Visão Consolidada) ---

	grupo_plano_final	min	q1	med	q3	max
3	00_Sem_Cobranca	-152.545200	-71.081910	-58.479486	-50.033419	1950.000000
2	01_Curto	-133.591014	-50.690076	-20.688140	86.539581	945.658211
1	02_Mensal	-152.545200	45.939913	73.072146	92.534515	1737.859653
0	03_Longo	-152.545200	-77.973307	-65.152461	-57.391381	1541.641613

=====

```
In [128]: plot_boxplot(df_base_trans, ["grupo_plano_final"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: grupo_plano_final...



--- Estatísticas: grupo_plano_final (Visão Consolidada) ---

	grupo_plano_final	min	q1	med	mean	q3	max
3	00_Sem_Cobranca	-108.964925	-71.081910	-58.479486	-35.254039	-50.033419	128.953521
2	01_Curto	-108.964925	-50.690076	-20.688140	9.456909	86.539581	128.953521
1	02_Mensal	-108.964925	45.939913	73.072146	62.986019	92.534515	128.953521
0	03_Longo	-108.964925	-77.973307	-65.152461	-69.579926	-57.391381	128.953521

=====

~80% da base tem plano mensal, uma flag aparenta ser melhor, mesmo que os casos longos tenha estatísticas bem distintas

```
In [129]: df_base_trans = df_base_trans.withColumn("flag_plano_mensal", F.when(F.col("payment_plan_days").isin(30, 31), 1).otherwise(0))
```

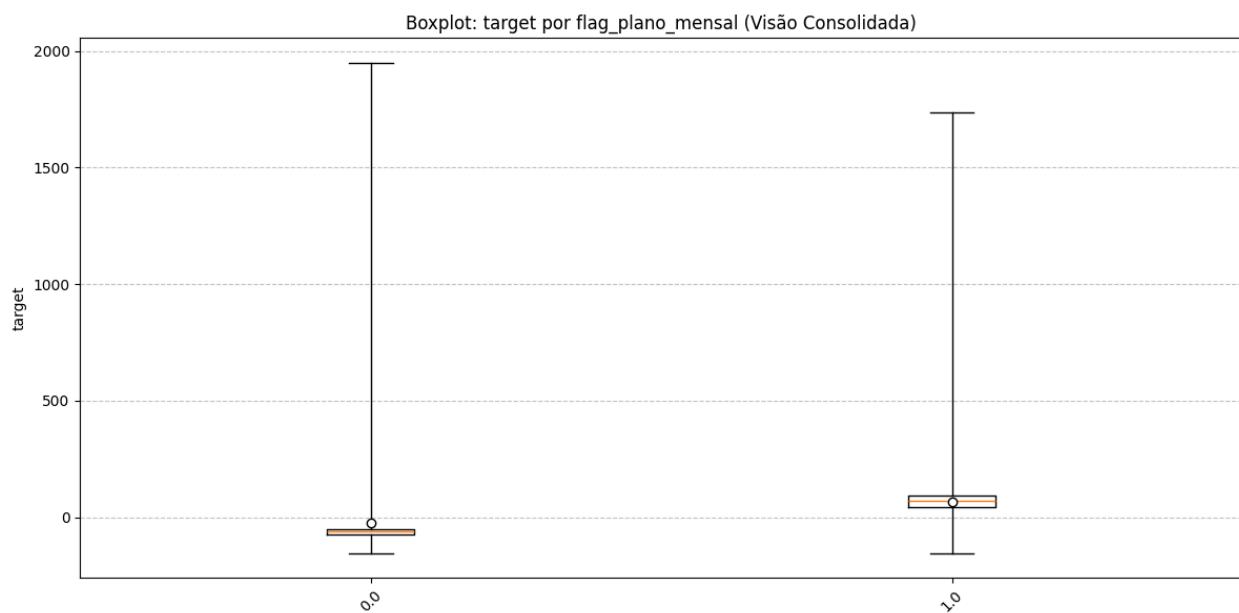
```
In [ ]: calcular_distribuicao(df_base_trans, ["flag_plano_mensal"], n_show=2)
```

flag_plano_mensal	total	pct_total
1	8952852	79.63
0	2290013	20.37

```
DataFrame[flag_plano_mensal: int, total: bigint, pct_total: double]
```

```
In [130]: plot_boxplot(df_base_trans, ["flag_plano_mensal"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_plano_mensal...



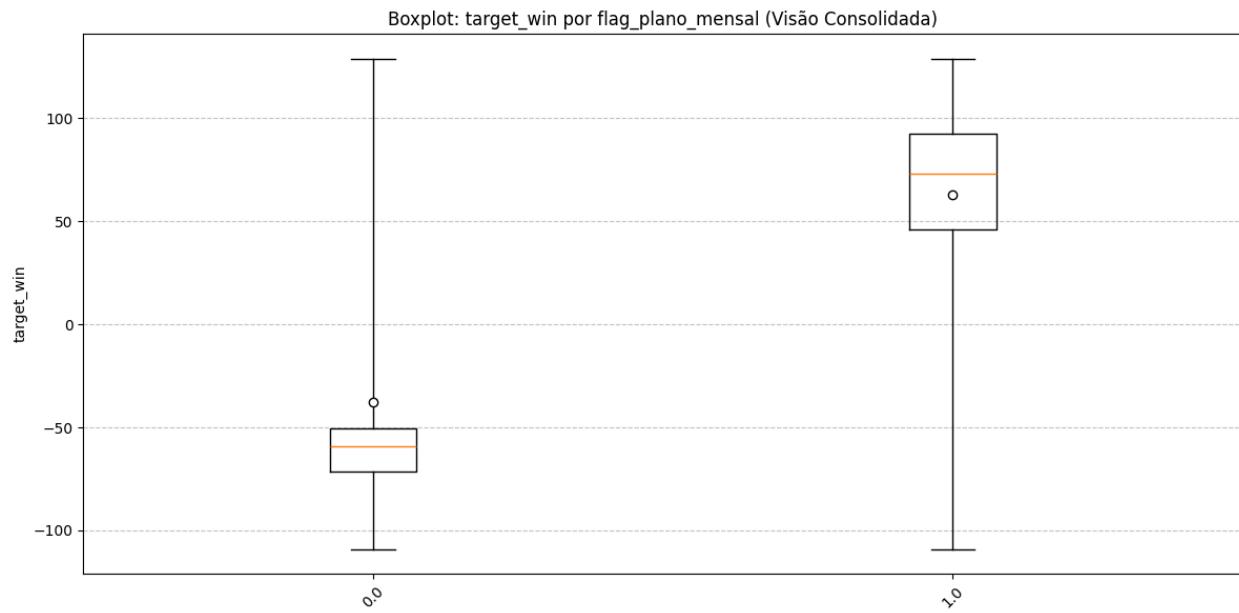
--- Estatísticas: flag_plano_mensal (Visão Consolidada) ---

	flag_plano_mensal	min	q1	med	mean	q3	max
1	0	-152.5452	-71.650163	-59.051857	-23.450302	-50.464816	1950.000000
0	1	-152.5452	45.939642	73.070747	64.387989	92.533925	1737.859653

=====

```
In [131]: plot_boxplot(df_base_trans, ["flag_plano_mensal"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_plano_mensal...



--- Estatísticas: flag_plano_mensal (Visão Consolidada) ---

	flag_plano_mensal	min	q1	med	mean	q3	max
1	0	-108.964925	-71.650163	-59.051857	-37.712949	-50.464816	128.953521
0	1	-108.964925	45.939642	73.070747	62.981315	92.533925	128.953521

=====

Interessante levar somente esta flag, pois substitui bem grupo_plano_final

Flag para usuarios longos (talvez nao precise, apenas para ver)

```
In [ ]: df_base_trans = df_base_trans.withColumn("is_long_term_user", F.when(F.col("grupo_plano_final") == "03_Longo", 1).otherwise(0))
```

```
In [ ]: calcular_distribuicao(df_base_trans, ["is_long_term_user"], n_show=2)
```

is_long_term_user	total	pct_total
0	11116388	98.88
1	126477	1.12

```
DataFrame[is_long_term_user: int, total: bigint, pct_total: double]
```

Baixa volumetria, acaba ja entrando na flag_plano_mensal .

8.2.4. daily_revenue_efficiency

Quanto o cliente "gera de receita por dia contratado"

```
In [134]: df_base_trans = df_base_trans.withColumn("daily_revenue_efficiency", F.when(F.col("payment_plan_days") > 0, F.col("actual_amount_paid") / F.col("payment_plan_days")).otherwise(F.col("actual_amount_paid")))
```

Descriptivas

```
In [ ]: df_base_trans.select("daily_revenue_efficiency").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary	daily_revenue_efficiency
count	9390483
mean	4.295660546947555
stddev	1.1862952262420043
min	0.0
1%	0.0
5%	3.3
25%	3.3
50%	4.966666666666667
75%	4.966666666666667
95%	6.0
99.5%	6.0
max	6.0

Distribuicao quase discreta, e nao continua, o que faz sentido, porque vimos que tem em esmagadora maioria planos padronizados.

```
In [ ]: calcular_distribuicao(df_base_trans, ["daily_revenue_efficiency"])
```

daily_revenue_efficiency	total	pct_total
4.966666666666667	5022656	44.67
3.3	2705388	24.06
NULL	1852382	16.48
4.3	515454	4.58
6.0	482787	4.29
0.0	364000	3.24
5.0	144758	1.29
4.584615384615384	45103	0.4
3.333333333333335	25959	0.23
2.977777777777778	21550	0.19

only showing top 10 rows

```
DataFrame[daily_revenue_efficiency: double, total: bigint, pct_total: double]
```

```
In [ ]: df_base_trans.select("daily_revenue_efficiency", "target").corr("daily_revenue_efficiency", "target")
```

```
0.398700573287757
```

```
In [136]: df_base_trans.select("daily_revenue_efficiency", "target_win").corr("daily_revenue_efficiency", "target_win")
```

```
Out[136]: 0.5995192903605263
```

A variável se mostra redundante com `flag_plano_mensal`. Parece uma categorica disfarcada. Apesar da alta correlação, é interessante atentar-se a possibilidade de `_leakage_`: apesar de eu suspeitar que não seja, mas o motivo de alta correção é ligado ao fato dessas variáveis estar diretamente ligada com o cálculo de margem líquida.

Agrupamento semântico

Agrupamento feito com base nos degraus de preço identificados no EDA.

```
In [ ]: df_base_trans = df_base_trans.withColumn("revenue_tier",
    F.when(F.col("flag_has_transactions").isin(0), "00_unknown") # Sem transação
    .when(F.col("daily_revenue_efficiency") == 0, "01_free_isencao") # Isenções/vouchers
    .when(F.col("daily_revenue_efficiency") < 3.0, "02_low_tier") # Planos residuais/testes
    .when(F.col("daily_revenue_efficiency").between(3.0, 3.5), "03_standard_99") # Plano básico (R$ 99/mês)
    .when(F.col("daily_revenue_efficiency").between(4.5, 5.5), "04_premium_149") # Plano mais popular (R$ 149/mês)
    .when(F.col("daily_revenue_efficiency") > 5.5, "05_high_tier") # Planos premium (R$ 180+)
    .otherwise("06_others")) # Planos especiais/promocionais
```

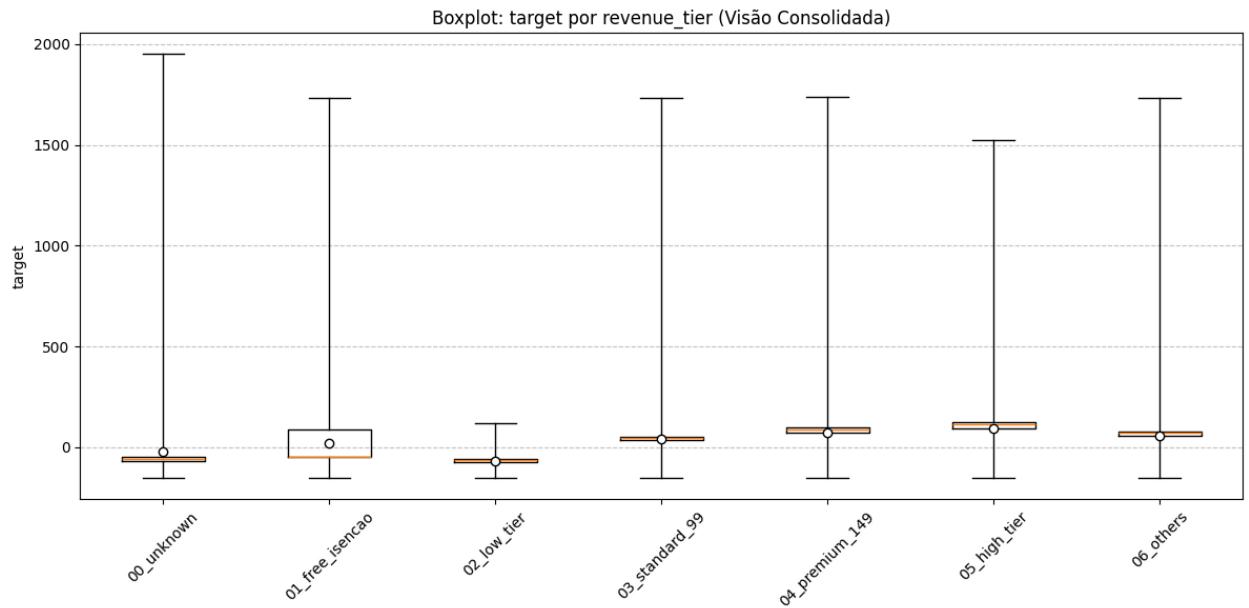
```
In [141]: calcular_distribuicao(df_base_trans, ["revenue_tier"])
```

revenue_tier	total	pct_total
04_premium_149	5234015	46.55
03_standard_99	2736522	24.34
00_unknown	1852382	16.48
06_others	551535	4.91
05_high_tier	482787	4.29
01_free_isencao	364000	3.24
02_low_tier	21624	0.19

```
Out[141]: DataFrame[revenue_tier: string, total: bigint, pct_total: double]
```

```
In [142]: plot_boxplot(df_base_trans, ["revenue_tier"], "target", table=True)
```

Processando estatísticas para: revenue_tier...



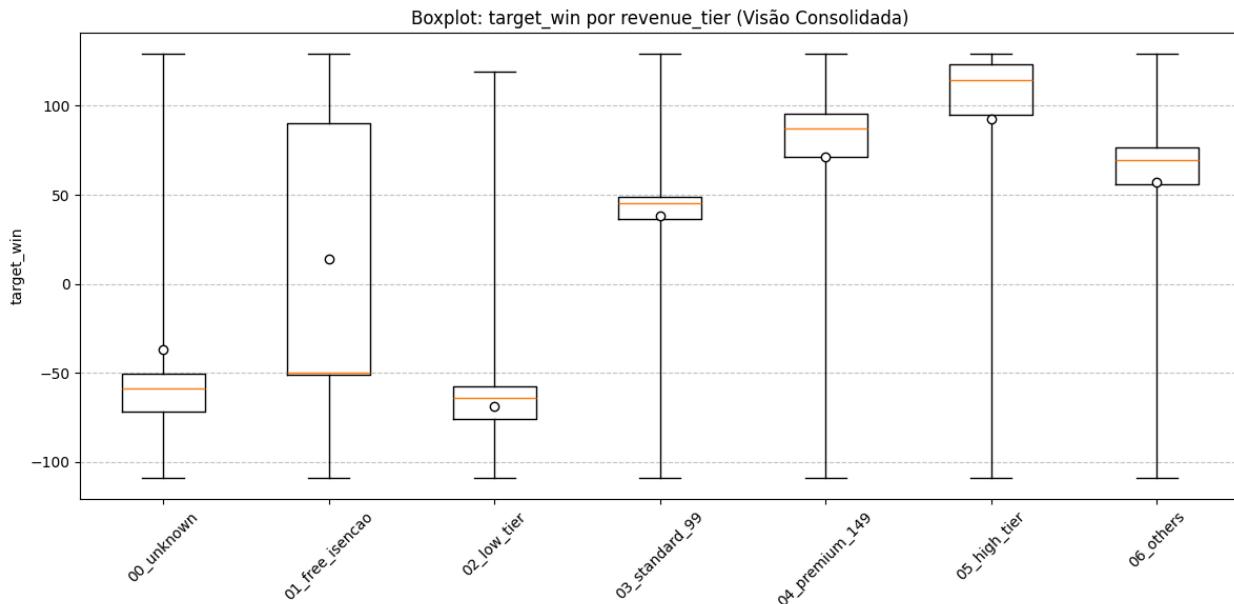
--- Estatísticas: revenue_tier (Visão Consolidada) ---

	revenue_tier	min	q1	med	mean	q3	max
3	00_unknown	-152.5452	-71.491939	-58.890695	-21.018945	-50.309033	1950.000000
0	01_free_isencao	-152.5452	-50.775706	-50.000000	21.177937	89.921758	1735.412335
5	02_low_tier	-152.5452	-76.106851	-64.322442	-70.277172	-57.319091	119.149782
4	03_standard_99	-152.5452	36.320474	45.179627	38.433489	48.806734	1731.843747
1	04_premium_149	-152.5452	71.254999	87.453640	73.506382	95.572968	1737.859653
6	05_high_tier	-152.5452	94.755995	114.571004	94.853670	123.325999	1521.663308
2	06_others	-152.5452	55.900144	69.415306	57.061065	76.435115	1735.011429

=====

```
In [143]: plot_boxplot(df_base_trans, ["revenue_tier"], "target_win", table=True)
```

Processando estatísticas para: revenue_tier...



--- Estatísticas: revenue_tier (Visão Consolidada) ---

	revenue_tier	min	q1	med	mean	q3	max
3	00_unknown	-108.964925	-71.491939	-58.890695	-36.612835	-50.309033	128.953521
0	01_free_isencao	-108.964925	-50.775706	-50.000000	13.872270	89.921758	128.953521
5	02_low_tier	-108.964925	-76.106851	-64.322442	-69.046208	-57.319091	119.149782
4	03_standard_99	-108.964925	36.320474	45.179627	38.430922	48.806734	128.953521
1	04_premium_149	-108.964925	71.254999	87.453640	71.373278	95.572968	128.953521
6	05_high_tier	-108.964925	94.755995	114.571004	92.362078	123.325999	128.953521
2	06_others	-108.964925	55.900144	69.415306	57.013605	76.435115	128.953521

=====

Conclusão

Levar a variável `daily_revenue_efficiency` (receita diária normalizada) apresentou uma distribuição multimodal concentrada em `"price points"` específicos do negócio (R\$3.30, R\$4.97, R\$ 6.00), refletindo a tabela de preços de planos de assinatura.

Para capturar o efeito não-linear da elasticidade de preço na margem líquida, criamos a variável categórica `revenue_tier`, que agrupa clientes em faixas de receita diária:

- * 00_unknown: Sem transação
- * 01_free_isencao: Isenções/_vouchers_
- * 02_low_tier: Planos de baixo valor
- * 03_standard_99: Plano básico (R\$ 99/mês)
- * 04_premium_149: Plano _premium_ (R\$ 149/mês)
- * 05_high_tier: Planos de alto valor
- * 06_others: Planos especiais/promocionais

Estratégia por modelo:

- * Elastic Net: Utiliza apenas `revenue_tier` (OHE) para evitar multicolinearidade e capturar não-linearidade.
- * LightGBM/RF: Mantém ambas as variáveis, permitindo que o modelo escolha a melhor representação pela `_feature importance_`.

8.2.5. payment_method_id + is_auto_renew

8.2.5.1. is_auto_renew

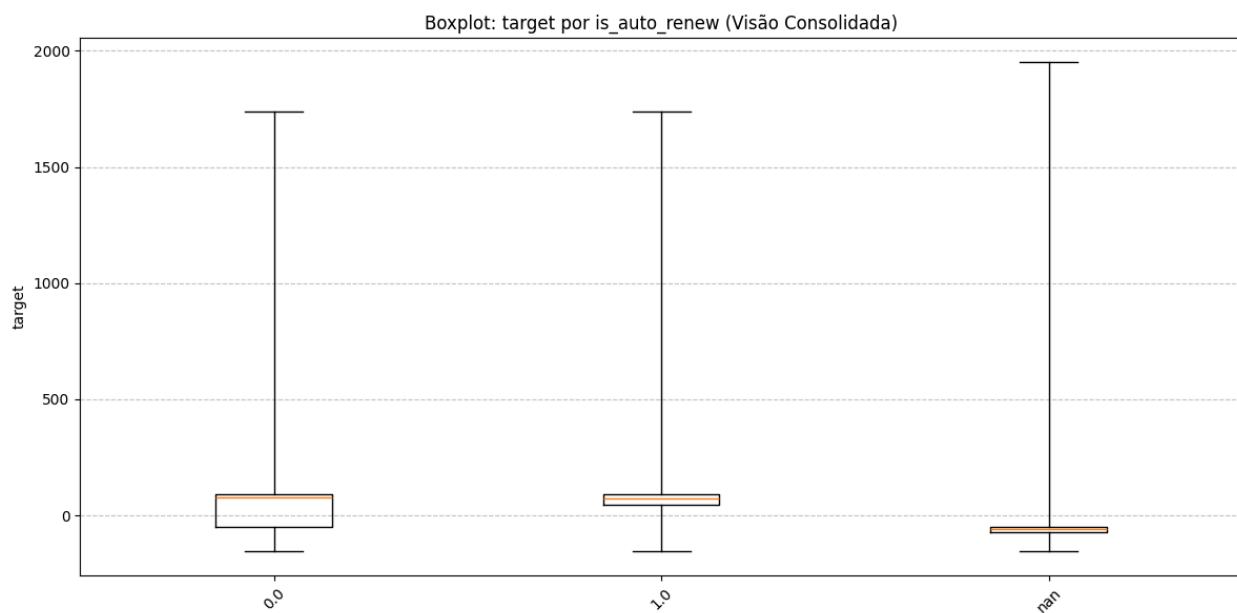
```
In [ ]: calcular_distribuicao(df_base_trans, ["is_auto_renew"], n_show=3)
```

is_auto_renew	total	pct_total
1	7863515	69.94
NULL	1852382	16.48
0	1526968	13.58

```
DataFrame[is_auto_renew: int, total: bigint, pct_total: double]
```

```
In [ ]: plot_boxplot(df_base_trans, ["is_auto_renew"], "target", agrupar_por_safra=False, table=True)
```

```
Processando estatísticas para: is_auto_renew...
```



```
--- Estatísticas: is_auto_renew (Visão Consolidada) ---
```

	is_auto_renew	min	q1	med	q3	max
2	0.0	-152.5452	-51.490648	76.731637	93.358386	1737.859653
1	1.0	-152.5452	45.941681	71.079558	92.169249	1737.041373
0	NaN	-152.5452	-71.491939	-58.890695	-50.309033	1950.000000

```
=====
```

8.2.5.2. payment_method_group

```
In [144]: df_payment_method_group = (
    df_base_trans
    .groupBy("payment_method_id")
    .agg(F.mean("is_auto_renew").alias("pct_auto_renew"))
    .withColumn("payment_method_group",
        F.when(F.col("pct_auto_renew").isNull(), "00_no_transactions")
        .when(F.col("pct_auto_renew") >= 0.90, "01_most_auto")
        .when(F.col("pct_auto_renew") >= 0.30, "02_mixed")
        .otherwise("03_most_manual"))
    .select("payment_method_id", "payment_method_group"))

df_base_trans = (
    df_base_trans
    .join(df_payment_method_group, "payment_method_id", "left"))

df_base_trans = df_base_trans.withColumn("payment_method_group", F.when(F.col("payment_method_group").isNull(),
"00_no_transactions").otherwise(F.col("payment_method_group"))))
```

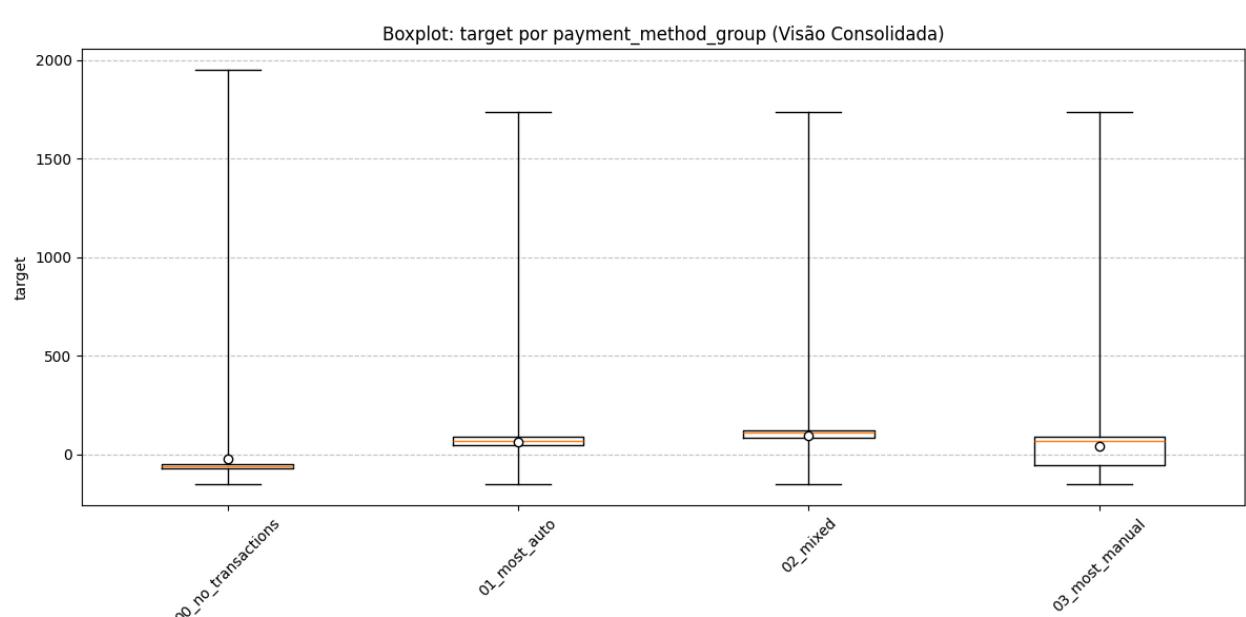
```
In [145]: calcular_distribuicao(df_base_trans, ["payment_method_group"], n_show=5)
```

payment_method_group	total	pct_total
01_most_auto	7629976	67.87
00_no_transactions	1852382	16.48
03_most_manual	1261339	11.22
02_mixed	499168	4.44

Out[145]: DataFrame[payment_method_group: string, total: bigint, pct_total: double]

```
In [146]: plot_boxplot(df_base_trans, ["payment_method_group"], "target", agrupar_por_safra=False, table=True)
```

[146]: Processando estatísticas para: payment_method_group...



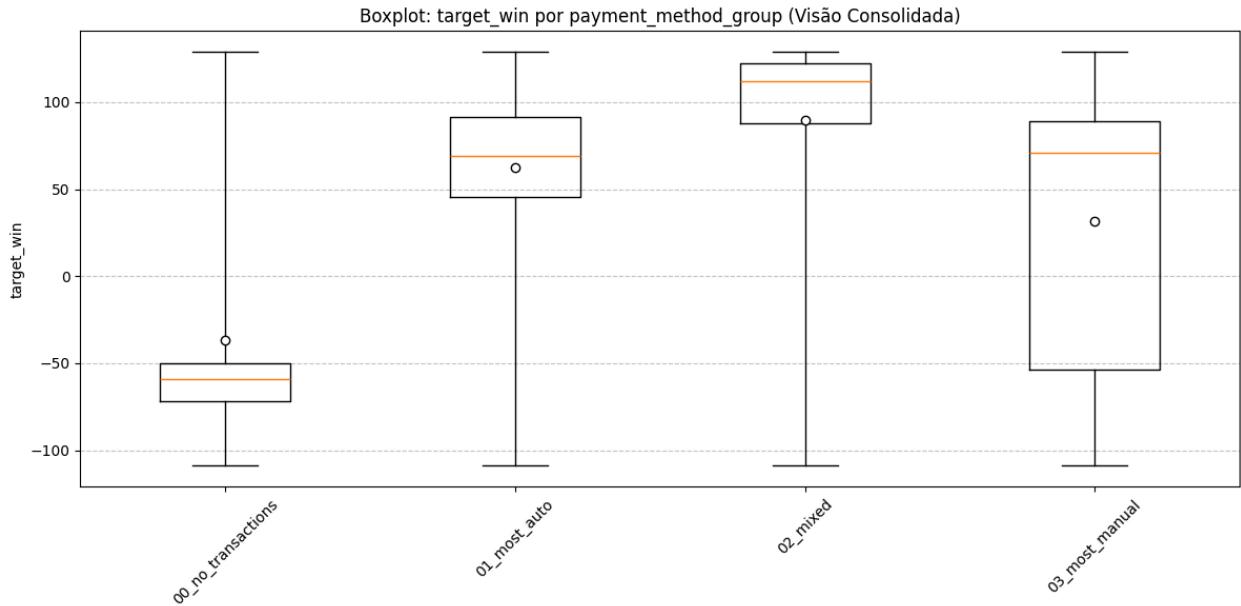
--- Estatísticas: payment_method_group (Visão Consolidada) ---

	payment_method_group	min	q1	med	mean	q3	max
0	00_no_transactions	-152.5452	-71.492186	-58.890549	-21.018945	-50.308344	1950.000000
2	01_most_auto	-152.5452	45.637513	69.378596	62.808500	91.288681	1737.041373
3	02_mixed	-152.5452	87.818467	111.897146	93.602949	122.573745	1737.859653
1	03_most_manual	-152.5452	-53.371603	70.838180	41.530809	88.995987	1736.987900

=====

```
In [147]: plot_boxplot(df_base_trans, ["payment_method_group"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: payment_method_group...



--- Estatísticas: payment_method_group (Visão Consolidada) ---

	payment_method_group	min	q1	med	mean	q3	max
0	00_no_transactions	-108.964925	-71.492186	-58.890549	-36.612835	-50.308344	128.953521
2	01_most_auto	-108.964925	45.637513	69.378596	62.608274	91.288681	128.953521
3	02_mixed	-108.964925	87.818467	111.897146	89.375366	122.573745	128.953521
1	03_most_manual	-108.964925	-53.371603	70.838180	31.897072	88.995987	128.953521

=====

Conclusao

Levar. A variavel `is_auto_renew` apresenta alta correlacao estrutural com `payment_method_id`, como visto na grafico da correlacao durante o EDA inicial, indicando redundancia informacional entre as duas. Apesar de `is_auto_renew` capturar o estado operacional no mes referencia, ela não representa um comportamento estrutural do cliente, variando conforme eventos pontuais.

Já a variavel `payment_method_group`, por outro lado, sintetiza o comportamento historico do meio de pagamento, refletindo padroes persistentes de renovacao (auto, manual ou mista). Ao analisar junto da `target`, percebe-se que que o grupo 02_mixed apresenta mediana de margem futura superior as demais, comportamento que não fica claramente distinguido por `is_auto_renew`.

Alem disso, a categoria 00_no_transactions isola de forma nitida clientes sem atividade financeira no mes, efeito altamente informativo que seria diluído ao utilizar apenas `is_auto_renew`.

8.2.6. diferença entre pagamento e preco

versao continua

```
In [ ]: # Razão entre pago e preço listado
df_base_trans = df_base_trans.withColumn("payment_price_ratio",
F.when(F.col("plan_list_price") > 0, F.col("actual_amount_paid") / F.col("plan_list_price"))))
```

```
In [ ]: df_base_trans.select("payment_price_ratio").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

summary payment_price_ratio	
count	9081504
mean	0.9938967180386512
stddev	0.07766425629635246
min	0.0
1%	1.0
5%	1.0
25%	1.0
50%	1.0
75%	1.0
95%	1.0
99.5%	1.0
max	1.007936507936508

```
In [ ]: calcular_distribuicao(df_base_trans, ["payment_price_ratio"])
```

 payment_price_ratio total pct_total 			
1.0	9024344	80.27	
NULL	2161361	19.22	
0.0	55022	0.49	
0.8657718120805369	1455	0.01	
0.6644295302013423	557	0.0	
0.7986577181208053	113	0.0	
1.007936507936508	7	0.0	
0.9933333333333333	3	0.0	
0.9496644295302014	1	0.0	
0.9955257270693513	1	0.0	

only showing top 10 rows

DataFrame[*payment_price_ratio*: double, *total*: bigint, *pct_total*: double]

```
In [ ]: df_base_trans.select("payment_price_ratio", "target").corr("payment_price_ratio", "target")
```

0.3564516665931757

Pela correlacao, conclui-se que aa variável carrega sinal, mas o sinal não vem da distribuicao contínua, e sim do regime (pagou / não pagou / não havia preço).

payment_price_ratio apresenta alta concentração em 1, com baixa variabilidade contínua. Apesar da correlacao ser moderada, nao significa que traz informacao nova e deve ser obrigatoriamente considerada. Modelos lineares vão tratar 0.95 vs 1 como diferença relevante (errado), arvores vao splitar em valores raríssimos e o efeito real (pagou vs não pagou) fica diluído.

versao categorica

```
In [152]: df_base_trans = df_base_trans.withColumn("payment_price_regime",
    F.when(F.col("plan_list_price").isNull(), "unknown")
    .when((F.col("plan_list_price") > 0) & (F.col("actual_amount_paid") == 0), "no_payment")
    .when((F.col("plan_list_price") > 0) & (F.col("actual_amount_paid") == F.col("plan_list_price")), "paid_as_expected")
    .when((F.col("plan_list_price") > 0) & (F.col("actual_amount_paid") > F.col("plan_list_price")), "overpay")
    .otherwise("other"))
```

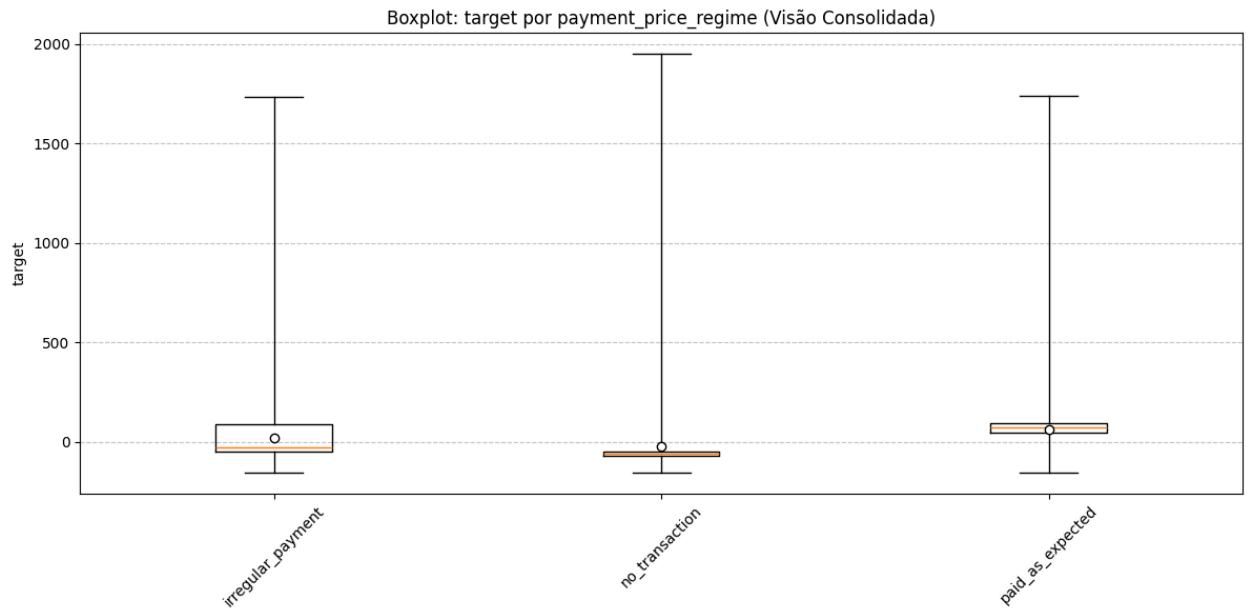
```
In [ ]: calcular_distribuicao(df_base_trans, ["payment_price_regime"], n_show=5)
```

 payment_price_regime total pct_total 			
paid_as_expected	9024344	80.27	
unknown	1852382	16.48	
other	311109	2.77	
no_payment	55022	0.49	
overpay	8	0.0	

DataFrame[*payment_price_regime*: string, *total*: bigint, *pct_total*: double]

```
In [151]: plot_boxplot(df_base_trans, ["payment_price_regime"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: payment_price_regime...



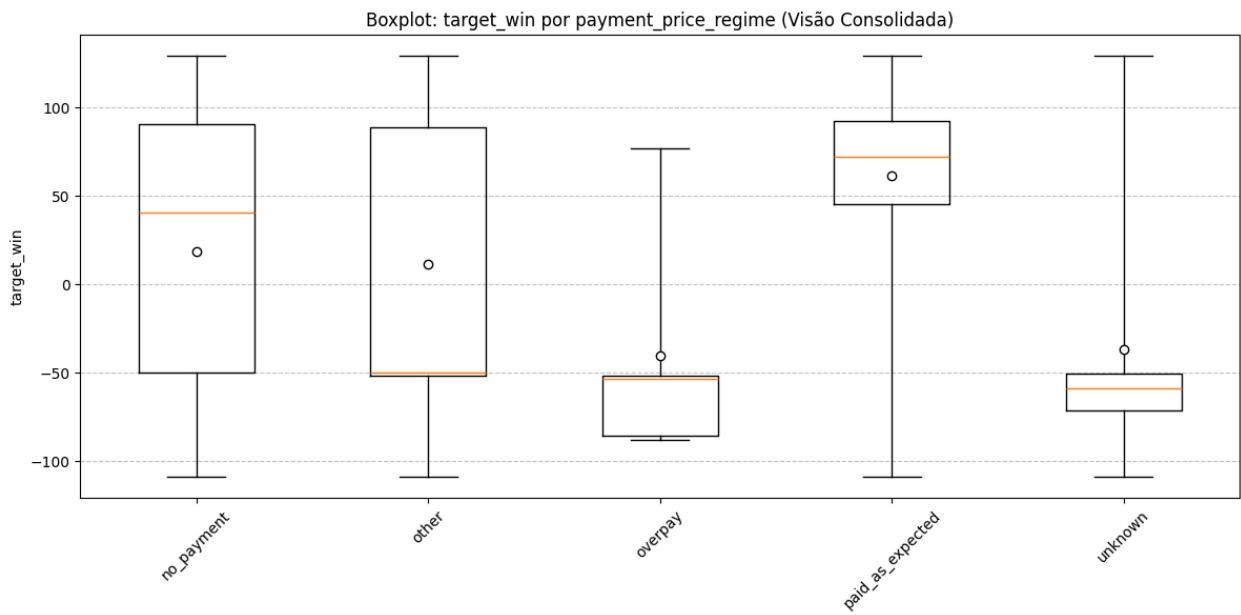
--- Estatísticas: payment_price_regime (Visão Consolidada) ---

	payment_price_regime	min	q1	med	mean	q3	max
1	irregular_payment	-152.5452	-50.701099	-24.933247	21.725562	89.644202	1735.412335
2	no_transaction	-152.5452	-71.491939	-58.890695	-21.018945	-50.309033	1950.000000
0	paid_as_expected	-152.5452	45.288213	72.160281	62.500997	92.343162	1737.859653

=====

```
In [153]: plot_boxplot(df_base_trans, ["payment_price_regime"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: payment_price_regime...



--- Estatísticas: payment_price_regime (Visão Consolidada) ---

	payment_price_regime	min	q1	med	mean	q3	max
4	no_payment	-108.964925	-50.101257	40.593152	18.753815	90.704765	128.953521
2	other	-108.964925	-51.907176	-50.000000	11.370332	88.472847	128.953521
0	overpay	-88.220061	-85.853643	-53.367111	-40.467399	-51.896180	77.000000
3	paid_as_expected	-108.964925	45.288213	72.160281	61.135881	92.343162	128.953521
1	unknown	-108.964925	-71.491939	-58.890695	-36.612835	-50.309033	128.953521

=====

Apesar dos resultados, a variavel em questao conversa com outras flags ja construidas (como `flag_no_payment`, a qual foi descartada, e `flag_valid_fee`, que contempla distribuicao muito similar com a categoria `paid_as_expected`). Ela nao cria um novo eixo explicativo, apenas reforca um ja existente. Ela compensa para responder a pergunta: “Esse usuário costuma pagar exatamente o que foi especificado?”, portanto seu papel esta mais relacionado a refinamento comportamental, nao estrutura principal. Decisao: refinar um pouco mais sua construcao, ja que na etapa de feature selection sera definido se compensa ou nao ir para o modelo escolhido.

```
In [148]: df_base_trans = df_base_trans.withColumn("payment_price_regime", F.when((F.col("plan_list_price") > 0) & (F.col("actual_amount_paid") == F.col("plan_list_price")), "paid_as_expected").when(F.col("plan_list_price").isNull(), "no_transaction").otherwise("irregular_payment"))
```

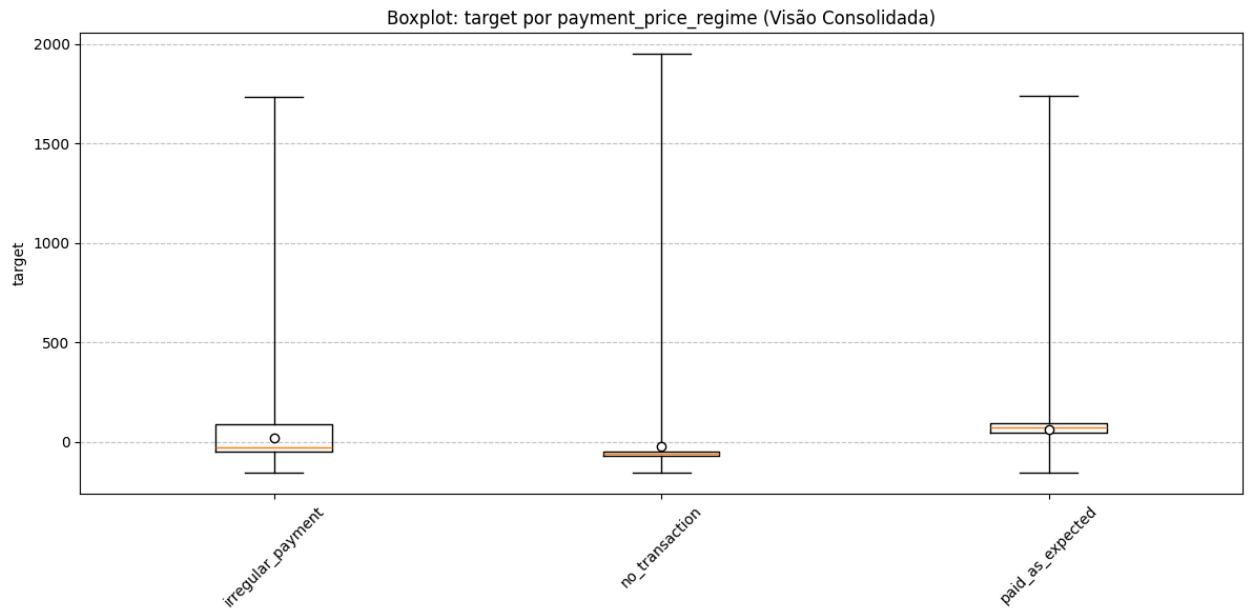
```
In [ ]: calcular_distribuicao(df_base_trans, ["payment_price_regime"], n_show=5)
```

payment_price_regime	total	pct_total
paid_as_expected	9024344	80.27
no_transaction	1852382	16.48
irregular_payment	366139	3.26

```
DataFrame[payment_price_regime: string, total: bigint, pct_total: double]
```

```
In [150]: plot_boxplot(df_base_trans, ["payment_price_regime"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: payment_price_regime...



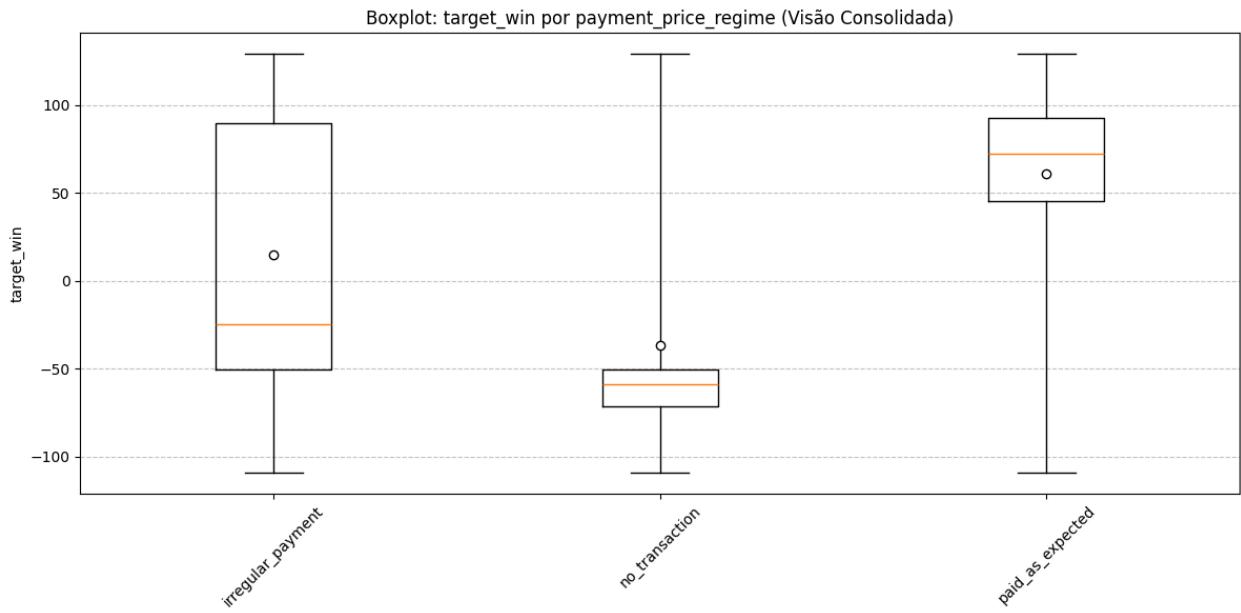
--- Estatísticas: payment_price_regime (Visão Consolidada) ---

	payment_price_regime	min	q1	med	mean	q3	max
1	irregular_payment	-152.5452	-50.701099	-24.933247	21.725562	89.644202	1735.412335
2	no_transaction	-152.5452	-71.491939	-58.890695	-21.018945	-50.309033	1950.000000
0	paid_as_expected	-152.5452	45.288213	72.160281	62.500997	92.343162	1737.859653

=====

```
In [149]: plot_boxplot(df_base_trans, ["payment_price_regime"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: payment_price_regime...



--- Estatísticas: payment_price_regime (Visão Consolidada) ---

	payment_price_regime	min	q1	med	mean	q3	max
1	irregular_payment	-108.964925	-50.701099	-24.933247	14.533862	89.644202	128.953521
2	no_transaction	-108.964925	-71.491939	-58.890695	-36.612835	-50.309033	128.953521
0	paid_as_expected	-108.964925	45.288213	72.160281	61.135881	92.343162	128.953521

=====

8.2.7. expire date invalida

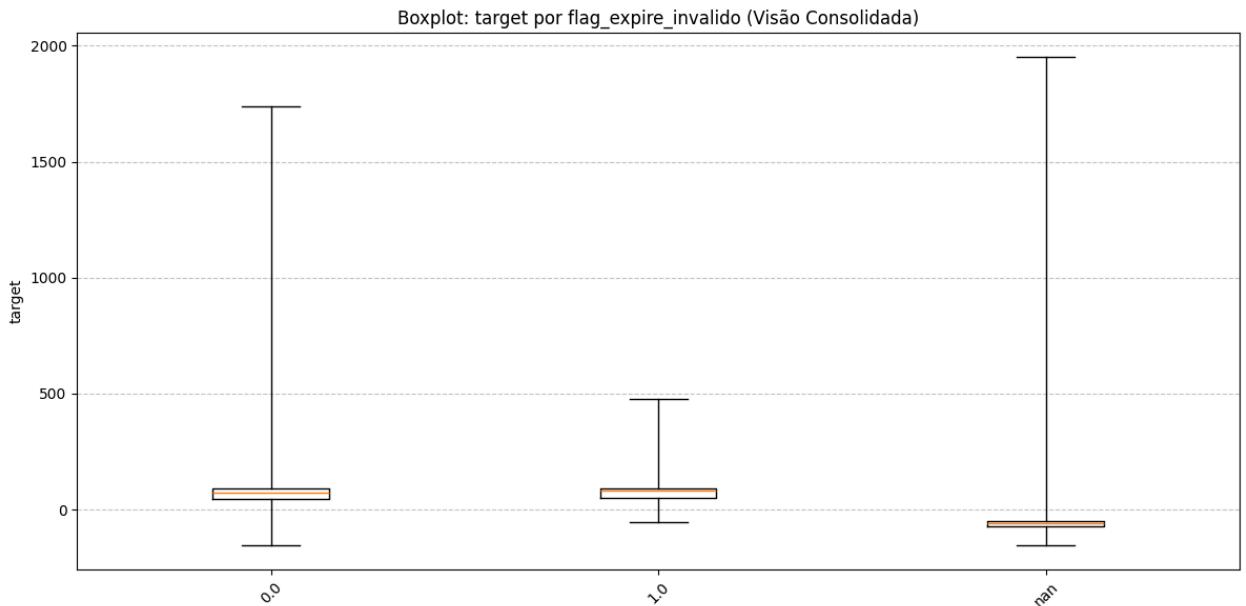
```
In [ ]: calcular_distribuicao(df_base_trans, ["flag_expire_invalido"], n_show=3)
```

flag_expire_invalido	total	pct_total
0	9390439	83.52
NULL	1852382	16.48
1	44	0.0

```
DataFrame[flag_expire_invalido: int, total: bigint, pct_total: double]
```

```
In [ ]: plot_boxplot(df_base_trans, ["flag_expire_invalido"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_expire_invalido...



--- Estatísticas: flag_expire_invalido (Visão Consolidada) ---

	flag_expire_invalido	min	q1	med	q3	max
2	0.0	-152.545200	45.088921	71.993359	92.325581	1737.859653
1	1.0	-55.752099	52.318236	82.121130	90.683575	476.451660
0	NaN	-152.545200	-71.491939	-58.890695	-50.309033	1950.000000

=====

Olhando a `flag_expire_invalido`, construída para os casos em que a data do final da assinatura do cliente estava com valores estranhos, vi que faria mais sentido separar entre os casos em que existe data e casos em que não (não houve transação). E ai reparo que acaba se tornando mais uma variável que distingue quando não há transação, pois todas as análises de `target` giram em torno das estatísticas com valores de -50, com exceção do máximo em 1950. Não compensa, no final das contas, somente uma variável que determina se tem ou não tem transação? O que pode se aplicar para logs, naturalmente.

8.3. Members

```
In [179]: members_vars = [
    "bd",
    "gender",
    "city",
    "registered_via",
    "registration_init_time",
    "idade_clean",
    "flag_idade_invalida",
    "gender_clean",
]
df_base_members = df_base.select("msno", "safra", "target", "target_win", *members_vars)
```

8.3.1. bd (idade)

```
In [180]: df_base_members = df_base_members.withColumn("faixa_idade",
F.when(F.col("flag_idade_invalida") == 1, "Desconhecido")
.when((F.col("idade_clean") >= 16) & (F.col("idade_clean") <= 22), "16-22")
.when((F.col("idade_clean") >= 23) & (F.col("idade_clean") <= 34), "23-34")
.when((F.col("idade_clean") >= 35) & (F.col("idade_clean") <= 44), "35-44")
.when((F.col("idade_clean") >= 45) & (F.col("idade_clean") <= 52), "45-52")
.otherwise("53+"))
```

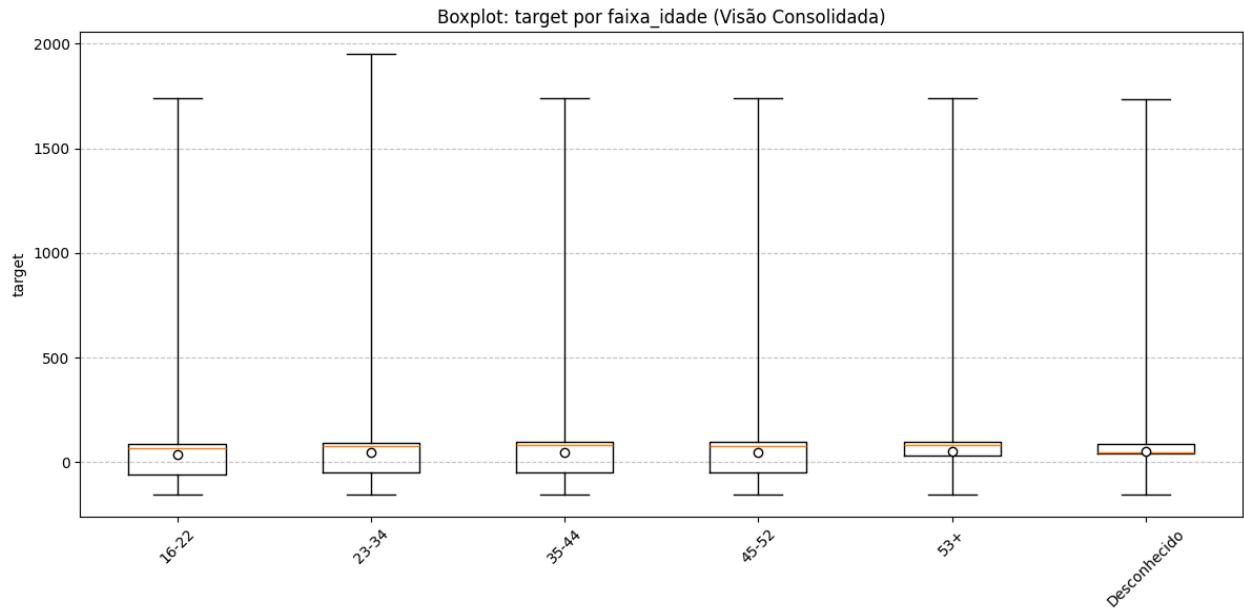
In [50]: calcular_distribuicao(df_base_members, ["faixa_idade"], n_show=5)

```
+-----+-----+
|faixa_idade |total |pct_total|
+-----+-----+
|Desconhecido|5478302|48.73   |
|23-34       |3192274|28.39   |
|16-22       |1118450|9.95    |
|35-44       |1020164|9.07    |
|45-52       |301403 |2.68    |
+-----+-----+
only showing top 5 rows
```

Out[50]: DataFrame[faixa_idade: string, total: bigint, pct_total: double]

In [177]: plot_boxplot(df_base_members, ["faixa_idade"], "target", agrupar_por_safra=False, table=True)

Processando estatísticas para: faixa_idade...



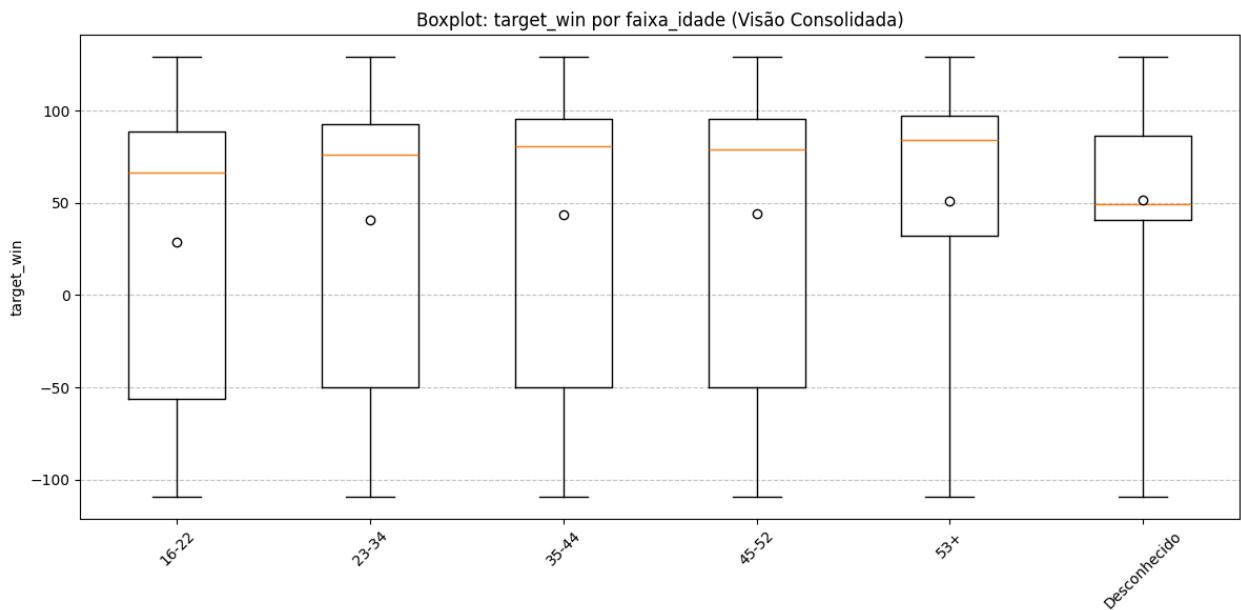
--- Estatísticas: faixa_idade (Visão Consolidada) ---

	faixa_idade	min	q1	med	mean	q3	max
5	16-22	-152.5452	-56.433437	66.439765	38.842285	88.639285	1736.987900
0	23-34	-152.5452	-50.146468	75.977863	46.142643	92.559926	1950.000000
3	35-44	-152.5452	-50.000000	80.585296	47.606177	95.623010	1737.929378
2	45-52	-152.5452	-50.000000	79.176308	47.673092	95.283569	1737.947047
4	53+	-152.5452	32.142554	84.064988	54.183717	97.022703	1737.137480
1	Desconhecido	-152.5452	40.672108	49.000000	52.681467	86.293879	1736.347631

=====

```
In [181]: plot_boxplot(df_base_members, ["faixa_idade"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: faixa_idade...



--- Estatísticas: faixa_idade (Visão Consolidada) ---

	faixa_idade	min	q1	med	mean	q3	max
5	16-22	-108.964925	-56.433437	66.439765	29.026862	88.639285	128.953521
0	23-34	-108.964925	-50.146468	75.977863	40.767472	92.559926	128.953521
3	35-44	-108.964925	-50.000000	80.585296	43.565887	95.623010	128.953521
2	45-52	-108.964925	-50.000000	79.176308	43.912887	95.283569	128.953521
4	53+	-108.964925	32.142554	84.064988	51.082567	97.022703	128.953521
1	Desconhecido	-108.964925	40.672108	49.000000	51.311194	86.293879	128.953521

=====

Faixas 35-44 e 45-52 sao proximas. Mesmo pela volumetria, entendo que seja interessante juntar.

```
In [ ]: df_base_members = df_base_members.withColumn("faixa_idade",
    F.when(F.col("flag_idade_invalida") == 1, "Desconhecido")
        .when((F.col("idade_clean") >= 16) & (F.col("idade_clean") <= 22), "16-22")
        .when((F.col("idade_clean") >= 23) & (F.col("idade_clean") <= 34), "23-34")
        .when((F.col("idade_clean") >= 35) & (F.col("idade_clean") <= 52), "35-52")
        .otherwise("53+"))
```

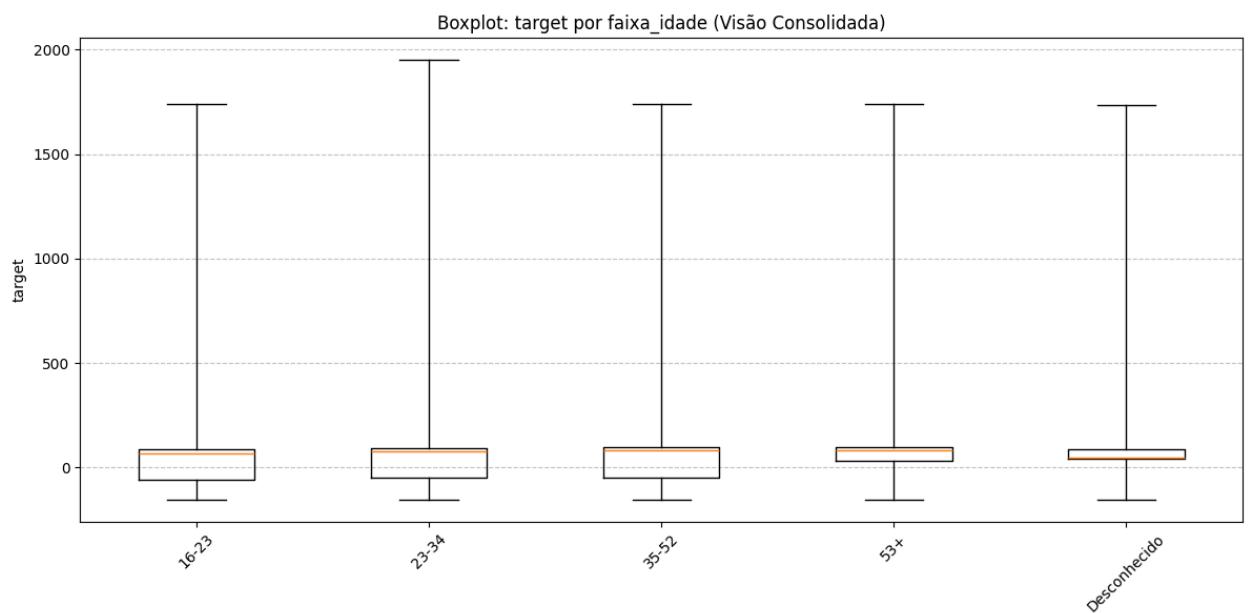
```
In [48]: calcular_distribuicao(df_base_members, ["faixa_idade"], n_show=5)
```

faixa_idade	total	pct_total
Desconhecido	5478302	48.73
23-34	3192274	28.39
35-52	1321567	11.75
16-23	1118450	9.95
53+	132272	1.18

```
Out[48]: DataFrame[faixa_idade: string, total: bigint, pct_total: double]
```

```
In [47]: plot_boxplot(df_base_members, ["faixa_idade"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: faixa_idade...



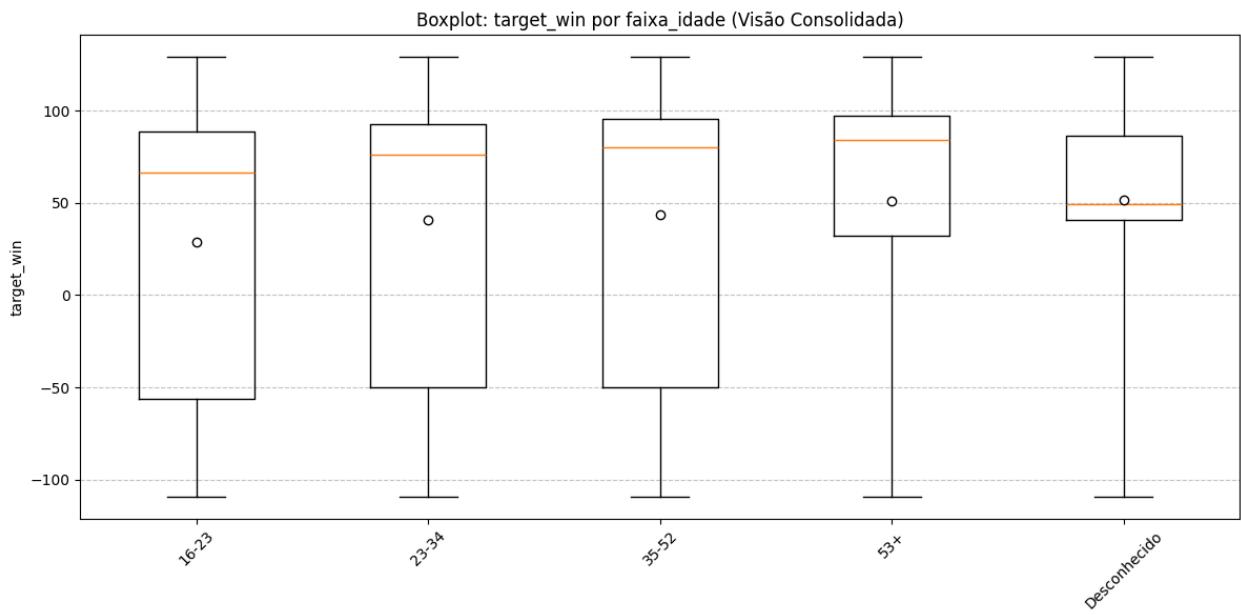
--- Estatísticas: faixa_idade (Visão Consolidada) ---

	faixa_idade	min	q1	med	q3	max
1	16-23	-152.5452	-56.433437	66.439765	88.639285	1736.987900
0	23-34	-152.5452	-50.146468	75.977863	92.559926	1950.000000
4	35-52	-152.5452	-50.000000	80.246616	95.552088	1737.947047
3	53+	-152.5452	32.142554	84.064988	97.022703	1737.137480
2	Desconhecido	-152.5452	40.672108	49.000000	86.293879	1736.347631

=====

```
In [183]: plot_boxplot(df_base_members, ["faixa_idade"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: faixa_idade...



--- Estatísticas: faixa_idade (Visão Consolidada) ---

	faixa_idade	min	q1	med	mean	q3	max
1	16-23	-108.964925	-56.433437	66.439765	29.026862	88.639285	128.953521
0	23-34	-108.964925	-50.146468	75.977863	40.767472	92.559926	128.953521
4	35-52	-108.964925	-50.000000	80.246616	43.645246	95.552088	128.953521
3	53+	-108.964925	32.142554	84.064988	51.082567	97.022703	128.953521
2	Desconhecido	-108.964925	40.672108	49.000000	51.311194	86.293879	128.953521

=====

Essa vai ser a versão final, considerando principalmente as medianas

8.3.2. city

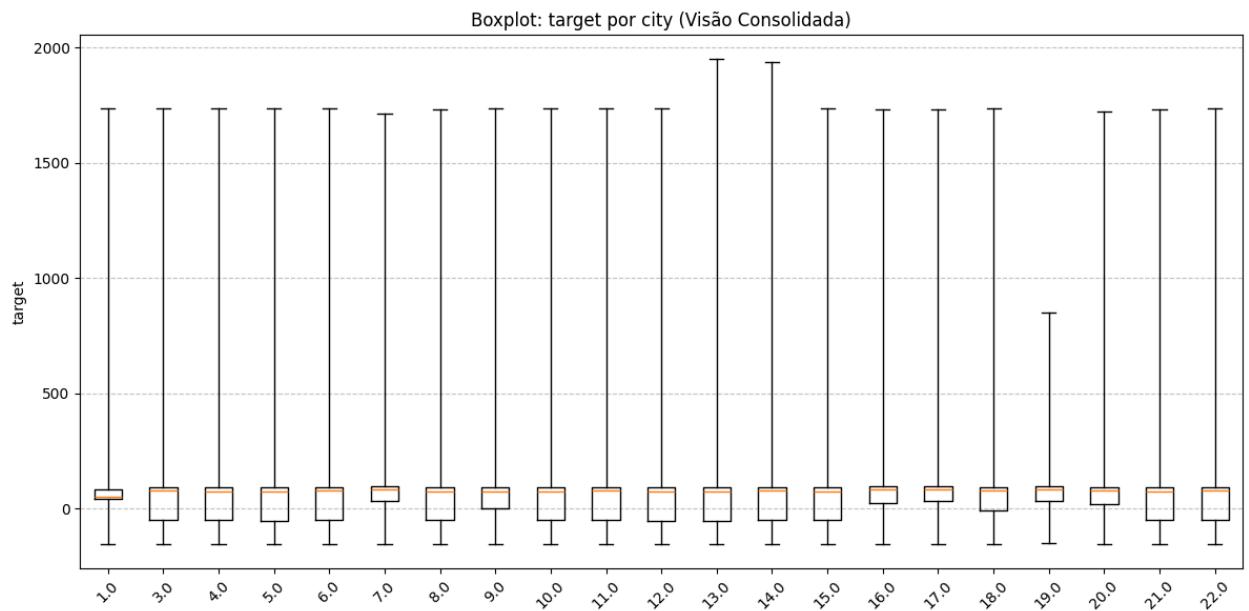
```
In [ ]: calcular_distribuicao(df_base_members, ["city"], n_show=25)
```

city	total	pct_total
1	4986145	44.35
13	1481765	13.18
5	1097664	9.76
4	714795	6.36
15	624620	5.56
22	603971	5.37
6	368930	3.28
14	293125	2.61
12	176023	1.57
9	157790	1.4
11	122816	1.09
8	113924	1.01
18	112453	1.0
10	96482	0.86
21	80295	0.71
3	75874	0.67
17	74709	0.66
7	35673	0.32
16	13862	0.12
20	9965	0.09
19	1984	0.02

```
DataFrame[city: int, total: bigint, pct_total: double]
```

```
In [53]: plot_boxplot(df_base_members, ["city"], "target", agrupar_por_safra=False)
```

Processando estatísticas para: city...



Estrutura de quatro níveis:

```
In [185]: df_base_members = df_base_members.withColumn("city_group",
    F.when(F.col("city").isin(1), "city_one")\
    .when(F.col("city").isin([13, 5, 4]), "top_cities")\
    .when(F.col("city").isin([15, 22, 6, 14]), "mid_cities")\
    .otherwise("other_cities"))
```

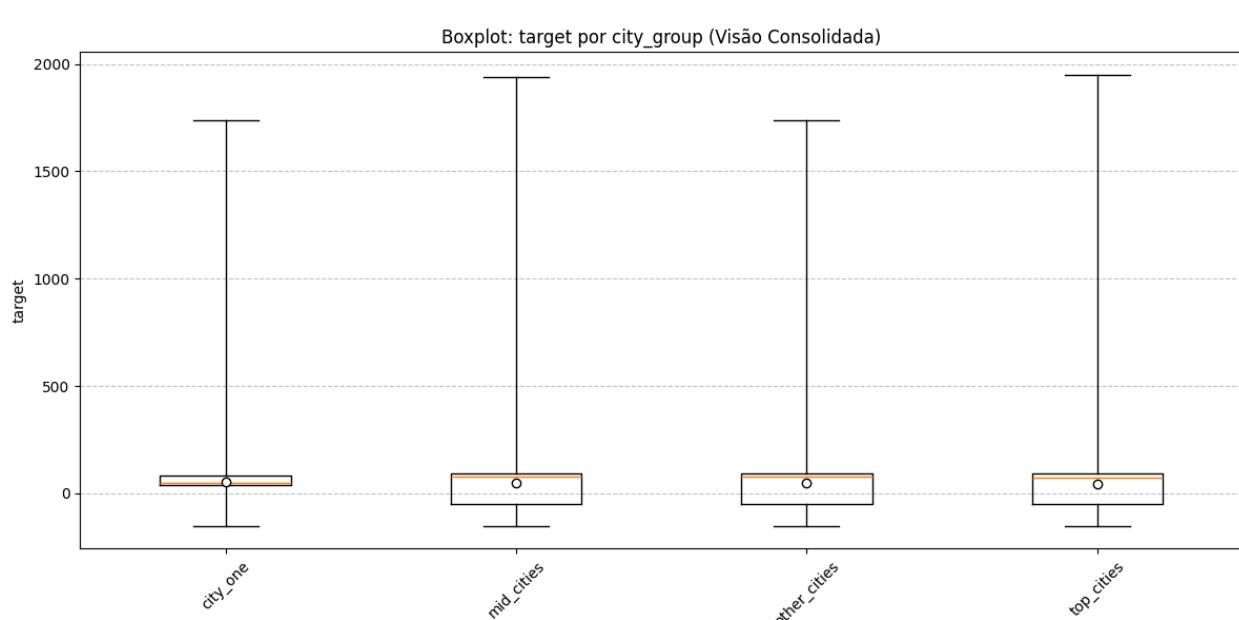
```
In [55]: calcular_distribuicao(df_base_members, ["city_group"])
```

```
+-----+-----+
|city_group |total |pct_total|
+-----+-----+
|city_one   |4986145|44.35   |
|top_cities |3294224|29.3    |
|mid_cities |1890646|16.82   |
|other_cities|1071850|9.53   |
+-----+-----+
```

```
Out[55]: DataFrame[city_group: string, total: bigint, pct_total: double]
```

```
In [186]: plot_boxplot(df_base_members, ["city_group"], "target", agrupar_por_safra=False, table=True)
```

```
[186]: Processando estatísticas para: city_group...
```



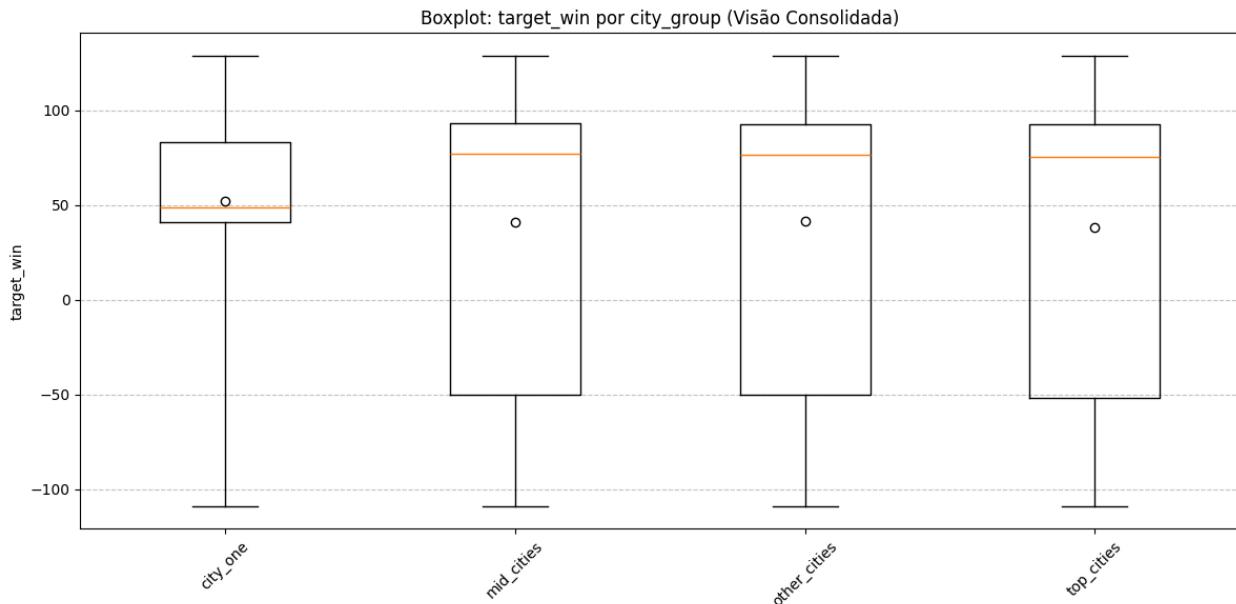
```
--- Estatísticas: city_group (Visão Consolidada) ---
```

	city_group	min	q1	med	mean	q3	max
0	city_one	-152.5452	41.113585	49.000000	53.290157	83.450254	1736.130944
1	mid_cities	-152.5452	-50.368506	77.159952	46.341483	93.363502	1936.799455
3	other_cities	-152.5452	-50.000000	76.347676	47.298061	92.935213	1737.640612
2	top_cities	-152.5452	-51.705813	75.436377	44.295062	92.954920	1950.000000

```
=====
```

```
In [187]: plot_boxplot(df_base_members, ["city_group"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: city_group...



--- Estatísticas: city_group (Visão Consolidada) ---

	city_group	min	q1	med	mean	q3	max
0	city_one	-108.964925	41.113585	49.000000	52.174945	83.450254	128.953521
1	mid_cities	-108.964925	-50.368506	77.159952	40.896772	93.363502	128.953521
3	other_cities	-108.964925	-50.000000	76.347676	41.852474	92.935213	128.953521
2	top_cities	-108.964925	-51.705813	75.436377	38.423098	92.954920	128.953521

=====

Boxplots muito similares para mid_cities, top_cities e other_cities. O agrupamento não cria poder discriminatório real. Ele só reorganiza categorias que já se comportam igual em termos de margem.

```
In [188]: df_base_members = df_base_members.withColumn("flag_city_one", F.when(F.col("city").isin(1), 1).otherwise(0))
```

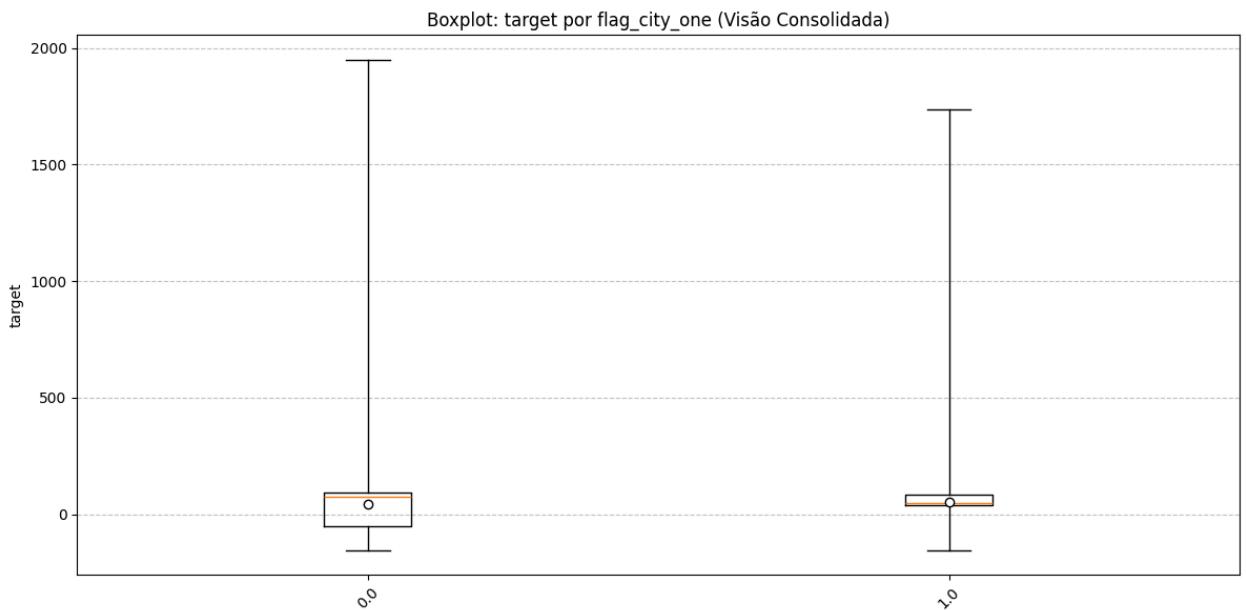
```
In [59]: calcular_distribuicao(df_base_members, ["flag_city_one"])
```

flag_city_one	total	pct_total
0	6256720	55.65
1	4986145	44.35

```
Out[59]: DataFrame[flag_city_one: int, total: bigint, pct_total: double]
```

```
In [189]: plot_boxplot(df_base_members, ["flag_city_one"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_city_one...



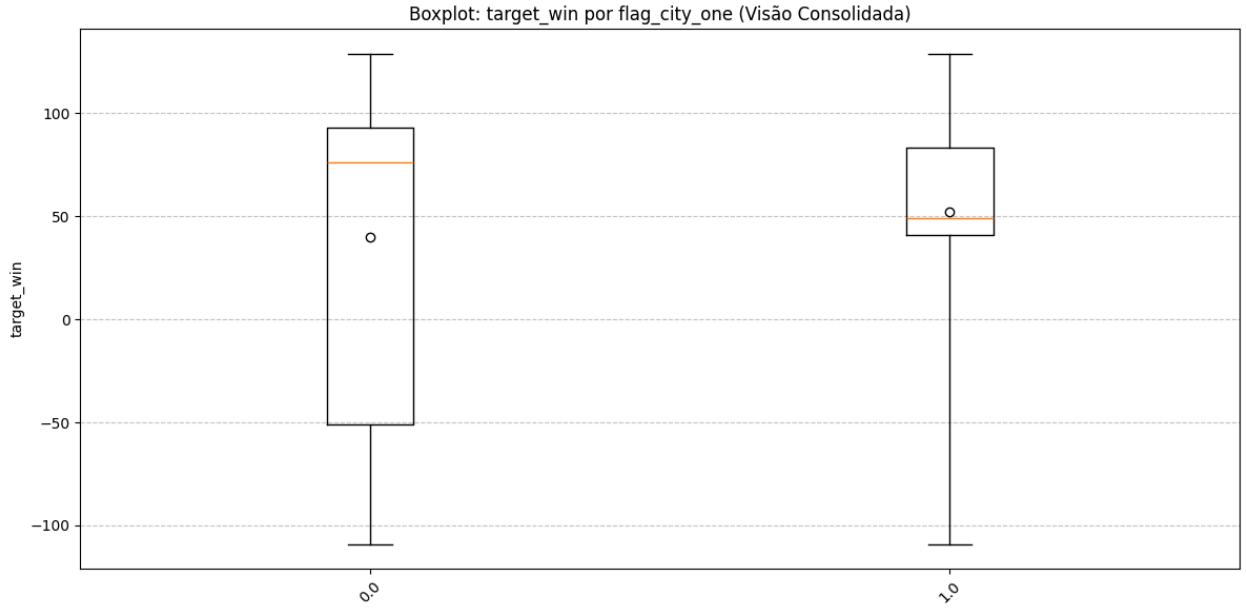
--- Estatísticas: flag_city_one (Visão Consolidada) ---

	flag_city_one	min	q1	med	mean	q3	max
1	0	-152.5452	-50.876032	76.12281	45.423162	93.077254	1950.000000
0	1	-152.5452	41.113585	49.00000	53.290157	83.450254	1736.130944

=====

```
In [190]: plot_boxplot(df_base_members, ["flag_city_one"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_city_one...



--- Estatísticas: flag_city_one (Visão Consolidada) ---

	flag_city_one	min	q1	med	mean	q3	max
1	0	-108.964925	-50.876032	76.12281	39.752541	93.077254	128.953521
0	1	-108.964925	41.113585	49.00000	52.174945	83.450254	128.953521

=====

A análise mostrou que a distribuição da margem líquida mensal é praticamente idêntica entre a maioria das cidades, tanto no nível individual quanto em agrupamentos por frequência. A única exceção consistente é a cidade 1, que apresenta uma distribuição distinta em termos de mediana e dispersão.

Diante disso, optou-se por representar a variável cidade por uma flag binária (`flag_city_one`), capturando o único efeito geográfico relevante sem introduzir complexidade desnecessária ou ruído ao modelo.

8.3.3. gender

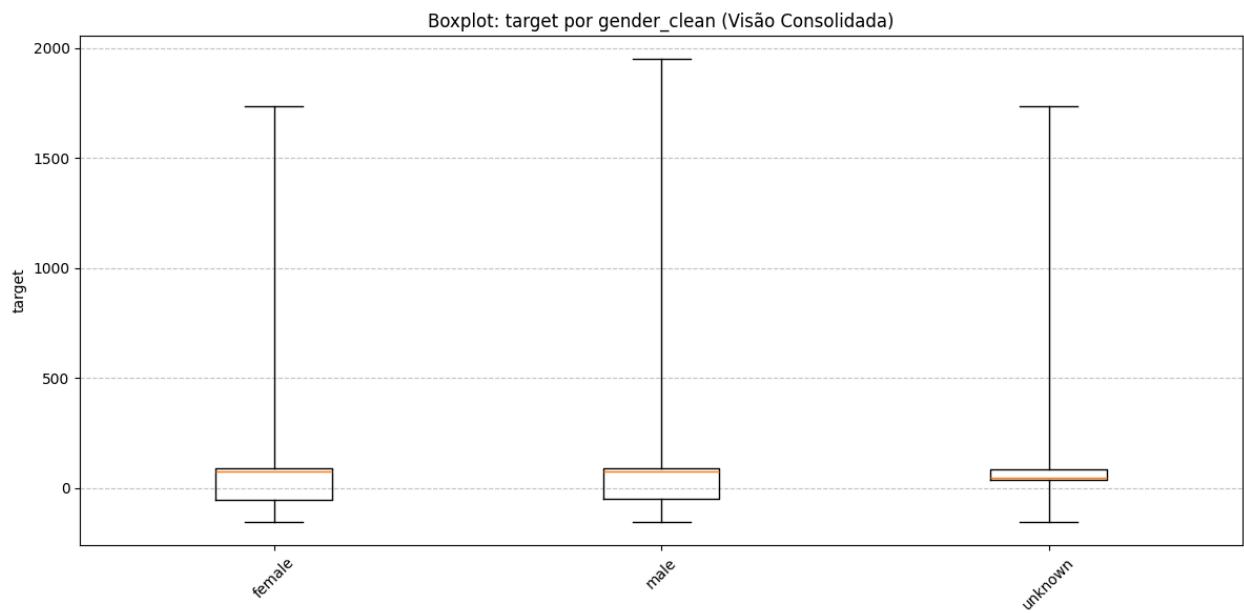
```
In [ ]: calcular_distribuicao(df_base_members, ["gender_clean"])
```

```
+-----+-----+
|gender_clean|total |pct_total|
+-----+-----+
|unknown    |5453964|48.51   |
|male       |3036611|27.01   |
|female     |2752290|24.48   |
+-----+-----+
```

```
DataFrame[gender_clean: string, total: bigint, pct_total: double]
```

```
In [60]: plot_boxplot(df_base_members, ["gender_clean"], "target", agrupar_por_safra=False, table=True)
```

```
Processando estatísticas para: gender_clean...
```



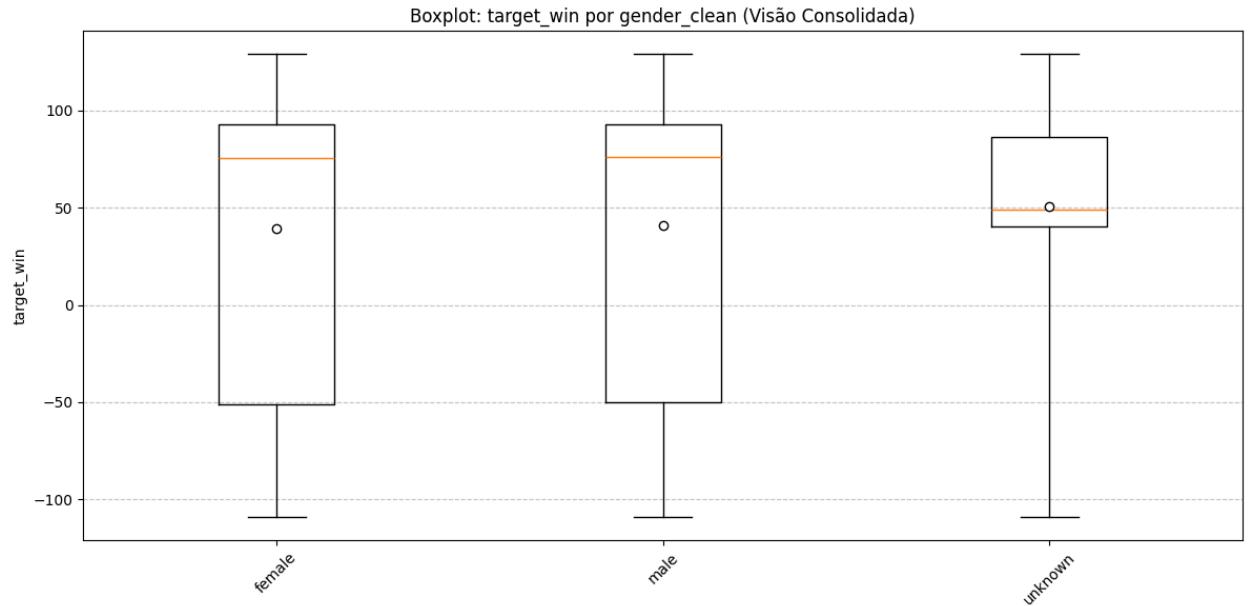
```
--- Estatísticas: gender_clean (Visão Consolidada) ---
```

	gender_clean	min	q1	med	q3	max
1	female	-152.5452	-51.160340	75.598110	93.097249	1737.859653
2	male	-152.5452	-50.028441	75.850492	92.812681	1950.000000
0	unknown	-152.5452	40.427550	49.000000	86.101611	1736.347631

```
=====
```

```
In [191]: plot_boxplot(df_base_members, ["gender_clean"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: gender_clean...



--- Estatísticas: gender_clean (Visão Consolidada) ---

	gender_clean	min	q1	med	mean	q3	max
1	female	-108.964925	-51.160340	75.598110	39.086724	93.097249	128.953521
2	male	-108.964925	-50.028441	75.850492	40.842283	92.812681	128.953521
0	unknown	-108.964925	40.427550	49.000000	50.752770	86.101611	128.953521

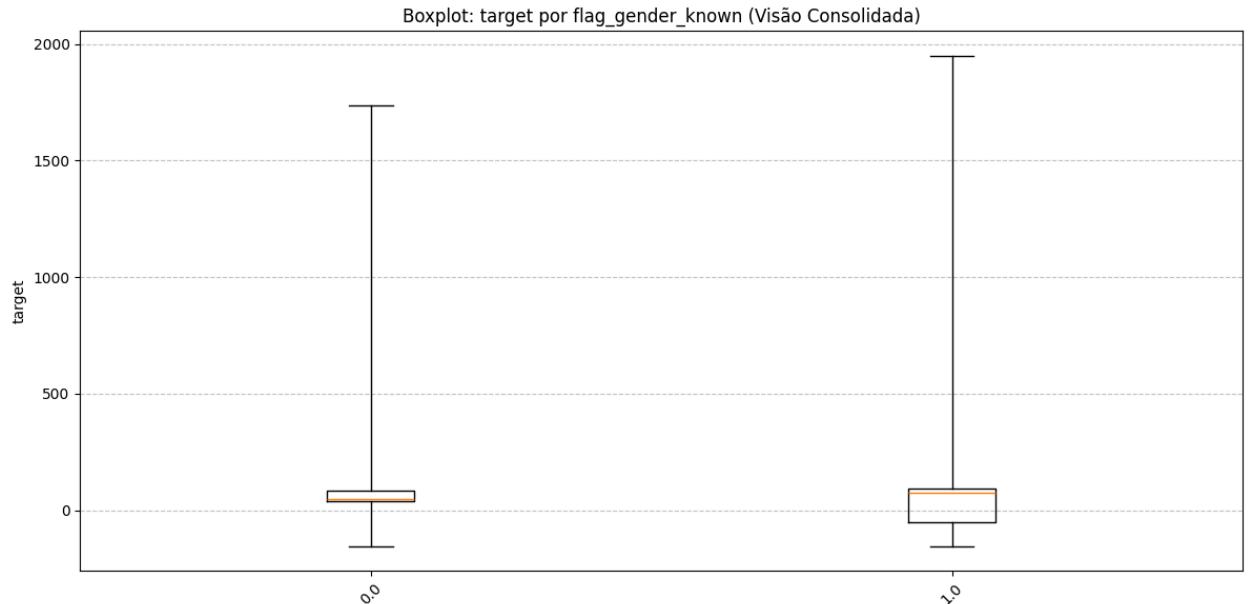
=====

Aparentemente, tambem faz mais sentido manter apenas uma flag para marcar se o genero esta registrado ou nao.

```
In [192]: df_base_members = df_base_members.withColumn("flag_gender_known", F.when(F.col("gender_clean") != "unknown", 1).otherwise(0))
```

```
In [62]: plot_boxplot(df_base_members, ["flag_gender_known"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_gender_known...



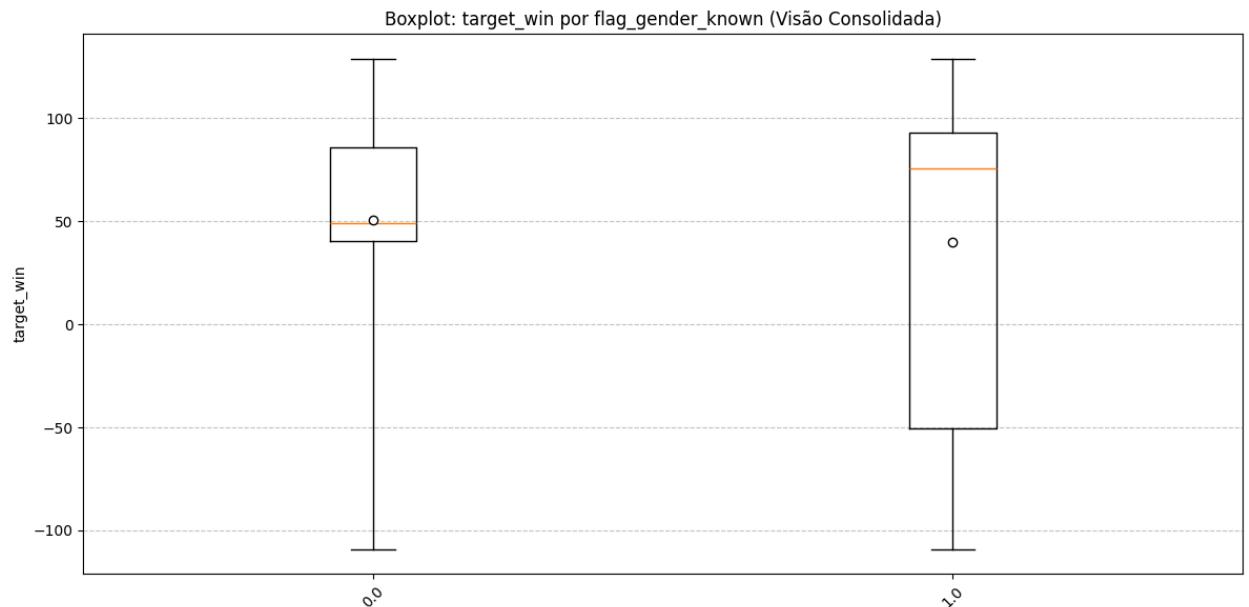
--- Estatísticas: flag_gender_known (Visão Consolidada) ---

	flag_gender_known	min	q1	med	q3	max
1	0	-152.5452	40.427550	49.000000	86.101611	1736.347631
0	1	-152.5452	-50.519827	75.734401	92.949399	1950.000000

=====

```
In [193]: plot_boxplot(df_base_members, ["flag_gender_known"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_gender_known...



--- Estatísticas: flag_gender_known (Visão Consolidada) ---

	flag_gender_known	min	q1	med	mean	q3	max
1	0	-108.964925	40.427550	49.000000	50.752770	86.101611	128.953521
0	1	-108.964925	-50.519827	75.734401	40.007911	92.949399	128.953521

=====

8.3.4. registered_via

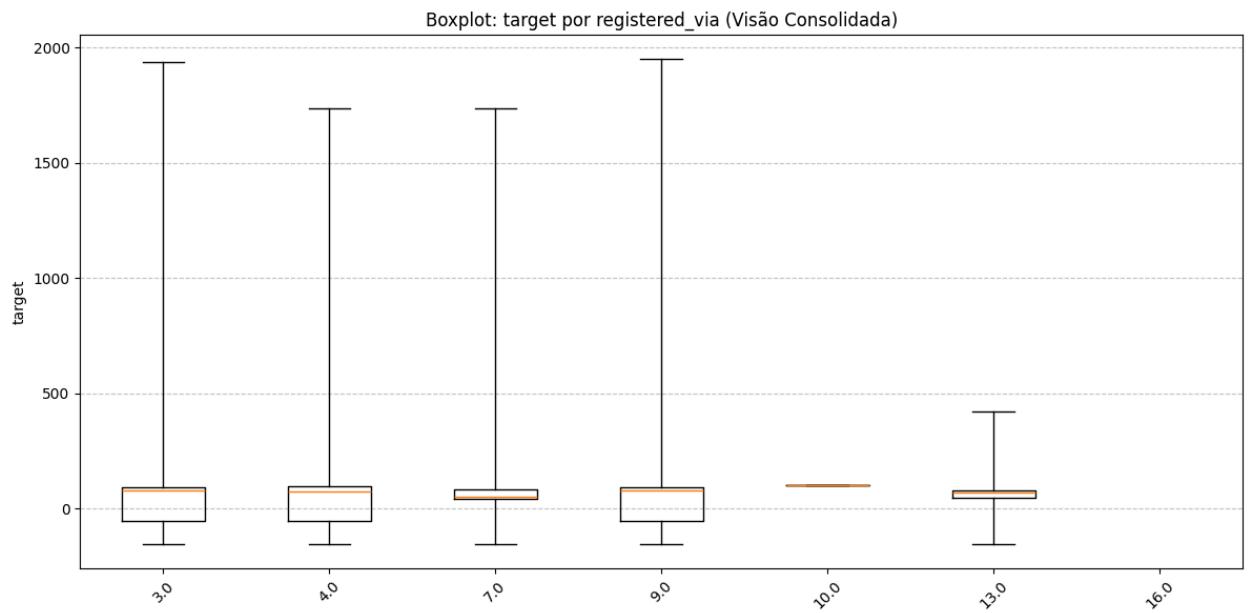
```
In [ ]: calcular_distribuicao(df_base_members, ["registered_via"], n_show=25)
```

registered_via	total	pct_total
7	5201676	46.27
9	3704733	32.95
3	1670379	14.86
4	658926	5.86
13	7140	0.06
10	10	0.0
16	1	0.0

DataFrame[registered_via: int, total: bigint, pct_total: double]

```
In [63]: plot_boxplot(df_base_members, ["registered_via"], "target", agrupar_por_safras=False, table=True)
```

Processando estatísticas para: registered_via...



--- Estatísticas: registered_via (Visão Consolidada) ---

	registered_via	min	q1	med	q3	max
1	3	-152.5452	-55.296765	76.197374	93.504638	1936.799455
4	4	-152.5452	-54.597162	76.029400	95.046895	1736.844913
5	7	-152.5452	41.489970	49.000000	81.934388	1736.522159
3	9	-152.5452	-52.670357	79.218606	94.035633	1950.000000
6	10	99.0000	99.000000	99.000000	99.000000	99.000000
0	13	-152.5452	44.402274	70.978867	79.000000	420.757018
2	16	NaN	NaN	NaN	NaN	NaN

=====

A variável `registered_via` representa o canal de aquisição do cliente.

A análise exploratória mostrou que, embora existam diversos códigos possíveis, mais de 99% da base está concentrada em quatro valores (3, 4, 7 e 9), e que esses canais apresentam comportamentos distintos de margem líquida.

Com base na distribuição da margem e nas estatísticas descritivas, foi possível identificar regimes econômicos claros:

- **High Value (3 e 9)**

Medianas de margem mais altas e maior potencial de rentabilidade, indicando perfis mais engajados ou com maior propensão a conversão.

- **Low Value (7)**

Maior volume da base, porém mediana de margem inferior e menor upside, sugerindo aquisição mais massificada e menos rentável.

- **Mid Value (4)**

Comportamento intermediário, com métricas de margem entre os grupos de alto e baixo valor.

- **Other**

Canais com amostras muito pequenas, evitando ruído e risco de overfitting no modelo.

Essa categorização reduz dimensionalidade, melhora a interpretabilidade e captura diferenças estruturais de rentabilidade entre canais de aquisição, sendo adequada tanto para modelos lineares quanto para modelos baseados em árvores.

```
In [194]: df_base_members = df_base_members.withColumn("registered_via_group", F.when(F.col("registered_via").isin(3, 9), "high_value") .when(F.col("registered_via") == 7, "low_value") .when(F.col("registered_via") == 4, "mid_value") .otherwise("other"))
```

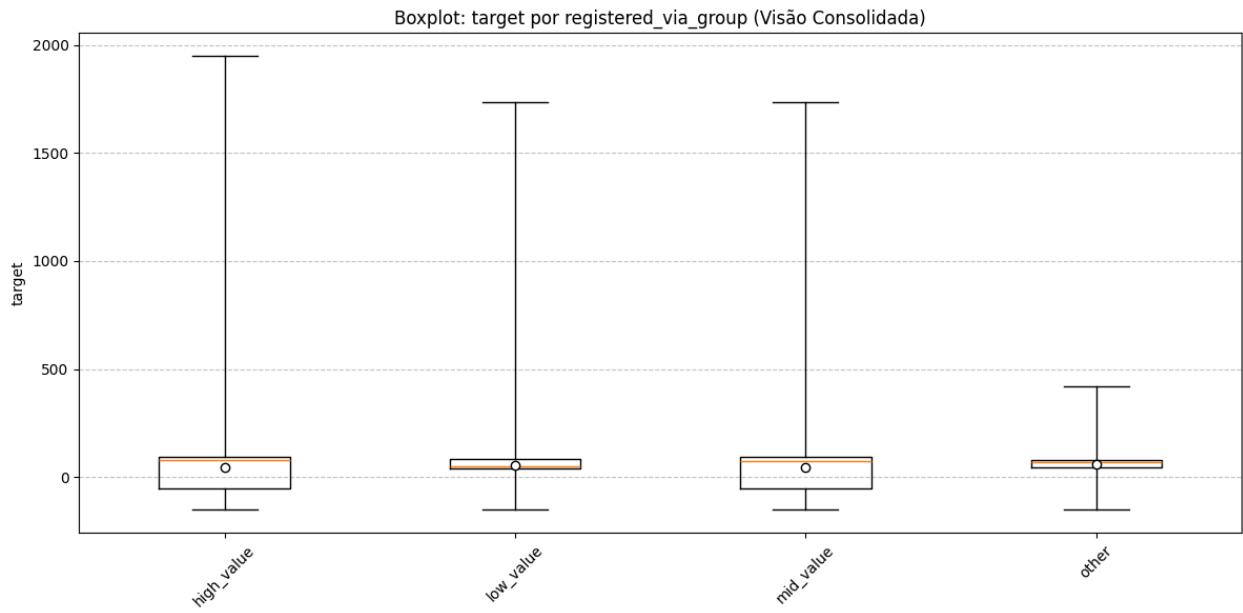
```
In [ ]: calcular_distribuicao(df_base_members, ["registered_via_group"], n_show=25)
```

registered_via_group	total	pct_total
high_value	5375112	47.81
low_value	5201676	46.27
mid_value	658926	5.86
other	7151	0.06

```
DataFrame[registered_via_group: string, total: bigint, pct_total: double]
```

```
In [195]: plot_boxplot(df_base_members, ["registered_via_group"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: registered_via_group...



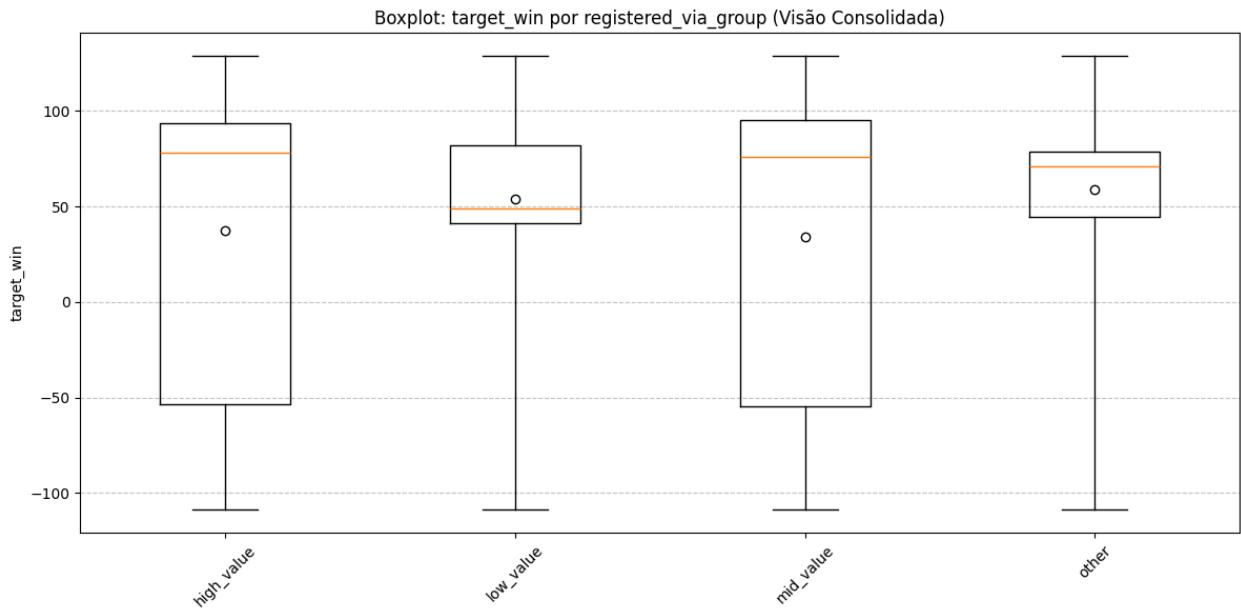
--- Estatísticas: registered_via_group (Visão Consolidada) ---

	registered_via_group	min	q1	med	mean	q3	max
1	high_value	-152.5452	-53.510725	78.419145	44.075802	93.889577	1950.000000
2	low_value	-152.5452	41.489970	49.000000	54.219620	81.934388	1736.522159
3	mid_value	-152.5452	-54.597162	76.029400	42.434752	95.046895	1736.844913
0	other	-152.5452	44.438603	70.998755	58.878411	79.000000	420.757018

=====

```
In [196]: plot_boxplot(df_base_members, ["registered_via_group"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: registered_via_group...



--- Estatísticas: registered_via_group (Visão Consolidada) ---

	registered_via_group	min	q1	med	mean	q3	max
1	high_value	-108.964925	-53.510725	78.419145	37.471944	93.889577	128.953521
2	low_value	-108.964925	41.489970	49.000000	53.870472	81.934388	128.953521
3	mid_value	-108.964925	-54.597162	76.029400	34.232197	95.046895	128.953521
0	other	-108.964925	44.438603	70.998755	58.777226	79.000000	128.953521

=====

Os valores de high_value e mid_value sao bem similares, talvez faça sentido uni-los.

```
In [197]: df_base_members = df_base_members.withColumn("registered_via_group",
    F.when(F.col("registered_via").isin([3, 9, 4]), "high_value")
    .when(F.col("registered_via").isin(7), "low_value")
    .otherwise("other"))
```

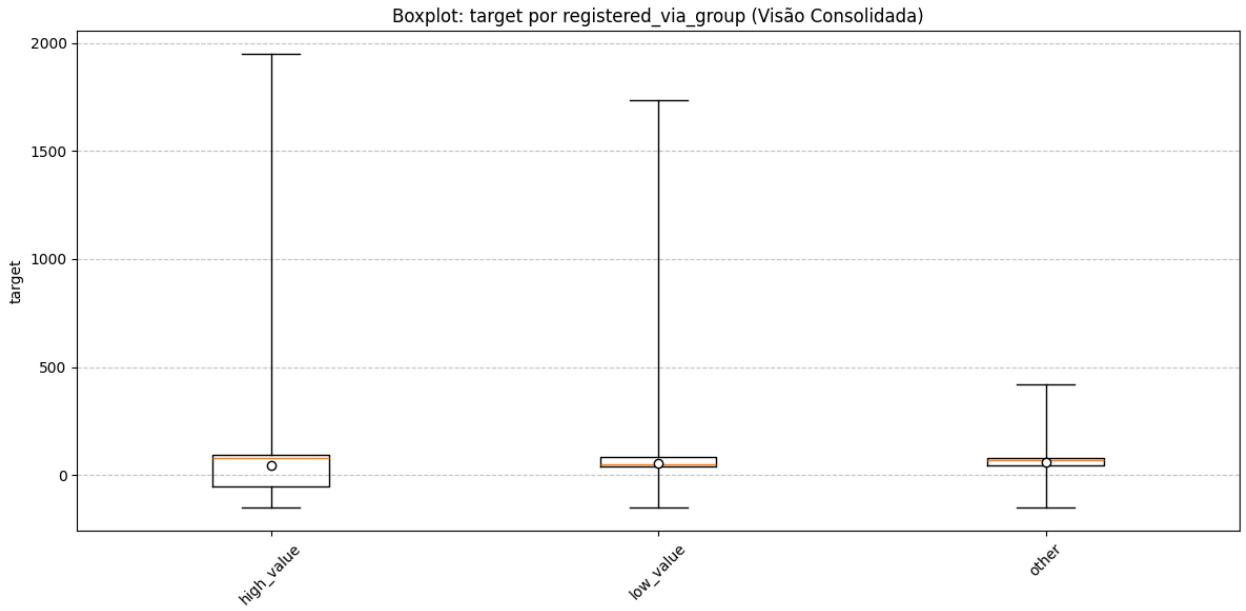
```
In [69]: calcular_distribuicao(df_base_members, ["registered_via_group"], n_show=25)
```

```
+-----+-----+-----+
|registered_via_group|total |pct_total|
+-----+-----+-----+
|high_value          |6034038|53.67   |
|low_value           |5201676|46.27   |
|other               |7151    |0.06    |
+-----+-----+-----+
```

Out[69]: DataFrame[registered_via_group: string, total: bigint, pct_total: double]

```
In [198]: plot_boxplot(df_base_members, ["registered_via_group"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: registered_via_group...



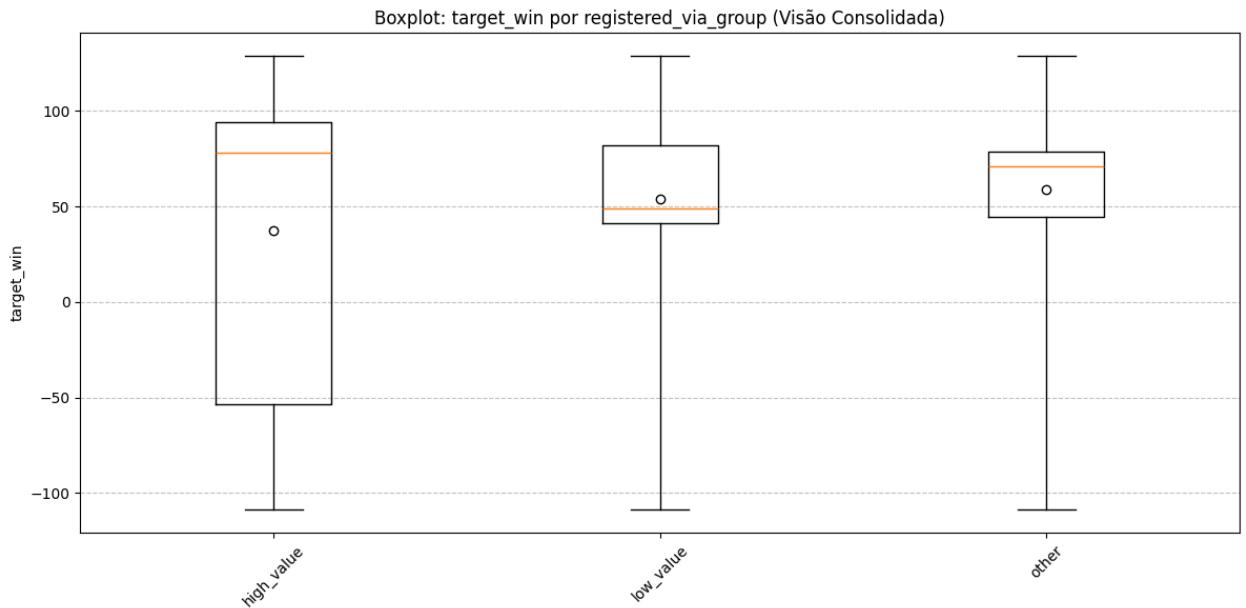
--- Estatísticas: registered_via_group (Visão Consolidada) ---

	registered_via_group	min	q1	med	mean	q3	max
1	high_value	-152.5452	-53.618059	78.259896	43.938683	93.967321	1950.000000
2	low_value	-152.5452	41.489970	49.000000	54.219620	81.934388	1736.522159
0	other	-152.5452	44.438603	70.998755	58.878411	79.000000	420.757018

=====

```
In [199]: plot_boxplot(df_base_members, ["registered_via_group"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: registered_via_group...



--- Estatísticas: registered_via_group (Visão Consolidada) ---

	registered_via_group	min	q1	med	mean	q3	max
1	high_value	-108.964925	-53.618059	78.259896	37.201246	93.967321	128.953521
2	low_value	-108.964925	41.489970	49.000000	53.870472	81.934388	128.953521
0	other	-108.964925	44.438603	70.998755	58.777226	79.000000	128.953521

=====

Como a classe other tem volume irrisorio, uma alternativa mais robusta para regressao linear (ou pode ser que para a arvore tambem, veremos eventualmente) seria uma flag para os casos de high value (3, 9 e 4).

```
In [200]: df_base_members = df_base_members.withColumn("flag_high_value_registered_via", F.when(F.col("registered_via").isin([3, 9, 4]), 1).otherwise(0))
```

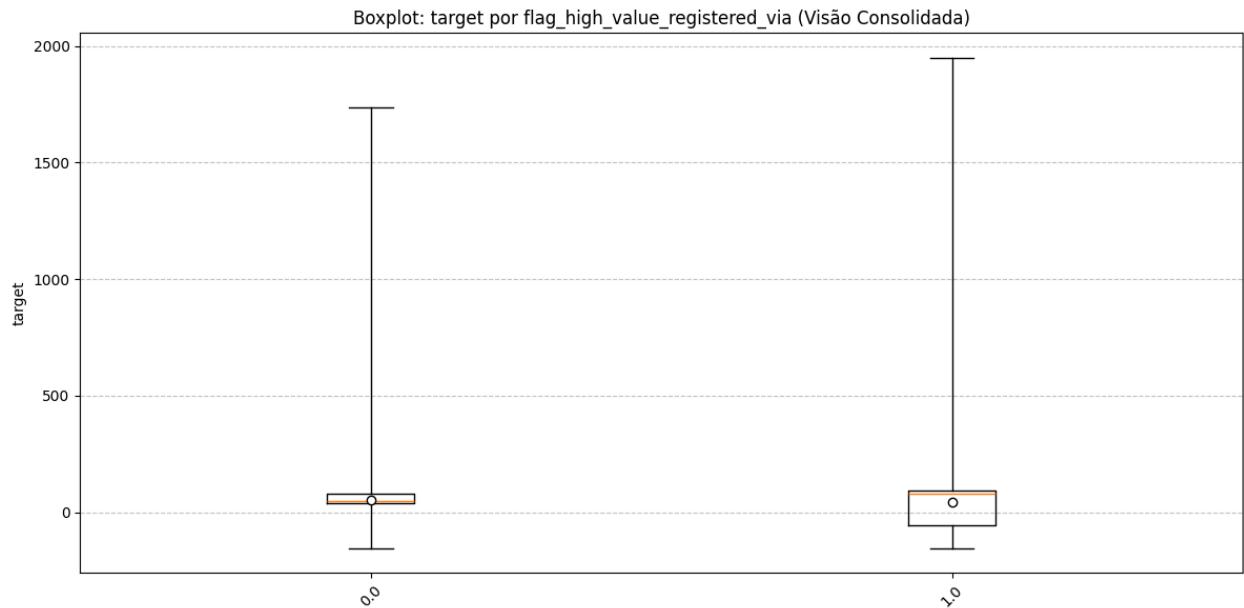
```
In [72]: calcular_distribuicao(df_base_members, ["flag_high_value_registered_via"], n_show=25)
```

flag_high_value_registered_via	total	pct_total
1	6034038	53.67
0	5208827	46.33

```
Out[72]: DataFrame[flag_high_value_registered_via: int, total: bigint, pct_total: double]
```

```
In [201]: plot_boxplot(df_base_members, ["flag_high_value_registered_via"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_high_value_registered_via...



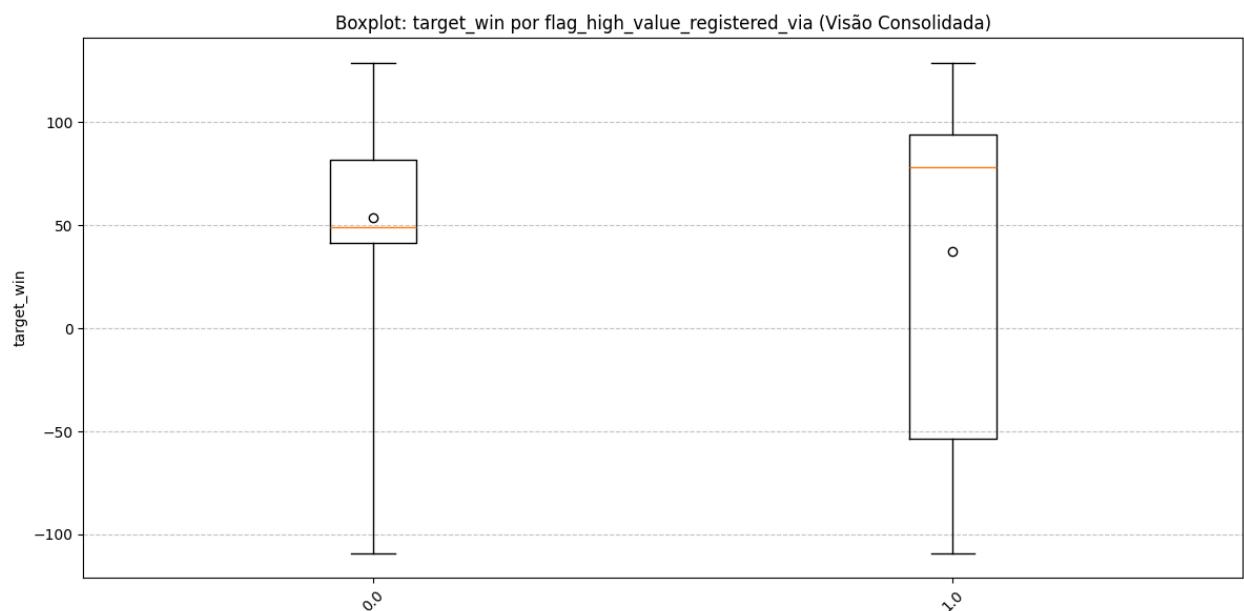
--- Estatísticas: flag_high_value_registered_via (Visão Consolidada) ---

	flag_high_value_registered_via	min	q1	med	mean	q3	max
1	0	-152.5452	41.491442	49.000000	54.223868	81.930892	1736.522159
0	1	-152.5452	-53.618059	78.259896	43.938683	93.967321	1950.000000

=====

```
In [202]: plot_boxplot(df_base_members, ["flag_high_value_registered_via"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_high_value_registered_via...



--- Estatísticas: flag_high_value_registered_via (Visão Consolidada) ---

	flag_high_value_registered_via	min	q1	med	mean	q3	max
1	0	-108.964925	41.491442	49.000000	53.874946	81.930892	128.953521
0	1	-108.964925	-53.618059	78.259896	37.201246	93.967321	128.953521

=====

8.3.5. registration_init_time

As variáveis derivadas da data de registro podem capturar efeitos de coorte e sazonalidade, enquanto o `tenure` representa o estágio de vida do cliente. Essa separação evita vazamento de informação futura e melhora a estabilidade temporal do modelo.

8.3.5.1. Variaveis derivadas da data de registro crua:

```
In [207]: df_base_members = (df_base_members
    .withColumn("mes_registro", F.month("registration_init_time"))
    .withColumn("trimestre_registro", F.quarter("registration_init_time"))
    .withColumn("ano_registro", F.year("registration_init_time")))
```

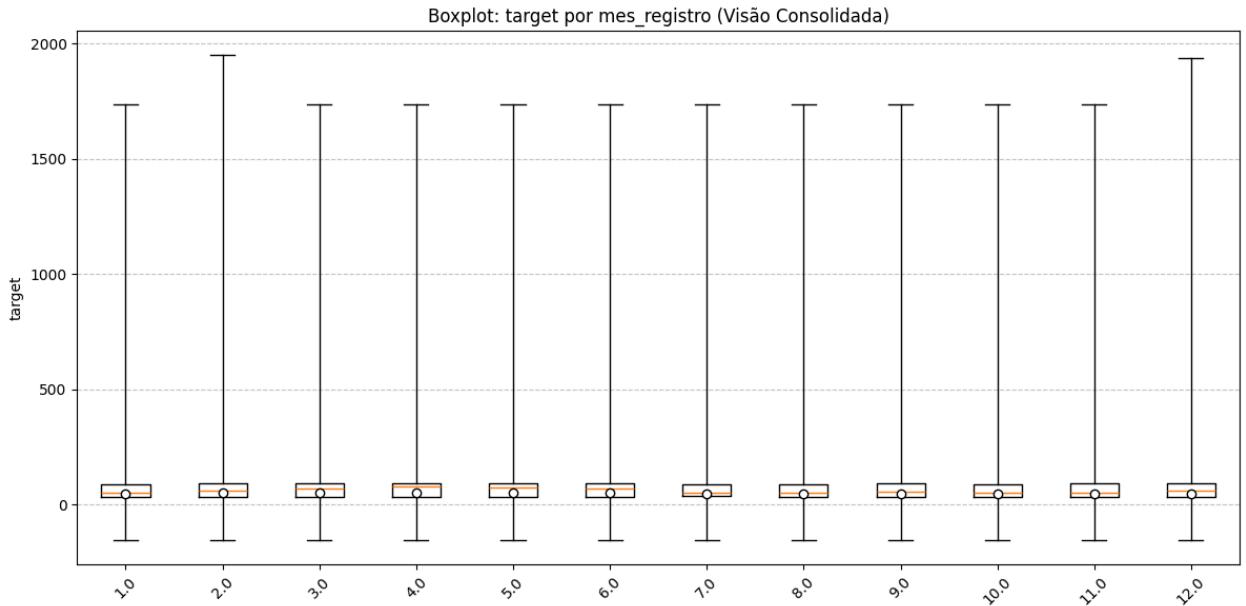
```
In [ ]: calcular_distribuicao(df_base_members, ["mes_registro"], n_show=12)
```

mes_registro	total	pct_total
1	1138465	10.13
10	1044471	9.29
12	1028114	9.14
2	1011064	8.99
11	985695	8.77
8	970168	8.63
7	945952	8.41
3	933838	8.31
9	909882	8.09
6	782525	6.96
5	767196	6.82
4	725495	6.45

```
DataFrame[mes_registro: int, total: bigint, pct_total: double]
```

```
In [208]: plot_boxplot(df_base_members, ["mes_registro"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: mes_registro...



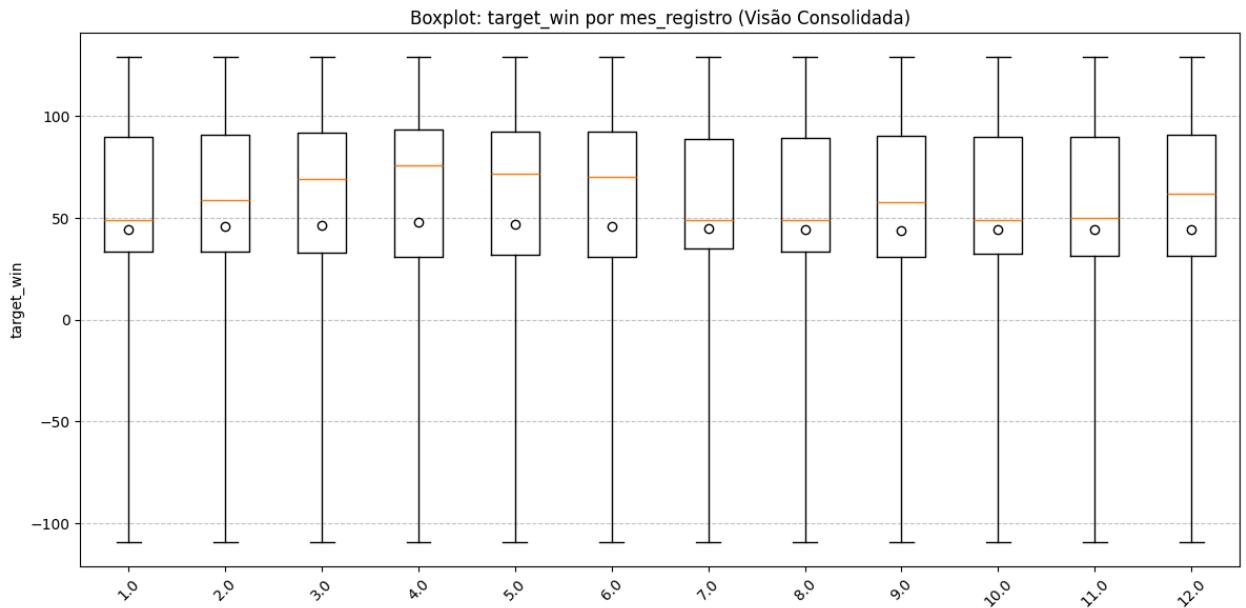
--- Estatísticas: mes_registro (Visão Consolidada) ---

	mes_registro	min	q1	med	mean	q3	max
1	1	-152.5452	33.413119	49.000000	48.179317	89.735415	1736.630919
11	2	-152.5452	33.467306	58.954065	49.423662	90.711468	1950.000000
3	3	-152.5452	32.884107	69.052488	50.326457	92.112409	1737.484560
6	4	-152.5452	30.952132	76.066708	51.784572	93.644680	1737.859653
4	5	-152.5452	32.134202	71.795872	50.708674	92.580299	1736.196782
2	6	-152.5452	30.688468	70.333983	49.988396	92.232552	1736.529083
8	7	-152.5452	35.037880	49.000000	48.151462	89.016834	1737.929378
7	8	-152.5452	33.716622	49.000000	47.850995	89.344076	1737.947047
5	9	-152.5452	30.708981	57.641461	47.628053	90.416083	1737.711061
9	10	-152.5452	32.221684	49.000000	47.659819	89.643036	1737.137480
10	11	-152.5452	31.676893	50.000000	47.744721	89.877219	1736.679651
0	12	-152.5452	31.619568	62.087429	48.455022	90.769493	1936.799455

=====

```
In [209]: plot_boxplot(df_base_members, ["mes_registro"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: mes_registro...



--- Estatísticas: mes_registro (Visão Consolidada) ---

	mes_registro	min	q1	med	mean	q3	max
1	1	-108.964925	33.413119	49.000000	44.534899	89.735415	128.953521
11	2	-108.964925	33.467306	58.954065	45.720796	90.711468	128.953521
3	3	-108.964925	32.884107	69.052488	46.614519	92.112409	128.953521
6	4	-108.964925	30.952132	76.066708	47.952455	93.644680	128.953521
4	5	-108.964925	32.134202	71.795872	47.042047	92.580299	128.953521
2	6	-108.964925	30.688468	70.333983	46.131242	92.232552	128.953521
8	7	-108.964925	35.037880	49.000000	44.773423	89.016834	128.953521
7	8	-108.964925	33.716622	49.000000	44.249673	89.344076	128.953521
5	9	-108.964925	30.708981	57.641461	43.763939	90.416083	128.953521
9	10	-108.964925	32.221684	49.000000	44.086205	89.643036	128.953521
10	11	-108.964925	31.676893	50.000000	44.198344	89.877219	128.953521
0	12	-108.964925	31.619568	62.087429	44.600963	90.769493	128.953521

=====

Não vale a pena agrupar meses (perde sinal e não ganha robustez)

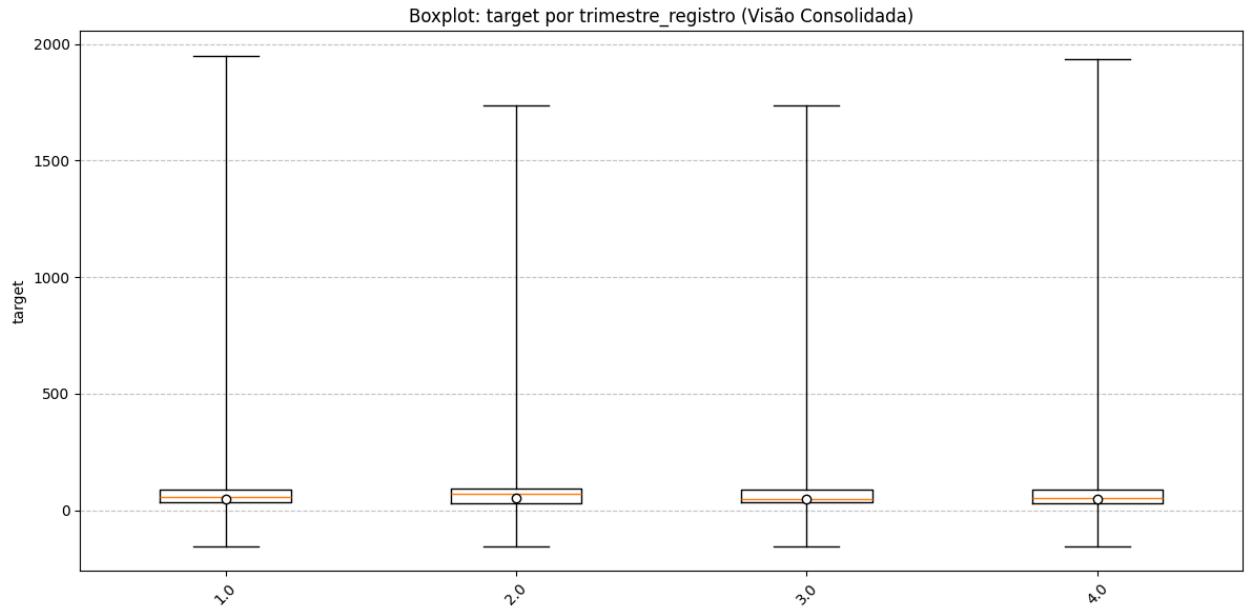
```
In [ ]: calcular_distribuicao(df_base_members, ["trimestre_registro"], n_show=4)
```

trimestre_registro	total	pct_total
1	3083367	27.43
4	3058280	27.2
3	2826002	25.14
2	2275216	20.24

```
DataFrame[trimestre_registro: int, total: bigint, pct_total: double]
```

```
In [210]: plot_boxplot(df_base_members, ["trimestre_registro"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: trimestre_registro...



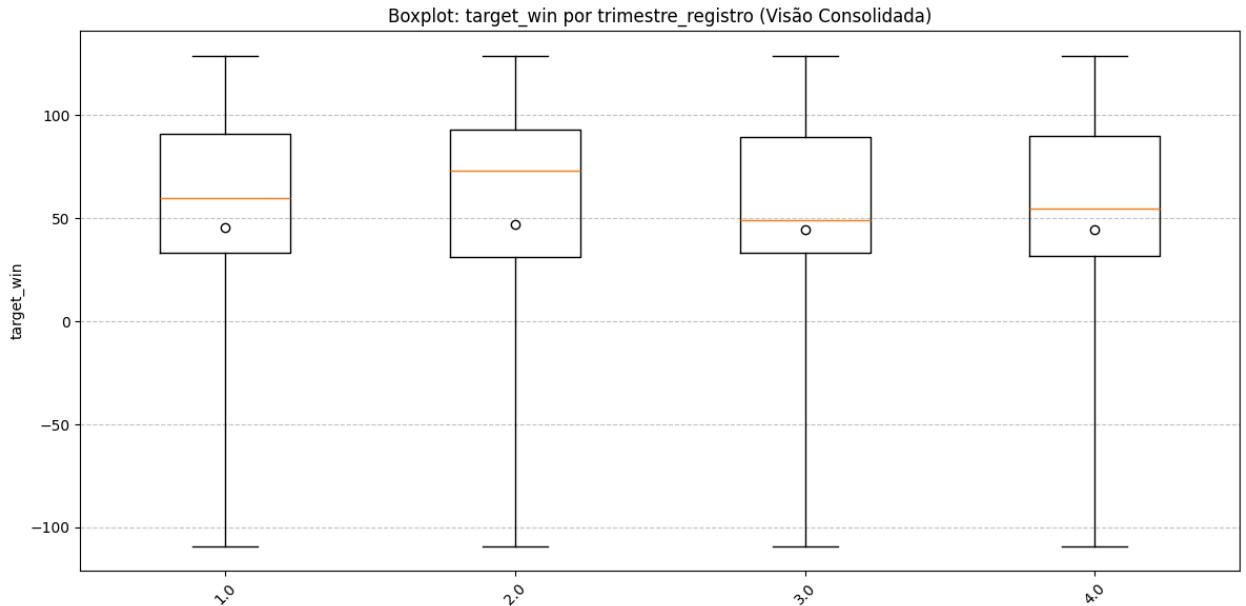
--- Estatísticas: trimestre_registro (Visão Consolidada) ---

	trimestre_registro	min	q1	med	mean	q3	max
0	1	-152.5452	33.292158	59.616697	49.233515	90.856801	1950.000000
3	2	-152.5452	31.267753	72.928165	50.806115	92.840961	1737.859653
1	3	-152.5452	33.433796	49.000000	47.880124	89.618422	1737.947047
2	4	-152.5452	31.869504	54.923554	47.952146	90.113312	1936.799455

=====

```
In [211]: plot_boxplot(df_base_members, ["trimestre_registro"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: trimestre_registro...



--- Estatísticas: trimestre_registro (Visão Consolidada) ---

	trimestre_registro	min	q1	med	mean	q3	max
0	1	-108.964925	33.292158	59.616697	45.549618	90.856801	128.953521
3	2	-108.964925	31.267753	72.928165	47.021298	92.840961	128.953521
1	3	-108.964925	33.433796	49.000000	44.269203	89.618422	128.953521
2	4	-108.964925	31.869504	54.923554	44.293884	90.113312	128.953521

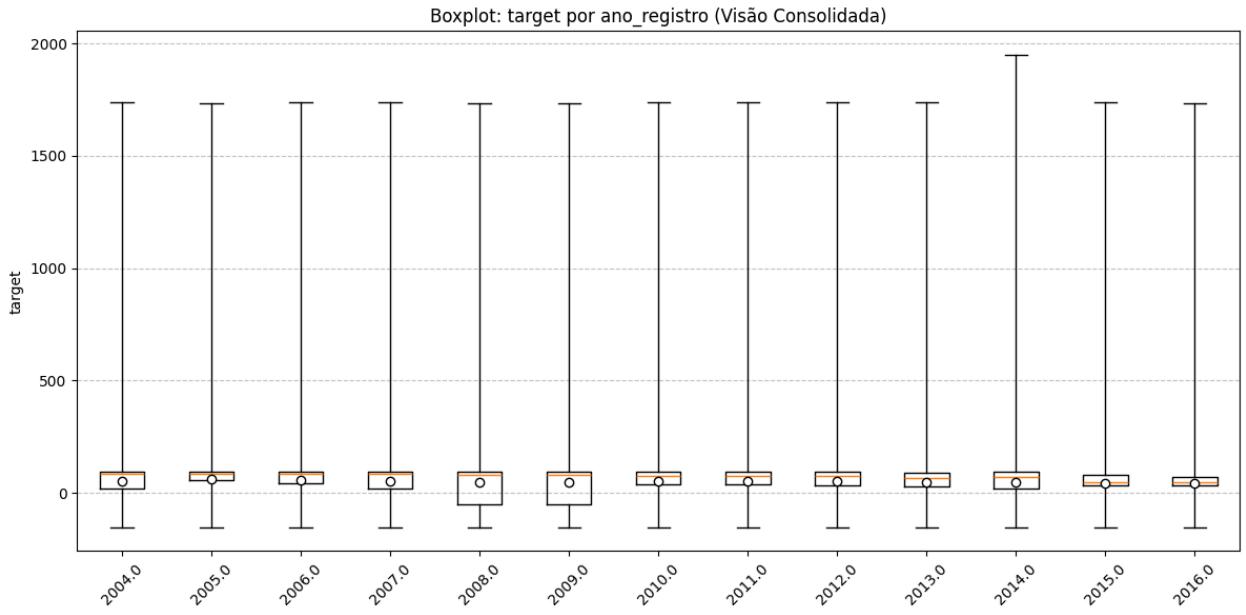
=====

Trimestre é melhor que mês para reduzir dimensionalidade

```
In [ ]: calcular_distribuicao(df_base_members, ["ano_registro"], n_show=12)
```

```
In [212]: plot_boxplot(df_base_members, ["ano_registro"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: ano_registro...



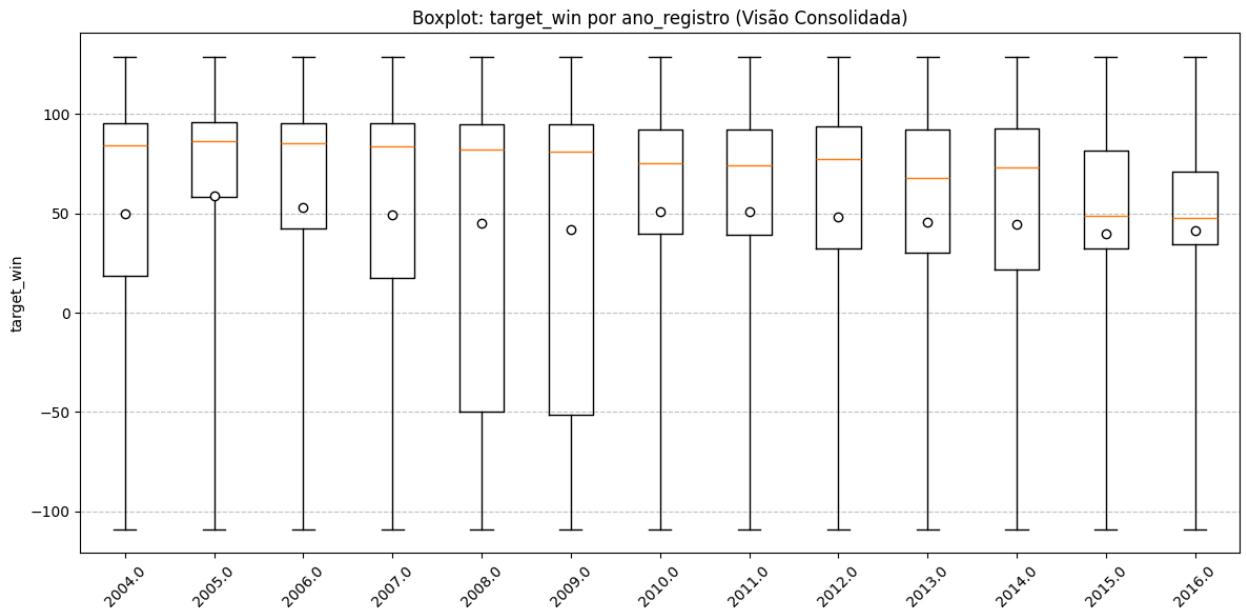
--- Estatísticas: ano_registro (Visão Consolidada) ---

	ano_registro	min	q1	med	mean	q3	max
5	2004	-152.5452	18.457178	84.406246	53.718707	95.425231	1737.137480
9	2005	-152.5452	58.455852	86.562108	61.592437	96.045108	1735.148557
2	2006	-152.5452	42.620012	85.275801	56.844760	95.682332	1737.241320
0	2007	-152.5452	17.382691	83.625064	53.413926	95.431529	1737.484560
12	2008	-152.5452	-50.000000	82.145659	49.530973	94.828378	1735.253213
7	2009	-152.5452	-51.178324	80.941019	46.843007	94.666346	1735.341114
10	2010	-152.5452	39.725118	75.311775	53.683214	92.167655	1737.947047
11	2011	-152.5452	39.096454	74.257053	53.867484	92.365950	1736.048807
6	2012	-152.5452	32.150807	77.213247	52.438135	93.608246	1737.640612
3	2013	-152.5452	30.347267	67.842465	49.359400	92.033185	1737.711061
4	2014	-152.5452	21.638423	73.141249	49.201687	92.768451	1950.000000
1	2015	-152.5452	32.362501	48.870233	44.258791	81.859895	1737.929378
8	2016	-152.5452	34.686824	47.748474	43.585844	71.174377	1735.710665

=====

```
In [213]: plot_boxplot(df_base_members, ["ano_registro"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: ano_registro...



--- Estatísticas: ano_registro (Visão Consolidada) ---

	ano_registro	min	q1	med	mean	q3	max
5	2004	-108.964925	18.457178	84.406246	49.734316	95.425231	128.953521
9	2005	-108.964925	58.455852	86.562108	58.630234	96.045108	128.953521
2	2006	-108.964925	42.620012	85.275801	53.210222	95.682332	128.953521
0	2007	-108.964925	17.382691	83.625064	49.425787	95.431529	128.953521
12	2008	-108.964925	-50.000000	82.145659	44.937708	94.828378	128.953521
7	2009	-108.964925	-51.178324	80.941019	42.121384	94.666346	128.953521
10	2010	-108.964925	39.725118	75.311775	50.814920	92.167655	128.953521
11	2011	-108.964925	39.096454	74.257053	51.075658	92.365950	128.953521
6	2012	-108.964925	32.150807	77.213247	48.478680	93.608246	128.953521
3	2013	-108.964925	30.347267	67.842465	45.654735	92.033185	128.953521
4	2014	-108.964925	21.638423	73.141249	44.438089	92.768451	128.953521
1	2015	-108.964925	32.362501	48.870233	40.039829	81.859895	128.953521
8	2016	-108.964925	34.686824	47.748474	41.332752	71.174377	128.953521

=====

Agrupar por ano parece bem melhor.

O que aparece claramente:

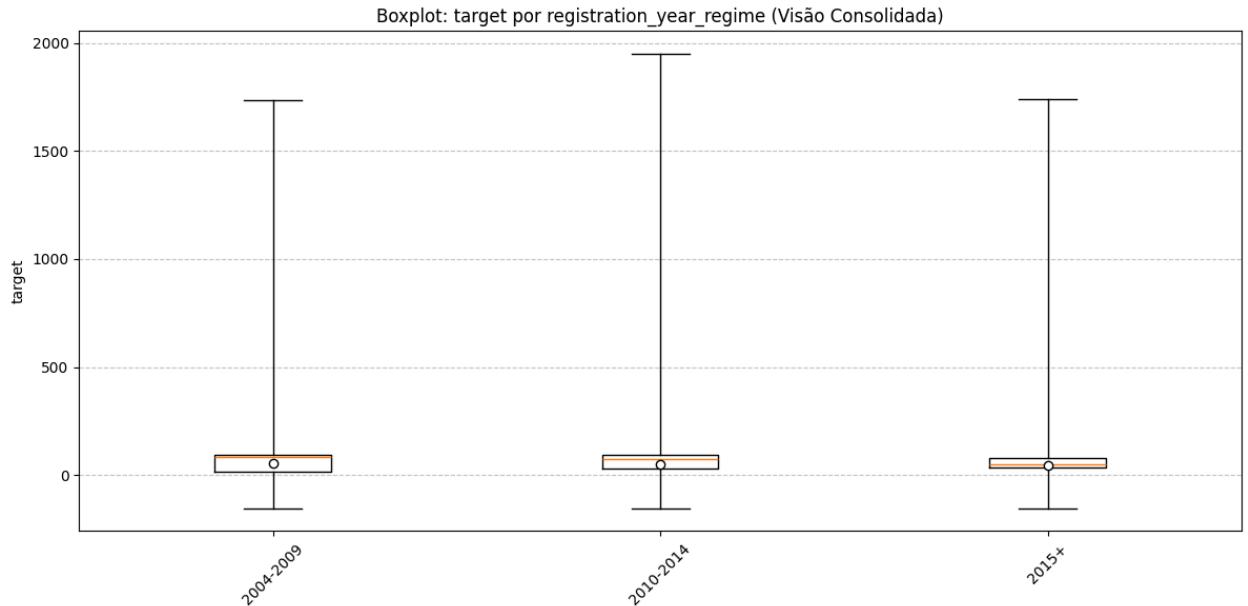
- * 2004–2009: Mediana alta (~80+), Perfil mais volátil → base antiga, contratos distintos
- * 2010–2014: Mediana ~68–77, Base mais “normalizada”
- * 2015–2016: Mediana cai para ~48, Q3 cai forte (~ 74–82), aparenta ser outro regime de negócio

Isso não é ruído estatístico, mas pode significar mudança de produto, mudança de pricing e/ou mudança de estratégia comercial

```
In [214]: df_base_members = df_base_members.withColumn("registration_year_regime",
    F.when(F.col("ano_registro").isin([2004, 2005, 2006, 2007, 2008, 2009]), "2004-2009") \
    .when(F.col("ano_registro").isin([2010, 2011, 2012, 2013, 2014]), "2010-2014") \
    .otherwise("2015+"))
```

```
In [215]: plot_boxplot(df_base_members, ["registration_year_regime"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: registration_year_regime...



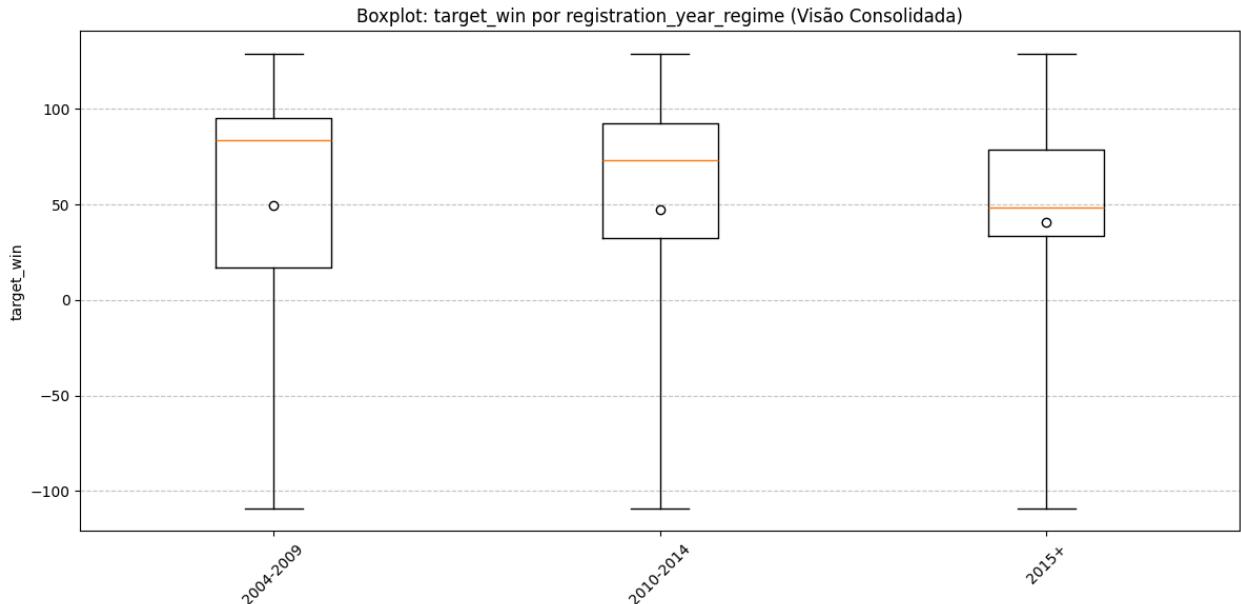
--- Estatísticas: registration_year_regime (Visão Consolidada) ---

	registration_year_regime	min	q1	med	mean	q3	max
1	2004-2009	-152.5452	17.305714	83.909292	53.531776	95.381007	1737.484560
0	2010-2014	-152.5452	32.279152	73.448129	51.115988	92.638294	1950.000000
2	2015+	-152.5452	33.520024	48.451471	43.989022	78.717034	1737.929378

=====

```
In [216]: plot_boxplot(df_base_members, ["registration_year_regime"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: registration_year_regime...



--- Estatísticas: registration_year_regime (Visão Consolidada) ---

	registration_year_regime	min	q1	med	mean	q3	max
1	2004-2009	-108.964925	17.305714	83.909292	49.535985	95.381007	128.953521
0	2010-2014	-108.964925	32.279152	73.448129	47.343875	92.638294	128.953521
2	2015+	-108.964925	33.520024	48.451471	40.558132	78.717034	128.953521

=====

8.3.5.2. Tenure e faixa tenure

```
In [217]: df_base_members = df_base_members.withColumn("safra_date", F.to_date(F.concat(F.col("safra").cast("string"), F.lit("01"))), "yyyyMMdd"))
```

```
In [218]: # tenure --> depois, faixa de tenure
df_base_members = (df_base_members
    .withColumn("tenure_meses", F.floor(F.months_between("safra_date", "registration_init_time")))
    .withColumn("tenure_meses", F.when(F.col("tenure_meses") < 0, 0).otherwise(F.col("tenure_meses"))))
```

```
In [ ]: df_base_members.select("tenure_meses").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+-----+
|summary|      tenure_meses|
+-----+-----+
| count| 11242865|
| mean| 37.65380247828289|
| stddev|35.163475062462894|
| min| 0|
| 1%| 0|
| 5%| 1|
| 25%| 9|
| 50%| 29|
| 75%| 55|
| 95%| 115|
| 99.5%| 142|
| max| 152|
+-----+-----+
```

```
In [219]: # grau de correlação de tenure com target
df_base_members.select("tenure_meses", "target").corr("tenure_meses", "target")
```

Out[219]: 0.050579142535209023

```
In # grau de correlacao de tenure com target  
[220]: df_base_members.select("tenure_meses", "target_win").corr("tenure_meses", "target_win")
```

```
Out[220]: 0.06988654552048591
```

Tenure (continuo) tem efeito nao tão linear na target (aprox 7%).

```
In df_base_members = df_base_members.withColumn("tenure_faixa",  
[221]: F.when(F.col("tenure_meses").isNull(), "unknown")  
.when((F.col("tenure_meses") >= 0) & (F.col("tenure_meses") <= 1), "0-1")  
.when((F.col("tenure_meses") >= 2) & (F.col("tenure_meses") <= 3), "2-3")  
.when((F.col("tenure_meses") >= 4) & (F.col("tenure_meses") <= 6), "4-6")  
.when((F.col("tenure_meses") >= 7) & (F.col("tenure_meses") <= 11), "7-11")  
.otherwise("12+"))
```

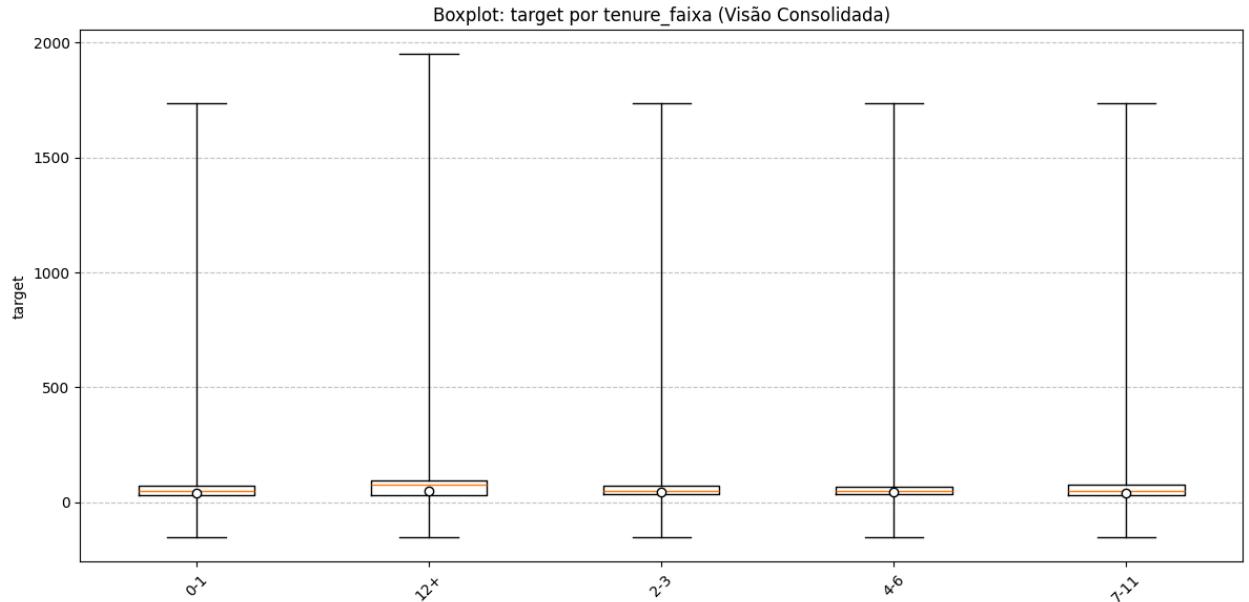
```
In [ ]: calcular_distribuicao(df_base_members, ["tenure_faixa"])
```

```
+-----+-----+-----+  
|tenure_faixa|total |pct_total|  
+-----+-----+-----+  
|12+        |7794615|69.33   |  
|7-11       |1214335|10.8    |  
|4-6         |863607 |7.68    |  
|0-1         |802000 |7.13    |  
|2-3         |568308 |5.05    |  
+-----+-----+-----+
```

```
DataFrame[tenure_faixa: string, total: bigint, pct_total: double]
```

```
In plot_boxplot(df_base_members, ["tenure_faixa"], "target", agrupar_por_safra=False, table=True)  
[222]:
```

Processando estatísticas para: tenure_faixa...



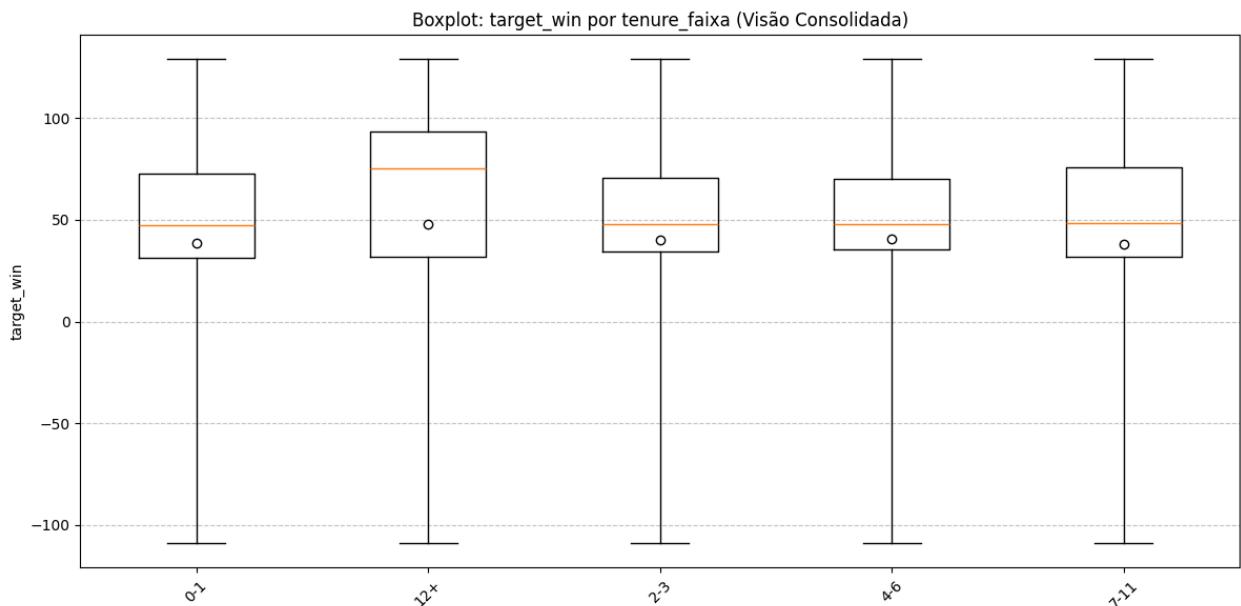
--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
4	0-1	-152.5452	31.538681	47.465809	41.129891	72.814103	1736.987900
3	12+	-152.5452	31.925361	75.409595	51.604072	93.259867	1950.000000
0	2-3	-152.5452	34.456012	47.828648	42.698356	70.870518	1734.872871
2	4-6	-152.5452	35.243553	48.090873	45.575029	70.098882	1737.929378
1	7-11	-152.5452	31.961867	48.353974	41.364531	75.637094	1736.589444

=====

```
In [223]: plot_boxplot(df_base_members, ["tenure_faixa"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: tenure_faixa...



--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
4	0-1	-108.964925	31.538681	47.465809	38.347386	72.814103	128.953521
3	12+	-108.964925	31.925361	75.409595	47.778133	93.259867	128.953521
0	2-3	-108.964925	34.456012	47.828648	40.373396	70.870518	128.953521
2	4-6	-108.964925	35.243553	48.090873	40.787534	70.098882	128.953521
1	7-11	-108.964925	31.961867	48.353974	38.208869	75.637094	128.953521

=====

Percebe-se que tenure_faixa em 12+ equivale a quase 70% da base, e as estatísticas das categorias distintas desta são muito similares.

Talvez considerar mais agrupamentos pra faixa_tenure ?

```
In [224]: df_base_members = df_base_members.withColumn("tenure_faixa", F.when(F.col("tenure_meses").isNull(), "unknown") .when((F.col("tenure_meses") >= 0) & (F.col("tenure_meses") <= 11), "0-11") .when((F.col("tenure_meses") >= 12) & (F.col("tenure_meses") <= 23), "12-23") .otherwise("24+"))
```

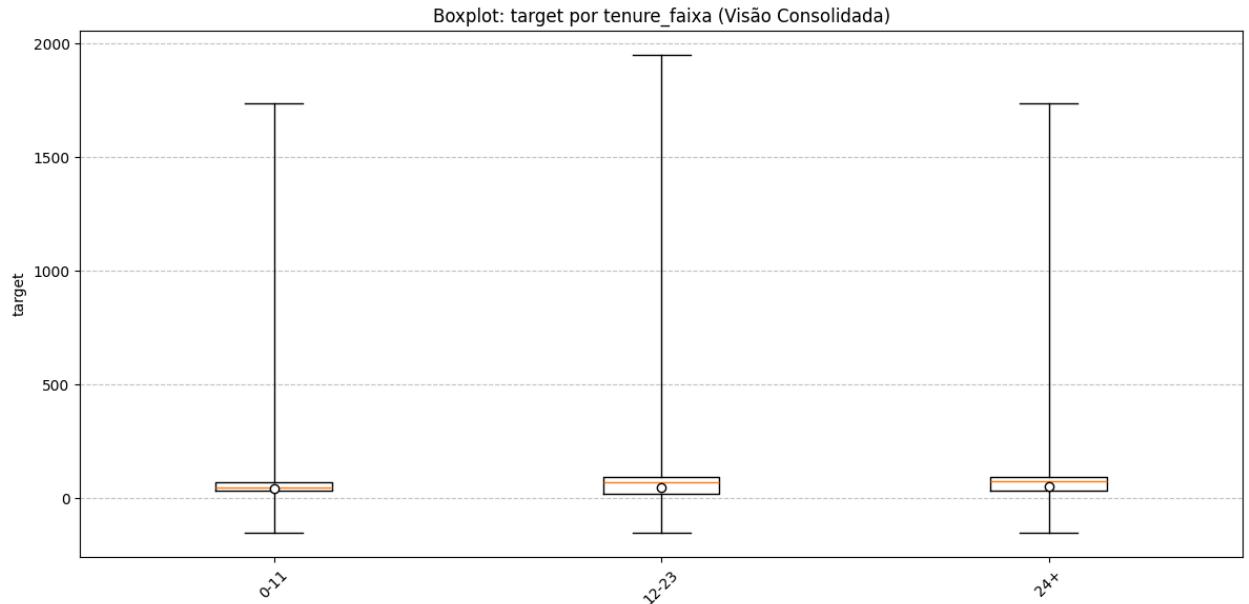
```
In [ ]: calcular_distribuicao(df_base_members, ["tenure_faixa"])
```

```
+-----+-----+-----+
|tenure_faixa|total |pct_total|
+-----+-----+-----+
|24+        |6318740|56.2   |
|0-11       |3448250|30.67  |
|12-23      |1475875|13.13  |
+-----+-----+-----+
```

DataFrame[tenure_faixa: string, total: bigint, pct_total: double]

```
In [225]: plot_boxplot(df_base_members, ["tenure_faixa"], "target", agrupar_por_safr=False, table=True)
```

Processando estatísticas para: tenure_faixa...



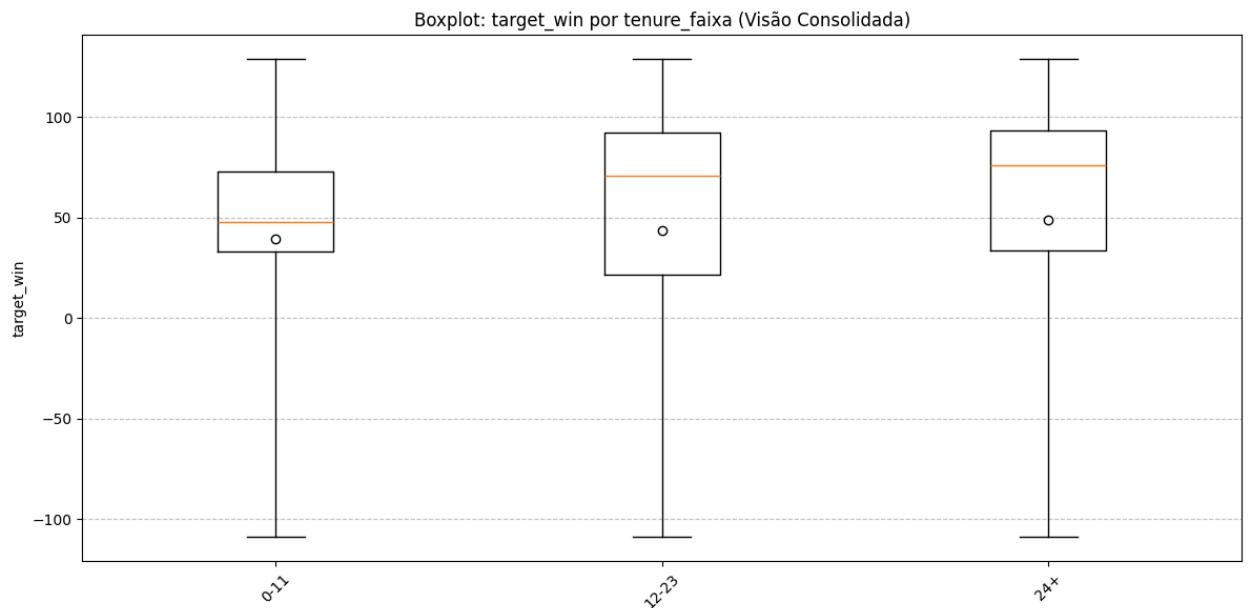
--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
2	0-11	-152.5452	33.258711	47.998934	42.593287	73.089370	1737.929378
1	12-23	-152.5452	21.884712	70.753704	48.995900	92.209199	1950.000000
0	24+	-152.5452	33.449292	76.199077	52.178311	93.457863	1737.947047

=====

```
In [226]: plot_boxplot(df_base_members, ["tenure_faixa"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: tenure_faixa...



--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
2	0-11	-108.964925	33.258711	47.998934	39.250497	73.089370	128.953521
1	12-23	-108.964925	21.884712	70.753704	43.855336	92.209199	128.953521
0	24+	-108.964925	33.449292	76.199077	48.641813	93.457863	128.953521

=====

```
In [227]: df_base_members = df_base_members.withColumn("tenure_faixa",
    F.when(F.col("tenure_meses").isNull(), "unknown")
    .when((F.col("tenure_meses") >= 0) & (F.col("tenure_meses") <= 11), "0-11")
    .when((F.col("tenure_meses") >= 12) & (F.col("tenure_meses") <= 23), "12-23")
    .when((F.col("tenure_meses") >= 24) & (F.col("tenure_meses") <= 35), "24-35")
    .otherwise("36+"))
```

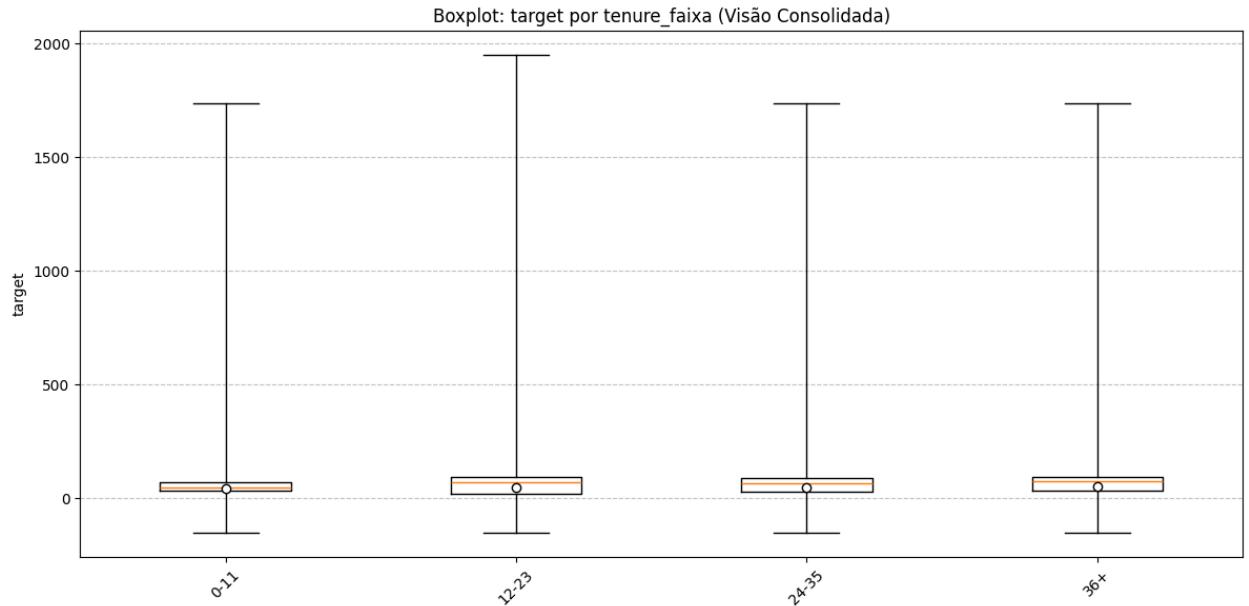
```
In [ ]: calcular_distribuicao(df_base_members, ["tenure_faixa"])
```

```
+-----+-----+-----+
|tenure_faixa|total |pct_total|
+-----+-----+-----+
|36+       |4758009|42.32   |
|0-11      |3448250|30.67   |
|24-35     |1560731|13.88   |
|12-23     |1475875|13.13   |
+-----+-----+-----+
```

```
DataFrame[tenure_faixa: string, total: bigint, pct_total: double]
```

```
In [228]: plot_boxplot(df_base_members, ["tenure_faixa"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: tenure_faixa...



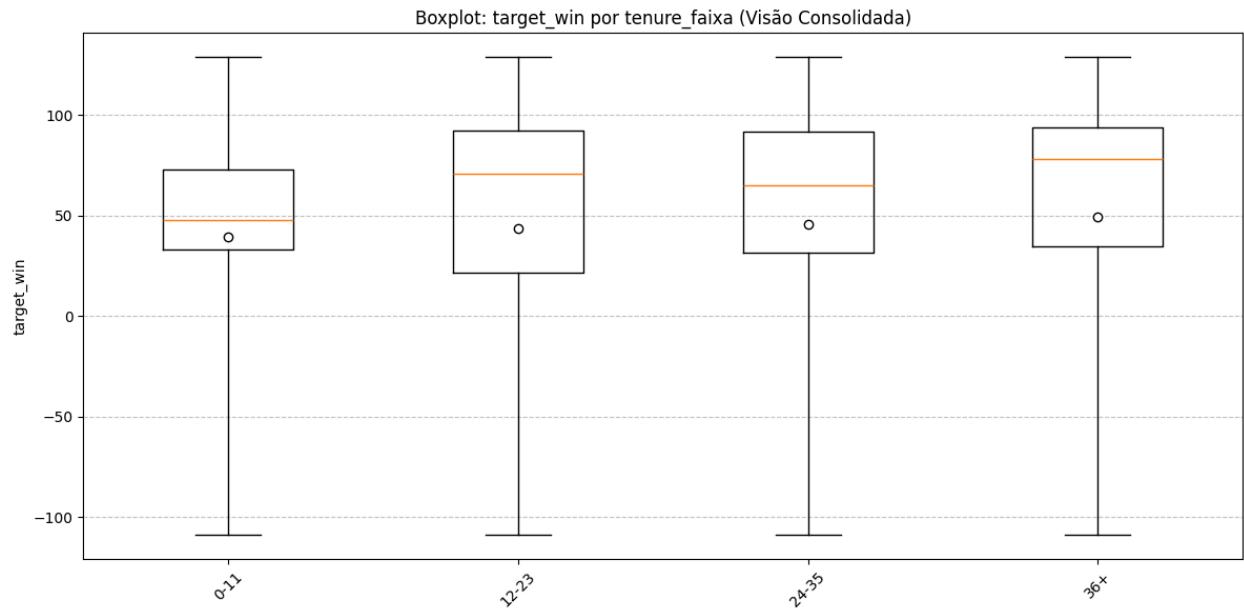
--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
3	0-11	-152.5452	33.258711	47.998934	42.593287	73.089370	1737.929378
1	12-23	-152.5452	21.884712	70.753704	48.995900	92.209199	1950.000000
0	24-35	-152.5452	31.513138	65.153832	49.456383	91.685049	1737.711061
2	36+	-152.5452	34.457785	78.161693	53.087818	93.915124	1737.947047

=====

```
In [229]: plot_boxplot(df_base_members, ["tenure_faixa"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: tenure_faixa...



--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
3	0-11	-108.964925	33.258711	47.998934	39.250497	73.089370	128.953521
1	12-23	-108.964925	21.884712	70.753704	43.855336	92.209199	128.953521
0	24-35	-108.964925	31.513138	65.153832	45.681294	91.685049	128.953521
2	36+	-108.964925	34.457785	78.161693	49.631042	93.915124	128.953521

=====

Valores estão começando a se aproximar mais. Talvez seja melhor considerar a de menor número de categorias. Vou agrupar as duas intermediárias e subir mais.

```
In [230]: df_base_members = df_base_members.withColumn("tenure_faixa",
    F.when(F.col("tenure_meses").isNull(), "unknown")
    .when((F.col("tenure_meses") >= 0) & (F.col("tenure_meses") <= 11), "0-11")
    .when((F.col("tenure_meses") >= 12) & (F.col("tenure_meses") <= 35), "12-35")
    .when((F.col("tenure_meses") >= 36) & (F.col("tenure_meses") <= 47), "36-47")
    .otherwise("48+"))
```

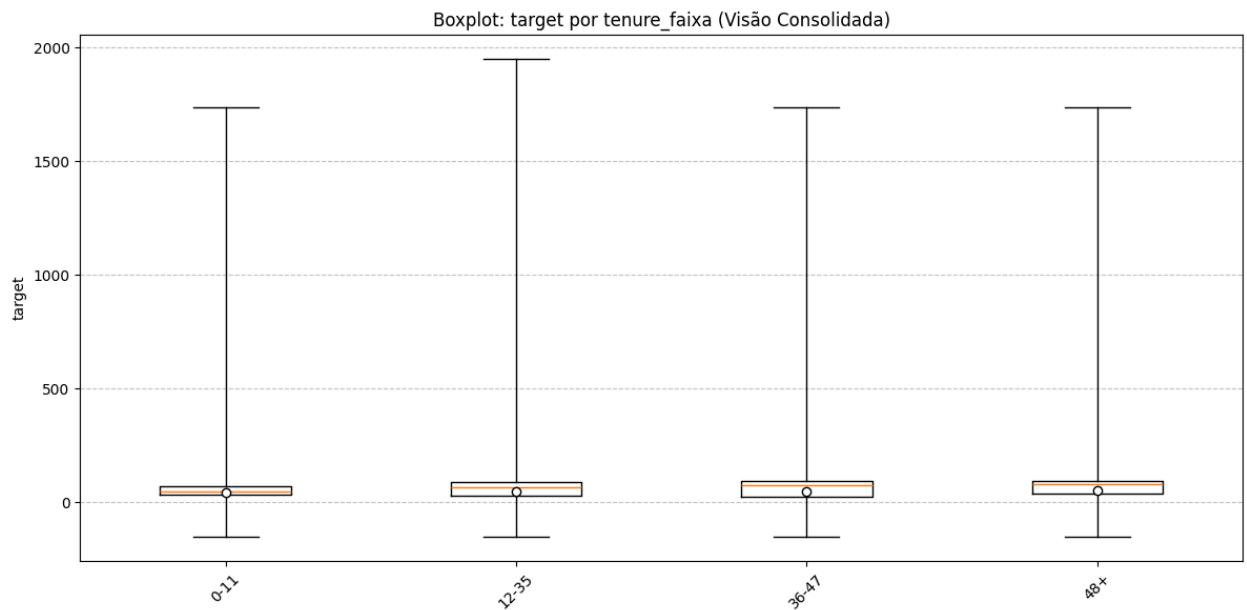
```
In [ ]: calcular_distribuicao(df_base_members, ["tenure_faixa"])
```

```
+-----+-----+-----+
|tenure_faixa|total |pct_total|
+-----+-----+-----+
|48+       |3484122|30.99   |
|0-11      |3448250|30.67   |
|12-35     |3036606|27.01   |
|36-47     |1273887|11.33   |
+-----+-----+-----+
```

```
DataFrame[tenure_faixa: string, total: bigint, pct_total: double]
```

```
In [231]: plot_boxplot(df_base_members, ["tenure_faixa"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: tenure_faixa...



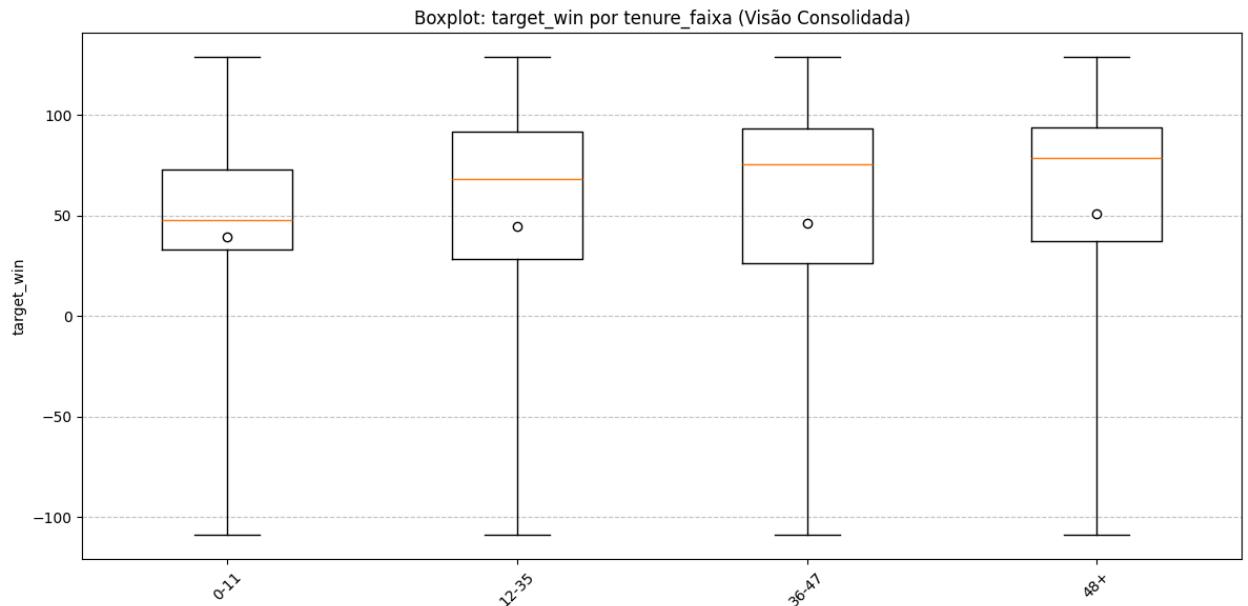
--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
3	0-11	-152.5452	33.258711	47.998934	42.593287	73.089370	1737.929378
2	12-35	-152.5452	28.534129	68.220781	49.240957	91.940411	1950.000000
1	36-47	-152.5452	26.563645	75.510011	50.486417	93.348304	1737.640612
0	48+	-152.5452	37.091280	78.810260	54.018694	94.098520	1737.947047

=====

```
In [232]: plot_boxplot(df_base_members, ["tenure_faixa"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: tenure_faixa...



--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
3	0-11	-108.964925	33.258711	47.998934	39.250497	73.089370	128.953521
2	12-35	-108.964925	28.534129	68.220781	44.827065	91.940411	128.953521
1	36-47	-108.964925	26.563645	75.510011	46.371666	93.348304	128.953521
0	48+	-108.964925	37.091280	78.810260	50.797366	94.098520	128.953521

=====

```
In [233]: df_base_members = df_base_members.withColumn("tenure_faixa",
    F.when(F.col("tenure_meses").isNull(), "unknown")
    .when((F.col("tenure_meses") >= 0) & (F.col("tenure_meses") <= 11), "0-11")
    .when((F.col("tenure_meses") >= 12) & (F.col("tenure_meses") <= 35), "12-35")
    .otherwise("36+"))
```

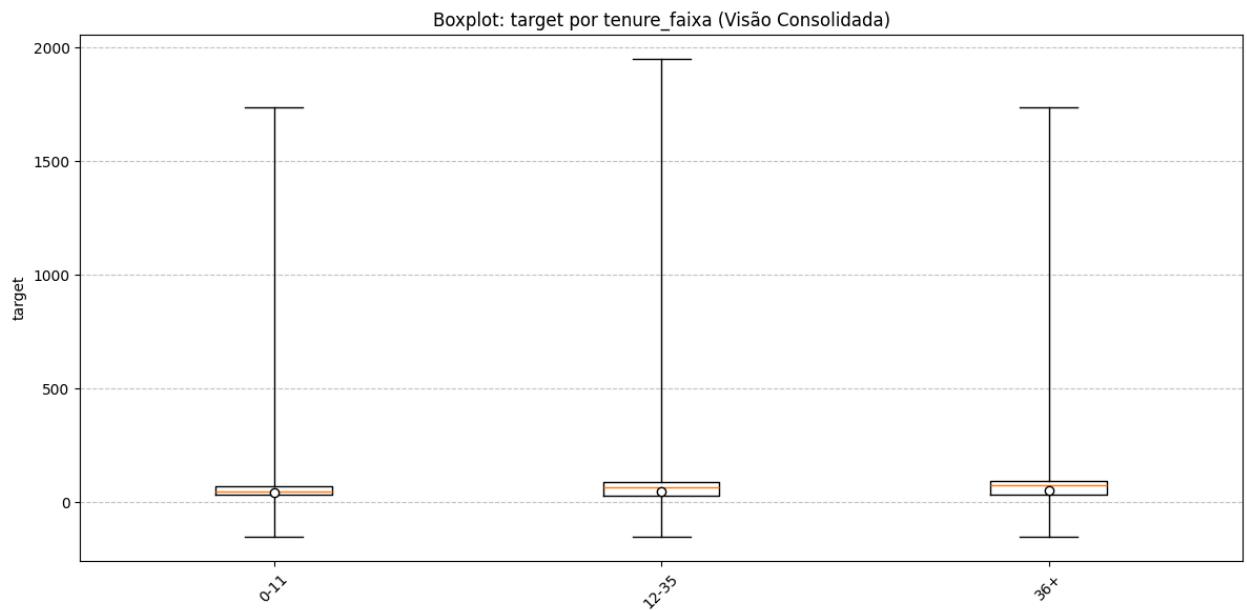
```
In [ ]: calcular_distribuicao(df_base_members, ["tenure_faixa"])
```

```
+-----+-----+-----+
|tenure_faixa|total |pct_total|
+-----+-----+-----+
|36+      |4758009|42.32   |
|0-11     |3448250|30.67   |
|12-35    |3036606|27.01   |
+-----+-----+-----+
```

```
DataFrame[tenure_faixa: string, total: bigint, pct_total: double]
```

```
In [234]: plot_boxplot(df_base_members, ["tenure_faixa"], "target", agrupar_por_safr=False, table=True)
```

Processando estatísticas para: tenure_faixa...



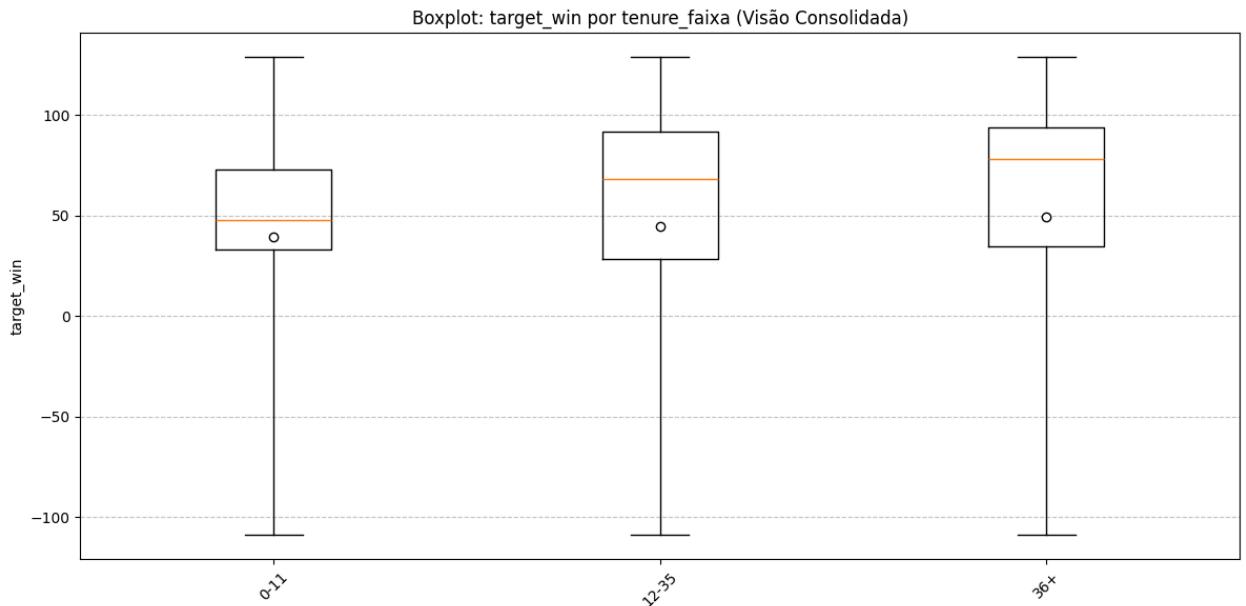
--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
2	0-11	-152.5452	33.258711	47.998934	42.593287	73.089370	1737.929378
0	12-35	-152.5452	28.534129	68.220781	49.240957	91.940411	1950.000000
1	36+	-152.5452	34.457785	78.161693	53.087818	93.915124	1737.947047

=====

```
In [235]: plot_boxplot(df_base_members, ["tenure_faixa"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: tenure_faixa...



--- Estatísticas: tenure_faixa (Visão Consolidada) ---

	tenure_faixa	min	q1	med	mean	q3	max
2	0-11	-108.964925	33.258711	47.998934	39.250497	73.089370	128.953521
0	12-35	-108.964925	28.534129	68.220781	44.827065	91.940411	128.953521
1	36+	-108.964925	34.457785	78.161693	49.631042	93.915124	128.953521

=====

Esta aparenta ser a melhor versão, a considerar a análise das medianas (como a distribuição não é simétrica, melhor concluir pela mediana, como estamos fazendo ao longo do projeto). A primeira divisão feita (0-11 / 12-23 / 24+) também era boa, porém, mistura usuários muito maduros (36+, 48+) com "intermediários".

Criação de uma flag para simbolizar long tenure

```
In [236]: df_base_members = df_base_members.withColumn("flag_long_tenure", F.when(F.col("tenure_meses") >= 36, 1).otherwise(0))
```

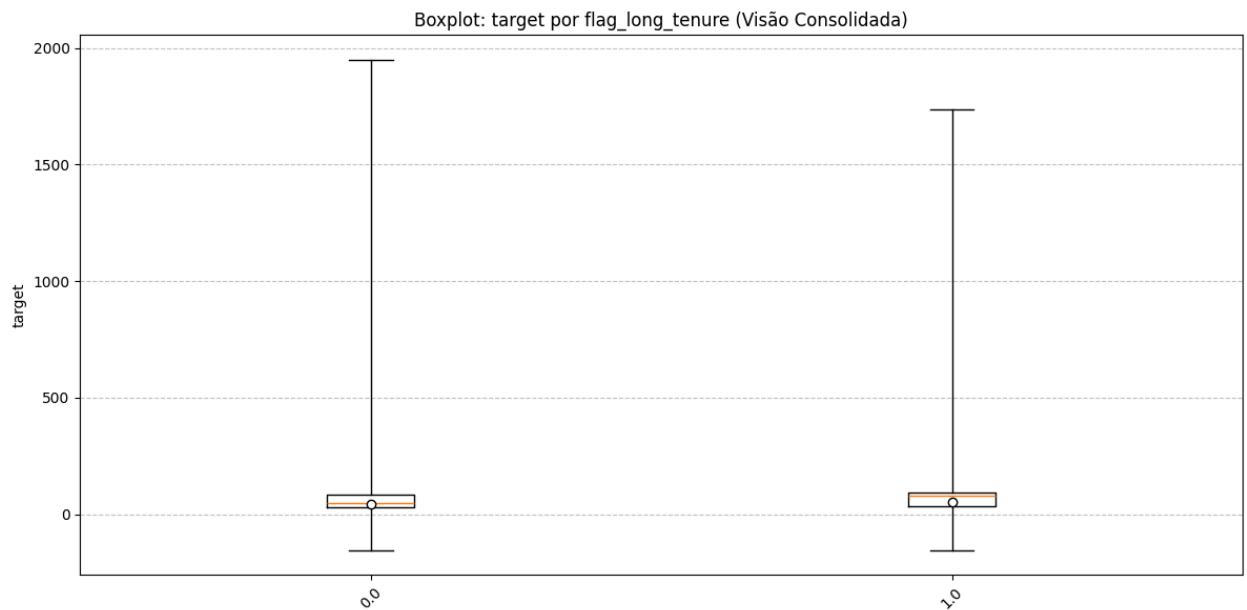
```
In [ ]: calcular_distribuicao(df_base_members, ["flag_long_tenure"], n_show=2)
```

flag_long_tenure	total	pct_total
0	6484856	57.68
1	4758009	42.32

```
DataFrame[flag_long_tenure: int, total: bigint, pct_total: double]
```

```
In [237]: plot_boxplot(df_base_members, ["flag_long_tenure"], "target", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: flag_long_tenure...



--- Estatísticas: flag_long_tenure (Visão Consolidada) ---

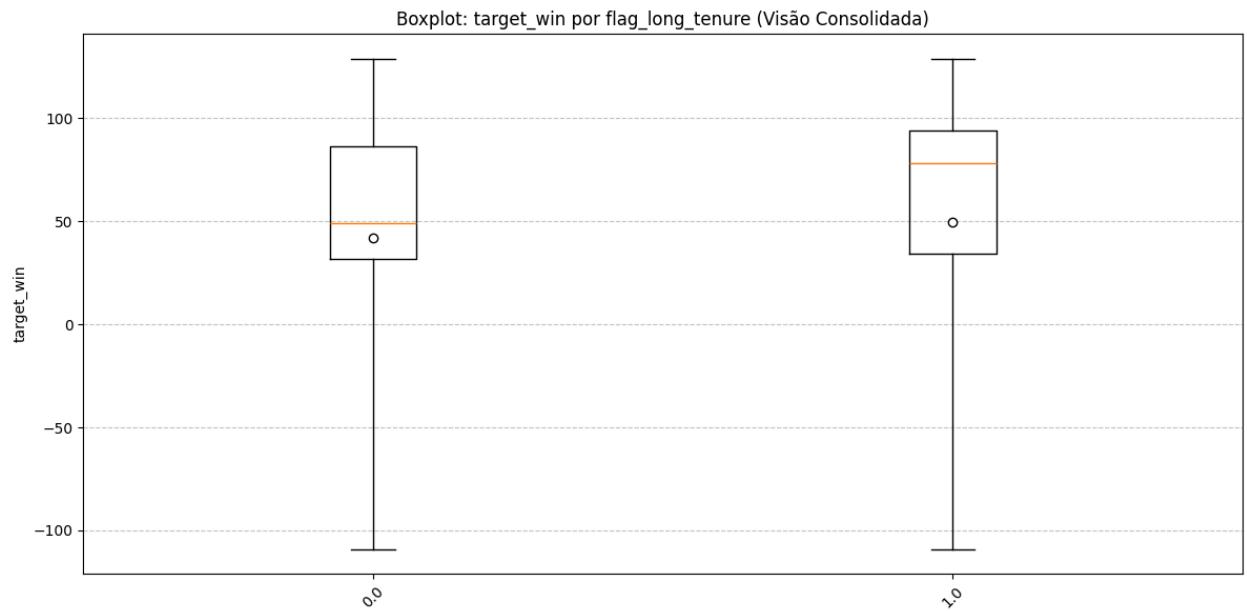
	flag_long_tenure	min	q1	med	mean	q3	max
1	0	-152.5452	31.933910	49.000000	45.708881	86.440688	1950.000000
0	1	-152.5452	34.457785	78.161693	53.087818	93.915124	1737.947047

=====

```
In [238]: plot_boxplot(df_base_members, ["flag_long_tenure"], "target_win", agrupar_por_safra=False, table=True)
```

[238]:

Processando estatísticas para: flag_long_tenure...



--- Estatísticas: flag_long_tenure (Visão Consolidada) ---

	flag_long_tenure	min	q1	med	mean	q3	max
1	0	-108.964925	31.933910	49.000000	41.864093	86.440688	128.953521
0	1	-108.964925	34.457785	78.161693	49.631042	93.915124	128.953521

=====

Conclusao

Levar tanto a flag quanto a variavel agrupada para feature engineering. A flag se mostra muito util para Elastic Net, enquanto a variavel agrupada para os modelos baseados em arvores.

9. _Feature Engineering_ - Etapa 1: _Books_ de variáveis

9.1. Book Logs

9.1.1. Carregando bases

```
In [3]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver"
df_logs_fix = spark.read.parquet(f"{silver_path}/df_logs_fix.parquet")
df_spine = spark.read.parquet(f"{silver_path}/df_spine")
```

```
In [6]: df_logs_fix.count()
```

```
Out[6]: 26758971
```

```
In [7]: df_spine.count()
```

```
Out[7]: 11242865
```

```
In [9]: df_logs_fix_spine = df_spine.join(df_logs_fix, ["msno", "safras"], "left")
```

```
In [10]: df_logs_fix_spine.count()
```

```
Out[10]: 11242865
```

```
In [5]: df_logs_fix_spine.printSchema()
```

```
root
|-- msno: string (nullable = true)
|-- safras: integer (nullable = true)
|-- safras: integer (nullable = true)
|-- num_25: double (nullable = true)
|-- num_50: double (nullable = true)
|-- num_75: double (nullable = true)
|-- num_985: double (nullable = true)
|-- num_100: double (nullable = true)
|-- num_unq: double (nullable = true)
|-- total_secs: double (nullable = true)
```

9.1.2. flag_has_logs

```
In [14]: df_logs_fix_spine = df_logs_fix_spine.withColumn("flag_has_logs", F.when(F.col("total_secs").isNull(), 0).otherwise(1))
```

9.1.3. total_plays

```
In [15]: df_logs_fix_spine = df_logs_fix_spine.withColumn("total_plays", F.col("num_25") + F.col("num_50") + F.col("num_75") + F.col("num_985") + F.col("num_100"))

# Agrupamento por quartis - metodo arriscado: usar os valores brutos do EDA para definir os cortes traz risco para a performance futura,
# dado que a distribuicao dos valores da var pode alterar

# df_logs_fix_spine = df_logs_fix_spine.withColumn("total_plays_group",
#     F.when(F.col("total_plays").isin(0) | F.col("total_plays").isNull(), "00_unknown")\
#     .when(F.col("total_plays").between(1, 164), "01_casual_listener")
#     .when(F.col("total_plays").between(165, 443), "02_regular_listener")
#     .when(F.col("total_plays").between(444, 932), "03_frequent_listener")
#     .otherwise("04_power_user"))

# Metodo adequado
df_logs_fix_spine = segment_by_percentile(df_logs_fix_spine, "total_plays", "logs")

map_total_plays = {
    "unknown": "00_unknown",
    "tier_1": "01_casual_listener", # Q1 - Usuário que consome pouco, possivelmente esporádico.
    "tier_2": "02_regular_listener", # Q2 - Usuário recorrente, mas com volume moderado.
    "tier_3": "03_frequent_listener", # Q3 - Usuário engajado, consome acima da média.
    "tier_4": "04_power_user" # Q4 - Usuário altamente engajado, possivelmente fã da plataforma.
}

df_logs_fix_spine = df_logs_fix_spine.replace(map_total_plays, subset=["total_plays_group"])
```

```
In [16]: # Aplicar transformação logarítmica - considerar levar esta como o valor continuo no modelo linear
df_logs_fix_spine = df_logs_fix_spine.withColumn("log_total_plays", F.log1p(F.col("total_plays")))
```

9.1.4. completed_songs_rate

```
In [17]: # Cálculo da taxa de completude com tratamento de divisão por zero
df_logs_fix_spine = df_logs_fix_spine.withColumn("completed_songs_rate",
    F.when(F.col("total_plays") > 0, F.col("num_100") / F.col("total_plays")).otherwise(0.0))

# Agrupamento por quartis - inadequado

# df_logs_fix_spine = df_logs_fix_spine.withColumn("completed_songs_rate_group",
#     F.when(F.col("completed_songs_rate").isin(0) | F.col("completed_songs_rate").isNull(), "00_unknown")
#     .when((F.col("completed_songs_rate") > 0.00) & (F.col("completed_songs_rate") <= 0.477) , "01_bouncer")
#     .when((F.col("completed_songs_rate") > 0.477) & (F.col("completed_songs_rate") <= 0.694), "02_skipping_listener")
#     .when((F.col("completed_songs_rate") > 0.694) & (F.col("completed_songs_rate") <= 0.835), "03_engaged_listener")
#     .otherwise("04_completionist"))

# Metodo adequado
df_logs_fix_spine = segment_by_percentile(df_logs_fix_spine, "completed_songs_rate", "logs")

map_completion = {
    "unknown": "00_unknown",
    "tier_1": "01_bouncer", # Q1 - Usuário que "pula" músicas rápido (skip rate alto).
    "tier_2": "02_skipping_listener", # Q2 - Ouve partes da música, mas raramente termina.
    "tier_3": "03_engaged_listener", # Q3 - Ouve a maior parte do conteúdo; bom sinal de interesse.
    "tier_4": "04_completionist" # Q4 - Usuário que ouve tudo até o fim.
}

df_logs_fix_spine = df_logs_fix_spine.replace(map_completion, subset=["completed_songs_rate_group"])
```

9.1.5. avg_secs_per_unq

```
In [18]: df_logs_fix_spine = df_logs_fix_spine.withColumn("avg_secs_per_unq", F.when(F.col("num_unq") > 0, F.col("total_secs") / F.col("num_unq")).otherwise(0.0))

# Winsorizar pra cima, no p995 visto no eda - pode trazer problemas de performance. O correto seria colocar o proprio p995

# df_logs_fix_spine = df_logs_fix_spine.withColumn("avg_secs_per_unq_cap",
#     # F.when(F.col("avg_secs_per_unq") > 1830, 1830).otherwise(F.col("avg_secs_per_unq")))

# Maneira ideal de cap
p995_secs_per_unq = df_logs_fix_spine.approxQuantile("avg_secs_per_unq", [0.995], 0.001)[0]
df_logs_fix_spine = df_logs_fix_spine.withColumn("avg_secs_per_unq_cap",
    F.when(F.col("avg_secs_per_unq") > p995_secs_per_unq, p995_secs_per_unq).otherwise(F.col("avg_secs_per_unq")))

# Agrupamento por quartis - inadequado

# df_logs_fix_spine = df_logs_fix_spine.withColumn("avg_secs_per_unq_cap_group",
#     # F.when(F.col("avg_secs_per_unq_cap").isin(0) | F.col("avg_secs_per_unq_cap").isNull(), "00_unknown")
#     # .when((F.col("avg_secs_per_unq_cap") > 0) & (F.col("avg_secs_per_unq_cap") <= 173.965), "01_skimmer")
#     # .when((F.col("avg_secs_per_unq_cap") > 173.965) & (F.col("avg_secs_per_unq_cap") <= 230.141), "02_sampler")
#     # .when((F.col("avg_secs_per_unq_cap") > 230.141) & (F.col("avg_secs_per_unq_cap") <= 276.917), "03_focused_listener")
#     # .otherwise("04_deep_listener"))

# Metodo adequado
df_logs_fix_spine = segment_by_percentile(df_logs_fix_spine, "avg_secs_per_unq_cap", "logs")

map_seconds = {
    "unknown": "00_unknown",
    "tier_1": "01_skimmer", # Q1 - 0 "surfista": ouve apenas o inicio, talvez procurando algo especifico
    "tier_2": "02_sampler", # Q2 - 0 "degustador": ouve o parte da musica, mas nao a faixa toda
    "tier_3": "03_focused_listener", # Q3 - 0 usuario padrao: permanece na musica durante o tempo core
    "tier_4": "04_deep_listener" # Q4 - 0 imersivo: ouve faixas completas e provavelmente de generos mais longos.
}

df_logs_fix_spine = df_logs_fix_spine.replace(map_seconds, subset=["avg_secs_per_unq_cap_group"])
```

```
In [19]: # Aplicar transformacao logaritmica - considerar levar esta como o valor continuo no modelo linear
df_logs_fix_spine = df_logs_fix_spine.withColumn("log_avg_secs_per_unq", F.log1p(F.col("avg_secs_per_unq")))
```

9.1.6. plays_per_unq_cap_group

```
In [20]: df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_per_unq", F.when(F.col("num_unq") > 0, F.col("num_100") / F.col("num_unq")).otherwise(0.0))

# Winsorizar pra cima, no p995 visto no eda - nao usar valor bruto

# df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_per_unq_cap",
#     # F.when(F.col("plays_per_unq") > 7.14, 7.14).otherwise(F.col("plays_per_unq")))

# Maneira correta de cap
p995_plays_per_unq = df_logs_fix_spine.approxQuantile("plays_per_unq", [0.995], 0.001)[0]
df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_per_unq_cap",
    F.when(F.col("plays_per_unq") > p995_plays_per_unq, p995_plays_per_unq).otherwise(F.col("plays_per_unq")))

# Agrupamento semantic, baseado no EDA: separacao estatistica (percentis) nao faz sentido aqui
df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_per_unq_behavior",
    F.when(F.col("plays_per_unq_cap").isin(0) | F.col("plays_per_unq_cap").isNull(), "00_unknown")
    .when((F.col("plays_per_unq_cap") > 0) & (F.col("plays_per_unq_cap") < 1.1), "01_explorer")
    .when((F.col("plays_per_unq_cap") >= 1.1) & (F.col("plays_per_unq_cap") < 1.5), "02_light_repeat")
    .when((F.col("plays_per_unq_cap") >= 1.5) & (F.col("plays_per_unq_cap") < 3.0), "03_repeat")
    .otherwise("04_heavy_repeat")))
```

9.1.7. agrupadas

```
In [21]: total_logs = df_logs_fix_spine.count()
threshold = 0.01 # para ambas vou considerar 1 % como limite maximo de percentual de distribuicao de categorias: serao colapsadas na categoria 99_others
```

```
#### plays_behavior_vs_volume
```

```
In [22]: df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_behavior_vs_volume", F.concat_ws("_", F.col("plays_per_unq_behavior"),
F.col("total_plays_group")))
```

```
In [23]: freq_table_bhv_vol = df_logs_fix_spine.groupBy("plays_behavior_vs_volume").count()
freq_table_bhv_vol = freq_table_bhv_vol.withColumn("pct", F.col("count") / total_logs)

rare_categories = (freq_table_bhv_vol
    .filter(F.col("pct") < threshold)
    .select("plays_behavior_vs_volume")
    .rdd.flatMap(lambda x: x).collect())

df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_behavior_vs_volume_collapsed",
    F.when(F.col("plays_behavior_vs_volume").isin(rare_categories), "99_other")
    .otherwise(F.col("plays_behavior_vs_volume")))
```

plays_behavior_vs_completion

```
In [24]: df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_behavior_vs_completion", F.concat_ws("_", F.col("plays_per_unq_behavior"),
F.col("completed_songs_rate_group")))
```

```
In [25]: freq_table_bhv_comp = df_logs_fix_spine.groupBy("plays_behavior_vs_completion").count()
freq_table_bhv_comp = freq_table_bhv_comp.withColumn("pct", F.col("count") / total_logs)

rare_categories = (freq_table_bhv_comp
    .filter(F.col("pct") < threshold)
    .select("plays_behavior_vs_completion")
    .rdd.flatMap(lambda x: x).collect())

df_logs_fix_spine = df_logs_fix_spine.withColumn("plays_behavior_vs_completion_collapsed",
    F.when(F.col("plays_behavior_vs_completion").isin(rare_categories), "99_other")
    .otherwise(F.col("plays_behavior_vs_completion")))
```

9.1.8. catalog_exploration_ratio

```
In [26]: df_logs_fix_spine = df_logs_fix_spine.withColumn("catalog_exploration_ratio", F.when(F.col("num_100") > 0, F.col("num_unq") /
F.col("num_100")).otherwise(0.0))

# Calcular o p99.5 de catalog_exploration_ratio
p995_value_expl_ratio = df_logs_fix_spine.approxQuantile("catalog_exploration_ratio", [0.995], 0.001)[0]

# # Winsorizar no p99.5
df_logs_fix_spine = df_logs_fix_spine.withColumn("catalog_exploration_ratio_cap",
    F.when(F.col("catalog_exploration_ratio") > p995_value_expl_ratio, p995_value_expl_ratio)
    .otherwise(F.col("catalog_exploration_ratio")))
```

9.1.9. early_drop_rate

```
In [27]: df_logs_fix_spine = df_logs_fix_spine.withColumn("early_drop_rate",
    F.when(F.col("flag_has_logs").isin(1), ((F.col("num_25") + F.col("num_50")) / F.col("total_plays"))).otherwise(0.0))

df_logs_fix_spine = df_logs_fix_spine.withColumn("early_drop_rate_group",
    F.when(F.col("flag_has_logs").isin(0), "00_unknown")
    .when(F.col("early_drop_rate") == 0.0, "01_completionist") # Ouvinte Passivo/Fiel: Ouve 100% do que começa.
    .when(F.col("early_drop_rate") <= 0.20, "02_engaged") # Ouvinte Engajado: Pula até 1 em cada 5 músicas.
    .when(F.col("early_drop_rate") <= 0.5, "03_explorer") # Explorador: Pula entre 20% e 50%. Está ativamente buscando novas
músicas ou pulando o que não conhece.
    .otherwise("04_skipping_heavy")) # Skipper Crônico: Pula mais da metade do que começa
```

9.1.10. completion_efficiency

```
In [28]: df_logs_fix_spine = df_logs_fix_spine.withColumn("completion_efficiency",
    F.when(F.col("flag_has_logs").isin(1), (F.col("num_100") / (F.col("num_75") + F.col("num_985") +
F.col("num_100")))).otherwise(0.0))
```

9.1.11. flag_shallow_user

```
In [29]: df_logs_fix_spine = df_logs_fix_spine.withColumn("flag_shallow_user", F.when((F.col("num_25") > 0) & (F.col("num_100") == 0),
1).otherwise(0))
```

9.1.12. log_total_secs

A presente variavel sera criada considerando sua aplicacao no modelo linear, o qual se mostra sensivel a alta escala de valores.

```
In [30]: df_logs_fix_spine = df_logs_fix_spine.withColumn("log_total_secs", F.log1p(F.col("total_secs")))
```

9.1.13. features_de_tendencia_num

```
In [34]: selected_vars = [
    # core drivers de custo e marge: representam volume/custo direto
    "num_unq",
    "total_secs",
    "total_plays",
    "log_total_secs",
    "log_total_plays",
    # eficiencias de consumo / engajamento / intensidade media
    "avg_secs_per_unq_cap",
    "plays_per_unq_cap",
    "catalog_exploration_ratio_cap",
    "completion_efficiency",
    "completed_songs_rate",
    "early_drop_rate"
]
df_logs_book = features_de_tendencia_num(df_logs_fix_spine, selected_vars)
```

As transformações temporais contínuas foram aplicadas apenas a variáveis numéricas que representam intensidade, eficiência ou custo de consumo. Ate considerei inserir as demais variaveis de log brutas (num_n), pois permitem capturar mudanca de padrao na escuta (bom para arvores), mas se mostram altamente colineares entre si (como visto no EDA) e sao redundantes com early_drop_rate , completion_efficiency .

9.1.14. features_de_tendencia_cat

```
In [35]: selected_vars = [
    "flag_has_logs",
    "flag_shallow_user"
]
df_logs_book = features_de_tendencia_cat(df_logs_book, selected_vars)
```

Considerando que flags mensuram estado, uma transformação temporal de flag serve para responder as perguntas:

- * Com que frequência esse estado ocorre ao longo do tempo?
- * Esse comportamento é persistente ou pontual?

```
flag_xpto :  
  
* flag_xpto_sum_n → quantos meses no estado x nos últimos n meses  
* flag_xpto_mean_n → frequência de atividade (0–1) nos últimos n meses  
* flag_xpto_max_n → esteve em x ao menos uma vez últimos n meses?
```

9.1.15. Salvar base

In [36]: df_logs_book.printSchema()

```
root
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- num_25: double (nullable = true)
|-- num_50: double (nullable = true)
|-- num_75: double (nullable = true)
|-- num_985: double (nullable = true)
|-- num_100: double (nullable = true)
|-- num_unq: double (nullable = true)
|-- total_secs: double (nullable = true)
|-- total_plays: double (nullable = true)
|-- flag_has_logs: integer (nullable = false)
|-- total_plays_group: string (nullable = false)
|-- log_total_plays: double (nullable = true)
|-- completed_songs_rate: double (nullable = true)
|-- completed_songs_rate_group: string (nullable = false)
|-- avg_secs_per_unq: double (nullable = true)
|-- avg_secs_per_unq_cap: double (nullable = true)
|-- avg_secs_per_unq_cap_group: string (nullable = false)
|-- log_avg_secs_per_unq: double (nullable = true)
|-- plays_per_unq: double (nullable = true)
|-- plays_per_unq_cap: double (nullable = true)
|-- plays_per_unq_behavior: string (nullable = false)
|-- plays_behavior_vs_volume: string (nullable = false)
|-- plays_behavior_vs_volume_collapsed: string (nullable = false)
|-- plays_behavior_vs_completion: string (nullable = false)
|-- plays_behavior_vs_completion_collapsed: string (nullable = false)
|-- catalog_exploration_ratio: double (nullable = true)
|-- catalog_exploration_ratio_cap: double (nullable = true)
|-- early_drop_rate: double (nullable = true)
|-- early_drop_rate_group: string (nullable = false)
|-- completion_efficiency: double (nullable = true)
|-- flag_shallow_user: integer (nullable = false)
|-- log_total_secs: double (nullable = true)
|-- num_unq_raw: double (nullable = true)
|-- num_unq_mean_3: double (nullable = true)
|-- num_unq_min_3: double (nullable = true)
|-- num_unq_max_3: double (nullable = true)
|-- num_unq_ratio_ref_mean_3: double (nullable = true)
|-- num_unq_ratio_ref_min_3: double (nullable = true)
|-- num_unq_ratio_ref_max_3: double (nullable = true)
|-- num_unq_mean_6: double (nullable = true)
|-- num_unq_min_6: double (nullable = true)
|-- num_unq_max_6: double (nullable = true)
|-- num_unq_ratio_ref_mean_6: double (nullable = true)
|-- num_unq_ratio_ref_min_6: double (nullable = true)
|-- num_unq_ratio_ref_max_6: double (nullable = true)
|-- total_secs_raw: double (nullable = true)
|-- total_secs_mean_3: double (nullable = true)
|-- total_secs_min_3: double (nullable = true)
|-- total_secs_max_3: double (nullable = true)
|-- total_secs_ratio_ref_mean_3: double (nullable = true)
|-- total_secs_ratio_ref_min_3: double (nullable = true)
|-- total_secs_ratio_ref_max_3: double (nullable = true)
|-- total_secs_mean_6: double (nullable = true)
|-- total_secs_min_6: double (nullable = true)
|-- total_secs_max_6: double (nullable = true)
|-- total_secs_ratio_ref_mean_6: double (nullable = true)
|-- total_secs_ratio_ref_min_6: double (nullable = true)
|-- total_secs_ratio_ref_max_6: double (nullable = true)
|-- total_plays_raw: double (nullable = true)
|-- total_plays_mean_3: double (nullable = true)
|-- total_plays_min_3: double (nullable = true)
|-- total_plays_max_3: double (nullable = true)
|-- total_plays_ratio_ref_mean_3: double (nullable = true)
|-- total_plays_ratio_ref_min_3: double (nullable = true)
|-- total_plays_ratio_ref_max_3: double (nullable = true)
|-- total_plays_mean_6: double (nullable = true)
|-- total_plays_min_6: double (nullable = true)
|-- total_plays_max_6: double (nullable = true)
|-- total_plays_ratio_ref_mean_6: double (nullable = true)
|-- total_plays_ratio_ref_min_6: double (nullable = true)
|-- total_plays_ratio_ref_max_6: double (nullable = true)
|-- log_total_secs_raw: double (nullable = true)
|-- log_total_secs_mean_3: double (nullable = true)
|-- log_total_secs_min_3: double (nullable = true)
|-- log_total_secs_max_3: double (nullable = true)
|-- log_total_secs_ratio_ref_mean_3: double (nullable = true)
|-- log_total_secs_ratio_ref_min_3: double (nullable = true)
|-- log_total_secs_ratio_ref_max_3: double (nullable = true)
```



```
|-- early_drop_rate_ratio_ref_mean_3: double (nullable = true)
|-- early_drop_rate_ratio_ref_min_3: double (nullable = true)
|-- early_drop_rate_ratio_ref_max_3: double (nullable = true)
|-- early_drop_rate_mean_6: double (nullable = true)
|-- early_drop_rate_min_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_mean_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_min_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_max_6: double (nullable = true)
|-- flag_has_logs_raw: integer (nullable = false)
|-- flag_has_logs_lag_0: integer (nullable = true)
|-- flag_has_logs_lag_1: integer (nullable = true)
|-- flag_has_logs_lag_2: integer (nullable = true)
|-- flag_has_logs_sum_3: integer (nullable = true)
|-- flag_has_logs_mean_3: double (nullable = true)
|-- flag_has_logs_max_3: integer (nullable = true)
|-- flag_has_logs_lag_3: integer (nullable = true)
|-- flag_has_logs_lag_4: integer (nullable = true)
|-- flag_has_logs_lag_5: integer (nullable = true)
|-- flag_has_logs_sum_6: integer (nullable = true)
|-- flag_has_logs_mean_6: double (nullable = true)
|-- flag_has_logs_max_6: integer (nullable = true)
|-- flag_shallow_user_raw: integer (nullable = false)
|-- flag_shallow_user_lag_0: integer (nullable = true)
|-- flag_shallow_user_lag_1: integer (nullable = true)
|-- flag_shallow_user_lag_2: integer (nullable = true)
|-- flag_shallow_user_sum_3: integer (nullable = true)
|-- flag_shallow_user_mean_3: double (nullable = true)
|-- flag_shallow_user_max_3: integer (nullable = true)
|-- flag_shallow_user_lag_3: integer (nullable = true)
|-- flag_shallow_user_lag_4: integer (nullable = true)
|-- flag_shallow_user_lag_5: integer (nullable = true)
|-- flag_shallow_user_sum_6: integer (nullable = true)
|-- flag_shallow_user_mean_6: double (nullable = true)
|-- flag_shallow_user_max_6: integer (nullable = true)
```

```
In [37]: # 1. cache
df_logs_book = df_logs_book.persist()
df_logs_book.count()

# 2. salvar particionado
df_logs_book.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_logs_book")
```

9.1.16. Testes

```
In [38]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver"
df_logs_book = spark.read.parquet(f'{silver_path}/df_logs_book')
df_spine = spark.read.parquet(f'{silver_path}/df_spine')

df_logs_spine = df_spine.join(df_logs_book, ["msno", "safra"], "left")
```

```
In [39]: df_logs_spine.groupBy("flag_has_logs").count().show()
```

flag_has_logs	count
1	9975880
0	1266985

9.2. Book Transactions

9.2.1. Carregando bases

```
In [40]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver"
df_transactions_fix = spark.read.parquet(f'{silver_path}/df_transactions_fix.parquet')
df_spine = spark.read.parquet(f'{silver_path}/df_spine')
```

```
In [41]: df_transactions_fix.count()
Out[41]: 20712225
```

```
In [42]: df_spine.count()
Out[42]: 11242865
```

```
In [43]: df_transactions_fix.printSchema()
[163]: root
|-- msno: string (nullable = true)
|-- payment_method_id: integer (nullable = true)
|-- payment_plan_days: integer (nullable = true)
|-- plan_list_price: float (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- is_auto_renew: integer (nullable = true)
|-- transaction_date: date (nullable = true)
|-- membership_expire_date: date (nullable = true)
|-- is_cancel: integer (nullable = true)
|-- safra: integer (nullable = true)
|-- flag_expire_invalido: integer (nullable = true)
```

```
In [44]: df_transactions_fix_spine = df_spine.join(df_transactions_fix, ["msno", "safra"], "left")
Out[44]: 11242865
```

```
In [45]: df_transactions_fix_spine.printSchema()
root
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- payment_method_id: integer (nullable = true)
|-- payment_plan_days: integer (nullable = true)
|-- plan_list_price: float (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- is_auto_renew: integer (nullable = true)
|-- transaction_date: date (nullable = true)
|-- membership_expire_date: date (nullable = true)
|-- is_cancel: integer (nullable = true)
|-- flag_expire_invalido: integer (nullable = true)
```

Importante: as variaveis construidas podem sim ter grande correlacao entre si, dando indicios de multicolinearidade. Ressalta-se que, no momento atual, estamos apenas elaborando as variaveis possiveis de serem construidas e que agregam alguma informacao por si so, ou seja, individualmente. A analise da correlacao destas variaveis e melhor escolha de qual delas vai para o modelo final se dara na etapa de _feature selection_.

9.2.2. flag_has_transactions_

```
In [46]: df_transactions_fix_spine = df_transactions_fix_spine.withColumn("flag_has_transactions",
F.when(F.col("flag_expire_invalido").isNotNull(), 1).otherwise(0))
```

9.2.3. flags de cobranca e pagamento

Descobertas a partir do EDA em payment_plan_days

```
In [47]: df_transactions_fix_spine = (df_transactions_fix_spine
.withColumn("flag_valid_fee", # Flag: cobrança válida (plano real)
F.when((F.col("payment_plan_days") > 0) & (F.col("plan_list_price") > 0), 1).otherwise(0))
.withColumn("flag_exemption", # Flag: isenção (nem cobrou nem pagou, sem cancelamento)
F.when((F.col("actual_amount_paid") == 0) & (F.col("plan_list_price") == 0) & (F.col("is_cancel") == 0), 1).otherwise(0)))
```

Essas flags explicam anomalias de receita, algo que o modelo não aprende só olhando valores contínuos.

9.2.4. indicadores de compromisso / estabilidade

```
In [48]: df_transactions_fix_spine = df_transactions_fix_spine.withColumn("flag_plano_mensal", F.when(F.col("payment_plan_days").isin([30, 31]), 1).otherwise(0))

In [49]: df_transactions_fix_spine = df_transactions_fix_spine.withColumn("daily_revenue_efficiency", # Eficiência de receita diária
    F.when(F.col("payment_plan_days") > 0, F.col("actual_amount_paid") /
    F.col("payment_plan_days")).otherwise(F.col("actual_amount_paid")))

df_transactions_fix_spine = df_transactions_fix_spine.withColumn("revenue_tier",
    F.when(F.col("flag_has_transactions").isin(0), "00_unknown") # Sem transação
    .when(F.col("daily_revenue_efficiency") == 0, "01_free_isencao") # Isenções/vouchers
    .when(F.col("daily_revenue_efficiency") < 3.0, "02_low_tier") # Planos residuais/testes
    .when(F.col("daily_revenue_efficiency").between(3.0, 3.5), "03_standard_99") # Plano básico (R$ 99/mês)
    .when(F.col("daily_revenue_efficiency").between(4.5, 5.5), "04_premium_149") # Plano mais popular (R$ 149/mês)
    .when(F.col("daily_revenue_efficiency") > 5.5, "05_high_tier") # Planos premium (R$ 180+)
    .otherwise("06_others")) # Planos especiais/promocionais
```

9.2.5. grupo de método de pagamento

Baseado na dependência forte entre `payment_method_id` e `is_auto_renew`.

```
In [ ]: df_payment_method_group = (df_transactions_fix_spine
    .groupBy("payment_method_id")
    .agg(F.mean("is_auto_renew").alias("pct_auto_renew"))
    .withColumn("payment_method_group",
        F.when(F.col("pct_auto_renew").isNull(), "00_no_transaction")
        .when(F.col("pct_auto_renew") >= 0.90, "01_most_auto")
        .when(F.col("pct_auto_renew") >= 0.30, "02_mixed")
        .otherwise("03_most_manual"))
    .select("payment_method_id", "payment_method_group"))

df_transactions_fix_spine = (df_transactions_fix_spine.join(df_payment_method_group, "payment_method_id", "left"))

df_transactions_fix_spine = df_transactions_fix_spine.withColumn("payment_method_group",
    F.when(F.col("payment_method_group").isNull(), "00_no_transaction").otherwise(F.col("payment_method_group")))
```

* Remove quase-colinearidade em regressão

* Preserva sinal comportamental

* Facilita storytelling de negócio

9.2.6. payment_price_regime

Baseada na diferença de pagamentos

```
In [51]: df_transactions_fix_spine = df_transactions_fix_spine.withColumn("payment_price_regime",
    F.when(F.col("plan_list_price").isNull(), "00_no_transaction")
    .when((F.col("plan_list_price") > 0) & (F.col("actual_amount_paid") == F.col("plan_list_price")), "01_paid_as_expected")
    .otherwise("02_irregular_payment"))
```

9.2.7. features_de_tendencia_num

```
In [52]: selected_vars = [
    "daily_revenue_efficiency",
]

df_transactions_book = features_de_tendencia_num(df_transactions_fix_spine, selected_vars)
```

A `daily_revenue_efficiency` é o termômetro financeiro do cliente. Transformá-la temporalmente permite ao modelo distinguir entre:

* Estabilidade: Um cliente que mantém a mesma eficiência diária há 6 meses (perfil fiel);

* Degradação: Um cliente que teve uma queda na eficiência (ex: mudou de um plano Premium para um Standard ou promocional), o que impacta diretamente a margem futura;

* Volatilidade: Clientes que alternam entre meses pagos e meses gratuitos (isenção), sinalizando instabilidade financeira.

9.2.8. features_de_tendencia_cat

```
In [53]: selected_flags = [
    "flag_has_transactions",
    "flag_valid_fee",
    "flag_exemption",
    "flag_plano_mensal",
    "flag_expire_invalido",
]

df_transactions_book = features_de_tendencia_cat(df_transactions_book, selected_flags)
```

Para as flags, a transformação temporal não busca "média", mas sim persistência e frequência.

- `flag_has_transactions`

Motivo: Criar a Taxa de Adimplênciा. Um mean_6 de 1.0 indica um pagador perfeito; um mean_6 de 0.5 indica um cliente que paga apenas metade do tempo. É o preditor mais forte de margem futura;

- `flag_valid_fee & flag_exemption`

Motivo: Identificar o Perfil de Custo de Aquisição. Usuários que vivem de isenções (flag_exemption) de forma recorrente têm um LTV (Lifetime Value) muito menor. A recorrência dessas flags ajuda a separar o "usuário oportunista" do "usuário pagador";

- `flag_plano_mensal :`

Motivo: Medir a Fidelidade ao Modelo de Assinatura. A mudança de um plano mensal para um plano de outra duração (ou a interrupção do mensal) é um sinal clássico de alteração no comportamento de retenção.

- `flag_expire_invalido :`

Motivo: Capturar Ruído Operacional Recorrente. Se essa flag aparece com frequência para um usuário, pode indicar problemas técnicos na conta ou tentativas de fraude, ambos correlacionados com instabilidade de margem.

9.2.10. Salvar base

In [54]: df_transactions_book.printSchema()

```
root
|-- payment_method_id: integer (nullable = true)
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- payment_plan_days: integer (nullable = true)
|-- plan_list_price: float (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- is_auto_renew: integer (nullable = true)
|-- transaction_date: date (nullable = true)
|-- membership_expire_date: date (nullable = true)
|-- is_cancel: integer (nullable = true)
|-- flag_expire_invalido: integer (nullable = true)
|-- flag_has_transactions: integer (nullable = false)
|-- flag_valid_fee: integer (nullable = false)
|-- flag_exemption: integer (nullable = false)
|-- flag_plano_mensal: integer (nullable = false)
|-- daily_revenue_efficiency: double (nullable = true)
|-- revenue_tier: string (nullable = false)
|-- payment_method_group: string (nullable = true)
|-- payment_price_regime: string (nullable = false)
|-- daily_revenue_efficiency_raw: double (nullable = true)
|-- daily_revenue_efficiency_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_min_3: double (nullable = true)
|-- daily_revenue_efficiency_max_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
|-- daily_revenue_efficiency_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_min_6: double (nullable = true)
|-- daily_revenue_efficiency_max_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_6: double (nullable = true)
|-- flag_has_transactions_raw: integer (nullable = false)
|-- flag_has_transactions_lag_0: integer (nullable = true)
|-- flag_has_transactions_lag_1: integer (nullable = true)
|-- flag_has_transactions_lag_2: integer (nullable = true)
|-- flag_has_transactions_sum_3: integer (nullable = true)
|-- flag_has_transactions_mean_3: double (nullable = true)
|-- flag_has_transactions_max_3: integer (nullable = true)
|-- flag_has_transactions_lag_3: integer (nullable = true)
|-- flag_has_transactions_lag_4: integer (nullable = true)
|-- flag_has_transactions_lag_5: integer (nullable = true)
|-- flag_has_transactions_sum_6: integer (nullable = true)
|-- flag_has_transactions_mean_6: double (nullable = true)
|-- flag_has_transactions_max_6: integer (nullable = true)
|-- flag_valid_fee_raw: integer (nullable = false)
|-- flag_valid_fee_lag_0: integer (nullable = true)
|-- flag_valid_fee_lag_1: integer (nullable = true)
|-- flag_valid_fee_lag_2: integer (nullable = true)
|-- flag_valid_fee_sum_3: integer (nullable = true)
|-- flag_valid_fee_mean_3: double (nullable = true)
|-- flag_valid_fee_max_3: integer (nullable = true)
|-- flag_valid_fee_lag_3: integer (nullable = true)
|-- flag_valid_fee_lag_4: integer (nullable = true)
|-- flag_valid_fee_lag_5: integer (nullable = true)
|-- flag_valid_fee_sum_6: integer (nullable = true)
|-- flag_valid_fee_mean_6: double (nullable = true)
|-- flag_valid_fee_max_6: integer (nullable = true)
|-- flag_exemption_raw: integer (nullable = false)
|-- flag_exemption_lag_0: integer (nullable = true)
|-- flag_exemption_lag_1: integer (nullable = true)
|-- flag_exemption_lag_2: integer (nullable = true)
|-- flag_exemption_sum_3: integer (nullable = true)
|-- flag_exemption_mean_3: double (nullable = true)
|-- flag_exemption_max_3: integer (nullable = true)
|-- flag_exemption_lag_3: integer (nullable = true)
|-- flag_exemption_lag_4: integer (nullable = true)
|-- flag_exemption_lag_5: integer (nullable = true)
|-- flag_exemption_sum_6: integer (nullable = true)
|-- flag_exemption_mean_6: double (nullable = true)
|-- flag_exemption_max_6: integer (nullable = true)
|-- flag_plano_mensal_raw: integer (nullable = false)
|-- flag_plano_mensal_lag_0: integer (nullable = true)
|-- flag_plano_mensal_lag_1: integer (nullable = true)
|-- flag_plano_mensal_lag_2: integer (nullable = true)
|-- flag_plano_mensal_sum_3: integer (nullable = true)
|-- flag_plano_mensal_mean_3: double (nullable = true)
|-- flag_plano_mensal_max_3: integer (nullable = true)
|-- flag_plano_mensal_lag_3: integer (nullable = true)
```

```
|-- flag_plano_mensal_lag_4: integer (nullable = true)
|-- flag_plano_mensal_lag_5: integer (nullable = true)
|-- flag_plano_mensal_sum_6: integer (nullable = true)
|-- flag_plano_mensal_mean_6: double (nullable = true)
|-- flag_plano_mensal_max_6: integer (nullable = true)
|-- flag_expire_invalido_raw: integer (nullable = true)
|-- flag_expire_invalido_lag_0: integer (nullable = true)
|-- flag_expire_invalido_lag_1: integer (nullable = true)
|-- flag_expire_invalido_lag_2: integer (nullable = true)
|-- flag_expire_invalido_sum_3: integer (nullable = true)
|-- flag_expire_invalido_mean_3: double (nullable = true)
|-- flag_expire_invalido_max_3: integer (nullable = true)
|-- flag_expire_invalido_lag_3: integer (nullable = true)
|-- flag_expire_invalido_lag_4: integer (nullable = true)
|-- flag_expire_invalido_lag_5: integer (nullable = true)
|-- flag_expire_invalido_sum_6: integer (nullable = true)
|-- flag_expire_invalido_mean_6: double (nullable = true)
|-- flag_expire_invalido_max_6: integer (nullable = true)
```

```
In [55]: # 1. cache
df_transactions_book = df_transactions_book.persist()
df_transactions_book.count()

# 2. salvar particionado
df_transactions_book.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_transactions_book")
```

9.3. Book Members

9.3.1. Carregando bases

```
In [56]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver"
df_members_fix = spark.read.parquet(f"{silver_path}/df_members_fix.parquet")
df_spine = spark.read.parquet(f"{silver_path}/df_spine")
```

```
In [57]: df_members_fix.count()
```

```
Out[57]: 63867246
```

```
In [58]: df_spine.count()
```

```
Out[58]: 11242865
```

```
In [59]: df_members_fix.printSchema()
[174]:
root
|-- msno: string (nullable = true)
|-- safra: string (nullable = true)
|-- registration_init_time: date (nullable = true)
|-- city: integer (nullable = true)
|-- bd: integer (nullable = true)
|-- gender: string (nullable = true)
|-- registered_via: integer (nullable = true)
|-- is_ativo: integer (nullable = true)
|-- flag_idade_invalida: integer (nullable = true)
|-- idade_clean: integer (nullable = true)
|-- gender_clean: string (nullable = true)
```

```
In [59]: df_members_fix_spine = df_spine.join(df_members_fix, ["msno", "safra"], "left")
```

```
In [60]: df_members_fix_spine.count()
```

```
Out[60]: 11242865
```

```
In [61]: df_members_fix_spine.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- registration_init_time: date (nullable = true)
 |-- city: integer (nullable = true)
 |-- bd: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- registered_via: integer (nullable = true)
 |-- is_ativo: integer (nullable = true)
 |-- flag_idade_invalida: integer (nullable = true)
 |-- idade_clean: integer (nullable = true)
 |-- gender_clean: string (nullable = true)
```

9.3.2. faixa_idade

```
In [62]: df_members_book = df_members_fix_spine.withColumn("faixa_idade",
    F.when(F.col("flag_idade_invalida") == 1, "Desconhecido")
        .when((F.col("idade_clean") >= 16) & (F.col("idade_clean") <= 22), "16-22")
        .when((F.col("idade_clean") >= 23) & (F.col("idade_clean") <= 34), "23-34")
        .when((F.col("idade_clean") >= 35) & (F.col("idade_clean") <= 52), "35-52")
        .otherwise("53+"))
```

9.3.3. flags for city and gender

```
#### city
```

```
In [63]: df_members_book = df_members_book.withColumn("flag_city_one", F.when(F.col("city").isin(1), 1).otherwise(0))
```

```
#### gender
```

```
In [65]: df_members_book = df_members_book.drop("gender")
```

```
In [64]: df_members_book = df_members_book.withColumn("flag_gender_known", F.when(F.col("gender_clean") != "unknown", 1).otherwise(0))
```

9.3.4. registered_via

```
#### flag
```

```
In [66]: df_members_book = df_members_book.withColumn("flag_high_value_registered_via", F.when(F.col("registered_via").isin(3, 9), 1).otherwise(0))
```

```
#### grouped
```

```
In [67]: df_members_book = df_members_book.withColumn("registered_via_group",
    F.when(F.col("registered_via").isin(3, 9), "high_value")
        .when(F.col("registered_via") == 7, "low_value")
        .when(F.col("registered_via") == 4, "mid_value")
        .otherwise("other"))
```

9.3.5. registration_year_regime

```
In [68]: df_members_book = df_members_book.withColumn("ano_registro", F.year("registration_init_time"))

df_members_book = df_members_book.withColumn("registration_year_regime",
    F.when(F.col("ano_registro").isin([2004, 2005, 2006, 2007, 2008, 2009]), "2004-2009")\
        .when(F.col("ano_registro").isin([2010, 2011, 2012, 2013, 2014]), "2010-2014")\
        .otherwise("2015+"))

df_members_book = df_members_book.drop("ano_registro")
```

9.3.6. tenure

continuo

```
In [69]: df_members_book = df_members_book.withColumn("safra_date", F.to_date(F.concat(F.col("safra").cast("string"), F.lit("01")),"yyyyMMdd"))

df_members_book = (df_members_book
    .withColumn("tenure_meses", F.floor(F.months_between("safra_date", "registration_init_time")))
    .withColumn("tenure_meses", F.when(F.col("tenure_meses") < 0, 0).otherwise(F.col("tenure_meses"))))

df_members_book = df_members_book.drop("safra_date")
```

flag + faixa

```
In [70]: df_members_book = df_members_book.withColumn("flag_long_tenure", F.when(F.col("tenure_meses") >= 36, 1).otherwise(0))
```

```
In [71]: df_members_book = df_members_book.withColumn("tenure_faixa",
    F.when(F.col("tenure_meses").isNull(), "00_unknown")
    .when((F.col("tenure_meses") >= 0) & (F.col("tenure_meses") <= 11), "01_0-11months")
    .when((F.col("tenure_meses") >= 12) & (F.col("tenure_meses") <= 35), "02_12-35months")
    .otherwise("03_36+months"))
```

9.3.7. features_de_tendencia

Para as variaveis em questao, comprehendo que nenhuma transformacao seja necessaria, pois como observado no EDA, a unica variavel que pode variar com o tempo seria `tenure_meses`, a qual nao faz sentido capturar tendencia. As demais nao se alteraram ao longo do tempo pelo que foi analisado ate entao, e mesmo que possam eventualmente se alterar quando o modelo for implantado em producao, uma vez nao tendo diferenca entre as flags durante o treino, as variaveis seriam naturalmente "iguais". Poderiamos ate supor que as variaveis de tendencia dos ultimos n meses trariam informacoes mais valiosas, uma vez que consiste em revelar habitos e nao um unico momento em especifico, mas ainda assim, optei por seguir sem transformacoes de tendencia.

9.3.8. Salvar base

```
In [72]: df_members_book.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- registration_init_time: date (nullable = true)
 |-- city: integer (nullable = true)
 |-- bd: integer (nullable = true)
 |-- registered_via: integer (nullable = true)
 |-- is_ativo: integer (nullable = true)
 |-- flag_idade_invalida: integer (nullable = true)
 |-- idade_clean: integer (nullable = true)
 |-- gender_clean: string (nullable = true)
 |-- faixa_idade: string (nullable = false)
 |-- flag_city_one: integer (nullable = false)
 |-- flag_gender_known: integer (nullable = false)
 |-- flag_high_value_registered_via: integer (nullable = false)
 |-- registered_via_group: string (nullable = false)
 |-- registration_year_regime: string (nullable = false)
 |-- tenure_meses: long (nullable = true)
 |-- flag_long_tenure: integer (nullable = false)
 |-- tenure_faixa: string (nullable = false)
```

```
In [73]: # 1. cache
df_members_book = df_members_book.persist()
df_members_book.count()

# 2. salvar particionado
df_members_book.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_members_book")
```

10. _Feature Engineering_ - Etapa 2: _Cross-Domain Features_

Para capturar a complexidade da relação entre custo e receita, foram desenvolvidas `_features_` que cruzam os domínios de Logs, Transactions e Members. Diferente das isoladas, as `_cross-features_` permitem identificar perfis comportamentais específicos, como o 'Assinante Fantasma' (alta adimplênciam com baixo uso) e o 'Explorador de Isenções' (alto consumo de catálogo com baixa recorrência de pagamento).

O uso de componentes já transformados por tendências temporais (como médias de 3/6 meses) nessas fórmulas garante que as variáveis resultantes refletem a estabilidade do comportamento do usuário, reduzindo a volatilidade do modelo e aumentando sua robustez em cenários de produção.

10.1. Carregando e juntando spine + books

```
In [161]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver"
          df_spine = spark.read.parquet(f'{silver_path}/df_spine")
          df_logs_book = spark.read.parquet(f'{silver_path}/df_logs_book")
          df_transactions_book = spark.read.parquet(f'{silver_path}/df_transactions_book")
          df_members_book = spark.read.parquet(f'{silver_path}/df_members_book")
```

```
In [75]: df_spine.count()
```

```
Out[75]: 11242865
```

```
In [162]: df_final_vars = df_spine.join(df_logs_book, ["msno", "safra"], "left") \
           .join(df_transactions_book, ["msno", "safra"], "left") \
           .join(df_members_book, ["msno", "safra"], "left")
```

```
In [77]: df_final_vars.count()
```

```
Out[77]: 11242865
```

In [23]: df_final_vars.printSchema()

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- num_25: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- num_100: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_secs: double (nullable = true)
 |-- total_plays: double (nullable = true)
 |-- flag_has_logs: integer (nullable = true)
 |-- total_plays_group: string (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- completed_songs_rate: double (nullable = true)
 |-- completed_songs_rate_group: string (nullable = true)
 |-- avg_secs_per_unq: double (nullable = true)
 |-- avg_secs_per_unq_cap: double (nullable = true)
 |-- avg_secs_per_unq_cap_group: string (nullable = true)
 |-- log_avg_secs_per_unq: double (nullable = true)
 |-- plays_per_unq: double (nullable = true)
 |-- plays_per_unq_cap: double (nullable = true)
 |-- plays_per_unq_behavior: string (nullable = true)
 |-- plays_behavior_vs_volume: string (nullable = true)
 |-- plays_behavior_vs_volume_collapsed: string (nullable = true)
 |-- plays_behavior_vs_completion: string (nullable = true)
 |-- plays_behavior_vs_completion_collapsed: string (nullable = true)
 |-- catalog_exploration_ratio: double (nullable = true)
 |-- catalog_exploration_ratio_cap: double (nullable = true)
 |-- early_drop_rate: double (nullable = true)
 |-- early_drop_rate_group: string (nullable = true)
 |-- completion_efficiency: double (nullable = true)
 |-- flag_shallow_user: integer (nullable = true)
 |-- log_total_secs: double (nullable = true)
 |-- num_unq_raw: double (nullable = true)
 |-- num_unq_mean_3: double (nullable = true)
 |-- num_unq_min_3: double (nullable = true)
 |-- num_unq_max_3: double (nullable = true)
 |-- num_unq_ratio_ref_mean_3: double (nullable = true)
 |-- num_unq_ratio_ref_min_3: double (nullable = true)
 |-- num_unq_ratio_ref_max_3: double (nullable = true)
 |-- num_unq_mean_6: double (nullable = true)
 |-- num_unq_min_6: double (nullable = true)
 |-- num_unq_max_6: double (nullable = true)
 |-- num_unq_ratio_ref_mean_6: double (nullable = true)
 |-- num_unq_ratio_ref_min_6: double (nullable = true)
 |-- num_unq_ratio_ref_max_6: double (nullable = true)
 |-- total_secs_raw: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- total_secs_min_3: double (nullable = true)
 |-- total_secs_max_3: double (nullable = true)
 |-- total_secs_ratio_ref_mean_3: double (nullable = true)
 |-- total_secs_ratio_ref_min_3: double (nullable = true)
 |-- total_secs_ratio_ref_max_3: double (nullable = true)
 |-- total_secs_mean_6: double (nullable = true)
 |-- total_secs_min_6: double (nullable = true)
 |-- total_secs_max_6: double (nullable = true)
 |-- total_secs_ratio_ref_mean_6: double (nullable = true)
 |-- total_secs_ratio_ref_min_6: double (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- total_plays_raw: double (nullable = true)
 |-- total_plays_mean_3: double (nullable = true)
 |-- total_plays_min_3: double (nullable = true)
 |-- total_plays_max_3: double (nullable = true)
 |-- total_plays_ratio_ref_mean_3: double (nullable = true)
 |-- total_plays_ratio_ref_min_3: double (nullable = true)
 |-- total_plays_ratio_ref_max_3: double (nullable = true)
 |-- total_plays_mean_6: double (nullable = true)
 |-- total_plays_min_6: double (nullable = true)
 |-- total_plays_max_6: double (nullable = true)
 |-- total_plays_ratio_ref_mean_6: double (nullable = true)
 |-- total_plays_ratio_ref_min_6: double (nullable = true)
 |-- total_plays_ratio_ref_max_6: double (nullable = true)
 |-- log_total_secs_raw: double (nullable = true)
 |-- log_total_secs_mean_3: double (nullable = true)
 |-- log_total_secs_min_3: double (nullable = true)
 |-- log_total_secs_max_3: double (nullable = true)
 |-- log_total_secs_ratio_ref_mean_3: double (nullable = true)
 |-- log_total_secs_ratio_ref_min_3: double (nullable = true)
 |-- log_total_secs_ratio_ref_max_3: double (nullable = true)
 |-- log_total_secs_mean_6: double (nullable = true)
 |-- log_total_secs_min_6: double (nullable = true)
 |-- log_total_secs_max_6: double (nullable = true)
 |-- log_total_secs_ratio_ref_mean_6: double (nullable = true)
 |-- log_total_secs_ratio_ref_min_6: double (nullable = true)
```



```
|-- early_drop_rate_max_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_mean_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_min_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_max_6: double (nullable = true)
|-- flag_has_logs_raw: integer (nullable = true)
|-- flag_has_logs_lag_0: integer (nullable = true)
|-- flag_has_logs_lag_1: integer (nullable = true)
|-- flag_has_logs_lag_2: integer (nullable = true)
|-- flag_has_logs_sum_3: integer (nullable = true)
|-- flag_has_logs_mean_3: double (nullable = true)
|-- flag_has_logs_max_3: integer (nullable = true)
|-- flag_has_logs_lag_3: integer (nullable = true)
|-- flag_has_logs_lag_4: integer (nullable = true)
|-- flag_has_logs_lag_5: integer (nullable = true)
|-- flag_has_logs_sum_6: integer (nullable = true)
|-- flag_has_logs_mean_6: double (nullable = true)
|-- flag_has_logs_max_6: integer (nullable = true)
|-- flag_shallow_user_raw: integer (nullable = true)
|-- flag_shallow_user_lag_0: integer (nullable = true)
|-- flag_shallow_user_lag_1: integer (nullable = true)
|-- flag_shallow_user_lag_2: integer (nullable = true)
|-- flag_shallow_user_sum_3: integer (nullable = true)
|-- flag_shallow_user_mean_3: double (nullable = true)
|-- flag_shallow_user_max_3: integer (nullable = true)
|-- flag_shallow_user_lag_3: integer (nullable = true)
|-- flag_shallow_user_lag_4: integer (nullable = true)
|-- flag_shallow_user_lag_5: integer (nullable = true)
|-- flag_shallow_user_sum_6: integer (nullable = true)
|-- flag_shallow_user_mean_6: double (nullable = true)
|-- flag_shallow_user_max_6: integer (nullable = true)
|-- payment_method_id: integer (nullable = true)
|-- payment_plan_days: integer (nullable = true)
|-- plan_list_price: float (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- is_auto_renew: integer (nullable = true)
|-- transaction_date: date (nullable = true)
|-- membership_expire_date: date (nullable = true)
|-- is_cancel: integer (nullable = true)
|-- flag_expire_invalido: integer (nullable = true)
|-- flag_has_transactions: integer (nullable = true)
|-- flag_valid_fee: integer (nullable = true)
|-- flag_exemption: integer (nullable = true)
|-- flag_plano_mensal: integer (nullable = true)
|-- daily_revenue_efficiency: double (nullable = true)
|-- revenue_tier: string (nullable = true)
|-- payment_method_group: string (nullable = true)
|-- payment_price_regime: string (nullable = true)
|-- daily_revenue_efficiency_raw: double (nullable = true)
|-- daily_revenue_efficiency_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_min_3: double (nullable = true)
|-- daily_revenue_efficiency_max_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
|-- daily_revenue_efficiency_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_min_6: double (nullable = true)
|-- daily_revenue_efficiency_max_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_6: double (nullable = true)
|-- flag_has_transactions_raw: integer (nullable = true)
|-- flag_has_transactions_lag_0: integer (nullable = true)
|-- flag_has_transactions_lag_1: integer (nullable = true)
|-- flag_has_transactions_lag_2: integer (nullable = true)
|-- flag_has_transactions_sum_3: integer (nullable = true)
|-- flag_has_transactions_mean_3: double (nullable = true)
|-- flag_has_transactions_max_3: integer (nullable = true)
|-- flag_has_transactions_lag_3: integer (nullable = true)
|-- flag_has_transactions_lag_4: integer (nullable = true)
|-- flag_has_transactions_lag_5: integer (nullable = true)
|-- flag_has_transactions_sum_6: integer (nullable = true)
|-- flag_has_transactions_mean_6: double (nullable = true)
|-- flag_has_transactions_max_6: integer (nullable = true)
|-- flag_valid_fee_raw: integer (nullable = true)
|-- flag_valid_fee_lag_0: integer (nullable = true)
|-- flag_valid_fee_lag_1: integer (nullable = true)
|-- flag_valid_fee_lag_2: integer (nullable = true)
|-- flag_valid_fee_sum_3: integer (nullable = true)
|-- flag_valid_fee_mean_3: double (nullable = true)
|-- flag_valid_fee_max_3: integer (nullable = true)
|-- flag_valid_fee_lag_3: integer (nullable = true)
|-- flag_valid_fee_lag_4: integer (nullable = true)
|-- flag_valid_fee_lag_5: integer (nullable = true)
|-- flag_valid_fee_sum_6: integer (nullable = true)
|-- flag_valid_fee_mean_6: double (nullable = true)
|-- flag_valid_fee_max_6: integer (nullable = true)
|-- flag_exemption_raw: integer (nullable = true)
|-- flag_exemption_lag_0: integer (nullable = true)
```

```

|-- flag_exemption_lag_1: integer (nullable = true)
|-- flag_exemption_lag_2: integer (nullable = true)
|-- flag_exemption_sum_3: integer (nullable = true)
|-- flag_exemption_mean_3: double (nullable = true)
|-- flag_exemption_max_3: integer (nullable = true)
|-- flag_exemption_lag_3: integer (nullable = true)
|-- flag_exemption_lag_4: integer (nullable = true)
|-- flag_exemption_lag_5: integer (nullable = true)
|-- flag_exemption_sum_6: integer (nullable = true)
|-- flag_exemption_mean_6: double (nullable = true)
|-- flag_exemption_max_6: integer (nullable = true)
|-- flag_plano_mensal_raw: integer (nullable = true)
|-- flag_plano_mensal_lag_0: integer (nullable = true)
|-- flag_plano_mensal_lag_1: integer (nullable = true)
|-- flag_plano_mensal_lag_2: integer (nullable = true)
|-- flag_plano_mensal_sum_3: integer (nullable = true)
|-- flag_plano_mensal_mean_3: double (nullable = true)
|-- flag_plano_mensal_max_3: integer (nullable = true)
|-- flag_plano_mensal_lag_3: integer (nullable = true)
|-- flag_plano_mensal_lag_4: integer (nullable = true)
|-- flag_plano_mensal_lag_5: integer (nullable = true)
|-- flag_plano_mensal_sum_6: integer (nullable = true)
|-- flag_plano_mensal_mean_6: double (nullable = true)
|-- flag_plano_mensal_max_6: integer (nullable = true)
|-- flag_expire_invalido_raw: integer (nullable = true)
|-- flag_expire_invalido_lag_0: integer (nullable = true)
|-- flag_expire_invalido_lag_1: integer (nullable = true)
|-- flag_expire_invalido_lag_2: integer (nullable = true)
|-- flag_expire_invalido_sum_3: integer (nullable = true)
|-- flag_expire_invalido_mean_3: double (nullable = true)
|-- flag_expire_invalido_max_3: integer (nullable = true)
|-- flag_expire_invalido_lag_3: integer (nullable = true)
|-- flag_expire_invalido_lag_4: integer (nullable = true)
|-- flag_expire_invalido_lag_5: integer (nullable = true)
|-- flag_expire_invalido_sum_6: integer (nullable = true)
|-- flag_expire_invalido_mean_6: double (nullable = true)
|-- flag_expire_invalido_max_6: integer (nullable = true)
|-- registration_init_time: date (nullable = true)
|-- city: integer (nullable = true)
|-- bd: integer (nullable = true)
|-- gender: string (nullable = true)
|-- registered_via: integer (nullable = true)
|-- is_ativo: integer (nullable = true)
|-- flag_idade_invalida: integer (nullable = true)
|-- idade_clean: integer (nullable = true)
|-- gender_clean: string (nullable = true)
|-- faixa_idade: string (nullable = true)
|-- flag_city_one: integer (nullable = true)
|-- flag_gender_known: integer (nullable = true)
|-- flag_high_value_registered_via: integer (nullable = true)
|-- registered_via_group: string (nullable = true)
|-- registration_year_regime: string (nullable = true)
|-- tenure_meses: long (nullable = true)
|-- flag_long_tenure: integer (nullable = true)
|-- tenure_faixa: string (nullable = true)

```

Testes

In [78]: `df_final_vars.groupBy("flag_has_logs").count().show()`

```
+-----+-----+
|flag_has_logs| count|
+-----+-----+
|          1|9975880|
|          0|1266985|
+-----+-----+
```

In [79]: `df_final_vars.groupBy("flag_has_transactions").count().show()`

```
+-----+-----+
|flag_has_transactions| count|
+-----+-----+
|           1|9390483|
|           0|1852382|
+-----+-----+
```

```
In [80]: df_final_vars.groupBy("plays_per_unq_behavior").count().show()
```

plays_per_unq_behavior	count
02_light_repeat	1470819
00_unknown	1409847
03_repeat	665352
01_explorer	7491601
04_heavy_repeat	205246

Os joins foram corretos, aparentemente não tem casos de nulos mais.

10.2. Margem líquida + target

```
In [ ]: v_paid = F.coalesce(F.col("actual_amount_paid"), F.lit(0))
v_unq = F.coalesce(F.col("num_unq"), F.lit(0))
v_secs = F.coalesce(F.col("total_secs"), F.lit(0))

# Custo fixo e variável
custo_variavel = (0.0051 * v_unq) + (0.0001 * v_secs)
custo_total = 50 + custo_variavel

df_final_vars = df_final_vars.withColumn("margem_liquida_mensal",
    F.when( F.col("actual_amount_paid").isNull() & F.col("num_unq").isNull() & F.col("total_secs").isNull(),
    F.lit(-50)).otherwise(v_paid - custo_total))

w = Window.partitionBy("msno").orderBy("safra")
# Target: margem do mês seguinte
df_final_vars = df_final_vars.withColumn("target", F.lead("margem_liquida_mensal", 1).over(w))

df_final_vars = aplicar_winsorizacao(df_final_vars, ["target"])
```

Coluna target: Limite Inferior=-108.8147221999999, Limite Superior=129.0401291

10.3. revenue_per_hour_listened

10.3.1. Definição

Eficiência de Custo por Engajamento. Responde à pergunta: "Quanto de receita diária o usuário gera por hora de conteúdo consumido?"

\$\$ \text{revenue_per_hour_listened} = \frac{\text{daily_revenue_efficiency_mean_3}}{\left(\frac{\text{total_secs_mean_3}}{3600}\right) + 1} \$\$

- **Numerador:** daily_revenue_efficiency_mean_3 representa a receita diária média dos últimos 3 meses (NTD/dia).
- **Denominador:** total_secs_mean_3 / 3600 converte segundos em horas. O +1 serve como:

1. Proteção contra divisão por zero: Usuários sem logs (total_secs = 0) teriam denominador indefinido;
2. Regularização: Evita explosão do ratio para usuários com consumo muito baixo (ex: 10 segundos);
3. Interpretação: O denominador mínimo de 1 hora garante que a métrica seja sempre bem-definida.

Interpretação de Valores

Valor Perfil Impacto na Margem
----- ----- -----
Alto (>5.0) Assinante Fantasma: paga bem e usa pouco ● Margem Líquida Máxima
Médio (2.0-5.0) Usuário Equilibrado ● Margem Saudável
<b (<2.0)<="" b="" baixo=""> Heavy User: consome muito em relação ao que paga ● Margem Comprimida/Risco

10.3.2. Construção

```
In [164]: df_final_vars = df_final_vars.withColumn("revenue_per_hour_listened",
[F.when((F.col("daily_revenue_efficiency_mean_3") >= 0) & (F.col("total_secs_mean_3") >= 0), # Tratamento para o caso dos
códigos sentinelas desenvolvidos (ex.: -99999)
      F.col("daily_revenue_efficiency_mean_3") / ((F.col("total_secs_mean_3") / 3600.0) + 1))
.otherwise(None))]
```

10.3.3. "_Fast EDA_" - Checagem rápida para validar a variável

Descriptivas + Correlação

```
In [32]: df_final_vars.select("revenue_per_hour_listened").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

	summary revenue_per_hour_listened
count	7595681
mean	0.39007977080327977
stddev	1.2629436194802681
min	9.222850774873949E-4
1%	0.02081915265478148
5%	0.03671249556477943
25%	0.09127676276173609
50%	0.17973124027703918
75%	0.4013412952209249
95%	1.483265136015288
99.5%	3.2654715105873966
max	715.2211255314578

```
In [33]: df_final_vars.select("revenue_per_hour_listened", "target").corr("revenue_per_hour_listened", "target")
```

Out[33]: 0.0655314249010374

```
In [34]: df_final_vars.select("revenue_per_hour_listened", "target_win").corr("revenue_per_hour_listened", "target_win")
```

Out[34]: 0.09807103374416482

Aproximadamente 32% dos registros continham códigos sentinelas negativos provenientes da ausência de histórico de receita válido (`daily_revenue_efficiency_mean_3`). A variável apresentou uma distribuição estável (média = 0.39, stddev = 1.26) com correlação moderada com a target winsorizada. A correlação relativamente baixa reflete a natureza não-linear da relação entre receita por hora e margem líquida, que depende também da exploração do catálogo (`num_unq`).

Testes para tratamentos finais

Agrupamento

```
In [42]: df_teste = df_final_vars.select("msno", "safra", "revenue_per_hour_listened", "target_win").withColumn("revenue_per_hour_tier",
F.when(F.col("revenue_per_hour_listened").isNull(), "00_unknown")
.when(F.col("revenue_per_hour_listened") < 0.1, "01_very_low")
.when(F.col("revenue_per_hour_listened") < 0.3, "02_low")
.when(F.col("revenue_per_hour_listened") < 0.6, "03_medium")
.when(F.col("revenue_per_hour_listened") < 1.5, "04_high")
.otherwise("05_very_high"))
```

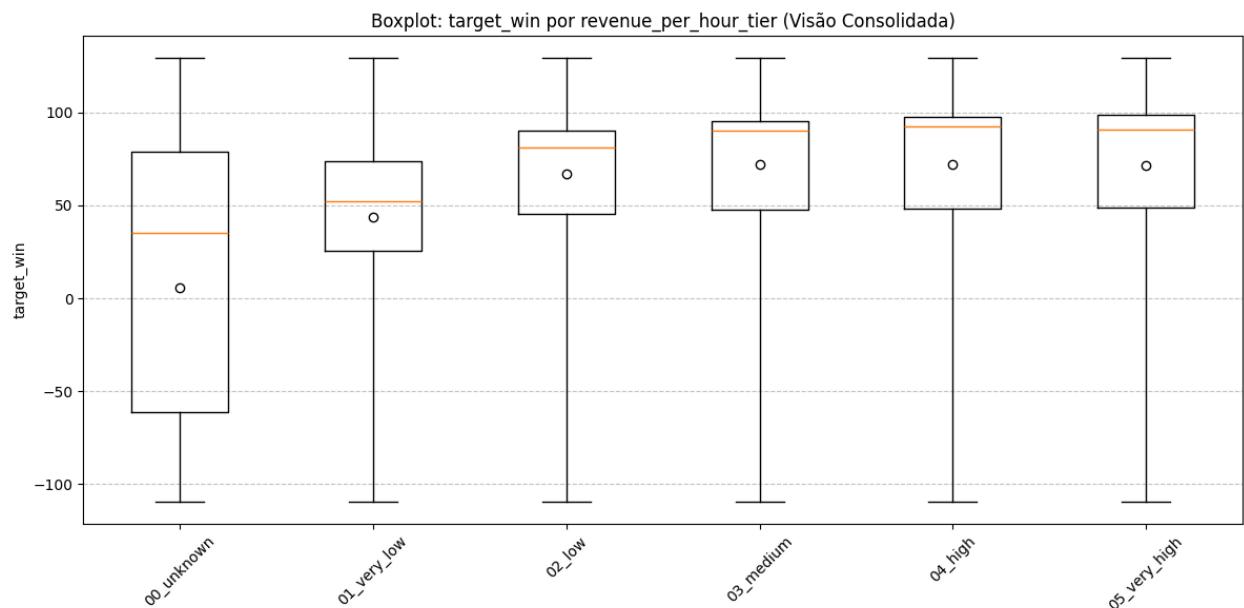
```
In [ ]: calcular_distribuicao(df_teste, ["revenue_per_hour_tier"])
```

	total	pct_total
00_unknown	3647184	32.44
02_low	2964952	26.37
01_very_low	2136040	19.0
03_medium	1242019	11.05
04_high	879350	7.82
05_very_high	373320	3.32

```
DataFrame[revenue_per_hour_tier: string, total: bigint, pct_total: double]
```

```
In [ ]: plot_boxplot(df_teste, ["revenue_per_hour_tier"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: revenue_per_hour_tier...



--- Estatísticas: revenue_per_hour_tier (Visão Consolidada) ---

	revenue_per_hour_tier	min	q1	med	mean	q3	max
0	00_unknown	-109.315469	-61.191562	35.243518	5.913455	79.000000	129.051366
3	01_very_low	-109.315469	25.736726	52.302961	43.546908	73.855340	129.051366
4	02_low	-109.315469	45.247835	81.004225	66.663018	90.271398	129.051366
1	03_medium	-109.315469	47.378830	89.883936	71.910907	95.488236	129.051366
2	04_high	-109.315469	48.322100	92.163950	72.274097	97.469390	129.051366
5	05_very_high	-109.315469	48.928189	90.974603	71.364724	98.642785	129.051366

```
=====
```

```
In [ ]: df_teste = df_teste.withColumn("revenue_per_hour_tier",
    F.when(F.col("revenue_per_hour_listened").isNull(), "00_unknown")
    .when(F.col("revenue_per_hour_listened") < 0.3, "01_low")
    .when(F.col("revenue_per_hour_listened") < 0.6, "02_medium")
    .otherwise("03_high"))
```

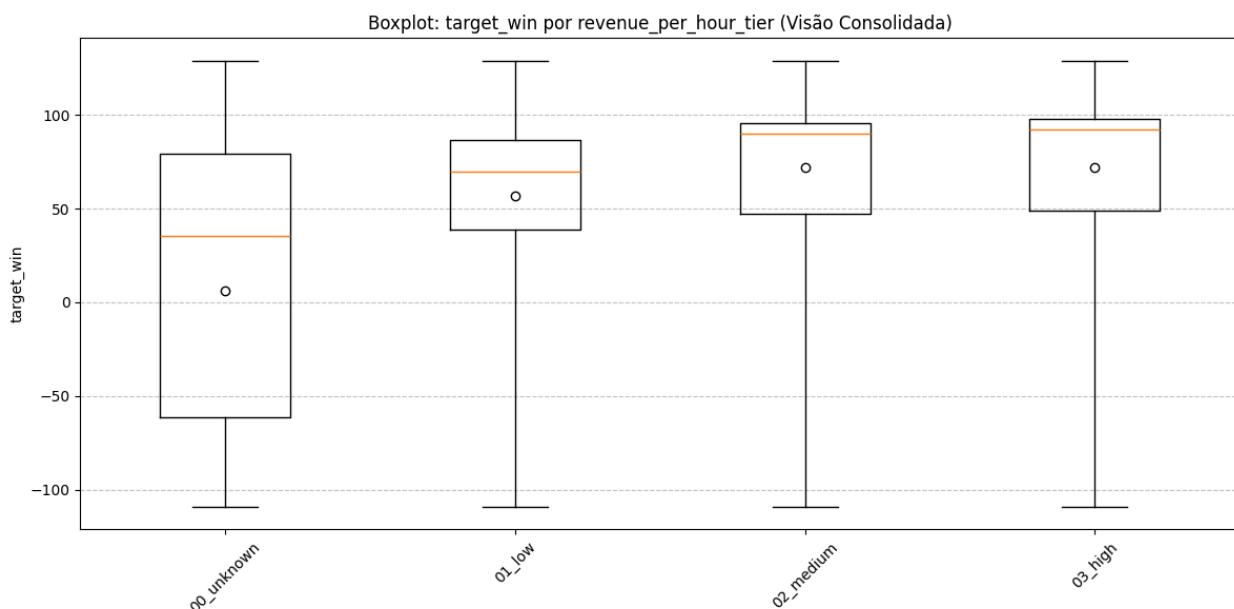
```
In [ ]: calcular_distribuicao(df_teste, ["revenue_per_hour_tier"])
```

revenue_per_hour_tier	total	pct_total
01_low	5100992	45.37
00_unknown	3647184	32.44
03_high	1252670	11.14
02_medium	1242019	11.05

```
DataFrame[revenue_per_hour_tier: string, total: bigint, pct_total: double]
```

```
In [ ]: plot_boxplot(df_teste, ["revenue_per_hour_tier"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: revenue_per_hour_tier...



```
--- Estatísticas: revenue_per_hour_tier (Visão Consolidada) ---
```

	revenue_per_hour_tier	min	q1	med	mean	q3	max
2	00_unknown	-109.315469	-61.191562	35.243518	5.913455	79.000000	129.051366
0	01_low	-109.315469	38.996684	69.701271	56.992275	86.578686	129.051366
1	02_medium	-109.315469	47.378830	89.883936	71.910907	95.488236	129.051366
3	03_high	-109.315469	48.613136	92.062052	72.002942	97.911087	129.051366

```
=====
```

```
In [43]: df_teste = df_teste.withColumn("revenue_per_hour_tier", F.when(F.col("revenue_per_hour_listened").isNull(), "00_unknown") .when(F.col("revenue_per_hour_listened") < 0.1, "01_low") .when(F.col("revenue_per_hour_listened") < 0.3, "02_medium") .otherwise("03_high"))
```

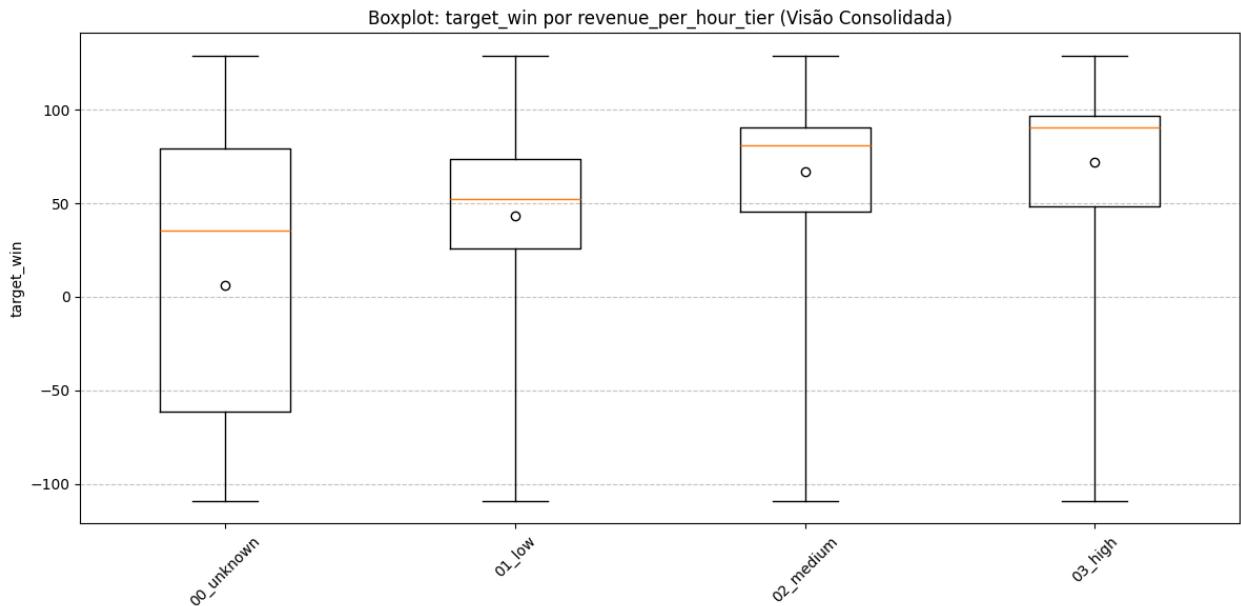
```
In [44]: calcular_distribuicao(df_teste, ["revenue_per_hour_tier"])
```

revenue_per_hour_tier	total	pct_total
00_unknown	3647184	32.44
02_medium	2964952	26.37
03_high	2494689	22.19
01_low	2136040	19.0

```
Out[44]: DataFrame[revenue_per_hour_tier: string, total: bigint, pct_total: double]
```

```
In [45]: plot_boxplot(df_teste, ["revenue_per_hour_tier"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: revenue_per_hour_tier...



--- Estatísticas: revenue_per_hour_tier (Visão Consolidada) ---

	revenue_per_hour_tier	min	q1	med	mean	q3	max
2	00_unknown	-109.315469	-61.191562	35.243518	5.913455	79.000000	129.051366
0	01_low	-109.315469	25.736726	52.302961	43.546908	73.855340	129.051366
1	02_medium	-109.315469	45.247835	81.004225	66.663018	90.271398	129.051366
3	03_high	-109.315469	48.271231	90.449188	71.957152	96.941563	129.051366

=====

A categorização da eficiência de receita por hora provou ser um discriminador robusto de lucratividade. A análise dos tiers revelou uma progressão não-linear na margem líquida, com o tier 03_high apresentando uma mediana de margem 72% superior ao tier 01_low (90.44 vs 52.30).

O grupo 00_unknown (usuários sem histórico de receita válido) foi identificado como o segmento de maior risco, apresentando a menor média de margem (5.91), o que justifica sua manutenção como uma categoria distinta para capturar o comportamento de usuários inadimplentes ou sem transações.

10.3.4. Conclusao

```
In [165]: # Winsorizar pra cima, no p995 - nao usar valor bruto
p995_rev_per_hour = df_final_vars.approxQuantile("revenue_per_hour_listened", [0.995], 0.001)[0]
df_final_vars = df_final_vars.withColumn("revenue_per_hour_listened_cap",
    F.when(F.col("revenue_per_hour_listened") > p995_rev_per_hour,
    p995_rev_per_hour).otherwise(F.col("revenue_per_hour_listened")))

# Tratar os casos de nulos
df_final_vars = df_final_vars.withColumn("revenue_per_hour_listened_cap",
    F.when(F.col("revenue_per_hour_listened_cap").isNull(), 0.0).otherwise(F.col("revenue_per_hour_listened_cap")))
```

```
In [166]: df_final_vars = df_final_vars.withColumn("revenue_per_hour_tier",
    F.when(F.col("revenue_per_hour_listened").isNull(), "00_unknown")
    .when(F.col("revenue_per_hour_listened") < 0.1, "01_low")
    .when(F.col("revenue_per_hour_listened") < 0.3, "02_medium")
    .otherwise("03_high"))

df_final_vars = df_final_vars.drop("revenue_per_hour_listened")
```

10.4. exemption_exploitation_index

10.4.1. Definição

Índice de Aproveitamento de Isenção. identifica o "**Usuário Explorador Gratuito**" através da interação entre:

- **Fator 1:** flag_exemption_mean_6 = Taxa de meses com isenção nos últimos 6 meses (0 a 1).
- **Fator 2:** catalog_exploration_ratio_cap_mean_3 = Intensidade de exploração de catálogo (músicas únicas / músicas completadas)

```
$$
\text{exemption\_exploitation\_index} = \text{flag\_exemption\_mean\_6} \times \text{catalog\_exploration\_ratio\_cap\_mean\_3}
$$
```

O produto (em vez de soma ou ratio) captura a interação não-linear:

- Se flag_exemption_mean_6 ≈ 0 (usuário pagante regular), o índice zera independentemente da exploração.
- Se flag_exemption_mean_6 ≈ 1 (usuário com isenções recorrentes), o índice reflete diretamente a exploração de catálogo.

Valor Perfil Impacto na Margem
----- ----- -----
Alto (>3.0) Explorador de Isenções: descobre muitas músicas sem pagar ● Alto Custo Variável (num_unq), Receita Zero
Médio (1.0-3.0) Isenção Ocasional com Uso Moderado ● Risco Moderado
Baixo (<1.0) Pagante Regular ou Isento com Baixo Uso ● Baixo Risco

10.4.2. Construção

```
In [87]: df_final_vars = df_final_vars.withColumn("exemption_exploitation_index",
    F.when((F.col("flag_exemption_mean_6") >= 0) & (F.col("flag_exemption_mean_6") <= 1) & # Garantir que é uma proporção válida
        (F.col("catalog_exploration_ratio_cap_mean_3") >= 0), # Garantir que é positivo
        F.col("flag_exemption_mean_6") * F.col("catalog_exploration_ratio_cap_mean_3"))
    .otherwise(None)) # Casos inválidos viram null
```

10.4.3. "_Fast EDA_"

Descriptivas + Correlação

```
In [49]: df_final_vars.select("exemption_exploitation_index").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+-----+
|summary|exemption_exploitation_index|
+-----+-----+
| count| 7705254|
| mean | 0.008924474607944327|
| stddev| 0.07440198111705876|
| min | 0.0|
| 1% | 0.0|
| 5% | 0.0|
| 25%| 0.0|
| 50%| 0.0|
| 75%| 0.0|
| 95%| 0.0|
| 99.5%| 0.41274008257045414|
| max | 10.152116402116404|
+-----+-----+
```

```
In [50]: df_final_vars.select("exemption_exploitation_index", "target").corr("exemption_exploitation_index", "target")
```

```
Out[50]: -0.028446912366603884
```

```
In [51]: df_final_vars.select("exemption_exploitation_index", "target_win").corr("exemption_exploitation_index", "target_win")
```

```
Out[51]: -0.03997726904790445
```

O sinal negativo faz sentido (isenção = menos margem), mas a magnitude é irrelevante.

10.4.4. Conclusao

A hipótese original era boa: "usuários com isenção recorrente que exploram muito catálogo são caros". Mas na prática pouquíssimos usuários têm isenção recorrente (`flag_exemption_mean_6 > 0`), então o produto vira zero para 95% da base. Os 5% restantes não têm poder estatístico suficiente para mover a correlação. Descartar.

```
In [88]: df_final_vars = df_final_vars.drop("exemption_exploitation_index")
```

10.5. usage_intensity_per_tenure

10.5.1. Definição

Intensidade de Uso por Maturidade. Normaliza o volume de `_plays_` recente pelo tempo de vida do cliente, respondendo: "Qual a intensidade de uso mensal ajustada pela maturidade do cliente?"

```
$$
\text{usage\_intensity\_per\_tenure} = \frac{\text{total\_plays\_mean\_3}}{\text{tenure\_meses}} + 1
$$
```

- **Numerador:** `total_plays_mean_3` = Média de plays dos últimos 3 meses.
- **Denominador:** `tenure_meses + 1` onde o `+1` serve para:

1. Evitar divisão por zero: Clientes no primeiro mês (`tenure = 0`) teriam denominador indefinido.
2. Estabilização: Reduz a volatilidade para clientes muito novos (`tenure < 3` meses).
3. Interpretação: Transforma a métrica em "plays por mês de vida (com mínimo de 1 mês)".

Contexto Valor Alto Valor Baixo
----- ----- -----
Usuário Novo (<code>tenure < 6</code>) ● Engajamento inicial saudável (esperado) ● Risco de churn precoce
Usuário Maduro (<code>tenure > 12</code>) ● Core fidelizado (alto LTV) ● Sinal preditivo de churn iminente

10.5.2. Construção

```
In [167]: df_final_vars = df_final_vars.withColumn("usage_intensity_per_tenure",
    F.when((F.col("total_plays_mean_3") >= 0) & (F.col("tenure_meses") >= 0), # Elimina sentinelas negativos e garante tenure
    válido
        F.col("total_plays_mean_3") / (F.col("tenure_meses") + 1))
    .otherwise(None))
```

10.5.3. _Fast EDA_

```
#### Descriptivas + Correlação
```

```
In [54]: df_final_vars.select("usage_intensity_per_tenure").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

 summary usage_intensity_per_tenure 	
+-----+-----+	+-----+
count 9548384	
mean 55.537243253089606	
stddev 140.65100329141413	
min 0.006802721088435374	
1% 0.32367149758454106	
5% 1.1521739130434783	
25% 5.87378640776699	
50% 16.246913580246915	
75% 45.833333333333336	
95% 230.2333333333335	
99.5% 918.0	
max 4630.0	
+-----+-----+	+-----+

```
In [55]: df_final_vars.select("usage_intensity_per_tenure", "target").corr("usage_intensity_per_tenure", "target")
```

```
Out[55]: -0.06605125168386554
```

```
In [56]: df_final_vars.select("usage_intensity_per_tenure", "target_win").corr("usage_intensity_per_tenure", "target_win")
```

```
Out[56]: -0.09445128652909669
```

A análise exploratória revelou uma distribuição assimétrica à direita, típica de métricas de engajamento, com mediana de 16.25 e cauda longa para usuários com tenure baixo e uso intensivo. A correlação negativa com a target winsorizada confirma o racional de negócio: maior intensidade de uso implica maior custo operacional, pressionando a margem quando não acompanhada de receita proporcional.

Agrupamento via quartil

```
In [58]: percentiles_usage_intensity = df_final_vars.stat.approxQuantile("usage_intensity_per_tenure", [0.25, 0.5, 0.75], 0.001)
p25, p50, p75 = percentiles_usage_intensity[0], percentiles_usage_intensity[1], percentiles_usage_intensity[2]

df_teste = df_final_vars.select("msno", "safra", "usage_intensity_per_tenure", "target_win").withColumn("usage_intensity_tier",
    F.when(F.col("usage_intensity_per_tenure").isNull(), "00_unknown")
    .when(F.col("usage_intensity_per_tenure") <= p25, "01_low_engagement")
    .when(F.col("usage_intensity_per_tenure") <= p50, "02_moderate_engagement")
    .when(F.col("usage_intensity_per_tenure") <= p75, "03_high_engagement")
    .otherwise("04_power_user"))
```

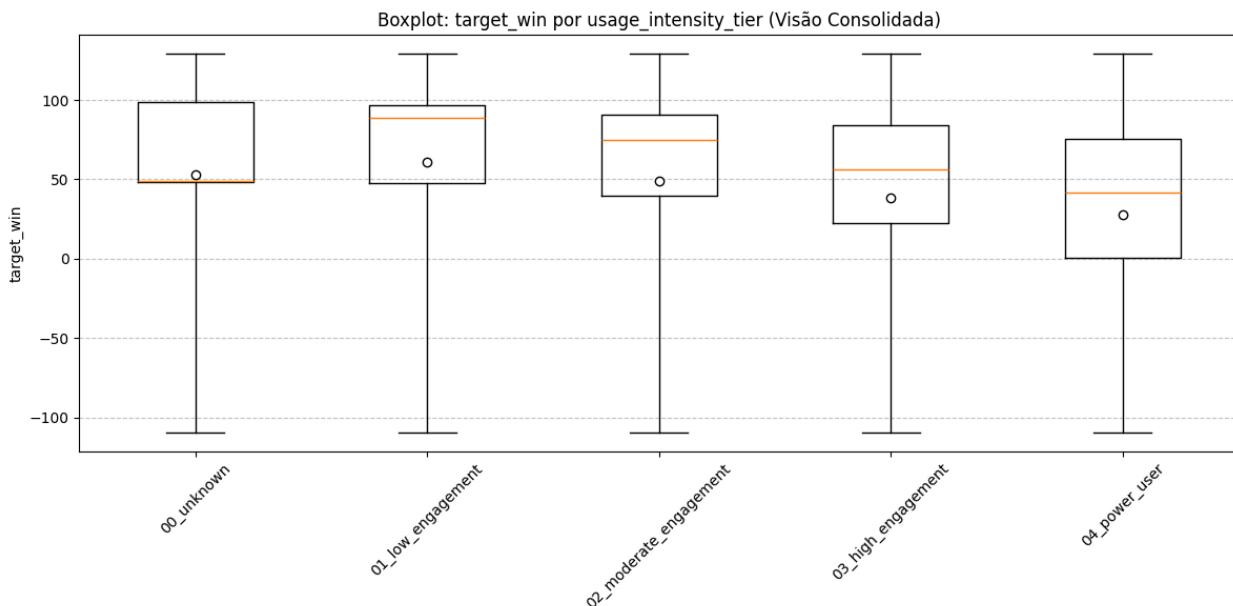
```
In [59]: calcular_distribuicao(df_teste, "usage_intensity_tier")
```

+-----+-----+-----+	+-----+-----+-----+	+-----+
usage_intensity_tier	total	pct_total
+-----+-----+-----+	+-----+-----+-----+	+-----+
04_power_user	2392310 21.28	
03_high_engagement	2386559 21.23	
01_low_engagement	2384790 21.21	
02_moderate_engagement	2384725 21.21	
00_unknown	1694481 15.07	
+-----+-----+-----+	+-----+-----+-----+	+-----+

```
Out[59]: DataFrame[usage_intensity_tier: string, total: bigint, pct_total: double]
```

```
In [61]: plot_boxplot(df_teste, ["usage_intensity_tier"], "target_win", agrupar_por_safra=False, table=True)
```

Processando estatísticas para: usage_intensity_tier...



--- Estatísticas: usage_intensity_tier (Visão Consolidada) ---

	usage_intensity_tier	min	q1	med	mean	q3	max
1	00_unknown	-109.315469	48.190266	49.000000	52.748234	98.881314	129.051366
0	01_low_engagement	-109.315469	47.757770	88.952823	60.821331	96.750829	129.051366
4	02_moderate_engagement	-109.315469	39.784859	74.811358	49.308118	90.506625	129.051366
3	03_high_engagement	-109.315469	22.444884	56.542948	38.381403	84.055290	129.051366
2	04_power_user	-109.315469	0.597280	41.908183	28.040117	75.818896	129.051366

=====

A variável `usage_intensity_tier` foi construída a partir da normalização do volume médio de plays pelo tempo de vida do cliente, sendo posteriormente discretizada em faixas de engajamento. A análise revelou uma relação monotônica clara entre intensidade de uso e margem líquida: usuários com menor engajamento apresentaram as maiores medianas de margem, enquanto usuários classificados como power users apresentaram margens significativamente menores.

Essa evidência confirma o racional de custo operacional do negócio, onde maior intensidade de consumo implica maior custo variável. A discretização em tiers permite capturar essa relação não-linear de forma eficiente em modelos lineares, sendo mantida como uma das principais features do modelo.

10.5.4. Conclusão

```
In [168]: # Winsorizar pra cima, no p995 - nao usar valor bruto
p995_int_per_ten = df_final_vars.approxQuantile("usage_intensity_per_tenure", [0.995], 0.001)[0]
df_final_vars = df_final_vars.withColumn("usage_intensity_per_tenure_cap",
F.when(F.col("usage_intensity_per_tenure") > p995_int_per_ten,
p995_int_per_ten).otherwise(F.col("usage_intensity_per_tenure")))

# Tratar os casos de nulos
df_final_vars = df_final_vars.withColumn("usage_intensity_per_tenure_cap",
F.when(F.col("usage_intensity_per_tenure_cap").isNull(), 0.0).otherwise(F.col("usage_intensity_per_tenure_cap")))
```

```
In [169]: percentiles_usage_intensity = df_final_vars.stat.approxQuantile("usage_intensity_per_tenure", [0.25, 0.5, 0.75], 0.001)
p25_ui, p50_ui, p75_ui = percentiles_usage_intensity[0], percentiles_usage_intensity[1], percentiles_usage_intensity[2]

df_final_vars = df_final_vars.withColumn("usage_intensity_tier",
F.when(F.col("usage_intensity_per_tenure").isNull(), "00_unknown")
.when(F.col("usage_intensity_per_tenure") <= p25_ui, "01_low_engagement")
.when(F.col("usage_intensity_per_tenure") <= p50_ui, "02_moderate_engagement")
.when(F.col("usage_intensity_per_tenure") <= p75_ui, "03_high_engagement")
.otherwise("04_power_user"))

df_final_vars = df_final_vars.drop("usage_intensity_per_tenure")
```

10.6. payment_engagement_gap

10.6.1. Definição

Gap de Adimplência vs. Engajamento. Mede a coerência entre pagar e usar através da diferença (não ratio) entre:

- `flag_has_transactions_mean_6` : Taxa de meses com pagamento nos últimos 6 meses (0 a 1);
- `flag_has_logs_mean_6` : Taxa de meses com atividade nos últimos 6 meses (0 a 1).

```
$$
\text{payment\_engagement\_gap} = \text{flag\_has\_transactions\_mean\_6} - \text{flag\_has\_logs\_mean\_6}
$$
```

Por que diferença e não ratio?

- Ambas as variáveis estão em [0,1], então a diferença fica em [-1, 1], bem comportada.
- Ratio tipo `transactions/logs` explode quando `logs ≈ 0` e perde interpretabilidade.
- A diferença tem significado direto e simétrico.

Interpretação de Valores

Valor Perfil Impacto na Margem
----- ----- -----
Positivo (+0.5 a +1.0) Assinante Fantasma: paga mas não usa ● Margem Líquida Máxima (receita sem custo)
Zero (~0) Usuário Coerente: paga quando usa ○ Margem Saudável
Negativo (-0.5 a -1.0) Inadimplente Ativo: usa mas não paga ● Prejuízo Operacional (custo sem receita)

10.6.2. Construção

```
In [92]: df_final_vars = df_final_vars.withColumn("payment_engagement_gap",
    # Garantir que é proporção válida
    F.when((F.col("flag_has_transactions_mean_6") >= 0) & (F.col("flag_has_transactions_mean_6") <= 1) &
        (F.col("flag_has_logs_mean_6") >= 0) & (F.col("flag_has_logs_mean_6") <= 1),
    F.col("flag_has_transactions_mean_6") - F.col("flag_has_logs_mean_6")).otherwise(None))
```

10.6.3. _Fast EDA_

Descritivas + Correlação

```
In [24]: df_final_vars.select("payment_engagement_gap").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

```
+-----+-----+
|summary|payment_engagement_gap|
+-----+-----+
|  count|      8242914|
|  mean| -0.01359031931345291|
| stddev|  0.11419972603654696|
|  min| -0.8333333333333334|
|   1%|      -0.5|
|   5%| -0.1666666666666666|
|  25%|      0.0|
|  50%|      0.0|
|  75%|      0.0|
|  95%|      0.0|
| 99.5%|      0.5|
|  max|      0.8333333333333334|
+-----+-----+
```

A variável “degenerou” em uma quase-constante. Com isso, entende-se que:

- * `flag_has_transactions_mean_6 ≈ flag_has_logs_mean_6`; ou
- * as duas flags são 0 no mesmo conjunto de meses, então a diferença dá 0 - que é o mais provável.

```
In [25]: df_final_vars.select("payment_engagement_gap", "target").corr("payment_engagement_gap", "target")
```

```
Out[25]: 0.032055136576022276
```

```
In [26]: df_final_vars.select("payment_engagement_gap", "target_win").corr("payment_engagement_gap", "target_win")
```

```
Out[26]: 0.05645751160251486
```

Sinal até existe, mas é fraco. Talvez faça mais sentido trabalhar com flags que separam _ghost payers_ (paga com frequência, mas quase não usa) de _freeloaders_ (usa com frequência, mas quase não paga).

Flag

```
In [27]: tx = F.col("flag_has_transactions_mean_6")
lg = F.col("flag_has_logs_mean_6")

valid = (tx.between(0, 1)) & (lg.between(0, 1))
```

```
In [ ]: df_teste = (df_final_vars
    .withColumn("flag_ghost_payer_6m", F.when(valid & (tx >= 0.5) & (lg <= 0.1666666667), 1).otherwise(0))
    .withColumn("flag_freeloader_6m", F.when(valid & (lg >= 0.5) & (tx <= 0.1666666667), 1).otherwise(0))
)
```

```
In [ ]: calcular_distribuicao(df_teste, ["flag_ghost_payer_6m"])
```

	total	pct_total
0	11220683	99.8
1	22182	0.2

```
DataFrame[flag_ghost_payer_6m: int, total: bigint, pct_total: double]
```

```
In [30]: calcular_distribuicao(df_teste, ["flag_freeloader_6m"])
```

	total	pct_total
0	11203945	99.65
1	38920	0.35

```
Out[30]: DataFrame[flag_freeloader_6m: int, total: bigint, pct_total: double]
```

10.6.4. Conclusão

Alem da raw, foram avaliadas flags destinadas a identificar perfis extremos de comportamento — usuários que utilizam o serviço sem pagar (freeloaders) e usuários que pagam sem utilizar (ghost payers). A análise revelou que tais perfis representam menos de 0.5% da base total, resultando em variáveis altamente esparsas e com forte _zero-inflation_.

Devido ao baixíssimo suporte estatístico, essas variáveis apresentam impacto global irrelevante na performance do modelo, além de elevado risco de instabilidade temporal e overfitting. Adicionalmente, o racional econômico subjacente a esses perfis já é capturado de forma mais robusta por variáveis contínuas e categóricas de engajamento e intensidade de uso.

Desconsiderar.

```
In [93]: df_final_vars = df_final_vars.drop("payment_engagement_gap")
```

10.7. revenue_volatility_6m

10.7.1 Definição

Volatilidade de Receita. Captura a instabilidade financeira do cliente através da amplitude relativa da receita diária nos últimos 6 meses.

```
$$
\text{revenue\_volatility\_6m} = \frac{\text{daily\_revenue\_efficiency\_max\_6} - \text{daily\_revenue\_efficiency\_min\_6}}{\text{daily\_revenue\_efficiency\_mean\_6} + 1}
$$
```

- **Numerador:** Amplitude (max - min) da receita diária.
- **Denominador:** Receita média + 1 (para evitar divisão por zero e normalizar).

Por que isso importa?

- Alta volatilidade indica mudanças frequentes de plano (_downgrades/_upgrades_) ou alternância entre pagamento e isenção.
- Clientes voláteis têm maior probabilidade de churn e margem imprevisível.

Interpretação de Valores

Valor	Perfil	Impacto na Margem
----- ----- -----		
Alto (> 1.0)	Instável: alterna entre planos ou isenções	🔴 Margem Imprevisível, Alto Risco de _Churn_
Médio (0.3-1.0)	Moderadamente Estável	🟡 Risco Moderado
Baixo (<0.3)	Estável: mantém o mesmo plano	🟢 Margem Previsível, Baixo Risco

10.7.2. Construção

```
In [94]: df_final_vars = df_final_vars.withColumn("revenue_volatility_6m",
    F.when((F.col("daily_revenue_efficiency_max_6") >= 0) & (F.col("daily_revenue_efficiency_min_6") >= 0) &
(F.col("daily_revenue_efficiency_mean_6") >= 0),
        (F.col("daily_revenue_efficiency_max_6") - F.col("daily_revenue_efficiency_min_6")) /
(F.col("daily_revenue_efficiency_mean_6") + 1))
.otherwise(None))
```

10.7.3. _Fast EDA_

```
#### Descriptivas + Correlação
```

```
In [54]: df_final_vars.select("revenue_volatility_6m").summary("count", "mean", "stddev", "min", "1%", "5%", "25%", "50%", "75%", "95%", "99.5%", "max").show()
```

+-----+-----+-----+	
summary revenue_volatility_6m	
-----+-----+-----	
count 8623297	
mean 0.13452032837706035	
stddev 0.5937154978488456	
min 0.0	
1% 0.0	
5% 0.0	
25% 0.0	
50% 0.0	
75% 0.0	
95% 0.9664864864864864	
99.5% 4.805560704355885	
max 5.936254980079681	
+-----+-----+-----+	

Pelo menos 75% da base tem volatilidade = 0, indicando que `daily_revenue_efficiency_max_6` = `daily_revenue_efficiency_min_6` (receita constante nos 6 meses). A variável só "acende" para os 5% superiores

```
In [55]: df_final_vars.select("revenue_volatility_6m", "target").corr("revenue_volatility_6m", "target")
```

```
Out[55]: 0.0600087999847909
```

```
In [56]: df_final_vars.select("revenue_volatility_6m", "target_win").corr("revenue_volatility_6m", "target_win")
```

```
Out[56]: 0.08424592298389764
```

10.7.4. Conclusao

Embora a variável apresente correlação positiva com a target, indicando que volatilidade pode capturar upgrades de plano, a magnitude do efeito é insuficiente para justificar sua inclusão. Adicionalmente, o racional de instabilidade financeira já é capturado de forma mais robusta por variáveis de isenção (flag_exemption_mean_6) e eficiência de receita (revenue_per_hour_tier). Descartar.

```
In [95]: df_final_vars = df_final_vars.drop("revenue_volatility_6m")
```

10.8. Salvando base com todos os _books_ + _cross-domain features_

```
In [170]: df_final_vars.printSchema()

root
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- num_25: double (nullable = true)
|-- num_50: double (nullable = true)
|-- num_75: double (nullable = true)
|-- num_985: double (nullable = true)
|-- num_100: double (nullable = true)
|-- num_unq: double (nullable = true)
|-- total_secs: double (nullable = true)
|-- total_plays: double (nullable = true)
|-- flag_has_logs: integer (nullable = true)
|-- total_plays_group: string (nullable = true)
|-- log_total_plays: double (nullable = true)
|-- completed_songs_rate: double (nullable = true)
|-- completed_songs_rate_group: string (nullable = true)
|-- avg_secs_per_unq: double (nullable = true)
|-- avg_secs_per_unq_cap: double (nullable = true)
|-- avg_secs_per_unq_cap_group: string (nullable = true)
|-- log_avg_secs_per_unq: double (nullable = true)
|-- plays_per_unq: double (nullable = true)
|-- plays_per_unq_cap: double (nullable = true)
|-- plays_per_unq_behavior: string (nullable = true)
|-- plays_behavior_vs_volume: string (nullable = true)
|-- plays_behavior_vs_volume_collapsed: string (nullable = true)
|-- plays_behavior_vs_completion: string (nullable = true)
|-- plays_behavior_vs_completion_collapsed: string (nullable = true)
|-- catalog_exploration_ratio: double (nullable = true)
|-- catalog_exploration_ratio_cap: double (nullable = true)
|-- early_drop_rate: double (nullable = true)
|-- early_drop_rate_group: string (nullable = true)
|-- completion_efficiency: double (nullable = true)
|-- flag_shallow_user: integer (nullable = true)
|-- log_total_secs: double (nullable = true)
|-- num_unq_raw: double (nullable = true)
|-- num_unq_mean_3: double (nullable = true)
|-- num_unq_min_3: double (nullable = true)
|-- num_unq_max_3: double (nullable = true)
|-- num_unq_ratio_ref_mean_3: double (nullable = true)
|-- num_unq_ratio_ref_min_3: double (nullable = true)
|-- num_unq_ratio_ref_max_3: double (nullable = true)
|-- num_unq_mean_6: double (nullable = true)
|-- num_unq_min_6: double (nullable = true)
|-- num_unq_max_6: double (nullable = true)
|-- num_unq_ratio_ref_mean_6: double (nullable = true)
|-- num_unq_ratio_ref_min_6: double (nullable = true)
|-- num_unq_ratio_ref_max_6: double (nullable = true)
|-- total_secs_raw: double (nullable = true)
|-- total_secs_mean_3: double (nullable = true)
|-- total_secs_min_3: double (nullable = true)
|-- total_secs_max_3: double (nullable = true)
|-- total_secs_ratio_ref_mean_3: double (nullable = true)
|-- total_secs_ratio_ref_min_3: double (nullable = true)
|-- total_secs_ratio_ref_max_3: double (nullable = true)
|-- total_secs_mean_6: double (nullable = true)
|-- total_secs_min_6: double (nullable = true)
|-- total_secs_max_6: double (nullable = true)
|-- total_secs_ratio_ref_mean_6: double (nullable = true)
|-- total_secs_ratio_ref_min_6: double (nullable = true)
|-- total_secs_ratio_ref_max_6: double (nullable = true)
|-- total_plays_raw: double (nullable = true)
|-- total_plays_mean_3: double (nullable = true)
|-- total_plays_min_3: double (nullable = true)
|-- total_plays_max_3: double (nullable = true)
|-- total_plays_ratio_ref_mean_3: double (nullable = true)
|-- total_plays_ratio_ref_min_3: double (nullable = true)
|-- total_plays_ratio_ref_max_3: double (nullable = true)
|-- total_plays_mean_6: double (nullable = true)
|-- total_plays_min_6: double (nullable = true)
|-- total_plays_max_6: double (nullable = true)
|-- total_plays_ratio_ref_mean_6: double (nullable = true)
|-- total_plays_ratio_ref_min_6: double (nullable = true)
|-- total_plays_ratio_ref_max_6: double (nullable = true)
|-- log_total_secs_raw: double (nullable = true)
|-- log_total_secs_mean_3: double (nullable = true)
|-- log_total_secs_min_3: double (nullable = true)
|-- log_total_secs_max_3: double (nullable = true)
|-- log_total_secs_ratio_ref_mean_3: double (nullable = true)
|-- log_total_secs_ratio_ref_min_3: double (nullable = true)
```



```
|-- early_drop_rate_max_3: double (nullable = true)
|-- early_drop_rate_ratio_ref_mean_3: double (nullable = true)
|-- early_drop_rate_ratio_ref_min_3: double (nullable = true)
|-- early_drop_rate_ratio_ref_max_3: double (nullable = true)
|-- early_drop_rate_mean_6: double (nullable = true)
|-- early_drop_rate_min_6: double (nullable = true)
|-- early_drop_rate_max_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_mean_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_min_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_max_6: double (nullable = true)
|-- flag_has_logs_raw: integer (nullable = true)
|-- flag_has_logs_lag_0: integer (nullable = true)
|-- flag_has_logs_lag_1: integer (nullable = true)
|-- flag_has_logs_lag_2: integer (nullable = true)
|-- flag_has_logs_sum_3: integer (nullable = true)
|-- flag_has_logs_mean_3: double (nullable = true)
|-- flag_has_logs_max_3: integer (nullable = true)
|-- flag_has_logs_lag_3: integer (nullable = true)
|-- flag_has_logs_lag_4: integer (nullable = true)
|-- flag_has_logs_lag_5: integer (nullable = true)
|-- flag_has_logs_sum_6: integer (nullable = true)
|-- flag_has_logs_mean_6: double (nullable = true)
|-- flag_has_logs_max_6: integer (nullable = true)
|-- flag_shallow_user_raw: integer (nullable = true)
|-- flag_shallow_user_lag_0: integer (nullable = true)
|-- flag_shallow_user_lag_1: integer (nullable = true)
|-- flag_shallow_user_lag_2: integer (nullable = true)
|-- flag_shallow_user_sum_3: integer (nullable = true)
|-- flag_shallow_user_mean_3: double (nullable = true)
|-- flag_shallow_user_max_3: integer (nullable = true)
|-- flag_shallow_user_lag_3: integer (nullable = true)
|-- flag_shallow_user_lag_4: integer (nullable = true)
|-- flag_shallow_user_lag_5: integer (nullable = true)
|-- flag_shallow_user_sum_6: integer (nullable = true)
|-- flag_shallow_user_mean_6: double (nullable = true)
|-- flag_shallow_user_max_6: integer (nullable = true)
|-- payment_method_id: integer (nullable = true)
|-- payment_plan_days: integer (nullable = true)
|-- plan_list_price: float (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- is_auto_renew: integer (nullable = true)
|-- transaction_date: date (nullable = true)
|-- membership_expire_date: date (nullable = true)
|-- is_cancel: integer (nullable = true)
|-- flag_expire_invalido: integer (nullable = true)
|-- flag_has_transactions: integer (nullable = true)
|-- flag_valid_fee: integer (nullable = true)
|-- flag_exemption: integer (nullable = true)
|-- flag_plano_mensal: integer (nullable = true)
|-- daily_revenue_efficiency: double (nullable = true)
|-- revenue_tier: string (nullable = true)
|-- payment_method_group: string (nullable = true)
|-- payment_price_regime: string (nullable = true)
|-- daily_revenue_efficiency_raw: double (nullable = true)
|-- daily_revenue_efficiency_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_min_3: double (nullable = true)
|-- daily_revenue_efficiency_max_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
|-- daily_revenue_efficiency_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_min_6: double (nullable = true)
|-- daily_revenue_efficiency_max_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_6: double (nullable = true)
|-- flag_has_transactions_raw: integer (nullable = true)
|-- flag_has_transactions_lag_0: integer (nullable = true)
|-- flag_has_transactions_lag_1: integer (nullable = true)
|-- flag_has_transactions_lag_2: integer (nullable = true)
|-- flag_has_transactions_sum_3: integer (nullable = true)
|-- flag_has_transactions_mean_3: double (nullable = true)
|-- flag_has_transactions_max_3: integer (nullable = true)
|-- flag_has_transactions_lag_3: integer (nullable = true)
|-- flag_has_transactions_lag_4: integer (nullable = true)
|-- flag_has_transactions_lag_5: integer (nullable = true)
|-- flag_has_transactions_sum_6: integer (nullable = true)
|-- flag_has_transactions_mean_6: double (nullable = true)
|-- flag_has_transactions_max_6: integer (nullable = true)
|-- flag_valid_fee_raw: integer (nullable = true)
|-- flag_valid_fee_lag_0: integer (nullable = true)
|-- flag_valid_fee_lag_1: integer (nullable = true)
|-- flag_valid_fee_lag_2: integer (nullable = true)
|-- flag_valid_fee_sum_3: integer (nullable = true)
|-- flag_valid_fee_mean_3: double (nullable = true)
|-- flag_valid_fee_max_3: integer (nullable = true)
|-- flag_valid_fee_lag_3: integer (nullable = true)
|-- flag_valid_fee_lag_4: integer (nullable = true)
```

```

|-- flag_valid_fee_lag_5: integer (nullable = true)
|-- flag_valid_fee_sum_6: integer (nullable = true)
|-- flag_valid_fee_mean_6: double (nullable = true)
|-- flag_valid_fee_max_6: integer (nullable = true)
|-- flag_exemption_raw: integer (nullable = true)
|-- flag_exemption_lag_0: integer (nullable = true)
|-- flag_exemption_lag_1: integer (nullable = true)
|-- flag_exemption_lag_2: integer (nullable = true)
|-- flag_exemption_sum_3: integer (nullable = true)
|-- flag_exemption_mean_3: double (nullable = true)
|-- flag_exemption_max_3: integer (nullable = true)
|-- flag_exemption_lag_3: integer (nullable = true)
|-- flag_exemption_lag_4: integer (nullable = true)
|-- flag_exemption_lag_5: integer (nullable = true)
|-- flag_exemption_sum_6: integer (nullable = true)
|-- flag_exemption_mean_6: double (nullable = true)
|-- flag_exemption_max_6: integer (nullable = true)
|-- flag_plano_mensal_raw: integer (nullable = true)
|-- flag_plano_mensal_lag_0: integer (nullable = true)
|-- flag_plano_mensal_lag_1: integer (nullable = true)
|-- flag_plano_mensal_lag_2: integer (nullable = true)
|-- flag_plano_mensal_sum_3: integer (nullable = true)
|-- flag_plano_mensal_mean_3: double (nullable = true)
|-- flag_plano_mensal_max_3: integer (nullable = true)
|-- flag_plano_mensal_lag_3: integer (nullable = true)
|-- flag_plano_mensal_lag_4: integer (nullable = true)
|-- flag_plano_mensal_lag_5: integer (nullable = true)
|-- flag_plano_mensal_sum_6: integer (nullable = true)
|-- flag_plano_mensal_mean_6: double (nullable = true)
|-- flag_plano_mensal_max_6: integer (nullable = true)
|-- flag_expire_invalido_raw: integer (nullable = true)
|-- flag_expire_invalido_lag_0: integer (nullable = true)
|-- flag_expire_invalido_lag_1: integer (nullable = true)
|-- flag_expire_invalido_lag_2: integer (nullable = true)
|-- flag_expire_invalido_sum_3: integer (nullable = true)
|-- flag_expire_invalido_mean_3: double (nullable = true)
|-- flag_expire_invalido_max_3: integer (nullable = true)
|-- flag_expire_invalido_lag_3: integer (nullable = true)
|-- flag_expire_invalido_lag_4: integer (nullable = true)
|-- flag_expire_invalido_lag_5: integer (nullable = true)
|-- flag_expire_invalido_sum_6: integer (nullable = true)
|-- flag_expire_invalido_mean_6: double (nullable = true)
|-- flag_expire_invalido_max_6: integer (nullable = true)
|-- registration_init_time: date (nullable = true)
|-- city: integer (nullable = true)
|-- bd: integer (nullable = true)
|-- registered_via: integer (nullable = true)
|-- is_ativo: integer (nullable = true)
|-- flag_idade_in valida: integer (nullable = true)
|-- idade_clean: integer (nullable = true)
|-- gender_clean: string (nullable = true)
|-- faixa_idade: string (nullable = true)
|-- flag_city_one: integer (nullable = true)
|-- flag_gender_known: integer (nullable = true)
|-- flag_high_value_registered_via: integer (nullable = true)
|-- registered_via_group: string (nullable = true)
|-- registration_year_regime: string (nullable = true)
|-- tenure_meses: long (nullable = true)
|-- flag_long_tenure: integer (nullable = true)
|-- tenure_faixa: string (nullable = true)
|-- margem_liquida_mensal: double (nullable = true)
|-- target: double (nullable = true)
|-- target_win: double (nullable = true)
|-- revenue_per_hour_listened_cap: double (nullable = true)
|-- revenue_per_hour_tier: string (nullable = false)
|-- usage_intensity_per_tenure_cap: double (nullable = true)
|-- usage_intensity_tier: string (nullable = false)

```

In [176]: df_final_vars.filter(F.col("margem_liquida_mensal").isNull()).count()

Out[176]: 0

In [177]: df_final_vars.filter(F.col("target").isNull()).count()

Out[177]: 1563999

In [173]: df_final_vars.filter(F.col("target_win").isNull()).count()

Out[173]: 1563999

```
In [175]: df_final_vars.filter(F.col("target_win").isNull()).groupBy("safra").count().show(5)
```

```
+-----+-----+
| safra| count|
+-----+-----+
|201605| 34322|
|201610|121953|
|201606| 26707|
|201611|146985|
|201607| 44169|
+-----+-----+
only showing top 5 rows
```

```
In [178]: # Se o número de nulos for próximo ao número de msno únicos, o diagnóstico está correto
print(f"Total de nulos no target: {df_final_vars.filter(F.col('target').isNull()).count()}")
print(f"Total de clientes únicos: {df_final_vars.select('msno').distinct().count()}")
```

```
Total de nulos no target: 1563999
Total de clientes únicos: 1563999
```

```
In [97]: # 1. cache
df_final_vars = df_final_vars.persist()
df_final_vars.count()

# 2. salvar particionado
df_final_vars.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_feature_store_intermediate")
```

11. Estratégia de Modelagem

11.1. Definição da Variável Target

A escolha da **variável target** é uma decisão estrutural que impacta diretamente a capacidade de generalização dos modelos. Neste projeto, foram geradas duas versões da margem líquida:

- target (raw): Margem líquida sem tratamento de outliers
- target_win (winsorizada): Margem líquida com winsorização nos percentis 1% e 99%

Análise Comparativa

Critério Target Raw Target Winsorizada
----- ----- -----
Outliers extremos Preserva valores extremos (positivos e negativos) Limita valores aos percentis 1% e 99%
Risco de overfitting Alto — modelos podem ajustar regras específicas para casos raros Reduzido — distribuição mais estável
Interpretabilidade Reflete a realidade "crua" do negócio Reflete a realidade "típica" do negócio
Estabilidade temporal Vulnerável a eventos pontuais (promoções, bugs, fraudes) Mais robusta a choques temporários
Performance em produção Pode degradar em períodos atípicos Tende a manter performance mais consistente

Decisão Técnica: utilizaremos `target_win` (winsorizada) como variável target principal.

Justificativa:

1. Robustez estatística: A winsorização reduz a influência de outliers extremos sem descartá-los completamente, preservando informação enquanto estabiliza a distribuição.
2. Generalização: Modelos treinados com target winsorizada tendem a aprender padrões mais generalizáveis, evitando ajustes excessivos a casos raros que podem não se repetir em produção.
3. Estabilidade em produção: Em ambientes de produção, é preferível um modelo que erra menos na média do que um modelo que acerta perfeitamente casos extremos mas falha na maioria dos casos típicos.
4. Alinhamento com o negócio: A margem winsorizada representa melhor o comportamento "esperado" da base de clientes, sendo mais útil para decisões estratégicas (specificação, segmentação, retenção).
5. Redução de viés induzido por eventos pontuais: Valores extremos podem ser resultado de erros de sistema, fraudes, ou promoções não-recorrentes. A winsorização mitiga o risco de o modelo aprender padrões pouco ou nada frequentes.

11.2. Escolha dos Algoritmos

A fim de equilibrar capacidade preditiva, interpretabilidade, custo computacional e viabilidade de implantação em produção, optou-se por uma abordagem 'multi-modelo', combinando famílias complementares, como mencionado anteriormente no projeto:

11.2.1. Elastic Net (Regressão Linear Regularizada)

Elastic Net combina as penalizações L1 (Lasso) e L2 (Ridge), oferecendo:

- Seleção automática de features (via L1): Zera coeficientes de variáveis irrelevantes
- Estabilidade numérica (via L2): Lida melhor com multicolinearidade que Lasso puro
- Interpretabilidade máxima: Coeficientes lineares permitem explicar o impacto marginal de cada feature
- Baixíssimo custo computacional: Treinamento e inferência extremamente rápidos
- Baseline robusto: Serve como referência para avaliar se modelos complexos realmente agregam valor

Por que Elastic Net?

Mesmo que a relação não seja linear, um modelo linear regularizado ($L_1 + L_2$) serve como um baseline de baixa variância. Ele captura a tendência global (o "viés principal") e impede que os modelos não-lineares fiquem "loucos" com ruídos locais. Ele é o "seguro" contra o overfit.

Comparação com Alternativas Lineares

Modelo Vantagens Desvantagens Por que não escolher?
----- ----- ----- -----
Regressão Linear Simples (OLS) Máxima interpretabilidade Sem regularização, vulnerável a overfitting e multicolinearidade Não lida bem com alta dimensionalidade
Ridge (L2) Estabiliza coeficientes Não faz seleção de features (mantém todas) Elastic Net oferece o melhor dos dois mundos
Lasso (L1) Seleção de features Instável com features correlacionadas Elastic Net é mais robusto
Elastic Net Seleção + Estabilidade Assume relações lineares Escolhido

Trade-off Bias-Variance + Interpretabilidade

- * Viés: Moderado/alto. Assume linearidade, "simplificando" demais o problema;
- * Variância: Baixa. Regularização controla complexidade;
- * Interpretabilidade: Máxima. Coeficientes diretos (β) indicam o impacto de cada variável.

11.2.2. LightGBM (Gradient Boosting com Otimizações)

Por que LightGBM (e não XGBoost ou CatBoost)? LightGBM é uma implementação moderna de Gradient Boosting otimizada para eficiência e escalabilidade.

Comparação Detalhada: LightGBM vs. XGBoost vs. CatBoost

Critério LightGBM XGBoost CatBoost Decisão
----- ----- ----- ----- -----
Velocidade de treinamento (5) Crescimento leaf-wise e amostragem GOSS reduzem custo por nó. (3) Crescimento level-wise exige mais divisões por nível. (2) O processamento de Ordered Boosting é computacionalmente caro LightGBM
Uso de memória (5) Algoritmo baseado em histogramas otimiza o uso de RAM. (3) Tradicionalmente exige estruturas de dados (DMatrix) mais pesadas. (3) Árvores simétricas e cache de categorias consomem recursos consideráveis. LightGBM
Performance preditiva (5) Estado da arte; excelente em capturar nuances em bases grandes. (5) Extremamente robusto e consistente em diversos domínios. (5) Frequentemente superior em bases com muitas variáveis categóricas. Empate técnico
Tratamento de categóricas (4) Agrupa categorias por valores, mas exige conversão prévia para int . (4) Apesar de melhora recente, ainda depende muito de pré-processamento. (5) Otimização nativa via Permutation Encoding (padrão ouro). CatBoost
Overfitting em bases pequenas (2) O crescimento folha a folha é muito agressivo para poucos dados. (5) O crescimento por nível e controle de profundidade agem como regularizadores. (5) Árvores simétricas evitam que o modelo "decore" ruídos locais. XGBoost/CatBoost
Maturidade e adoção (5) Padrão da indústria (5) O mais antigo e testado em ambientes de missão crítica. (3) Mais recente; comunidade e bibliotecas de apoio ainda em expansão. LightGBM/XGBoost
Facilidade de deploy (5) Integração nativa com MLflow, por exemplo. (4) Robusto, mas o ecossistema de serialização é levemente mais burocrático. (3) Menos ferramentas de terceiros suportam seus formatos nativos. LightGBM
Suporte a GPU (5) Excelente escalabilidade paralela. (5) Muito eficiente, especialmente em histogramas. (5) Projetado para brilhar em GPUs (treina muito mais rápido nelas). Empate
Documentação e comunidade (5) Vasta literatura e soluções de erros online. (5) Referência absoluta em fóruns técnicos. (4) Boa documentação, mas menos "receitas de bolo" disponíveis. LightGBM/XGBoost

Por que LightGBM? Justificativa:

1. Eficiência computacional superior: base com milhões de registros (~11M), LightGBM se mostra significativamente mais rápido que XGBoost e CatBoost, permitindo iterações mais rápidas de experimentação;
2. Menor consumo de memória: utiliza histogramas para discretizar features contínuas, reduzindo drasticamente o uso de RAM;
3. Arquitetura Leaf-wise: enquanto o XGBoost cresce nível por nível, o LightGBM foca nos nós que mais reduzem o erro (perda). Isso o torna performático em bases grandes, pois chega no "ponto ideal" mais rápido, resultando em modelos mais profundos com menos árvores;
4. Flexibilidade: feature importance nativo facilita mapear gain/split.

Por que não XGBoost?

1. XGBoost é excelente, mas mais lento e mais pesado em memória para bases grandes;
2. A diferença de performance preditiva entre LightGBM e XGBoost **bem-tunados** é marginal;
3. LightGBM oferece melhor custo-benefício para experimentação rápida.

Por que não CatBoost?

Apesar de melhor tratamento nativo de categóricas:

1. Muito mais lento para treinar (especialmente sem GPU);
2. Base já com encoding adequado das categóricas'
3. O ganho marginal não justifica o custo computacional adicional.

Trade-off Bias-Variance + Interpretabilidade

* Viés: Baixo. Modelos complexos capturam não-linearidades;

* Variância: Moderada/alta. Boosting sequencial reduz variância. Se mostra muito preciso, mas propenso a overfitting.;

* Interpretabilidade: Média. Requer ferramentas como SHAP values, feature importance, não sendo possível analisar coeficientes diretos.

11.2.3. Random Forest Regressor

Random Forest vs. Quantile Regression Forest (QRF)

Critério RFR QRF Justificativa (pensando nos +10M de registros)
----- ----- ----- -----
Objetivo do Modelo 5 5 O RFR foca na média e o QRF em quantis. Após a winsorização (1%/99%), a média torna-se um estimador estável para a margem líquida.
Robustez a Outliers 3 5 O QRF é nativamente robusto. No RFR, essa robustez foi "transferida" para o pré-processamento (winsorização), nivelando o jogo.
Incerteza Preditiva 1 5 O QRF fornece intervalos (P10-P90) nativos. O RFR entrega apenas a estimativa pontual, exigindo métodos externos para incerteza.
Velocidade de Treino 5 2 O RFR calcula apenas a média por folha. O QRF precisa ordenar e armazenar todas as observações em cada folha, gerando alto overhead.
Velocidade de Inferência 5 1 No RFR, a inferência é um acesso direto ao valor da folha. No QRF, o modelo precisa calcular quantis sobre a massa de dados em tempo real.
Uso de Memória (RAM) 5 1 O QRF armazena as labels de treino nas folhas. Com +10M de linhas, o risco de Out-of-Memory é crítico no QRF.
Interpretabilidade 4 4 Ambos permitem o uso de SHAP e Feature Importance de forma idêntica, sendo transparentes para auditorias.
Maturidade em Prod. 5 2 O RFR é padrão ouro no Scikit-Learn e Spark. O QRF depende de bibliotecas menos estáveis e com menor suporte da comunidade.
Facilidade de Deploy 5 2 Modelos RFR são leves (serialização simples). Arquivos de modelos QRF são massivos e de difícil transporte em pipelines de CI/CD.

A escolha inicial pelo Quantile Regression Forest (QRF) fundamentava-se na natureza estatística bruta da variável alvo (Margem Líquida M+1). Como visto, a target "raw" apresenta assimetria severa (skewed distribution), com caudas longas causadas por eventos extremos (imagino que sejam picos de custos de licenciamento ou usuários com consumo fora da curva, considerando o negocio de uma plataforma de streaming de musicas).

Sem tratamento prévio, o Random Forest Regressor tradicional falharia em representar o "cliente típico". Por minimizar o Erro Quadrático Médio (MSE), o RFR é "sequestrado" pelos outliers, pois o erro ao quadrado penaliza excessivamente os valores extremos, puxando a previsão em direção à média da cauda. O QRF, ao focar na mediana condicional, ignoraria a magnitude desses extremos, oferecendo uma previsão mais estável e centralizada.

O Cenário Pós-Winsorização: Por que o RFR vence?

Justificativa:

1. Alinhamento com o target winsorizado: com outliers via winsorização, a vantagem de robustez do QRF é parcialmente neutralizada;
2. Simplicidade operacional: Random Forest tem implantação simples em qualquer stack de produção (sklearn, ONNX, etc.), enquanto QRF requer bibliotecas específicas (ex: quantile-forest, scikit-garden);
3. Custo computacional em produção: QRF é significativamente mais lento, pois precisa calcular distribuições empíricas em cada folha. Em um sistema de scoring em lote (milhões de clientes), isso pode ser prejudicial;
4. Maturidade e suporte: Random Forest tem décadas de uso em produção, documentação extensa, e integração nativa com ferramentas de MLOps.

Quando QRF seria preferível?

- * Se o objetivo fosse gestão de risco (ex: "qual a margem no pior cenário P10?");
- * Se houvesse requisito explícito de quantificar incerteza (estudo será feito com modelos de agrupamento, no lugar);
- * Se a distribuição da target fosse altamente assimétrica mesmo após winsorização.

Em resumo: O QRF era a solução para um problema de dados brutos e ruidosos; o RFR é a solução otimizada para dados refinados e prontos para produção.

Trade-off Bias-Variance + Interpretabilidade

- * Viés: Moderado. Árvores profundas capturam não-linearidades, mas média de árvores suaviza;
- * Variância: Baixa. Bagging reduz drasticamente variância;
- * Interpretabilidade: Média (feature importance, partial dependence plots).

11.3. _Ensemble Learning_ e Arquitetura Multi-Modelo

11.3.1 Definição

Ensemble Learning (Aprendizado em Conjunto) é a estratégia de Machine Learning que combina múltiplos modelos preditivos para produzir um resultado superior ao que seria obtido por qualquer modelo individual isoladamente. A premissa fundamental é que a sabedoria coletiva de modelos diversos supera a capacidade de um único modelo especializado. A teoria estatística por trás dos ensembles baseia-se em três princípios:

11.3.1.1. Decomposição do Erro: Bias-Variance Trade-off

O erro esperado de um modelo pode ser decomposto em três componentes:

$$\text{Erro Total} = \text{Viés}^2 + \text{Variância} + \text{Ruído Irreduzível}$$

- * Viés: Erro sistemático causado por suposições simplificadoras do modelo (ex: assumir linearidade quando a relação é não-linear);
- * Variância: Sensibilidade do modelo a flutuações nos dados de treino (overfitting);
- * Ruído Irreduzível: Aleatoriedade inerente aos dados que nenhum modelo pode capturar. Erro impossível de ser mitigado.

Ensembles exploram esse trade-off de duas formas:

1. Redução de Variância (Bagging): Treinar múltiplos modelos em subconjuntos diferentes dos dados e agregar suas previsões reduz a variância sem aumentar o viés. Exemplo: Random Forest;
2. Redução de Viés (Boosting): Treinar modelos sequencialmente, onde cada novo modelo corrige os erros do anterior, reduz o viés mantendo a variância controlada. Exemplo: LightGBM.

11.3.1.2. Diversidade de Modelos

Para que um ensemble seja efetivo, os modelos componentes devem ser diversos — ou seja, devem cometer erros diferentes. Se todos os modelos erram da mesma forma, agregá-los não traz benefício. Fontes de diversidade:

- * Algoritmos diferentes: Modelos lineares vs. árvores vs. redes neurais
- * Representações diferentes: Features brutas vs. features engenheiradas vs. embeddings
- * Dados diferentes: Bootstrap samples, cross-validation folds, janelas temporais
- * Hiperparâmetros diferentes: Profundidade de árvores, regularização, learning rate

11.3.1.3. Teorema do "Wisdom of Crowds"

Se tivermos N modelos independentes, cada um com acurácia $p > 0.5$, a acurácia do ensemble (votação por maioria) aumenta exponencialmente com N . Mesmo que modelos individuais sejam fracos, sua combinação pode ser forte.

11.3.2. Primeira abordagem escolhida: Ensemble Heterogêneo Multi-Nível

A estratégia planejada a princípio para este projeto não seria um ensemble tradicional (como Random Forest ou Gradient Boosting isoladamente), mas sim um meta-ensemble heterogêneo, combinando:

- * Modelos de famílias algorítmicas distintas (linear, boosting, bagging);
- * Representações de features customizadas por modelo (OHE para Elastic Net, TEs para LightGBM e RFR);
- * Objetivos complementares (interpretabilidade vs. performance vs. robustez).

11.3.3 Decisão final

Apesar da primeira abordagem proposta se mostrar robusta, dado que cada modelo traria uma "perspectiva" diferente sobre a predição "x" e isso permitiria que se escolhesse uma média ponderada dos três outputs dos modelos para uma predição final mais precisa, a solução se mostra complexa de ser implantada em produção.

Considerar o modelo final como o meta-modelo de outros n algoritmos implica que, em estudos de monitoramento, por exemplo, o cientista responsável pelo processo analise os n algoritmos e a forma como cada um deles individualmente está performando. Além disso, as bases finais de execução dos modelos não contemplam as mesmas variáveis (no caso de EN vs. LGBM e RF, que são baseados em árvores) e nível de tratamento destas variáveis, caso em comum entre si (nulos sem tratamento para LGBM, tratados para RFR, mesmo sendo as mesmas variáveis).

Se as métricas consideradas como decisivas pra escolha de um único modelo final forem muito próximas, faz muito mais sentido escolher um modelo finalista dentre esses três do que implementar uma solução complexa para unir todos.

12. _Feature Selection_ - ABTs p/ Algoritmo Supervisionado - Camada _Gold_

12.1. Carregando base e separando tipos de variáveis

```
In [4]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver"
df_feature_store_intermediate = spark.read.parquet(f"{silver_path}/df_feature_store_intermediate")
```

In [5]: df_feature_store_intermediate.printSchema()

```
root
 |-- msno: string (nullable = true)
 |-- num_25: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- num_100: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_secs: double (nullable = true)
 |-- total_plays: double (nullable = true)
 |-- flag_has_logs: integer (nullable = true)
 |-- total_plays_group: string (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- completed_songs_rate: double (nullable = true)
 |-- completed_songs_rate_group: string (nullable = true)
 |-- avg_secs_per_unq: double (nullable = true)
 |-- avg_secs_per_unq_cap: double (nullable = true)
 |-- avg_secs_per_unq_cap_group: string (nullable = true)
 |-- log_avg_secs_per_unq: double (nullable = true)
 |-- plays_per_unq: double (nullable = true)
 |-- plays_per_unq_cap: double (nullable = true)
 |-- plays_per_unq_behavior: string (nullable = true)
 |-- plays_behavior_vs_volume: string (nullable = true)
 |-- plays_behavior_vs_volume_collapsed: string (nullable = true)
 |-- plays_behavior_vs_completion: string (nullable = true)
 |-- plays_behavior_vs_completion_collapsed: string (nullable = true)
 |-- catalog_exploration_ratio: double (nullable = true)
 |-- catalog_exploration_ratio_cap: double (nullable = true)
 |-- early_drop_rate: double (nullable = true)
 |-- early_drop_rate_group: string (nullable = true)
 |-- completion_efficiency: double (nullable = true)
 |-- flag_shallow_user: integer (nullable = true)
 |-- log_total_secs: double (nullable = true)
 |-- num_unq_raw: double (nullable = true)
 |-- num_unq_mean_3: double (nullable = true)
 |-- num_unq_min_3: double (nullable = true)
 |-- num_unq_max_3: double (nullable = true)
 |-- num_unq_ratio_ref_mean_3: double (nullable = true)
 |-- num_unq_ratio_ref_min_3: double (nullable = true)
 |-- num_unq_ratio_ref_max_3: double (nullable = true)
 |-- num_unq_mean_6: double (nullable = true)
 |-- num_unq_min_6: double (nullable = true)
 |-- num_unq_max_6: double (nullable = true)
 |-- num_unq_ratio_ref_mean_6: double (nullable = true)
 |-- num_unq_ratio_ref_min_6: double (nullable = true)
 |-- num_unq_ratio_ref_max_6: double (nullable = true)
 |-- total_secs_raw: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- total_secs_min_3: double (nullable = true)
 |-- total_secs_max_3: double (nullable = true)
 |-- total_secs_ratio_ref_mean_3: double (nullable = true)
 |-- total_secs_ratio_ref_min_3: double (nullable = true)
 |-- total_secs_ratio_ref_max_3: double (nullable = true)
 |-- total_secs_mean_6: double (nullable = true)
 |-- total_secs_min_6: double (nullable = true)
 |-- total_secs_max_6: double (nullable = true)
 |-- total_secs_ratio_ref_mean_6: double (nullable = true)
 |-- total_secs_ratio_ref_min_6: double (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- total_plays_raw: double (nullable = true)
 |-- total_plays_mean_3: double (nullable = true)
 |-- total_plays_min_3: double (nullable = true)
 |-- total_plays_max_3: double (nullable = true)
 |-- total_plays_ratio_ref_mean_3: double (nullable = true)
 |-- total_plays_ratio_ref_min_3: double (nullable = true)
 |-- total_plays_ratio_ref_max_3: double (nullable = true)
 |-- total_plays_mean_6: double (nullable = true)
 |-- total_plays_min_6: double (nullable = true)
 |-- total_plays_max_6: double (nullable = true)
 |-- total_plays_ratio_ref_mean_6: double (nullable = true)
 |-- total_plays_ratio_ref_min_6: double (nullable = true)
 |-- total_plays_ratio_ref_max_6: double (nullable = true)
 |-- log_total_secs_raw: double (nullable = true)
 |-- log_total_secs_mean_3: double (nullable = true)
 |-- log_total_secs_min_3: double (nullable = true)
 |-- log_total_secs_max_3: double (nullable = true)
 |-- log_total_secs_ratio_ref_mean_3: double (nullable = true)
 |-- log_total_secs_ratio_ref_min_3: double (nullable = true)
 |-- log_total_secs_ratio_ref_max_3: double (nullable = true)
 |-- log_total_secs_mean_6: double (nullable = true)
 |-- log_total_secs_min_6: double (nullable = true)
 |-- log_total_secs_max_6: double (nullable = true)
 |-- log_total_secs_ratio_ref_mean_6: double (nullable = true)
 |-- log_total_secs_ratio_ref_min_6: double (nullable = true)
 |-- log_total_secs_ratio_ref_max_6: double (nullable = true)
```



```
|-- early_drop_rate_ratio_ref_mean_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_min_6: double (nullable = true)
|-- early_drop_rate_ratio_ref_max_6: double (nullable = true)
|-- flag_has_logs_raw: integer (nullable = true)
|-- flag_has_logs_lag_0: integer (nullable = true)
|-- flag_has_logs_lag_1: integer (nullable = true)
|-- flag_has_logs_lag_2: integer (nullable = true)
|-- flag_has_logs_sum_3: integer (nullable = true)
|-- flag_has_logs_mean_3: double (nullable = true)
|-- flag_has_logs_max_3: integer (nullable = true)
|-- flag_has_logs_lag_3: integer (nullable = true)
|-- flag_has_logs_lag_4: integer (nullable = true)
|-- flag_has_logs_lag_5: integer (nullable = true)
|-- flag_has_logs_sum_6: integer (nullable = true)
|-- flag_has_logs_mean_6: double (nullable = true)
|-- flag_has_logs_max_6: integer (nullable = true)
|-- flag_shallow_user_raw: integer (nullable = true)
|-- flag_shallow_user_lag_0: integer (nullable = true)
|-- flag_shallow_user_lag_1: integer (nullable = true)
|-- flag_shallow_user_lag_2: integer (nullable = true)
|-- flag_shallow_user_sum_3: integer (nullable = true)
|-- flag_shallow_user_mean_3: double (nullable = true)
|-- flag_shallow_user_max_3: integer (nullable = true)
|-- flag_shallow_user_lag_3: integer (nullable = true)
|-- flag_shallow_user_lag_4: integer (nullable = true)
|-- flag_shallow_user_lag_5: integer (nullable = true)
|-- flag_shallow_user_sum_6: integer (nullable = true)
|-- flag_shallow_user_mean_6: double (nullable = true)
|-- flag_shallow_user_max_6: integer (nullable = true)
|-- payment_method_id: integer (nullable = true)
|-- payment_plan_days: integer (nullable = true)
|-- plan_list_price: float (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- is_auto_renew: integer (nullable = true)
|-- transaction_date: date (nullable = true)
|-- membership_expire_date: date (nullable = true)
|-- is_cancel: integer (nullable = true)
|-- flag_expire_invalido: integer (nullable = true)
|-- flag_has_transactions: integer (nullable = true)
|-- flag_valid_fee: integer (nullable = true)
|-- flag_exemption: integer (nullable = true)
|-- flag_plano_mensal: integer (nullable = true)
|-- daily_revenue_efficiency: double (nullable = true)
|-- revenue_tier: string (nullable = true)
|-- payment_method_group: string (nullable = true)
|-- payment_price_regime: string (nullable = true)
|-- daily_revenue_efficiency_raw: double (nullable = true)
|-- daily_revenue_efficiency_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_min_3: double (nullable = true)
|-- daily_revenue_efficiency_max_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
|-- daily_revenue_efficiency_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_min_6: double (nullable = true)
|-- daily_revenue_efficiency_max_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_mean_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_min_6: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_6: double (nullable = true)
|-- flag_has_transactions_raw: integer (nullable = true)
|-- flag_has_transactions_lag_0: integer (nullable = true)
|-- flag_has_transactions_lag_1: integer (nullable = true)
|-- flag_has_transactions_lag_2: integer (nullable = true)
|-- flag_has_transactions_sum_3: integer (nullable = true)
|-- flag_has_transactions_mean_3: double (nullable = true)
|-- flag_has_transactions_max_3: integer (nullable = true)
|-- flag_has_transactions_lag_3: integer (nullable = true)
|-- flag_has_transactions_lag_4: integer (nullable = true)
|-- flag_has_transactions_lag_5: integer (nullable = true)
|-- flag_has_transactions_sum_6: integer (nullable = true)
|-- flag_has_transactions_mean_6: double (nullable = true)
|-- flag_has_transactions_max_6: integer (nullable = true)
|-- flag_valid_fee_raw: integer (nullable = true)
|-- flag_valid_fee_lag_0: integer (nullable = true)
|-- flag_valid_fee_lag_1: integer (nullable = true)
|-- flag_valid_fee_lag_2: integer (nullable = true)
|-- flag_valid_fee_sum_3: integer (nullable = true)
|-- flag_valid_fee_mean_3: double (nullable = true)
|-- flag_valid_fee_max_3: integer (nullable = true)
|-- flag_valid_fee_lag_3: integer (nullable = true)
|-- flag_valid_fee_lag_4: integer (nullable = true)
|-- flag_valid_fee_lag_5: integer (nullable = true)
|-- flag_valid_fee_sum_6: integer (nullable = true)
|-- flag_valid_fee_mean_6: double (nullable = true)
|-- flag_valid_fee_max_6: integer (nullable = true)
|-- flag_exemption_raw: integer (nullable = true)
|-- flag_exemption_lag_0: integer (nullable = true)
|-- flag_exemption_lag_1: integer (nullable = true)
```

```

|-- flag_exemption_lag_2: integer (nullable = true)
|-- flag_exemption_sum_3: integer (nullable = true)
|-- flag_exemption_mean_3: double (nullable = true)
|-- flag_exemption_max_3: integer (nullable = true)
|-- flag_exemption_lag_3: integer (nullable = true)
|-- flag_exemption_lag_4: integer (nullable = true)
|-- flag_exemption_lag_5: integer (nullable = true)
|-- flag_exemption_sum_6: integer (nullable = true)
|-- flag_exemption_mean_6: double (nullable = true)
|-- flag_exemption_max_6: integer (nullable = true)
|-- flag_plano_mensal_raw: integer (nullable = true)
|-- flag_plano_mensal_lag_0: integer (nullable = true)
|-- flag_plano_mensal_lag_1: integer (nullable = true)
|-- flag_plano_mensal_lag_2: integer (nullable = true)
|-- flag_plano_mensal_sum_3: integer (nullable = true)
|-- flag_plano_mensal_mean_3: double (nullable = true)
|-- flag_plano_mensal_max_3: integer (nullable = true)
|-- flag_plano_mensal_lag_3: integer (nullable = true)
|-- flag_plano_mensal_lag_4: integer (nullable = true)
|-- flag_plano_mensal_lag_5: integer (nullable = true)
|-- flag_plano_mensal_sum_6: integer (nullable = true)
|-- flag_plano_mensal_mean_6: double (nullable = true)
|-- flag_plano_mensal_max_6: integer (nullable = true)
|-- flag_expire_invalido_raw: integer (nullable = true)
|-- flag_expire_invalido_lag_0: integer (nullable = true)
|-- flag_expire_invalido_lag_1: integer (nullable = true)
|-- flag_expire_invalido_lag_2: integer (nullable = true)
|-- flag_expire_invalido_sum_3: integer (nullable = true)
|-- flag_expire_invalido_mean_3: double (nullable = true)
|-- flag_expire_invalido_max_3: integer (nullable = true)
|-- flag_expire_invalido_lag_3: integer (nullable = true)
|-- flag_expire_invalido_lag_4: integer (nullable = true)
|-- flag_expire_invalido_lag_5: integer (nullable = true)
|-- flag_expire_invalido_sum_6: integer (nullable = true)
|-- flag_expire_invalido_mean_6: double (nullable = true)
|-- flag_expire_invalido_max_6: integer (nullable = true)
|-- registration_init_time: date (nullable = true)
|-- city: integer (nullable = true)
|-- bd: integer (nullable = true)
|-- registered_via: integer (nullable = true)
|-- is_ativo: integer (nullable = true)
|-- flag_idade_invalida: integer (nullable = true)
|-- idade_clean: integer (nullable = true)
|-- gender_clean: string (nullable = true)
|-- faixa_idade: string (nullable = true)
|-- flag_city_one: integer (nullable = true)
|-- flag_gender_known: integer (nullable = true)
|-- flag_high_value_registered_via: integer (nullable = true)
|-- registered_via_group: string (nullable = true)
|-- registration_year_regime: string (nullable = true)
|-- tenure_meses: long (nullable = true)
|-- flag_long_tenure: integer (nullable = true)
|-- tenure_faixa: string (nullable = true)
|-- margem_liquida_mensal: double (nullable = true)
|-- target: double (nullable = true)
|-- target_win: double (nullable = true)
|-- revenue_per_hour_listened_cap: double (nullable = true)
|-- revenue_per_hour_tier: string (nullable = true)
|-- usage_intensity_per_tenure_cap: double (nullable = true)
|-- usage_intensity_tier: string (nullable = true)
|-- safra: integer (nullable = true)

```

In [6]: df_feature_store_intermediate.count()

Out[6]: 11242865

In [23]: # Separar features por tipo
schema = df_feature_store_intermediate.schema

In [24]: # 1. Defina o que nunca deve ser feature
blacklist = ['msno', 'safra', 'target', 'target_win']

In [25]: # 2. Categoricas (Geralmente Strings)
features_categoricas = [
 f.name for f in schema.fields
 if isinstance(f.dataType, StringType) and f.name not in blacklist
]

```
In [26]: # 3. Flags (Binárias, Lags, Min, Max) - Tudo que é 0/1 ou limite inferior/superior
features_flags = [
    f.name for f in schema.fields
    if f.name not in blacklist
    and f.name.startswith('flag_')
    and not any(x in f.name for x in ['_mean_', '_ratio_', '_sum_']) # EXCLUI o que é contínuo
]
```



```
In [27]: # 4. Numéricas Contínuas (Médias, Razões, Somas e valores monetários/quantidades)
features_numericas_continuas = [
    f.name for f in schema.fields
    if f.name not in blacklist
    and f.name not in features_flags # GARANTE que não repete o que já é flag
    and f.name not in features_categoricas
    and isinstance(f.dataType, (DoubleType, FloatType, IntegerType))
]
```



```
In [28]: len(schema.fields)
```

Out[28]: 321


```
In [29]: print("=" * 80)
print("📊 INVENTÁRIO DE FEATURES")
print("=" * 80)

print(f"\n💡 Features Numéricas Contínuas + Flags de tendencia media e/ou soma: {len(features_numericas_continuas)}")
print(f"    (ex: total_secs, num_unq, revenue_per_hour, etc.)")

print(f"\n▶ Flags Binárias Na Referencia, Com Lag e Transformadas com Maximo e Minimo: {len(features_flags)}")
print(f"    (ex: flag_shallow_user, flag_premium_sum_6, etc.)")

print(f"\n📝 Features Categóricas: {len(features_categoricas)}")
print(f"    (ex: city, gender, plays_behavior_vs_volume, etc.)")

print(f"\n✅ TOTAL: {len(features_numericas_continuas) + len(features_flags) + len(features_categoricas)})")

print("\n" + "=" * 80)

# Visualizar exemplos de cada tipo
print(f"\n🔍 Exemplos de Features Numéricas Contínuas (primeiras 10):")
for feat in features_numericas_continuas[:3]:
    print(f"    • {feat}")

print(f"\n🔍 Exemplos de Flags Binárias Puras (primeiras 3):")
for feat in features_flags[:3]:
    print(f"    • {feat}")

print(f"\n🔍 Exemplos de Features Categóricas (primeiras 3):")
for feat in features_categoricas[:3]:
    print(f"    • {feat}")
```

```
=====
📊 INVENTÁRIO DE FEATURES
=====

💡 Features Numéricas Contínuas + Flags de tendencia media e/ou soma: 219
(ex: total_secs, num_unq, revenue_per_hour, etc.)

▶ Flags Binárias Na Referencia, Com Lag e Transformadas com Maximo e Minimo: 75
(ex: flag_shallow_user, flag_premium_sum_6, etc.)

📝 Features Categóricas: 19
(ex: city, gender, plays_behavior_vs_volume, etc.)

✅ TOTAL: 313
=====

🔍 Exemplos de Features Numéricas Contínuas (primeiras 10):
• num_25
• num_50
• num_75

🔍 Exemplos de Flags Binárias Puras (primeiras 3):
• flag_has_logs
• flag_shallow_user
• flag_has_logs_raw

🔍 Exemplos de Features Categóricas (primeiras 3):
• total_plays_group
• completed_songs_rate_group
• avg_secs_per_unq_cap_group
```

Checando as que ficaram de fora

```
In [30]: set_schema = set([f.name for f in schema.fields])
set_features = set(features_numericas_continuas + features_flags + features_categoricas)
set_out = set_schema - set_features
print(f"Colunas fora das listas: {set_out}")
```

```
Colunas fora das listas: {'registration_init_time', 'tenure_meses', 'safra', 'transaction_date', 'target', 'msno', 'target_win', 'membership_expire_date'}
```

```
In [31]: df_features_check = df_feature_store_intermediate.select(*set_out)
df_features_check.printSchema()
```

```
root
 |-- registration_init_time: date (nullable = true)
 |-- tenure_meses: long (nullable = true)
 |-- safra: integer (nullable = true)
 |-- transaction_date: date (nullable = true)
 |-- target: double (nullable = true)
 |-- msno: string (nullable = true)
 |-- target_win: double (nullable = true)
 |-- membership_expire_date: date (nullable = true)
```

```
In [32]: features_numericas_continuas.append("tenure_meses")
features_categoricas.append('membership_expire_date')
features_categoricas.append('transaction_date')
features_categoricas.append('registration_init_time')
```

```
In [33]: print("=" * 80)
print("📊 INVENTÁRIO DE FEATURES")
print("=" * 80)

print(f"\n🔢 Features Numéricas Contínuas + Flags de tendencia media e/ou soma: {len(features_numericas_continuas)}")
print(f"  (ex: total_secs, num_unq, revenue_per_hour, etc.)")

print(f"\n▶ Flags Binárias Na Referencia, Com Lag e Transformadas com Maximo e Minimo: {len(features_flags)}")
print(f"  (ex: flag_shallow_user, flag_premium_sum_6, etc.)")

print(f"\n📌 Features Categóricas: {len(features_categoricas)}")
print(f"  (ex: city, gender, plays_behavior_vs_volume, etc.)")

print(f"\n✅ TOTAL: {len(features_numericas_continuas) + len(features_flags) + len(features_categoricas)})")

print("\n" + "=" * 80)

# Visualizar exemplos de cada tipo
print(f"\n🔍 Exemplos de Features Numéricas Contínuas (primeiras 10):")
for feat in features_numericas_continuas[:3]:
    print(f"  • {feat}")

print(f"\n🔍 Exemplos de Flags Binárias Puras (primeiras 3):")
for feat in features_flags[:3]:
    print(f"  • {feat}")

print(f"\n🔍 Exemplos de Features Categóricas (primeiras 3):")
for feat in features_categoricas[:3]:
    print(f"  • {feat}")

set_schema = set([f.name for f in schema.fields])
set_features = set(features_numericas_continuas + features_flags + features_categoricas)
set_out = set_schema - set_features
print(f"Colunas fora das listas: {set_out}")
```

```
=====
📊 INVENTÁRIO DE FEATURES
=====

🔢 Features Numéricas Contínuas + Flags de tendencia media e/ou soma: 220
  (ex: total_secs, num_unq, revenue_per_hour, etc.)

▶ Flags Binárias Na Referencia, Com Lag e Transformadas com Maximo e Minimo: 75
  (ex: flag_shallow_user, flag_premium_sum_6, etc.)

📌 Features Categóricas: 22
  (ex: city, gender, plays_behavior_vs_volume, etc.)

✅ TOTAL: 317

=====
🔍 Exemplos de Features Numéricas Contínuas (primeiras 10):
• num_25
• num_50
• num_75

🔍 Exemplos de Flags Binárias Puras (primeiras 3):
• flag_has_logs
• flag_shallow_user
• flag_has_logs_raw

🔍 Exemplos de Features Categóricas (primeiras 3):
• total_plays_group
• completed_songs_rate_group
• avg_secs_per_unq_cap_group

Colunas fora das listas: {'target_win', 'target', 'safra', 'msno'}
```

12.2. Tratamentos Gerais

Objetivo

Reducir as 318 features para um conjunto otimizado (~50-100 features) que maximize performance preditiva e interpretabilidade dos modelos.

Removendo os casos em que a target esta nula

```
In [197]: df_feature_store_intermediate.filter(F.col("target_win").isNull()).count()
```

```
Out[197]: 1563999
```

```
In [198]: # Identificar última safra de cada cliente
w_ultima_safra = Window.partitionBy('msno').orderBy(F.desc('safra'))


df_feature_store_intermediate_teste = df_feature_store_intermediate.withColumn(
    'is_ultima_safra',
    F.when(F.row_number().over(w_ultima_safra) == 1, 1).otherwise(0)
)

# ==
# ETAPA 2: Split Explícito
# ==

# Dataset de TREINO/VALIDAÇÃO (tem target)
df_train = df_feature_store_intermediate_teste.filter(F.col('target_win').isNotNull())


# Dataset de INFERÊNCIA (última safra, sem target)
df_inference = df_feature_store_intermediate_teste.filter(F.col('target_win').isNull())


# ==
# ETAPA 3: Verificação de Sanidade
# ==

print("=" * 80)
print("📊 SPLIT DE DADOS: Treino vs Inferência")
print("=" * 80)

total_linhas = df_feature_store_intermediate.count()
linhas_train = df_train.count()
linhas_inference = df_inference.count()

print(f"Total de linhas: {total_linhas:,}")
print(f"└ Com target (treino): {linhas_train:,} ({linhas_train/total_linhas*100:.1f}%)")
print(f"└ Sem target (infer): {linhas_inference:,} ({linhas_inference/total_linhas*100:.1f}%)")

# Verificar se última safra == sem target
ultima_safra_count = df_feature_store_intermediate_teste.filter(F.col('is_ultima_safra') == 1).count()
print(f"\nÚltimas safras (flag): {ultima_safra_count:,}")
print(f"Linhas sem target: {linhas_inference:,}")

if ultima_safra_count == linhas_inference:
    print("✅ CONFIRMADO: Nulos são exatamente as últimas safras")
else:
    print("⚠️ AVISO: Há nulos fora da última safra! Investigar...")

    # Diagnóstico adicional
    nulos_nao_ultima = df_feature_store_intermediate_teste.filter(
        (F.col('target_win').isNull()) & (F.col('is_ultima_safra') == 0)
    ).count()
    print(f"└ Nulos em safras intermediárias: {nulos_nao_ultima:,}")

print("=" * 80)
```

```
=====
```

```
📊 SPLIT DE DADOS: Treino vs Inferência
=====
Total de linhas: 11,242,865
└ Com target (treino): 9,678,866 (86.1%)
└ Sem target (infer): 1,563,999 (13.9%)
```

```
Últimas safras (flag): 1,563,999
Linhas sem target: 1,563,999
✅ CONFIRMADO: Nulos são exatamente as últimas safras
=====
```

```
In [34]: df_feature_store_intermediate = df_feature_store_intermediate.filter(F.col("target_win").isNotNull())
df_feature_store_intermediate.count()
```

```
Out[34]: 9678866
```

Etapa 1: Filter Methods

```
#### 1.1. Filtro de Variância Quase-Zero
```

Definicao

Fundamento Matemático:

A variância de uma feature X é definida como:

$$\text{Var}(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

Critério de remoção:

- **Features numéricas contínuas:** $\text{Var}(X) < 0.01$
- **Flags binárias puras (0/1):** Proporção da classe minoritária < 1% (ex: 99.5% zeros)
- **Flags de tendência (média de flags, entre 0 e 1):** $\text{Var}(X) < 0.001$ (threshold mais baixo)
- **Features categóricas:** Uma categoria representa > 99% dos valores

Justificativa: Features constantes ou quase-constantes não carregam informação discriminatória para o modelo.

_Obs.: os thresholds definidos são iniciais, sendo passíveis de mudanças.

Filtro de Variância para Features Numéricas Contínuas

Antes de definir os thresholds, vamos visualizar a distribuição de variâncias para escolher um threshold baseado em dados:

```
In [35]: def filtro_variancia_baixa_continuas(df, features_numericas, threshold=0.05):
    """
    Remove features numéricas contínuas com variância < threshold

    Parâmetros:
        df: DataFrame Spark
        features_numericas: Lista de nomes de features numéricas contínuas
        threshold: Limite mínimo de variância (default: 0.05)

    Saídas:
        features_mantidas: Lista de features com variância >= threshold
        removed_features: Lista de (feature, variância)
    """
    print(f"\n🔍 Calculando variância de {len(features_numericas)} features numéricas contínuas...")

    variance_stats = []

    for col in features_numericas:
        var_value = df.select(F.variance(col)).collect()[0][0]
        if var_value is not None:
            variance_stats.append((col, var_value))
        else:
            # Se variância é None, a coluna é constante (todas nulls ou mesmo valor)
            variance_stats.append((col, 0.0))

    # Ordenar por variância (crescente)
    variance_stats_sorted = sorted(variance_stats, key=lambda x: x[1])

    # Identificar features com baixa variância
    low_variance_features = [(col, var) for col, var in variance_stats_sorted if var < threshold]

    print(f"\n✖ Features com variância < {threshold}: {len(low_variance_features)}")

    if low_variance_features:
        print(f"\n⚠️ Features removidas (mostrando até 20):")
        for col, var in low_variance_features[:20]:
            print(f"  • {col}: Var = {var:.6f}")

    # Features mantidas
    features_mantidas = [col for col in features_numericas if col not in [f[0] for f in low_variance_features]]

    print(f"\n✅ Features mantidas: {len(features_mantidas)})")

    return features_mantidas, low_variance_features
```

```
In [36]: # Executar filtro com threshold = 0.01
features_continuas_filtradas, removed_continuas = filtro_variancia_baixa_continuas(
    df_feature_store_intermediate,
    features_numericas_continuas,
    threshold=0.05
)
```

🔍 Calculando variância de 220 features numéricas contínuas...

✖ Features com variância < 0.05: 16

📋 Features removidas (mostrando até 20):

- is_ativo: Var = 0.000000
- flag_expire_invalido_mean_6: Var = 0.000000
- flag_expire_invalido_mean_3: Var = 0.000001
- flag_expire_invalido_sum_3: Var = 0.000008
- flag_expire_invalido_sum_6: Var = 0.000011
- flag_exemption_mean_6: Var = 0.000378
- flag_exemption_mean_3: Var = 0.001100
- flag_shallow_user_mean_6: Var = 0.001327
- flag_shallow_user_mean_3: Var = 0.002860
- flag_exemption_sum_3: Var = 0.009900
- is_cancel: Var = 0.012357
- flag_exemption_sum_6: Var = 0.013617
- flag_shallow_user_sum_3: Var = 0.025743
- early_drop_rate: Var = 0.035181
- early_drop_rate_raw: Var = 0.035181
- flag_shallow_user_sum_6: Var = 0.047772

✓ Features mantidas: 204

Com threshold de variância = 0.05, saem apenas features **quase constantes** (no caso, a feature `is_ativo` naturalmente seria removida, dado que tem apenas um valor - 0). Isso é muito permissivo e deixa passar muitas features com baixo poder discriminatório.

Maldição da Dimensionalidade - Para Elastic Net, ter muitas features causa:

1. **Overfitting**: Modelo aprende ruído dos dados de treino;
2. **Instabilidade**: Coeficientes variam muito entre folds de validação;
3. **Custo computacional**: Grid search fica lento;
4. **Dificuldade de interpretação**: 150+ coeficientes são impossíveis de explicar.

```
In [37]: def analisar_distribuicao_variancia(df, features_numericas):
    """
    Analisa a distribuição de variâncias para ajudar a escolher threshold
    """
    print(f"🔍 Analisando distribuição de variância de {len(features_numericas)} features...")

    variance_stats = []

    for col in features_numericas:
        var_value = df.select(F.variance(col)).collect()[0][0]
        if var_value is not None:
            variance_stats.append((col, var_value))
        else:
            variance_stats.append((col, 0.0))

    # Criar DataFrame para análise
    df_var = pd.DataFrame(variance_stats, columns=['feature', 'variance'])
    df_var = df_var.sort_values('variance')

    # Estatísticas descritivas
    print(f"\n📊 Estatísticas de Variância:")
    print(f"  • Mínimo: {df_var['variance'].min():.6f}")
    print(f"  • Percentil 10%: {df_var['variance'].quantile(0.10):.6f}")
    print(f"  • Percentil 25%: {df_var['variance'].quantile(0.25):.6f}")
    print(f"  • Mediana: {df_var['variance'].median():.6f}")
    print(f"  • Percentil 75%: {df_var['variance'].quantile(0.75):.6f}")
    print(f"  • Percentil 90%: {df_var['variance'].quantile(0.90):.6f}")
    print(f"  • Máximo: {df_var['variance'].max():.6f}")

    # Contar features por faixa de variância
    print(f"\n📈 Distribuição por Faixa de Variância:")
    print(f"  • Var < 0.01: {len(df_var[df_var['variance'] < 0.01])} features")
    print(f"  • 0.01 ≤ Var < 0.05: {len(df_var[(df_var['variance'] >= 0.01) & (df_var['variance'] < 0.05)])} features")
    print(f"  • 0.05 ≤ Var < 0.10: {len(df_var[(df_var['variance'] >= 0.05) & (df_var['variance'] < 0.10)])} features")
    print(f"  • 0.10 ≤ Var < 0.50: {len(df_var[(df_var['variance'] >= 0.10) & (df_var['variance'] < 0.50)])} features")
    print(f"  • 0.50 ≤ Var < 1.00: {len(df_var[(df_var['variance'] >= 0.50) & (df_var['variance'] < 1.00)])} features")
    print(f"  • Var ≥ 1.00: {len(df_var[df_var['variance'] >= 1.00])} features")

    # Mostrar features com menor variância
    print(f"\n⚠️ 20 Features com Menor Variância:")
    for i, row in df_var.head(20).iterrows():
        print(f"  {i+1:2d}. {row['feature'][:50]} Var = {row['variance']:.6f}")

    return df_var
```

```
# Executar análise
df_var_continuas = analisar_distribuicao_variancia(df_feature_store_intermediate, features_numericas_continuas)
```

🔍 Analisando distribuição de variância de 220 features...

📊 Estatísticas de Variância:

- Mínimo: 0.000000
- Percentil 10%: 0.118462
- Percentil 25%: 317.408235
- Mediana: 1605964558.048935
- Percentil 75%: 1765310409.110675
- Percentil 90%: 1800308875.709165
- Máximo: 44332634694.402168

📈 Distribuição por Faixa de Variância:

- Var < 0.01: 10 features
- 0.01 ≤ Var < 0.05: 6 features
- 0.05 ≤ Var < 0.10: 4 features
- 0.10 ≤ Var < 0.50: 10 features
- 0.50 ≤ Var < 1.00: 5 features
- Var ≥ 1.00: 185 features

⚠️ 20 Features com Menor Variância:

215. is_ativo	Var = 0.000000
211. flag_expire_invalido_mean_6	Var = 0.000000
209. flag_expire_invalido_mean_3	Var = 0.000001
208. flag_expire_invalido_sum_3	Var = 0.000008
210. flag_expire_invalido_sum_6	Var = 0.000011
203. flag_exemption_mean_6	Var = 0.000378
201. flag_exemption_mean_3	Var = 0.001100
171. flag_shallow_user_mean_6	Var = 0.001327
169. flag_shallow_user_mean_3	Var = 0.002860
200. flag_exemption_sum_3	Var = 0.009900
177. is_cancel	Var = 0.012357
202. flag_exemption_sum_6	Var = 0.013617
168. flag_shallow_user_sum_3	Var = 0.025743
151. early_drop_rate_raw	Var = 0.035181
18. early_drop_rate	Var = 0.035181
170. flag_shallow_user_sum_6	Var = 0.047772
138. completed_songs_rate_raw	Var = 0.083243
10. completed_songs_rate	Var = 0.083243

125. completion_efficiency_raw Var = 0.087471
19. completion_efficiency Var = 0.087471

A distribuição de variâncias revela um cenário extremamente heterogêneo e problemático para aplicação direta de threshold único:

1. Bimodalidade Extrema

- * Grupo 1 (13 features): Variância < 1.0 (features em escala original ou transformadas)
- * Grupo 2 (178 features): Variância ≥ 1.0 , com mediana de 412 milhões e máximo de 42 bilhões

Interpretação: As features estão em escalas completamente diferentes. Features como `total_secs`, `num_unq`, e suas derivadas (somas, médias de janelas temporais) estão em escala bruta (milhões/bilhões), enquanto features transformadas (log, taxas, flags) estão em escala [0,1] ou logarítmica.

2. Problema do Threshold Absoluto

Um threshold de variância absoluto (ex: 0.05) é inadequado porque:

- * Remove features informativas em escala pequena (ex: `completion_efficiency`, `early_drop_rate`);
- * Mantém features em escala grande mesmo que sejam pouco informativas.

```

In [38]: def filtro_coeficiente_variacao(df, features_numericas, threshold_cv=0.05):
    """
        Remove features com Coeficiente de Variação (CV) < threshold

        CV = σ / μ (desvio padrão / média)

        Vantagens sobre variância absoluta:
        - Independente de escala
        - Identifica features "proporcionalmente constantes"

        Parâmetros:
            df: DataFrame Spark
            features_numericas: Lista de features numéricas contínuas
            threshold_cv: Limite mínimo de CV (default: 0.05 = 5%)

        Saídas:
            features_mantidas: Lista de features mantidas
            removed_features: Lista de (feature, média, stddev, CV)
    """

    print(f"🔍 Calculando Coeficiente de Variação de {len(features_numericas)} features...")
    print(f"    Threshold: CV < {threshold_cv} ({threshold_cv*100:.1f}%)")

    cv_stats = []

    for col in features_numericas:
        # Calcular média e desvio padrão
        stats = df.select(
            F.mean(col).alias('mean'),
            F.stddev(col).alias('stddev')
        ).collect()[0]

        mean_val = stats['mean']
        stddev_val = stats['stddev']

        # Calcular CV
        if mean_val is not None and stddev_val is not None and mean_val != 0:
            cv = abs(stddev_val / mean_val) # abs() para lidar com médias negativas
            cv_stats.append((col, mean_val, stddev_val, cv))
        else:
            # Se média é 0 ou None, a feature é constante
            cv_stats.append((col, mean_val, stddev_val, 0.0))

    # Ordenar por CV (crescente)
    cv_stats_sorted = sorted(cv_stats, key=lambda x: x[3])

    # Identificar features com baixo CV
    low_cv_features = [(col, mean_val, stddev_val, cv) for col, mean_val, stddev_val, cv in cv_stats_sorted if cv < threshold_cv]

    print(f"\n✖ Features com CV < {threshold_cv}: {len(low_cv_features)}")

    if low_cv_features:
        print(f"\n⚠️ Features removidas (mostrando todas):")
        for col, mean_val, stddev_val, cv in low_cv_features:
            print(f"    • {col:50s} μ={mean_val:12.2f}, σ={stddev_val:12.2f}, CV={cv:.6f}")

    # Features mantidas
    features_mantidas = [col for col in features_numericas if col not in [f[0] for f in low_cv_features]]

    print(f"\n✅ Features mantidas: {len(features_mantidas)})")

    # Mostrar estatísticas das features mantidas
    if features_mantidas:
        cv_mantidas = [cv for col, mean_val, stddev_val, cv in cv_stats_sorted if col in features_mantidas]
        print(f"\n📊 Estatísticas de CV das features mantidas:")
        print(f"    • Mínimo: {min(cv_mantidas):.4f}")
        print(f"    • Percentil 25%: {np.percentile(cv_mantidas, 25):.4f}")
        print(f"    • Mediana: {np.median(cv_mantidas):.4f}")
        print(f"    • Percentil 75%: {np.percentile(cv_mantidas, 75):.4f}")
        print(f"    • Máximo: {max(cv_mantidas):.4f}")

    return features_mantidas, low_cv_features

```

In [39]:

```
# Executar filtro com CV
print("=" * 80)
print("FILTER DE VARIÂNCIA REVISADO - COEFICIENTE DE VARIAÇÃO")
print("=" * 80)

features_continuas_filtradas_cv, removed_continuas_cv = filtro_coeficiente_variacao(
    df_feature_store_intermediate,
    features_numericas_continuas,
    threshold_cv=0.05 # 5% - features com variação < 5% da média
)

print("=" * 80)
```

```
=====
FILTER DE VARIÂNCIA REVISADO - COEFICIENTE DE VARIAÇÃO
=====
🔍 Calculando Coeficiente de Variação de 220 features...
Threshold: CV < 0.05 (5.0%)

✖ Features com CV < 0.05: 1

📝 Features removidas (mostrando todas):
• is_ativo           μ=      1.00, σ=      0.00, CV=0.000000

✅ Features mantidas: 219

📊 Estatísticas de CV das features mantidas:
• Mínimo: 0.0903
• Percentil 25%: 1.3372
• Mediana: 1.8351
• Percentil 75%: 2.0141
• Máximo: 364.1239
=====
```

Decisão final: manter a remoção das features filtradas por variancia. Justificativa:

* O CV não "discordou" do que foi filtrado anteriormente.

* Min em 9% significa que todas as features têm variação significativa em relação à média (desvio padrão \geq 9% da média)

* Mediana em 183% indica alta heterogeneidade. A maioria das features tem desvio padrão maior que a média, o que é excelente para poder discriminatório

Filtro de Variância para Flags

```
In [40]: def filtro_flags_desbalanceadas(df, features_flags, threshold=0.01):
    """
        Remove flags binárias onde a classe minoritária representa < threshold

        Para flags binárias (0/1), a variância máxima ocorre quando p=0.5:
        Var(X) = p(1-p), onde p é a proporção de 1s

        Parâmetros:
            df: DataFrame Spark
            features_flags: Lista de flags binárias
            threshold: Proporção mínima da classe minoritária (default: 0.01 = 1%)

        Saídas:
            features_mantidas: Lista de flags mantidas
            removed_features: Lista de (flag, proporção_minoritária, variância)
    """
    print(f"\n🔍 Analisando {len(features_flags)} flags binárias puras...")

    total_count = df.count()
    imbalanced_flags = []

    for col in features_flags:
        # Calcular proporção de 1s
        count_ones = df.filter(F.col(col) == 1).count()
        prop_ones = count_ones / total_count
        prop_zeros = 1 - prop_ones

        # Classe minoritária
        minority_prop = min(prop_ones, prop_zeros)

        # Variância de Bernoulli: p(1-p)
        variance = prop_ones * prop_zeros

        if minority_prop < threshold:
            imbalanced_flags.append((col, minority_prop, variance))

    # Ordenar por proporção minoritária (crescente)
    imbalanced_flags_sorted = sorted(imbalanced_flags, key=lambda x: x[1])

    print(f"\n✖ Flags com classe minoritária < {threshold:.1%}: {len(imbalanced_flags)}")

    if imbalanced_flags:
        print(f"\n📋 Flags removidas (mostrando até 10):")
        for col, minority_prop, var in imbalanced_flags_sorted[:10]:
            print(f"  • {col}: Classe minoritária = {minority_prop:.2%}, Var = {var:.6f}")

    # Features mantidas
    features_mantidas = [col for col in features_flags if col not in [f[0] for f in imbalanced_flags]]

    print(f"\n✅ Flags mantidas: {len(features_mantidas)}")

    return features_mantidas, imbalanced_flags
```

```
In [41]: # Executar filtro
features_flags_bin_filtradas, removed_flags_bin = filtro_flags_desbalanceadas(
    df_feature_store_intermediate,
    features_flags,
    threshold=0.01
)
```

🔍 Analisando 75 flags binárias puras...

✖ Flags com classe minoritária < 1.0%: 27

📋 Flags removidas (mostrando até 10):

- flag_expire_invalido_lag_5: Classe minoritária = 0.00%, Var = 0.000001
- flag_expire_invalido_lag_4: Classe minoritária = 0.00%, Var = 0.000001
- flag_expire_invalido_lag_3: Classe minoritária = 0.00%, Var = 0.000002
- flag_expire_invalido_lag_2: Classe minoritária = 0.00%, Var = 0.000002
- flag_expire_invalido_lag_1: Classe minoritária = 0.00%, Var = 0.000003
- flag_expire_invalido: Classe minoritária = 0.00%, Var = 0.000003
- flag_expire_invalido_raw: Classe minoritária = 0.00%, Var = 0.000003
- flag_expire_invalido_lag_0: Classe minoritária = 0.00%, Var = 0.000003
- flag_expire_invalido_max_3: Classe minoritária = 0.00%, Var = 0.000008
- flag_expire_invalido_max_6: Classe minoritária = 0.00%, Var = 0.000011

✅ Flags mantidas: 48

Filtro de Variância para Features Categóricas

```
In [42]: def filtro_categoricas_constantes(df, features_categoricas, threshold=0.99):
    """
        Remove features categóricas onde uma categoria domina > threshold
    """

    Args:
        df: DataFrame Spark
        features_categoricas: Lista de nomes de features categóricas
        threshold: Proporção máxima de dominância (default: 0.99)

    Returns:
        features_mantidas: Lista de features mantidas
        removed_features: Lista de (feature, proporção_dominante)
    """
    print(f"🔍 Analisando {len(features_categoricas)} features categóricas...")

    total_count = df.count()
    constant_categoricals = []

    for col in features_categoricas:
        # Calcular proporção da categoria mais frequente
        max_freq = df.groupBy(col).count().agg(F.max("count")).collect()[0][0]
        max_proportion = max_freq / total_count

        if max_proportion > threshold:
            constant_categoricals.append((col, max_proportion))

    print(f"\n✖ Features categóricas com dominância > {threshold:.0%}: {len(constant_categoricals)}")

    if constant_categoricals:
        print(f"\n❌ Features removidas:")
        for col, prop in constant_categoricals:
            print(f"  • {col}: {prop:.2%} de dominância")

    # Features mantidas
    features_mantidas = [
        col for col in features_categoricas
        if col not in [c[0] for c in constant_categoricals]
    ]

    print(f"\n✅ Features categóricas mantidas: {len(features_mantidas)}")

    return features_mantidas, constant_categoricals
```

```
In [43]: # Executar filtro
features_cat_filtradas, removed_constant_cat = filtro_categoricas_constantes(
    df_feature_store_intermediate,
    features_categoricas,
    threshold=0.99
)
```

🔍 Analisando 22 features categóricas...

✖ Features categóricas com dominância > 99%: 0

✅ Features categóricas mantidas: 22

Consolidado filtros de variancia

In [44]:

```
# Consolidar todas as features numéricas filtradas
features_numericas_filtradas = (
    features_continuas_filtradas +
    features_flags_bin_filtradas
)

print("=" * 80)
print("📊 RESUMO - FILTRO DE VARIÂNCIA")
print("=" * 80)

print(f"\n🔢 Features Numéricas Contínuas:")
print(f"  • Iniciais: {len(features_numericas_continuas)}")
print(f"  • Mantidas: {len(features_continuas_filtradas)}")
print(f"  • Removidas: {len(removed_continuas)}")

print(f"\n▶ Flags Binárias Puras:")
print(f"  • Iniciais: {len(features_flags)}")
print(f"  • Mantidas: {len(features_flags_bin_filtradas)}")
print(f"  • Removidas: {len(removed_flags_bin)}")

print(f"\n📂 Features Categóricas:")
print(f"  • Iniciais: {len(features_categoricas)}")
print(f"  • Mantidas: {len(features_cat_filtradas)}")
print(f"  • Removidas: {len(removed_constant_cat)}")

print(f"\n✅ TOTAL APÓS FILTRO DE VARIÂNCIA:")
print(f"  • Features numéricas: {len(features_numericas_filtradas)}")
print(f"  • Features categóricas: {len(features_cat_filtradas)}")
print(f"  • TOTAL: {len(features_numericas_filtradas) + len(features_cat_filtradas)}")

print("\n" + "=" * 80)
```

```
=====
📊 RESUMO - FILTRO DE VARIÂNCIA
=====

🔢 Features Numéricas Contínuas:
  • Iniciais: 220
  • Mantidas: 204
  • Removidas: 16

▶ Flags Binárias Puras:
  • Iniciais: 75
  • Mantidas: 48
  • Removidas: 27

📂 Features Categóricas:
  • Iniciais: 22
  • Mantidas: 22
  • Removidas: 0

✅ TOTAL APÓS FILTRO DE VARIÂNCIA:
  • Features numéricas: 252
  • Features categóricas: 22
  • TOTAL: 274
=====
```

Etapa 2: Análise de Associação com Target

2.1. FEATURES NUMÉRICAS: Pearson + Spearman

```
In [45]: def calcular_correlacao_dupla_target(df, features_numericas, target_col='target_win', threshold=0.02):
    """
    Calcula Pearson E Spearman para cada feature vs target
    Remove apenas se AMBAS as correlações forem < threshold

    CORREÇÃO: Usa VectorAssembler + Correlation.corr() para Spearman
    """
    print(f"\n🕒 Calculando Pearson + Spearman para {len(features_numericas)} features...")
    print(f"  Threshold: |ρ| < {threshold} (ambos os métodos)")

    correlations = []

    # Calcular Pearson (método rápido via df.corr)
    print(f"\n  [1/2] Calculando Pearson...")
    pearson_dict = {}
    for col in features_numericas:
        try:
            pearson_val = df.corr(col, target_col, method="pearson")
            if pearson_val is None:
                pearson_val = 0.0
            pearson_dict[col] = pearson_val
        except Exception as e:
            print(f"⚠️ Erro em {col}: {e}")
            pearson_dict[col] = 0.0

    # Calcular Spearman (método via VectorAssembler + Correlation)
    print(f"\n  [2/2] Calculando Spearman (pode demorar alguns minutos)...")

    # Preparar dados: todas as features + target em um único vetor
    all_cols = features_numericas + [target_col]

    assembler = VectorAssembler(
        inputCols=all_cols,
        outputCol="features_vector",
        handleInvalid="skip" # Ignorar nulls
    )

    df_vector = assembler.transform(df).select("features_vector")

    # Calcular matriz de correlação Spearman
    spearman_matrix = Correlation.corr(df_vector, "features_vector", method="spearman").collect()[0][0]
    spearman_matrix_array = spearman_matrix.toArray()

    # Extrair correlações com o target (última coluna/linha da matriz)
    target_idx = len(all_cols) - 1
    spearman_dict = {}

    for i, col in enumerate(features_numericas):
        spearman_val = spearman_matrix_array[i, target_idx]
        spearman_dict[col] = spearman_val

    # Consolidar resultados
    for col in features_numericas:
        pearson_val = pearson_dict[col]
        spearman_val = spearman_dict[col]

        abs_pearson = abs(pearson_val)
        abs_spearman = abs(spearman_val)

        # Usar o MAIOR dos dois como critério de manutenção
        max_corr = max(abs_pearson, abs_spearman)

        correlations.append((col, pearson_val, spearman_val, abs_pearson, abs_spearman, max_corr))

    # Ordenar por max_corr (decrescente)
    correlations_sorted = sorted(correlations, key=lambda x: x[5], reverse=True)

    # Identificar features com AMBAS correlações baixas
    low_corr_features = [
        col for col, p, s, abs_p, abs_s, max_c in correlations_sorted
        if max_c < threshold
    ]

    print(f"\n✖ Features com max(|Pearson|, |Spearman|) < {threshold}: {len(low_corr_features)}")

    if low_corr_features:
        print(f"\n🕒 Features removidas (mostrando até 20):")
        for col, p, s, abs_p, abs_s, max_c in correlations_sorted:
            if col in low_corr_features[:20]:
                tipo = "[FLAG_BIN]" if col.startswith('flag_') and not any(x in col for x in ['_mean_', '_sum_']) else \
                      "[FLAG_TEND]" if col.startswith('flag_') else "[CONTÍNUA]"
                print(f"  • {tipo:13s} {col:50s} Pearson={p:+.4f}, Spearman={s:+.4f}")

    print(f"\n💡 Top 30 features por correlação máxima:")
    for i, (col, p, s, abs_p, abs_s, max_c) in enumerate(correlations_sorted[:30], 1):
        tipo = "[FLAG_BIN]" if col.startswith('flag_') and not any(x in col for x in ['_mean_', '_sum_']) else \
              "[FLAG_TEND]" if col.startswith('flag_') else "[CONTÍNUA]"
        print(f"  • {tipo:13s} {col:50s} Pearson={p:+.4f}, Spearman={s:+.4f}"
```

```

# Indicar qual método foi mais forte
metodo_dominante = "📈 Pearson" if abs_p > abs_s else "📊 Spearman"

print(f" {i:2d}. {tipo:13s} {col:50s} P={p:+.4f}, S={s:+.4f} {metodo_dominante}")

# Features mantidas
features_mantidas = [
    col for col, p, s, abs_p, abs_s, max_c in correlations_sorted
    if max_c >= threshold
]

print(f"\n✅ Features mantidas: {len(features_mantidas)}")

# Análise de discrepância Pearson vs Spearman
print(f"\n📊 Análise de Não-Linearidade (|Pearson - Spearman| > 0.05):")
non_linear_features = [
    (col, p, s, abs(p - s))
    for col, p, s, abs_p, abs_s, max_c in correlations_sorted
    if abs(p - s) > 0.05 and col in features_mantidas
]

if non_linear_features:
    non_linear_features_sorted = sorted(non_linear_features, key=lambda x: x[3], reverse=True)
    print(f" {len(non_linear_features)} features com relação não-linear detectada:")
    for col, p, s, diff in non_linear_features_sorted[:10]:
        tipo = "[FLAG_BIN]" if col.startswith('flag_') and not any(x in col for x in ['_mean_', '_sum_']) else \
               "[FLAG_TEND]" if col.startswith('flag_') else "[CONTÍNUA]"
        print(f"     • {tipo:13s} {col:50s} P={p:+.4f}, S={s:+.4f}, Δ={diff:.4f}")
else:
    print(f" Nenhuma feature com discrepância significativa detectada.")

return features_mantidas, correlations_sorted, low_corr_features

```

```

In [46]: # Consolidar features numéricas
features_numericas_post_filter = (
    features_continuas_filtradas +
    features_flags_bin_filtradas
)

print(f"\n📋 Features numéricas para análise:")
print(f"     • Contínuas: {len(features_continuas_filtradas)}")
print(f"     • Flags binárias: {len(features_flags_bin_filtradas)}")
print(f"     • TOTAL: {len(features_numericas_post_filter)}")

```

📋 Features numéricas para análise:

- Contínuas: 204
- Flags binárias: 48
- TOTAL: 252

In [47]:

```
# Executar
features_num_corr_filtradas, all_correlations, removed_low_corr = calcular_correlacao_dupla_target(
    df_feature_store_intermediate,
    features_numericas_post_filter,
    target_col='target_win',
    threshold=0.02
)
```

Calculando Pearson + Spearman para 252 features...
Threshold: $|p| < 0.02$ (ambos os métodos)

[1/2] Calculando Pearson...

[2/2] Calculando Spearman (pode demorar alguns minutos)...

✖ Features com $\max(|\text{Pearson}|, |\text{Spearman}|) < 0.02: 13$

📋 Features removidas (mostrando até 20):

- [CONTÍNUA] avg_secs_per_unq_cap_ratio_ref_min_6
- [CONTÍNUA] early_drop_rate_ratio_ref_mean_3
- [CONTÍNUA] completed_songs_rate_ratio_ref_mean_3
- [CONTÍNUA] early_drop_rate_ratio_ref_max_6
- [CONTÍNUA] completed_songs_rate_ratio_ref_mean_6
- [CONTÍNUA] early_drop_rate_min_6
- [CONTÍNUA] plays_per_unq_cap_ratio_ref_min_3
- [CONTÍNUA] completion_efficiency_max_6
- [CONTÍNUA] early_drop_rate_ratio_ref_min_6
- [CONTÍNUA] completion_efficiency_ratio_ref_mean_3
- [CONTÍNUA] avg_secs_per_unq_cap_ratio_ref_min_3
- [CONTÍNUA] early_drop_rate_ratio_ref_min_3
- [CONTÍNUA] catalog_exploration_ratio_cap_min_6

🏆 Top 30 features por correlação máxima:

1. [CONTÍNUA] margem_liquida_mensal
2. [FLAG_BIN] flag_plano_mensal_max_3
3. [FLAG_BIN] flag_plano_mensal_max_6
4. [CONTÍNUA] daily_revenue_efficiency
5. [CONTÍNUA] daily_revenue_efficiency_raw
6. [FLAG_BIN] flag_plano_mensal
7. [FLAG_BIN] flag_plano_mensal_raw
8. [FLAG_BIN] flag_plano_mensal_lag_0
9. [FLAG_TEND] flag_plano_mensal_sum_3
10. [FLAG_TEND] flag_plano_mensal_mean_3
11. [FLAG_BIN] flag_valid_fee
12. [FLAG_BIN] flag_valid_fee_raw
13. [FLAG_BIN] flag_valid_fee_lag_0
14. [FLAG_BIN] flag_has_transactions
15. [FLAG_BIN] flag_has_transactions_raw
16. [FLAG_BIN] flag_has_transactions_lag_0
17. [CONTÍNUA] revenue_per_hour_listened_cap
18. [FLAG_BIN] flag_valid_fee_max_3
19. [FLAG_BIN] flag_has_transactions_max_3
20. [CONTÍNUA] payment_method_id
21. [FLAG_TEND] flag_valid_fee_mean_3
22. [FLAG_TEND] flag_valid_fee_sum_3
23. [FLAG_TEND] flag_has_transactions_mean_3
24. [FLAG_TEND] flag_has_transactions_sum_3
25. [CONTÍNUA] log_total_secs_raw
26. [CONTÍNUA] log_total_secs
27. [CONTÍNUA] total_secs
28. [CONTÍNUA] total_secs_raw
29. [CONTÍNUA] num_100
30. [CONTÍNUA] total_plays

✓ Features mantidas: 239

📊 Análise de Não-Linearidade ($|\text{Pearson} - \text{Spearman}| > 0.05$):

195 features com relação não-linear detectada:

- [CONTÍNUA] payment_method_id
- [CONTÍNUA] daily_revenue_efficiency_ratio_ref_mean_3
- [FLAG_BIN] flag_valid_fee_max_3
- [FLAG_BIN] flag_valid_fee
- [FLAG_BIN] flag_valid_fee_raw
- [FLAG_BIN] flag_valid_fee_lag_0
- [CONTÍNUA] log_total_secs_mean_3
- [CONTÍNUA] log_total_plays_mean_3
- [CONTÍNUA] total_plays_mean_3
- [CONTÍNUA] log_total_secs_max_3

Pearson=+0.0078, Spearman=-0.0179
Pearson=-0.0035, Spearman=+0.0152
Pearson=-0.0009, Spearman=-0.0135
Pearson=+0.0011, Spearman=-0.0090
Pearson=+0.0032, Spearman=-0.0087
Pearson=+0.0011, Spearman=-0.0081
Pearson=-0.0009, Spearman=-0.0067
Pearson=+0.0055, Spearman=+0.0045
Pearson=+0.0011, Spearman=-0.0049
Pearson=+0.0017, Spearman=+0.0046
Pearson=+0.0039, Spearman=-0.0017
Pearson=-0.0035, Spearman=+0.0033
Pearson=+0.0032, Spearman=+0.0025

P=+0.2331, S=+0.7084 Spearman
P=+0.6946, S=+0.1860 Pearson
P=+0.6761, S=+0.1824 Pearson
P=+0.6556, S=+0.5098 Pearson
P=+0.6556, S=+0.5098 Pearson
P=+0.6475, S=+0.2137 Pearson
P=+0.6475, S=+0.2137 Pearson
P=+0.6475, S=+0.2137 Pearson
P=+0.6090, S=+0.1225 Pearson
P=+0.6090, S=+0.1225 Pearson
P=+0.6003, S=+0.0389 Pearson
P=+0.6003, S=+0.0389 Pearson
P=+0.6003, S=+0.0389 Pearson
P=+0.5998, S=+nan Spearman
P=+0.5998, S=+nan Spearman
P=+0.5998, S=+nan Spearman
P=+0.2139, S=+0.5909 Spearman
P=+0.5892, S=+0.0230 Pearson
P=+0.5887, S=+nan Spearman
P=+0.5801, S=-0.2264 Pearson
P=+0.5749, S=+0.1029 Pearson
P=+0.5749, S=+0.1029 Pearson
P=+0.5748, S=+0.0957 Pearson
P=+0.5748, S=+0.0957 Pearson
P=+0.1448, S=-0.5517 Spearman
P=+0.1448, S=-0.5517 Spearman
P=-0.2620, S=-0.5517 Spearman
P=-0.2620, S=-0.5517 Spearman
P=-0.2603, S=-0.5477 Spearman
P=-0.2588, S=-0.5471 Spearman

P=+0.5801, S=-0.2264, Δ=0.8065
P=+0.4690, S=-0.1086, Δ=0.5776
P=+0.5892, S=+0.0230, Δ=0.5662
P=+0.6003, S=+0.0389, Δ=0.5614
P=+0.6003, S=+0.0389, Δ=0.5614
P=+0.6003, S=+0.0389, Δ=0.5614
P=+0.0228, S=-0.5382, Δ=0.5610
P=+0.0228, S=-0.5345, Δ=0.5573
P=+0.0191, S=-0.5377, Δ=0.5568
P=+0.0228, S=-0.5294, Δ=0.5523

In [48]:

```
print("\n" + "=" * 80)
print("📊 RESUMO CAMADA 2.1")
print("=" * 80)
print(f" • Features iniciais: {len(features_numericas_post_filter)}")
print(f" • Features removidas: {len(removed_low_corr)}")
print(f" • Features mantidas: {len(features_num_corr_filtradas)}")
print(f" • Taxa de redução: {len(removed_low_corr) / len(features_numericas_post_filter) * 100:.1f}%"）
print("=" * 80)
```

```
=====
📊 RESUMO CAMADA 2.1
=====
• Features iniciais: 252
• Features removidas: 13
• Features mantidas: 239
• Taxa de redução: 5.2%
=====
```

Problemas encontrados

- Problema 1: Features `_raw` são DUPLICATAS EXATAS;
- Problema 2: 261 Features é EXCESSIVO: overfitting garantido em modelos lineares, multicolinearidade severa.

Plano de Ação Imediato

Remover Features `_raw` e `_lag_0` Duplicadas

```
In [49]: print("=" * 80)
print("🔧 ETAPA 1: REMOÇÃO DE FEATURES _raw E _lag_0 DUPLICADAS")
print("=" * 80)

# Identificar features _raw duplicadas
features_raw = [col for col in features_num_corr_filtradas if col.endswith('_raw')]
features_raw_duplicadas = [col for col in features_raw if col.replace('_raw', '') in features_num_corr_filtradas]

print(f"\n📋 Features _raw que têm versão sem sufixo:")
print(f"  • Total de features _raw: {len(features_raw)}")
print(f"  • Features _raw duplicadas: {len(features_raw_duplicadas)}")

if features_raw_duplicadas:
    print(f"\n📋 Amostra de features _raw a serem removidas (mostrando até 20):")
    for col in features_raw_duplicadas[:20]:
        versao_original = col.replace('_raw', '')
        print(f"  • {col:60s} → Mantém: {versao_original}")

# Identificar features _lag_0 duplicadas
features_lag_0 = [col for col in features_num_corr_filtradas if col.endswith('_lag_0')]
features_lag_0_duplicadas = [col for col in features_lag_0 if col.replace('_lag_0', '') in features_num_corr_filtradas]

print(f"\n📋 Features _lag_0 que têm versão sem sufixo:")
print(f"  • Total de features _lag_0: {len(features_lag_0)}")
print(f"  • Features _lag_0 duplicadas: {len(features_lag_0_duplicadas)}")

if features_lag_0_duplicadas:
    print(f"\n📋 Amostra de features _lag_0 a serem removidas (mostrando até 20):")
    for col in features_lag_0_duplicadas[:20]:
        versao_original = col.replace('_lag_0', '')
        print(f"  • {col:60s} → Mantém: {versao_original}")

# Remover features duplicadas (_raw e _lag_0)
features_duplicadas = features_raw_duplicadas + features_lag_0_duplicadas
features_num_sem_duplicatas = [col for col in features_num_corr_filtradas if col not in features_duplicadas]
```

```
=====
🔧 ETAPA 1: REMOÇÃO DE FEATURES _raw E _lag_0 DUPLICADAS
=====

📋 Features _raw que têm versão sem sufixo:
• Total de features _raw: 15
• Features _raw duplicadas: 15

📋 Amostra de features _raw a serem removidas (mostrando até 20):
• daily_revenue_efficiency_raw → Mantém: daily_revenue_efficiency
• flag_plano_mensal_raw → Mantém: flag_plano_mensal
• flag_valid_fee_raw → Mantém: flag_valid_fee
• flag_has_transactions_raw → Mantém: flag_has_transactions
• log_total_secs_raw → Mantém: log_total_secs
• total_secs_raw → Mantém: total_secs
• total_plays_raw → Mantém: total_plays
• log_total_plays_raw → Mantém: log_total_plays
• num_unq_raw → Mantém: num_unq
• plays_per_unq_cap_raw → Mantém: plays_per_unq_cap
• catalog_exploration_ratio_cap_raw → Mantém: catalog_exploration_ratio_cap
• completed_songs_rate_raw → Mantém: completed_songs_rate
• avg_secs_per_unq_cap_raw → Mantém: avg_secs_per_unq_cap
• completion_efficiency_raw → Mantém: completion_efficiency
• flag_has_logs_raw → Mantém: flag_has_logs

📋 Features _lag_0 que têm versão sem sufixo:
• Total de features _lag_0: 4
• Features _lag_0 duplicadas: 4

📋 Amostra de features _lag_0 a serem removidas (mostrando até 20):
• flag_plano_mensal_lag_0 → Mantém: flag_plano_mensal
• flag_valid_fee_lag_0 → Mantém: flag_valid_fee
• flag_has_transactions_lag_0 → Mantém: flag_has_transactions
• flag_has_logs_lag_0 → Mantém: flag_has_logs
```

```
In [50]: # =====
# IMPORTANTE: Atualizar all_correlations para remover _raw e _lag_0
# =====

all_correlations = [
    (col, p, s, abs_p, abs_s, max_c)
    for col, p, s, abs_p, abs_s, max_c in all_correlations
    if col not in features_duplicadas
]
```

Aumento do Threshold de Correlação (Agressivo)

```
In [52]: # Filtrar correlações do resultado anterior
# Usar all_correlations (já calculado) para evitar recalcular Spearman

def filtrar_por_threshold_agressivo(correlations_sorted, threshold=0.05):
    """
    Aplica threshold mais agressivo nas correlações já calculadas
    """

    print(f"\n🔍 Aplicando threshold agressivo: |ρ| < {threshold}")

    # Filtrar features com max_corr >= threshold
    features_mantidas = [
        col for col, p, s, abs_p, abs_s, max_c in correlations_sorted
        if max_c >= threshold
    ]

    features_removidas = [
        col for col, p, s, abs_p, abs_s, max_c in correlations_sorted
        if max_c < threshold
    ]

    print(f"\n✖ Features removidas (max_corr < {threshold}): {len(features_removidas)}")

    if features_removidas:
        print(f"\n📋 Features removidas (mostrando até 30):")
        for col, p, s, abs_p, abs_s, max_c in correlations_sorted:
            if col in features_removidas[:30]:
                tipo = "[FLAG_BIN]" if col.startswith('flag_') and not any(x in col for x in ['_mean_', '_sum_']) else \
                    "[FLAG_TEND]" if col.startswith('flag_') else "[CONTÍNUA]"
                print(f"  • {tipo:13s} {col:50s} P={p:+.4f}, S={s:+.4f}, max={max_c:.4f}")

    print(f"\n✅ Features mantidas: {len(features_mantidas)}")

    return features_mantidas, features_removidas
```

Threshold de 5%

```
In [53]: # Aplicar threshold agressivo
features_num_threshold_agressivo, removed_threshold_agressivo = filtrar_por_threshold_agressivo(
    all_correlations,
    threshold=0.05
)
```

🔍 Aplicando threshold agressivo: |ρ| < 0.05

✖ Features removidas (max_corr < 0.05): 39

📋 Features removidas (mostrando até 30):	P=+0.0011, S=+0.0488, max=0.0488
• [CONTÍNUA] early_drop_rate_ratio_ref_mean_6	P=-0.0009, S=-0.0473, max=0.0473
• [CONTÍNUA] catalog_exploration_ratio_cap_ratio_ref_max_3	P=+0.0032, S=+0.0461, max=0.0461
• [CONTÍNUA] catalog_exploration_ratio_cap_ratio_ref_mean_3	P=-0.0009, S=+0.0436, max=0.0436
• [CONTÍNUA] completion_efficiency_ratio_ref_min_6	P=+0.0055, S=+0.0415, max=0.0415
• [CONTÍNUA] plays_per_unq_cap_ratio_ref_mean_6	P=+0.0032, S=-0.0378, max=0.0378
• [CONTÍNUA] log_total_secs_ratio_ref_min_6	P=+0.0334, S=+0.0373, max=0.0373
• [CONTÍNUA] completion_efficiency_ratio_ref_min_3	P=+0.0017, S=+0.0369, max=0.0369
• [CONTÍNUA] catalog_exploration_ratio_cap_ratio_ref_min_6	P=+0.0032, S=+0.0364, max=0.0364
• [CONTÍNUA] num_unq_ratio_ref_min_6	P=+0.0334, S=+0.0074, max=0.0334
• [CONTÍNUA] total_plays_ratio_ref_min_6	P=+0.0334, S=+0.0067, max=0.0334
• [CONTÍNUA] total_secs_ratio_ref_min_6	P=+0.0334, S=+0.0063, max=0.0334
• [CONTÍNUA] plays_per_unq_cap_ratio_mean_3	P=-0.0009, S=-0.0317, max=0.0317
• [CONTÍNUA] early_drop_rate_ratio_ref_max_3	P=-0.0035, S=-0.0270, max=0.0270
• [CONTÍNUA] avg_secs_per_unq_cap_ratio_ref_mean_6	P=+0.0078, S=-0.0268, max=0.0268
• [CONTÍNUA] completion_efficiency_ratio_ref_mean_6	P=+0.0055, S=+0.0262, max=0.0262
• [CONTÍNUA] catalog_exploration_ratio_cap_ratio_ref_mean_3	P=-0.0009, S=+0.0249, max=0.0249
• [CONTÍNUA] completed_songs_rate_ratio_ref_min_3	P=-0.0009, S=+0.0240, max=0.0240
• [CONTÍNUA] completed_songs_rate_ratio_ref_min_6	P=+0.0032, S=+0.0230, max=0.0230
• [CONTÍNUA] num_unq_ratio_ref_min_3	P=+0.0228, S=-0.0073, max=0.0228
• [CONTÍNUA] total_plays_ratio_ref_min_3	P=+0.0228, S=-0.0117, max=0.0228
• [CONTÍNUA] log_total_plays_ratio_ref_min_3	P=+0.0228, S=+0.0162, max=0.0228
• [CONTÍNUA] total_secs_ratio_ref_min_3	P=+0.0228, S=+0.0088, max=0.0228
• [CONTÍNUA] total_secs_ratio_ref_min_3	P=+0.0228, S=-0.0079, max=0.0228
• [CONTÍNUA] avg_secs_per_unq_cap_ratio_ref_mean_3	P=+0.0039, S=-0.0227, max=0.0227
• [CONTÍNUA] plays_per_unq_cap_ratio_ref_min_6	P=+0.0032, S=-0.0221, max=0.0221
• [CONTÍNUA] avg_secs_per_unq_cap_ratio_ref_min_6	P=+0.0078, S=-0.0179, max=0.0179
• [CONTÍNUA] early_drop_rate_ratio_ref_mean_3	P=-0.0035, S=+0.0152, max=0.0152
• [CONTÍNUA] completed_songs_rate_ratio_ref_mean_3	P=-0.0009, S=-0.0135, max=0.0135
• [CONTÍNUA] early_drop_rate_ratio_ref_max_6	P=+0.0011, S=-0.0090, max=0.0090

✅ Features mantidas: 194

```
In [54]: print(f"\n📊 Resumo após Etapa 2:")
print(f"  • Após threshold 0.05: {len(features_num_threshold_agressivo)}")
```

📊 Resumo após Etapa 2:
• Após threshold 0.05: 194

Threshold de 10%

```
In [55]: # Aplicar threshold agressivo
features_num_threshold_agressivo_10, removed_threshold_agressivo_10 = filtrar_por_threshold_agressivo(
    all_correlations,
    threshold=0.1
)
```

🔍 Aplicando threshold agressivo: $|p| < 0.1$

✗ Features removidas ($\max_{corr} < 0.1$): 62

📝 Features removidas (mostrando até 30):

• [FLAG_BIN]	flag_idade_invalida	P=+0.0998, S=+nan, max=0.0998
• [FLAG_BIN]	flag_has_logs_lag_1	P=+0.0101, S=-0.0949, max=0.0949
• [FLAG_BIN]	flag_has_logs_lag_2	P=+0.0289, S=-0.0940, max=0.0940
• [FLAG_BIN]	flag_gender_known	P=-0.0905, S=-0.0056, max=0.0905
• [FLAG_BIN]	flag_has_logs_lag_3	P=+0.0325, S=-0.0904, max=0.0904
• [CONTÍNUA]	avg_secs_per_unq_cap_ratio_ref_max_3	P=+0.0039, S=-0.0895, max=0.0895
• [FLAG_BIN]	flag_has_logs_lag_4	P=+0.0348, S=-0.0880, max=0.0880
• [FLAG_BIN]	flag_shallow_user_max_6	P=+0.0326, S=+0.0843, max=0.0843
• [CONTÍNUA]	plays_per_unq_cap_max_6	P=+0.0032, S=-0.0842, max=0.0842
• [FLAG_BIN]	flag_has_logs_lag_5	P=+0.0346, S=-0.0804, max=0.0804
• [FLAG_BIN]	flag_has_logs	P=-0.0775, S=+nan, max=0.0775
• [CONTÍNUA]	city	P=-0.0737, S=+0.0092, max=0.0737
• [CONTÍNUA]	completion_efficiency_max_3	P=+0.0017, S=-0.0669, max=0.0669
• [FLAG_BIN]	flag_shallow_user_max_3	P=+0.0250, S=+0.0666, max=0.0666
• [CONTÍNUA]	avg_secs_per_unq_cap_max_6	P=+0.0078, S=-0.0663, max=0.0663
• [CONTÍNUA]	catalog_exploration_ratio_cap_min_3	P=-0.0009, S=+0.0657, max=0.0657
• [CONTÍNUA]	completed_songs_rate_max_6	P=+0.0032, S=-0.0616, max=0.0616
• [FLAG_BIN]	flag_has_logs_max_3	P=-0.0616, S=+nan, max=0.0616
• [CONTÍNUA]	catalog_exploration_ratio_cap_ratio_ref_max_6	P=+0.0032, S=-0.0610, max=0.0610
• [CONTÍNUA]	log_total_plays_ratio_ref_min_6	P=+0.0334, S=+0.0585, max=0.0585
• [FLAG_BIN]	flag_has_logs_max_6	P=-0.0565, S=+nan, max=0.0565
• [FLAG_BIN]	flag_exemption_max_6	P=-0.0438, S=-0.0520, max=0.0520
• [CONTÍNUA]	early_drop_rate_min_3	P=-0.0035, S=+0.0514, max=0.0514
• [CONTÍNUA]	early_drop_rate_ratio_ref_mean_6	P=+0.0011, S=+0.0488, max=0.0488
• [CONTÍNUA]	catalog_exploration_ratio_cap_ratio_ref_max_3	P=-0.0009, S=-0.0473, max=0.0473
• [CONTÍNUA]	catalog_exploration_ratio_cap_ratio_ref_mean_6	P=+0.0032, S=+0.0461, max=0.0461
• [CONTÍNUA]	catalog_exploration_ratio_cap_ratio_ref_min_3	P=-0.0009, S=+0.0436, max=0.0436
• [CONTÍNUA]	completion_efficiency_ratio_ref_mean_6	P=+0.0055, S=+0.0415, max=0.0415
• [CONTÍNUA]	plays_per_unq_cap_ratio_ref_mean_6	P=+0.0032, S=-0.0378, max=0.0378
• [CONTÍNUA]	log_total_secs_ratio_ref_min_6	P=+0.0334, S=+0.0373, max=0.0373

✓ Features mantidas: 171

```
In [56]: print(f"\n📊 Resumo após Etapa 2:")
print(f"  • Após threshold 0.1: {len(features_num_threshold_agressivo_10)}")
```

📊 Resumo após Etapa 2:
• Após threshold 0.1: 171

Threshold de 20%

```
In [57]: # Aplicar threshold agressivo
features_num_threshold_agressivo_20, removed_threshold_agressivo_20 = filtrar_por_threshold_agressivo(
    all_correlations,
    threshold=0.2
)
```

🔍 Aplicando threshold agressivo: $|\rho| < 0.2$

✖ Features removidas ($\text{max_corr} < 0.2$): 114

📋 Features removidas (mostrando até 30):

- [CONTÍNUA] total_secs_ratio_ref_max_3 $P=+0.0228, S=-0.1961, \max=0.1961$
- [CONTÍNUA] total_secs_ratio_ref_mean_6 $P=+0.0334, S=-0.1919, \max=0.1919$
- [CONTÍNUA] plays_per_unq_cap_mean_3 $P=-0.0009, S=-0.1917, \max=0.1917$
- [CONTÍNUA] early_drop_rate_max_3 $P=-0.0035, S=+0.1915, \max=0.1915$
- [CONTÍNUA] total_plays_ratio_ref_mean_6 $P=+0.0334, S=-0.1893, \max=0.1893$
- [CONTÍNUA] total_plays_ratio_ref_max_3 $P=+0.0228, S=-0.1892, \max=0.1892$
- [CONTÍNUA] num_unq_ratio_ref_max_3 $P=+0.0228, S=-0.1883, \max=0.1883$
- [CONTÍNUA] plays_per_unq $P=-0.0349, S=-0.1868, \max=0.1868$
- [CONTÍNUA] plays_per_unq_cap $P=-0.0677, S=-0.1868, \max=0.1868$
- [CONTÍNUA] num_unq_ratio_ref_mean_6 $P=+0.0334, S=-0.1832, \max=0.1832$
- [CONTÍNUA] plays_per_unq_cap_mean_6 $P=+0.0032, S=-0.1809, \max=0.1809$
- [CONTÍNUA] payment_plan_days $P=+0.1142, S=-0.1801, \max=0.1801$
- [CONTÍNUA] catalog_exploration_ratio $P=+0.0077, S=+0.1779, \max=0.1779$
- [CONTÍNUA] catalog_exploration_ratio_cap $P=+0.0139, S=+0.1779, \max=0.1779$
- [CONTÍNUA] completed_songs_rate $P=-0.1109, S=-0.1753, \max=0.1753$
- [CONTÍNUA] avg_secs_per_unq_cap_mean_3 $P=+0.0037, S=-0.1742, \max=0.1742$
- [CONTÍNUA] completed_songs_rate_ratio_ref_max_6 $P=+0.0032, S=-0.1728, \max=0.1728$
- [CONTÍNUA] log_avg_secs_per_unq $P=-0.0880, S=-0.1709, \max=0.1709$
- [CONTÍNUA] avg_secs_per_unq $P=-0.0334, S=-0.1709, \max=0.1709$
- [CONTÍNUA] avg_secs_per_unq_cap $P=-0.0618, S=-0.1709, \max=0.1709$
- [CONTÍNUA] completion_efficiency_ratio_ref_max_6 $P=+0.0055, S=-0.1684, \max=0.1684$
- [CONTÍNUA] avg_secs_per_unq_cap_mean_6 $P=+0.0076, S=-0.1646, \max=0.1646$
- [CONTÍNUA] catalog_exploration_ratio_cap_mean_3 $P=-0.0009, S=+0.1619, \max=0.1619$
- [CONTÍNUA] catalog_exploration_ratio_cap_mean_6 $P=+0.0032, S=+0.1556, \max=0.1556$
- [CONTÍNUA] tenure_meses $P=+0.0710, S=+0.1528, \max=0.1528$
- [CONTÍNUA] completion_efficiency $P=-0.0954, S=-0.1515, \max=0.1515$
- [CONTÍNUA] early_drop_rate_mean_3 $P=-0.0035, S=+0.1499, \max=0.1499$
- [FLAG_TEND] flag_has_logs_sum_6 $P=+0.0216, S=-0.1486, \max=0.1486$
- [FLAG_TEND] flag_has_logs_mean_6 $P=+0.0216, S=-0.1486, \max=0.1486$
- [CONTÍNUA] early_drop_rate_mean_6 $P=+0.0011, S=+0.1434, \max=0.1434$

✓ Features mantidas: 119

```
In [58]: print(f"\n📊 Resumo após Etapa 2:")
print(f"  • Após threshold 0.2: {len(features_num_threshold_agressivo_20)}")
```

📊 Resumo após Etapa 2:

- Após threshold 0.2: 119

Threshold de 25%

```
In [59]: # Aplicar threshold agressivo
features_num_threshold_agressivo_25, removed_threshold_agressivo_25 = filtrar_por_threshold_agressivo(
    all_correlations,
    threshold=0.25
)
```

🔍 Aplicando threshold agressivo: $|\rho| < 0.25$

✖ Features removidas ($\text{max_corr} < 0.25$): 130

📋 Features removidas (mostrando até 30):

- [CONTÍNUA] completed_songs_rate_min_3
- [CONTÍNUA] completion_efficiency_min_3
- [FLAG_BIN] flag_high_value_registered_via
- [CONTÍNUA] avg_secs_per_unq_cap_min_3
- [FLAG_BIN] flag_plano_mensal_lag_5
- [CONTÍNUA] catalog_exploration_ratio_cap_max_6
- [CONTÍNUA] completion_efficiency_mean_6
- [FLAG_BIN] flag_valid_fee_lag_5
- [FLAG_BIN] flag_has_transactions_lag_5
- [CONTÍNUA] log_total_plays_ratio_ref_max_3
- [CONTÍNUA] early_drop_rate_max_6
- [CONTÍNUA] catalog_exploration_ratio_cap_max_3
- [CONTÍNUA] log_total_secs_ratio_ref_max_3
- [CONTÍNUA] completed_songs_rate_mean_3
- [CONTÍNUA] completion_efficiency_mean_3
- [CONTÍNUA] completed_songs_rate_mean_6
- [CONTÍNUA] total_secs_ratio_ref_max_3
- [CONTÍNUA] total_secs_ratio_ref_mean_6
- [CONTÍNUA] plays_per_unq_cap_mean_3
- [CONTÍNUA] early_drop_rate_max_3
- [CONTÍNUA] total_plays_ratio_ref_mean_6
- [CONTÍNUA] total_plays_ratio_ref_max_3
- [CONTÍNUA] num_unq_ratio_ref_max_3
- [CONTÍNUA] plays_per_unq
- [CONTÍNUA] plays_per_unq_cap
- [CONTÍNUA] num_unq_ratio_ref_mean_6
- [CONTÍNUA] plays_per_unq_cap_mean_6
- [CONTÍNUA] payment_plan_days
- [CONTÍNUA] catalog_exploration_ratio
- [CONTÍNUA] catalog_exploration_ratio_cap

P=-0.0009, S=-0.2493, max=0.2493
P=+0.0017, S=-0.2482, max=0.2482
P=-0.1243, S=+0.2453, max=0.2453
P=+0.0037, S=-0.2401, max=0.2401
P=+0.2357, S=+0.0298, max=0.2357
P=+0.0032, S=+0.2273, max=0.2273
P=+0.0055, S=-0.2198, max=0.2198
P=+0.2153, S=+0.0276, max=0.2153
P=+0.2147, S=+0.0269, max=0.2147
P=+0.0228, S=-0.2115, max=0.2115
P=+0.0011, S=+0.2104, max=0.2104
P=-0.0009, S=+0.2099, max=0.2099
P=+0.0228, S=-0.2096, max=0.2096
P=-0.0009, S=-0.2080, max=0.2080
P=+0.0017, S=-0.2075, max=0.2075
P=+0.0032, S=-0.2073, max=0.2073
P=+0.0228, S=-0.1961, max=0.1961
P=+0.0334, S=-0.1919, max=0.1919
P=-0.0009, S=-0.1917, max=0.1917
P=-0.0035, S=+0.1915, max=0.1915
P=+0.0334, S=-0.1893, max=0.1893
P=+0.0228, S=-0.1892, max=0.1892
P=+0.0228, S=-0.1883, max=0.1883
P=-0.0349, S=-0.1868, max=0.1868
P=-0.0677, S=-0.1868, max=0.1868
P=+0.0334, S=-0.1832, max=0.1832
P=+0.0032, S=-0.1809, max=0.1809
P=+0.1142, S=-0.1801, max=0.1801
P=+0.0077, S=+0.1779, max=0.1779
P=+0.0139, S=+0.1779, max=0.1779

✓ Features mantidas: 103

```
In [60]: print(f"\n📊 Resumo após Etapa 2:")
print(f"  • Após threshold 0.25: {len(features_num_threshold_agressivo_25)}")
```

📊 Resumo após Etapa 2:

- Após threshold 0.25: 103

Threshold de 50%

```
In [61]: # Aplicar threshold agressivo
features_num_threshold_agressivo_50, removed_threshold_agressivo_50 = filtrar_por_threshold_agressivo(
    all_correlations,
    threshold=0.5
)
```

🔍 Aplicando threshold agressivo: $|\rho| < 0.5$

✖ Features removidas ($\text{max_corr} < 0.5$): 187

📋 Features removidas (mostrando até 30):

• [CONTÍNUA]	num_unq_min_3	P=+0.0206, S=-0.4991, max=0.4991
• [CONTÍNUA]	total_plays_max_6	P=+0.0289, S=-0.4982, max=0.4982
• [CONTÍNUA]	log_total_plays_max_6	P=+0.0334, S=-0.4982, max=0.4982
• [CONTÍNUA]	total_secs_max_6	P=-0.1657, S=-0.4975, max=0.4975
• [CONTÍNUA]	log_total_secs_max_6	P=+0.0334, S=-0.4975, max=0.4975
• [FLAG_TEND]	flag_plano_mensal_sum_6	P=+0.4929, S=+0.0437, max=0.4929
• [FLAG_TEND]	flag_plano_mensal_mean_6	P=+0.4929, S=+0.0437, max=0.4929
• [CONTÍNUA]	usage_intensity_per_tenure_cap	P=-0.1124, S=-0.4892, max=0.4892
• [FLAG_BIN]	flag_has_transactions_lag_1	P=+0.4863, S=+0.1168, max=0.4863
• [FLAG_BIN]	flag_valid_fee_lag_1	P=+0.4860, S=+0.1166, max=0.4860
• [CONTÍNUA]	num_unq_max_6	P=+0.0302, S=-0.4845, max=0.4845
• [CONTÍNUA]	daily_revenue_efficiency_min_3	P=+0.4690, S=+0.4830, max=0.4830
• [CONTÍNUA]	daily_revenue_efficiency_mean_3	P=+0.4690, S=+0.4807, max=0.4807
• [CONTÍNUA]	log_total_secs_min_6	P=+0.0334, S=-0.4794, max=0.4794
• [CONTÍNUA]	total_secs_min_6	P=-0.1687, S=-0.4794, max=0.4794
• [CONTÍNUA]	total_plays_min_6	P=+0.0306, S=-0.4763, max=0.4763
• [CONTÍNUA]	log_total_plays_min_6	P=+0.0334, S=-0.4763, max=0.4763
• [CONTÍNUA]	daily_revenue_efficiency_max_3	P=+0.4690, S=+0.4675, max=0.4690
• [CONTÍNUA]	daily_revenue_efficiency_ratio_ref_max_3	P=+0.4690, S=+0.1855, max=0.4690
• [CONTÍNUA]	daily_revenue_efficiency_ratio_ref_min_3	P=+0.4690, S=+0.1381, max=0.4690
• [CONTÍNUA]	daily_revenue_efficiency_ratio_ref_mean_3	P=+0.4690, S=-0.1086, max=0.4690
• [CONTÍNUA]	num_unq_min_6	P=+0.0314, S=-0.4645, max=0.4645
• [FLAG_TEND]	flag_valid_fee_mean_6	P=+0.4601, S=+0.0344, max=0.4601
• [FLAG_TEND]	flag_valid_fee_sum_6	P=+0.4601, S=+0.0344, max=0.4601
• [FLAG_TEND]	flag_has_transactions_sum_6	P=+0.4588, S=+0.0254, max=0.4588
• [FLAG_TEND]	flag_has_transactions_mean_6	P=+0.4588, S=+0.0254, max=0.4588
• [CONTÍNUA]	daily_revenue_efficiency_min_6	P=+0.3852, S=+0.4522, max=0.4522
• [CONTÍNUA]	daily_revenue_efficiency_max_6	P=+0.3852, S=+0.4390, max=0.4390
• [CONTÍNUA]	daily_revenue_efficiency_mean_6	P=+0.3852, S=+0.4299, max=0.4299
• [FLAG_BIN]	flag_plano_mensal_lag_2	P=+0.4194, S=+0.0978, max=0.4194

✓ Features mantidas: 46

```
In [62]: print(f"\n📊 Resumo após Etapa 2:")
print(f"  • Após threshold 0.5: {len(features_num_threshold_agressivo_50)}")
```

📊 Resumo após Etapa 2:
 • Após threshold 0.5: 46

Threshold	Features (sem _raw)	Avaliação	Recomendação
0.05 201 ✖ EXCESSIVO - Overfitting garantido ✖ Não usar			
0.10 166 ✖ MUITO ALTO - Multicolinearidade severa ✖ Não usar			
0.20 98 ⚠ ALTO - Ainda tem redundância ⚠ Precisa Filtros 2+3			
0.25 110 ✓ BOM - Ponto de equilíbrio ✓ RECOMENDADO			
0.50 62 ⚠ AGRESSIVO - Perda de features importantes ⚠ Muito restritivo			

Remover Redundância de Features de Tendência

```

In [63]: def remover_redundancia_tendencia(correlations_sorted, features_mantidas):
    """
    Para cada feature base (ex: total_secs), mantém apenas:
    - A versão original (sem sufixo de tendência)
    - A versão de tendência com MAIOR correlação

    Remove: _mean_3, _mean_6, _max_3, _max_6, _sum_3, _sum_6 redundantes
    Remove: _lag_1, _lag_2, etc. redundantes (mantém apenas top 1)

    NOTA: _lag_0 e _raw já foram removidos na Etapa 1
    """
    print(f"\n🔍 Identificando famílias de features de tendência...")

    # Criar dicionário: feature_base -> [(versão, correlação)]
    familias = {}

    for col, p, s, abs_p, abs_s, max_c in correlations_sorted:
        if col not in features_mantidas:
            continue

        # Identificar feature base (remover sufixos de tendência)
        base = col
        for sufixo in ['_mean_3', '_mean_6', '_max_3', '_max_6', '_sum_3', '_sum_6', '_min_3', '_min_6', '_lag_1', '_lag_2', '_lag_3', '_lag_4', '_lag_5']:
            if col.endswith(sufixo):
                base = col.replace(sufixo, '')
                break

        if base not in familias:
            familias[base] = []

        familias[base].append((col, max_c))

    # Para cada família, manter apenas:
    # 1. Versão original (sem sufixo)
    # 2. Top 1 versão de tendência (_mean, _max, _sum)
    # 3. Top 1 versão de lag

    features_a_manter = set()
    features_a_remover = []

    for base, versoes in familias.items():
        if len(versoes) == 1:
            # Apenas uma versão, manter
            features_a_manter.add(versoes[0][0])
            continue

        # Separar por tipo
        versao_original = None
        versoes_tendencia = [] # _mean, _max, _sum
        versoes_lag = []

        for col, corr in versoes:
            if col == base:
                versao_original = (col, corr)
            elif any(x in col for x in ['_mean_', '_max_', '_sum_', '_min_']):
                versoes_tendencia.append((col, corr))
            elif '_lag_' in col:
                versoes_lag.append((col, corr))
            else:
                # Outras versões (ex: log_)
                features_a_manter.add(col)

        # Manter versão original
        if versao_original:
            features_a_manter.add(versao_original[0])

        # Manter top 1 tendência
        if versoes_tendencia:
            versoes_tendencia_sorted = sorted(versoes_tendencia, key=lambda x: x[1], reverse=True)
            features_a_manter.add(versoes_tendencia_sorted[0][0])

            # Remover as outras
            for col, corr in versoes_tendencia_sorted[1:]:
                features_a_remover.append((col, corr))

        # Manter top 1 lag
        if versoes_lag:
            versoes_lag_sorted = sorted(versoes_lag, key=lambda x: x[1], reverse=True)
            features_a_manter.add(versoes_lag_sorted[0][0])

            # Remover as outras
            for col, corr in versoes_lag_sorted[1:]:
                features_a_remover.append((col, corr))

    print(f"\n📊 Análise de famílias:")
    print(f" • Famílias identificadas: {len(familias)}")

```

```

print(f" • Features a manter: {len(features_a_manter)}")
print(f" • Features a remover (redundância): {len(features_a_remover)}")

if features_a_remover:
    print(f"\n💡 Features removidas por redundância (mostrando até 30):")
    features_a_remover_sorted = sorted(features_a_remover, key=lambda x: x[1], reverse=True)
    for col, corr in features_a_remover_sorted[:30]:
        print(f" • {col}: {max_corr} max_corr = {corr:.4f}")

return list(features_a_manter), [col for col, corr in features_a_remover]

```

```
In [64]: print("\n" + "=" * 80)
print("💡 ETAPA 3: REMOÇÃO DE REDUNDÂNCIA EM FEATURES DE TENDÊNCIA")
print("=" * 80)
```

```
# Aplicar
features_num_final_etapa3, removed_redundancia = remover_redundancia_tendencia(
    all_correlations,
    features_num_threshold_agressivo_25
)
```

```
print(f"\n✅ Features numéricas finais após Etapa 3: {len(features_num_final_etapa3)})")
```

```
print("=" * 80)
```

```
=====
💡 ETAPA 3: REMOÇÃO DE REDUNDÂNCIA EM FEATURES DE TENDÊNCIA
=====
```

🔍 Identificando famílias de features de tendência...

📊 Análise de famílias:

- Famílias identificadas: 31
- Features a manter: 43
- Features a remover (redundância): 60

📋 Features removidas por redundância (mostrando até 30):

• flag_plano_mensal_max_6	max_corr = 0.6761
• flag_plano_mensal_sum_3	max_corr = 0.6090
• flag_plano_mensal_mean_3	max_corr = 0.6090
• flag_valid_fee_mean_3	max_corr = 0.5749
• flag_valid_fee_sum_3	max_corr = 0.5749
• flag_has_transactions_mean_3	max_corr = 0.5748
• flag_has_transactions_sum_3	max_corr = 0.5748
• total_secs_max_3	max_corr = 0.5294
• log_total_secs_max_3	max_corr = 0.5294
• total_plays_max_3	max_corr = 0.5275
• log_total_plays_max_3	max_corr = 0.5275
• total_secs_mean_6	max_corr = 0.5207
• total_plays_mean_6	max_corr = 0.5190
• log_total_secs_mean_6	max_corr = 0.5187
• log_total_secs_min_3	max_corr = 0.5173
• total_secs_min_3	max_corr = 0.5173
• log_total_plays_mean_6	max_corr = 0.5158
• total_plays_min_3	max_corr = 0.5131
• log_total_plays_min_3	max_corr = 0.5131
• num_unq_max_3	max_corr = 0.5121
• flag_valid_fee_max_6	max_corr = 0.5120
• flag_has_transactions_max_6	max_corr = 0.5107
• num_unq_mean_6	max_corr = 0.5044
• num_unq_min_3	max_corr = 0.4991
• total_plays_max_6	max_corr = 0.4982
• log_total_plays_max_6	max_corr = 0.4982
• total_secs_max_6	max_corr = 0.4975
• log_total_secs_max_6	max_corr = 0.4975
• flag_plano_mensal_sum_6	max_corr = 0.4929
• flag_plano_mensal_mean_6	max_corr = 0.4929

✅ Features numéricas finais após Etapa 3: 43

```
=====
```

Conclusao

In [65]:

```

# =====
# VERIFICAÇÃO FINAL
# =====

print("\n" + "=" * 80)
print("🔍 VERIFICAÇÃO: FEATURES NUMÉRICAS FINAIS")
print("=" * 80)

print(f"\n💡 Lista completa das features mantidas (ordenadas por correlação):")
count = 0
for col, p, s, abs_p, abs_s, max_c in all_correlations:
    if col in features_num_final_etapa3:
        count += 1
        tipo = "[FLAG_BIN]" if col.startswith('flag_') and not any(x in col for x in ['_mean_', '_sum_']) else \
            "[FLAG_TEND]" if col.startswith('flag_') else "[CONTÍNUA]"
        metodo = "☒ P" if abs_p > abs_s else "📊 S"
        print(f" {count:2d}. {tipo:13s} {col:5s} P={p:+.4f}, S={s:+.4f}, max={max_c:.4f} {metodo}")

print("\n" + "=" * 80)

# Análise por tipo de feature
print("\n📊 ANÁLISE POR TIPO DE FEATURE:")
print("=" * 80)

flags_bin = [col for col in features_num_final_etapa3 if col.startswith('flag_') and not any(x in col for x in ['_mean_', '_sum_', '_max_', '_min_'])]
flags_tend = [col for col in features_num_final_etapa3 if col.startswith('flag_') and any(x in col for x in ['_mean_', '_sum_', '_max_', '_min_'])]
continuas = [col for col in features_num_final_etapa3 if not col.startswith('flag_')]

print(f"\n • Flags Binárias (puras): {len(flags_bin)}")
print(f" • Flags de Tendência: {len(flags_tend)}")
print(f" • Features Continuas: {len(continuas)}")
print(f" • TOTAL: {len(features_num_final_etapa3)}")

# Análise por origem
print("\n📊 ANÁLISE POR ORIGEM:")
print("=" * 80)

logs_features = [col for col in features_num_final_etapa3 if any(x in col for x in ['total_secs', 'total_plays', 'num_unq', 'num_25', 'num_50', 'num_75', 'num_985', 'num_100', 'catalog_', 'plays_per_unq', 'avg_secs', 'completed_songs', 'completion_'])]
transactions_features = [col for col in features_num_final_etapa3 if any(x in col for x in ['flag_plano_mensal', 'flag_valid_fee', 'flag_has_transactions', 'flag_exemption', 'is_auto_renew', 'payment_', 'actual_amount', 'plan_list_price', 'daily_revenue'])]
members_features = [col for col in features_num_final_etapa3 if any(x in col for x in ['city', 'bd', 'gender', 'registered_via', 'registration_', 'idade', 'tenure', 'flag_long_tenure'])]

print(f"\n • Features de Logs: {len(logs_features)}")
print(f" • Features de Transactions: {len(transactions_features)}")
print(f" • Features de Members: {len(members_features)}")

print("\n" + "=" * 80)

```

```

=====
🔍 VERIFICAÇÃO: FEATURES NUMÉRICAS FINAIS
=====
```

💡 Lista completa das features mantidas (ordenadas por correlação):

1. [CONTÍNUA]	margem_liquida_mensal	P=+0.2331, S=+0.7084, max=0.7084	📊 S
2. [FLAG_BIN]	flag_plano_mensal_max_3	P=+0.6946, S=+0.1860, max=0.6946	☒ P
3. [CONTÍNUA]	daily_revenue_efficiency	P=+0.6556, S=+0.5098, max=0.6556	☒ P
4. [FLAG_BIN]	flag_plano_mensal	P=+0.6475, S=+0.2137, max=0.6475	☒ P
5. [FLAG_BIN]	flag_valid_fee	P=+0.6003, S=+0.0389, max=0.6003	☒ P
6. [FLAG_BIN]	flag_has_transactions	P=+0.5998, S=+nan, max=0.5998	📊 S
7. [CONTÍNUA]	revenue_per_hour_listened_cap	P=+0.2139, S=+0.5909, max=0.5909	📊 S
8. [FLAG_BIN]	flag_valid_fee_max_3	P=+0.5892, S=+0.0230, max=0.5892	☒ P
9. [FLAG_BIN]	flag_has_transactions_max_3	P=+0.5887, S=+nan, max=0.5887	📊 S
10. [CONTÍNUA]	payment_method_id	P=+0.5801, S=-0.2264, max=0.5801	☒ P
11. [CONTÍNUA]	log_total_secs	P=-0.1448, S=-0.5517, max=0.5517	📊 S
12. [CONTÍNUA]	total_secs	P=-0.2620, S=-0.5517, max=0.5517	📊 S
13. [CONTÍNUA]	num_100	P=-0.2603, S=-0.5477, max=0.5477	📊 S
14. [CONTÍNUA]	total_plays	P=-0.2588, S=-0.5471, max=0.5471	📊 S
15. [CONTÍNUA]	log_total_plays	P=-0.1749, S=-0.5471, max=0.5471	📊 S
16. [CONTÍNUA]	total_secs_mean_3	P=-0.1846, S=-0.5407, max=0.5407	📊 S
17. [CONTÍNUA]	log_total_secs_mean_3	P=+0.0228, S=-0.5382, max=0.5382	📊 S
18. [CONTÍNUA]	total_plays_mean_3	P=+0.0191, S=-0.5377, max=0.5377	📊 S
19. [CONTÍNUA]	log_total_plays_mean_3	P=+0.0228, S=-0.5345, max=0.5345	📊 S
20. [FLAG_BIN]	flag_plano_mensal_lag_1	P=+0.5313, S=+0.1171, max=0.5313	☒ P
21. [CONTÍNUA]	num_unq	P=-0.2511, S=-0.5303, max=0.5303	📊 S
22. [CONTÍNUA]	num_unq_mean_3	P=+0.0201, S=-0.5221, max=0.5221	📊 S
23. [CONTÍNUA]	is_auto_renew	P=+0.5119, S=+0.0887, max=0.5119	☒ P
24. [CONTÍNUA]	usage_intensity_per_tenure_cap	P=-0.1124, S=-0.4892, max=0.4892	📊 S
25. [FLAG_BIN]	flag_has_transactions_lag_1	P=+0.4863, S=+0.1168, max=0.4863	☒ P
26. [FLAG_BIN]	flag_valid_fee_lag_1	P=+0.4860, S=+0.1166, max=0.4860	☒ P
27. [CONTÍNUA]	daily_revenue_efficiency_min_3	P=+0.4690, S=+0.4830, max=0.4830	📊 S
28. [CONTÍNUA]	daily_revenue_efficiency_ratio_ref_max_3	P=+0.4690, S=+0.1855, max=0.4690	☒ P
29. [CONTÍNUA]	plan_list_price	P=+0.1868, S=+0.3896, max=0.3896	📊 S

30. [CONTÍNUA]	actual_amount_paid	P=+0.1887, S=+0.3871, max=0.3871	S
31. [CONTÍNUA]	num_985	P=-0.1477, S=-0.3849, max=0.3849	S
32. [CONTÍNUA]	num_75	P=-0.1494, S=-0.3816, max=0.3816	S
33. [CONTÍNUA]	num_50	P=-0.1369, S=-0.3573, max=0.3573	S
34. [CONTÍNUA]	num_25	P=-0.1230, S=-0.3394, max=0.3394	S
35. [CONTÍNUA]	log_total_plays_ratio_ref_max_6	P=+0.0334, S=-0.2987, max=0.2987	S
36. [CONTÍNUA]	log_total_secs_ratio_ref_max_6	P=+0.0334, S=-0.2905, max=0.2905	S
37. [CONTÍNUA]	total_secs_ratio_ref_max_6	P=+0.0334, S=-0.2713, max=0.2713	S
38. [CONTÍNUA]	completion_efficiency_min_6	P=+0.0055, S=-0.2703, max=0.2703	S
39. [CONTÍNUA]	total_plays_ratio_ref_max_6	P=+0.0334, S=-0.2676, max=0.2676	S
40. [CONTÍNUA]	plays_per_unq_cap_min_6	P=+0.0032, S=-0.2675, max=0.2675	S
41. [CONTÍNUA]	completed_songs_rate_min_6	P=+0.0032, S=-0.2664, max=0.2664	S
42. [CONTÍNUA]	num_unq_ratio_ref_max_6	P=+0.0334, S=-0.2625, max=0.2625	S
43. [CONTÍNUA]	avg_secs_per_unq_cap_min_6	P=+0.0076, S=-0.2562, max=0.2562	S

=====

📊 ANÁLISE POR TIPO DE FEATURE:

- Flags Binárias (puras): 6
- Flags de Tendência: 3
- Features Contínuas: 34
- TOTAL: 43

📊 ANÁLISE POR ORIGEM:

- Features de Logs: 24
- Features de Transactions: 16
- Features de Members: 1

=====

Decididas as variáveis contínuas finais.

```
In [66]: # Salvar lista final
features_num_corr_filtradas_final = features_num_final_etapa3
```

2.2. FEATURES CATEGÓRICAS: Cramer's V com Target Binário

```
In [67]: def calcular_cramer_v_target(df, features_categoricas, target_col='target_win', threshold=0.05):
    """
    Calcula Cramer's V entre cada feature categórica e o target (binarizado)

    Estratégia:
    1. Binarizar o target em "alto" vs "baixo" (mediana como corte)
    2. Calcular Cramer's V entre cada categórica e o target binário
    3. Remover features com V < threshold
    """

    print(f"\n🔍 Calculando Cramer's V para {len(features_categoricas)} features categóricas...")
    print(f"  Threshold: V < {threshold}")

    # Binarizar target (mediana como corte)
    median_target = df.select(F.expr(f"percentile_approx({target_col}, 0.5)").collect()[0][0])
    df_with_target_bin = df.withColumn(
        "target_bin",
        F.when(F.col(target_col) >= median_target, "alto").otherwise("baixo")
    )

    print(f"  Target binarizado: mediana = {median_target:.2f}")

    cramer_values = []

    for col in features_categoricas:
        print(f"    Calculando Cramer's V: {col} vs target_bin...")

        try:
            v = calcular_cramer_v(df_with_target_bin, col, "target_bin")

            # Verificar se retornou NaN (muitos valores únicos)
            if np.isnan(v):
                print(f"      🚨 {col} pulado (muitos valores únicos ou erro)")
                cramer_values.append((col, 0.0))
            else:
                cramer_values.append((col, v))
        except Exception as e:
            print(f"      🚨 Erro ao calcular {col}: {e}")
            cramer_values.append((col, 0.0))

    # Ordenar por Cramer's V (decrescente)
    cramer_values_sorted = sorted(cramer_values, key=lambda x: x[1], reverse=True)

    # Identificar features com baixa associação
    low_cramer_features = [col for col, v in cramer_values_sorted if v < threshold]

    print(f"\n✖ Features com Cramer's V < {threshold}: {len(low_cramer_features)}")

    if low_cramer_features:
        print(f"\n⚠️ Features removidas:")
        for col, v in cramer_values_sorted:
            if col in low_cramer_features:
                print(f"  • {col:50s} V = {v:.4f}")

    print(f"\n🏆 Ranking de features categóricas por Cramer's V:")
    for i, (col, v) in enumerate(cramer_values_sorted, 1):
        status = "✅" if col not in low_cramer_features else "✖"
        print(f"  {i:2d}. {status} {col:50s} V = {v:.4f}")

    # Features mantidas
    features_mantidas = [col for col in features_categoricas if col not in low_cramer_features]

    print(f"\n✅ Features categóricas mantidas: {len(features_mantidas)}")

    return features_mantidas, cramer_values_sorted, low_cramer_features
```

Threshold de 5%

In [68]:

```
# Executar
features_cat_corr_filtradas, cramer_values, removed_low_cramer = calcular_cramer_v_target(
    df_feature_store_intermediate,
    features_cat_filtradas,
    target_col='target_win',
    threshold=0.05
)
```

Calculando Cramer's V para 22 features categóricas...

Threshold: $V < 0.05$

Target binarizado: mediana = 60.41

Calculando Cramer's V: total_plays_group vs target_bin...

Calculando Cramer's V: completed_songs_rate_group vs target_bin...

Calculando Cramer's V: avg_secs_per_unq_cap_group vs target_bin...

Calculando Cramer's V: plays_per_unq_behavior vs target_bin...

Calculando Cramer's V: plays_behavior_vs_volume vs target_bin...

Calculando Cramer's V: plays_behavior_vs_volume_collapsed vs target_bin...

Calculando Cramer's V: plays_behavior_vs_completion vs target_bin...

Calculando Cramer's V: plays_behavior_vs_completion_collapsed vs target_bin...

Calculando Cramer's V: early_drop_rate_group vs target_bin...

Calculando Cramer's V: revenue_tier vs target_bin...

Calculando Cramer's V: payment_method_group vs target_bin...

Calculando Cramer's V: payment_price_regime vs target_bin...

Calculando Cramer's V: gender_clean vs target_bin...

Calculando Cramer's V: faixa_idade vs target_bin...

Calculando Cramer's V: registered_via_group vs target_bin...

Calculando Cramer's V: registration_year_regime vs target_bin...

Calculando Cramer's V: tenure_faixa vs target_bin...

Calculando Cramer's V: revenue_per_hour_tier vs target_bin...

Calculando Cramer's V: usage_intensity_tier vs target_bin...

Calculando Cramer's V: membership_expire_date vs target_bin...

Pulei membership_expire_date: muitos valores únicos (464). Verifique se é quantitativa.

⚠️ membership_expire_date pulado (muitos valores únicos ou erro)

Calculando Cramer's V: transaction_date vs target_bin...

Pulei transaction_date: muitos valores únicos (336). Verifique se é quantitativa.

⚠️ transaction_date pulado (muitos valores únicos ou erro)

Calculando Cramer's V: registration_init_time vs target_bin...

Pulei registration_init_time: muitos valores únicos (4630). Verifique se é quantitativa.

⚠️ registration_init_time pulado (muitos valores únicos ou erro)

✖️ Features com Cramer's V < 0.05: 3

📝 Features removidas:

- membership_expire_date $V = 0.0000$
- transaction_date $V = 0.0000$
- registration_init_time $V = 0.0000$

🏆 Ranking de features categóricas por Cramer's V:

1. ✓ revenue_tier	$V = 0.7395$
2. ✓ payment_method_group	$V = 0.3113$
3. ✓ revenue_per_hour_tier	$V = 0.2934$
4. ✓ tenure_faixa	$V = 0.2919$
5. ✓ payment_price_regime	$V = 0.2867$
6. ✓ registration_year_regime	$V = 0.2778$
7. ✓ registered_via_group	$V = 0.2253$
8. ✓ usage_intensity_tier	$V = 0.2095$
9. ✓ gender_clean	$V = 0.1958$
10. ✓ faixa_idade	$V = 0.1956$
11. ✓ plays_behavior_vs_volume	$V = 0.1709$
12. ✓ plays_behavior_vs_volume_collapsed	$V = 0.1662$
13. ✓ total_plays_group	$V = 0.1551$
14. ✓ plays_behavior_vs_completion	$V = 0.1021$
15. ✓ plays_behavior_vs_completion_collapsed	$V = 0.1020$
16. ✓ completed_songs_rate_group	$V = 0.0868$
17. ✓ plays_per_unq_behavior	$V = 0.0841$
18. ✓ early_drop_rate_group	$V = 0.0782$
19. ✓ avg_secs_per_unq_cap_group	$V = 0.0732$
20. ✖️ membership_expire_date	$V = 0.0000$
21. ✖️ transaction_date	$V = 0.0000$
22. ✖️ registration_init_time	$V = 0.0000$

✓ Features categóricas mantidas: 19

```
In [69]: print("\n" + "=" * 80)
print("📊 RESUMO CAMADA 2.2")
print("=" * 80)
print(f" • Features iniciais: {len(features_cat_filtradas)}")
print(f" • Features removidas: {len(removed_low_cramer)}")
print(f" • Features mantidas: {len(features_cat_corr_filtradas)}")
print(f" • Taxa de redução: {len(removed_low_cramer) / len(features_cat_filtradas) * 100:.1f}%")
print("=" * 80)
```

```
=====
📊 RESUMO CAMADA 2.2
=====
• Features iniciais: 22
• Features removidas: 3
• Features mantidas: 19
• Taxa de redução: 13.6%
=====
```

Conclusao - Criação da Feature Store Finalista - Antecedente as ABTs por modelo

Consolidar lista de features finalistas

```
In [70]: print("\n📝 ETAPA 1: Consolidando features finalistas...")

# Features numéricas (41)
features_numericas_finalistas = features_num_final_etapa3

# Features categóricas (17)
features_categoricas_finalistas = features_cat_filtradas

# Colunas de controle (sempre manter)
colunas_controle = ['msno', 'safra', 'target', 'target_win']

# Lista completa de colunas
colunas_finalistas = colunas_controle + features_numericas_finalistas + features_categoricas_finalistas

print(f"\n✅ Features consolidadas:")
print(f" • Colunas de controle: {len(colunas_controle)}")
print(f" • Features numéricas: {len(features_numericas_finalistas)}")
print(f" • Features categóricas: {len(features_categoricas_finalistas)}")
print(f" • TOTAL de colunas: {len(colunas_finalistas)}")
```

📝 ETAPA 1: Consolidando features finalistas...

✅ Features consolidadas:
• Colunas de controle: 4
• Features numéricas: 43
• Features categóricas: 22
• TOTAL de colunas: 69

Verificar se todas as colunas existem no DataFrame

```
In [71]: print("\n" + "=" * 80)
print("📝 ETAPA 2: Verificando existência das colunas...")
print("=" * 80)

# Colunas disponíveis no DataFrame
colunas_disponiveis = df_feature_store_intermediate.columns

# Verificar quais colunas estão faltando
colunas_faltando = [col for col in colunas_finalistas if col not in colunas_disponiveis]

if colunas_faltando:
    print(f"\n⚠️ ATENÇÃO: {len(colunas_faltando)} colunas não encontradas no DataFrame:")
    for col in colunas_faltando[:20]:
        print(f" • {col}")

    # Remover colunas faltando
    colunas_finalistas = [col for col in colunas_finalistas if col in colunas_disponiveis]
    print(f"\n✅ Colunas ajustadas: {len(colunas_finalistas)}")
else:
    print(f"\n✅ Todas as {len(colunas_finalistas)} colunas existem no DataFrame!")
```

```
=====
📝 ETAPA 2: Verificando existência das colunas...
=====

✅ Todas as 69 colunas existem no DataFrame!
```

```
#### Criar DataFrame finalista
```

```
In [72]: print("\n" + "=" * 80)
print("📝 ETAPA 3: Criando DataFrame finalista...")
print("=" * 80)

# Selecionar apenas as colunas finalistas
df_feature_store_final = df_feature_store_intermediate.select(colunas_finalistas)

print(f"\n✅ DataFrame finalista criado!")
print(f" • Total de linhas: {df_feature_store_final.count():,}")
print(f" • Total de colunas: {len(df_feature_store_final.columns)}")
```

```
=====
📝 ETAPA 3: Criando DataFrame finalista...
=====

✅ DataFrame finalista criado!
• Total de linhas: 9,678,866
• Total de colunas: 69
```

```
#### Verificar schema e tipos
```

```
In [73]: print("\n" + "=" * 80)
print("📝 ETAPA 4: Verificando schema do DataFrame finalista...")
print("=" * 80)

print(f"\n📊 Schema (mostrando primeiras 20 colunas):")
df_feature_store_final.select(df_feature_store_final.columns[:20]).printSchema()

# Contar tipos de colunas
from pyspark.sql.types import StringType, IntegerType, DoubleType, FloatType, LongType

tipos_colunas = {}
for field in df_feature_store_final.schema.fields:
    tipo = type(field.dataType).__name__
    if tipo not in tipos_colunas:
        tipos_colunas[tipo] = 0
    tipos_colunas[tipo] += 1

print(f"\n📊 Distribuição de tipos:")
for tipo, count in sorted(tipos_colunas.items(), key=lambda x: x[1], reverse=True):
    print(f" • {tipo}: {count} colunas")
```

```
=====
📝 ETAPA 4: Verificando schema do DataFrame finalista...
=====
```

```
📊 Schema (mostrando primeiras 20 colunas):
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- target: double (nullable = true)
 |-- target_win: double (nullable = true)
 |-- flag_valid_fee_lag_1: integer (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- num_unq_ratio_ref_max_6: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- flag_has_transactions_max_3: integer (nullable = true)
 |-- total_plays_ratio_ref_max_6: double (nullable = true)
 |-- flag_plano_mensal: integer (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- total_plays: double (nullable = true)
 |-- log_total_secs: double (nullable = true)
 |-- flag_valid_fee_max_3: integer (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- total_plays_mean_3: double (nullable = true)
 |-- num_50: double (nullable = true)
```

```
📊 Distribuição de tipos:
• DoubleType : 32 colunas
• StringType : 20 colunas
• IntegerType : 12 colunas
• DateType : 3 colunas
• FloatType : 2 colunas
```

Checagem de nulos

In [74]:

```

print("\n" + "=" * 80)
print("📝 ETAPA 5: Análise de valores nulos...")
print("=" * 80)

# Contar nulls por coluna (apenas para features, não para controle)
from pyspark.sql.functions import col, sum as spark_sum, count, isnan, when

features_para_verificar = features_numericas_finalistas + features_categoricas_finalistas

print(f"\n🔍 Verificando nulls em {len(features_para_verificar)} features...")

# Calcular % de nulls de forma segura
null_counts = []
total_rows = df_feature_store_final.count()

for col_name in features_para_verificar:
    # Obtém o tipo de dado da coluna
    dtype = dict(df_feature_store_final.dtypes)[col_name]

    # Condição base: verificar se é nulo (funciona para todos os tipos)
    condicao = col(col_name).isNull()

    # Se for numérica (double ou float), adiciona a verificação de NaN
    if dtype in ("double", "float"):
        condicao = condicao | isnan(col(col_name))

    null_count = df_feature_store_final.filter(condicao).count()

    if null_count > 0:
        null_pct = (null_count / total_rows) * 100
        null_counts.append((col_name, null_count, null_pct))

if null_counts:
    null_counts_sorted = sorted(null_counts, key=lambda x: x[2], reverse=True)
    print(f"\n⚠️ Features com valores nulos: {len(null_counts)}")
    print(f"\n📝 Top 20 features com mais nulls:")
    for col_name, count, pct in null_counts_sorted[:20]:
        tipo = "[CATEGÓRICA]" if col_name in features_categoricas_finalistas else "[NUMÉRICA]"
        print(f" • {tipo:13s} {col_name:50s} {count:10,} ({pct:5.2f}%)")
else:
    print(f"\n✅ Nenhuma feature com valores nulos!")

```

=====
📝 ETAPA 5: Análise de valores nulos...
=====

🔍 Verificando nulls em 65 features...

⚠️ Features com valores nulos: 26

📝 Top 20 features com mais nulls:	
• [NUMÉRICA] daily_revenue_efficiency	1,539,830 (15.91%)
• [NUMÉRICA] actual_amount_paid	1,539,830 (15.91%)
• [NUMÉRICA] is_auto_renew	1,539,830 (15.91%)
• [NUMÉRICA] payment_method_id	1,539,830 (15.91%)
• [NUMÉRICA] plan_list_price	1,539,830 (15.91%)
• [CATEGÓRICA] membership_expire_date	1,539,830 (15.91%)
• [CATEGÓRICA] transaction_date	1,539,830 (15.91%)
• [NUMÉRICA] flag_valid_fee_lag_1	1,259,830 (13.02%)
• [NUMÉRICA] flag_plano_mensal_lag_1	1,259,830 (13.02%)
• [NUMÉRICA] flag_has_transactions_lag_1	1,259,830 (13.02%)
• [NUMÉRICA] total_plays	987,360 (10.20%)
• [NUMÉRICA] log_total_secs	987,360 (10.20%)
• [NUMÉRICA] num_50	987,360 (10.20%)
• [NUMÉRICA] num_75	987,360 (10.20%)
• [NUMÉRICA] log_total_plays	987,360 (10.20%)
• [NUMÉRICA] total_secs	987,360 (10.20%)
• [NUMÉRICA] num_100	987,360 (10.20%)
• [NUMÉRICA] num_25	987,360 (10.20%)
• [NUMÉRICA] num_985	987,360 (10.20%)
• [NUMÉRICA] num_unq	987,360 (10.20%)

Descriptivas

In [75]:

```

print("\n" + "=" * 80)
print("▣ ETAPA 6: Estatísticas descritivas...")
print("=" * 80)

print(f"\n▣ Estatísticas do target:")
df_feature_store_final.select('target', 'target_win').describe().show()

print(f"\n▣ Estatísticas de features numéricas (amostra de 10):")
df_feature_store_final.select(features_numericas_finalistas[:10]).describe().show()

print(f"\n▣ Distribuição de features categóricas (amostra de 5):")
for col_name in features_categoricas_finalistas[:5]:
    print(f"\n • {col_name}:")
    df_feature_store_final.groupBy(col_name).count().orderBy('count', ascending=False).show(10, truncate=False)

```

```
=====
▣ ETAPA 6: Estatísticas descritivas...
=====
```

▣ Estatísticas do target:

summary	target	target_win
count	9678866	9678866
mean	48.862693949808225	45.18610950651179
stddev	81.72979107853216	59.31573504699382
min	-152.5452002	-108.8147222
max	1950.0	129.0401291

▣ Estatísticas de features numéricas (amostra de 10):

summary	flag_valid_fee_lag_1	total_secs_ratio_ref_max_6	num_unq_ratio_ref_max_6	usage_intensity_per_tenure_cap	daily_revenue_efficiency	log_total_plays_mean_3	flag_has_transactions_max_3	total_plays_ratio_ref_max_6	flag_plano_mensal	revenue_per_hour_listened_cap
count	8419036	9225026	9225026	9678866	41.38492705597649	4.417299877542849	-19779.43559340661	0.8886917124382133	-18936.737463609617	0.8232514015588189
8139036	9678866	9678866	9225026	9678866	0.2980915016527225	0.9313283541189036	39838.66047286536	0.31451353373412577	39179.88930690752	0.38145582499667646
9678866					0.573469834112672					
mean	0.8393520350786005	-18936.740127082154	-18936.73668421884	41.38492705597649	4.417299877542849	-19779.43559340661	0.8886917124382133	-18936.737463609617	0.8232514015588189	0.2980915016527225
4.417299877542849	-19779.43559340661	0.8886917124382133	-18936.737463609617	0.8232514015588189	0.2980915016527225					
0.2980915016527225										
stddev	0.3672059535248893	39179.88901956004	39179.889683613554	100.61556373198133	0.9313283541189036	39838.66047286536	0.31451353373412577	39179.88930690752	0.38145582499667646	0.573469834112672
0.9313283541189036	39838.66047286536	0.31451353373412577	39179.88930690752	0.38145582499667646	0.573469834112672					
0.573469834112672										
min	0	-99998.0	-99998.0	0	0	-99998.0	0	-99998.0	0	0
0.0	-99998.0	0	0	-99998.0	0	0	-99998.0	0	0	0
0.0										
max	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	869.25
6.0	8.440528106480752	1	1	1.0	1.0	1	1	1.0	1	869.25
3.8702505013143265										

▣ Distribuição de features categóricas (amostra de 5):

• total_plays_group:
+-----+-----+
total_plays_group count
+-----+-----+
04_power_user 2214194
03_frequent_listener 2211467
02_regular_listener 2188248
01_casual_listener 2077597
00_unknown 987360
+-----+-----+

• completed_songs_rate_group:
+-----+-----+
completed_songs_rate_group count
+-----+-----+
03_engaged_listener 2192471
04_completionist 2188834
02_skipping_listener 2186604
01_bouncer 2123597
00_unknown 987360
+-----+-----+

```

• avg_secs_per_unq_cap_group:
+-----+-----+
|avg_secs_per_unq_cap_group|count |
+-----+-----+
|02_sampler |2187475|
|03.Focused_listener |2186251|
|04.deep_listener |2181201|
|01.skimmer |2136579|
|00.unknown |987360 |
+-----+-----+


• plays_per_unq_behavior:
+-----+-----+
|plays_per_unq_behavior|count |
+-----+-----+
|01_explorer |6544782|
|02_light_repeat |1290959|
|00.unknown |1087138|
|03_repeat |580333 |
|04_heavy_repeat |175654 |
+-----+-----+


• plays_behavior_vs_volume:
+-----+-----+
|plays_behavior_vs_volume |count |
+-----+-----+
|01_explorer_02_regular_listener |1775012|
|01_explorer_01_casual_listener |1709157|
|01_explorer_03_frequent_listener |1696289|
|01_explorer_04_power_user |1364324|
|00.unknown_00.unknown |987360 |
|02.light_repeat_04_power_user |534778 |
|02.light_repeat_03_frequent_listener |342294|
|02.light_repeat_02_regular_listener |266082 |
|03.repeat_04_power_user |240448 |
|02.light_repeat_01_casual_listener |147805 |
+-----+-----+
only showing top 10 rows

```

Dicionario de features

```
In [76]: print("\n" + "=" * 80)
print("📝 ETAPA 7: Salvando listas de features...")
print("=" * 80)

# Criar dicionário com metadados
features_metadata = {
    'colunas_controle': colunas_controle,
    'features_numericas': features_numericas_finalistas,
    'features_categoricas': features_categoricas_finalistas,
    'total_features': len(features_numericas_finalistas) + len(features_categoricas_finalistas),
    'total_colunas': len(colunas_finalistas)
}

print(f"\n✅ Metadados salvos:")
print(f" • Colunas de controle: {len(features_metadata['colunas_controle'])}")
print(f" • Features numéricas: {len(features_metadata['features_numericas'])}")
print(f" • Features categóricas: {len(features_metadata['features_categoricas'])}")
print(f" • Total de features: {features_metadata['total_features']}")

=====
📝 ETAPA 7: Salvando listas de features...
=====
```

```

✅ Metadados salvos:
• Colunas de controle: 4
• Features numéricas: 43
• Features categóricas: 22
• Total de features: 65

```

Resumo final + Salvar base

```
In [77]: print("\n" + "=" * 80)
print("🕒 RESUMO FINAL - DATAFRAME FINALISTA")
print("=" * 80)

print(f"\n📊 Dimensões:")
print(f" • Linhas: {df_feature_store_final.count():,}")
print(f" • Colunas: {len(df_feature_store_final.columns)}")
print(f" • Features: {len(features_numericas_finalistas) + len(features_categoricas_finalistas)})")

print(f"\n📊 Composição:")
print(f" • Colunas de controle: {len(colunas_controle)}")
print(f" • Features numéricas: {len(features_numericas_finalistas)}")
print(f" • Features categóricas: {len(features_categoricas_finalistas)})")

print(f"\n📊 Qualidade:")
if null_counts:
    print(f" • Features com nulls: {len(null_counts)}")
    print(f" • Features sem nulls: {len(features_para_verificar) - len(null_counts)}")
else:
    print(f" • Features com nulls: 0")
    print(f" • Features sem nulls: {len(features_para_verificar)})")

print(f"\n✅ DataFrame finalista pronto para modelagem!")
print(f" Nome da base: df_feature_store_final")

print("\n" + "=" * 80)
```

```
=====
🕒 RESUMO FINAL - DATAFRAME FINALISTA
=====

📊 Dimensões:
• Linhas: 9,678,866
• Colunas: 69
• Features: 65

📊 Composição:
• Colunas de controle: 4
• Features numéricas: 43
• Features categóricas: 22

📊 Qualidade:
• Features com nulls: 26
• Features sem nulls: 39

✅ DataFrame finalista pronto para modelagem!
Nome da base: df_feature_store_final
=====
```

```
In [78]: # 1. cache
df_feature_store_final = df_feature_store_final.persist()
df_feature_store_final.count()

# 2. salvar particionado
df_feature_store_final.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_feature_store_final")
```

```
In [ ]: features_numericas_base = features_num_final_etapa3 # 43
features_categoricas_base = features_cat_filtradas # 22

print(f"Lista de features numéricas base: {features_numericas_base}")
print(f"Lista de features categóricas base: {features_categoricas_base}")

Lista de features numéricas base: ['flag_valid_fee_lag_1', 'total_secs_ratio_ref_max_6', 'num_unq_ratio_ref_max_6',
'usage_intensity_per_tenure_cap', 'daily_revenue_efficiency', 'log_total_plays_mean_3', 'flag_has_transactions_max_3',
'total_plays_ratio_ref_max_6', 'flag_plano_mensal', 'revenue_per_hour_listened_cap', 'total_plays', 'log_total_secs',
'flag_valid_fee_max_3', 'daily_revenue_efficiency_min_3', 'total_plays_mean_3', 'num_50', 'margem_liquida_mensal', 'num_75',
'log_total_plays', 'total_secs_mean_3', 'daily_revenue_efficiency_ratio_ref_max_3', 'total_secs', 'flag_plano_mensal_lag_1',
'flag_has_transactions_lag_1', 'flag_valid_fee', 'flag_has_transactions', 'actual_amount_paid',
'avg_secs_per_unq_cap_min_6', 'log_total_secs_ratio_ref_max_6', 'num_100', 'num_25', 'is_auto_renew',
'plays_per_unq_cap_min_6', 'payment_method_id', 'log_total_plays_ratio_ref_max_6', 'completion_efficiency_min_6',
'flag_plano_mensal_max_3', 'plan_list_price', 'num_unq_mean_3', 'log_total_secs_mean_3', 'num_985',
'completed_songs_rate_min_6', 'num_unq']
Lista de features categóricas base: ['total_plays_group', 'completed_songs_rate_group', 'avg_secs_per_unq_cap_group',
'plays_per_unq_behavior', 'plays_behavior_vs_volume', 'plays_behavior_vs_volume_collapsed', 'plays_behavior_vs_completion',
'plays_behavior_vs_completion_collapsed', 'early_drop_rate_group', 'revenue_tier', 'payment_method_group',
'payment_price_regime', 'gender_clean', 'faixa_idade', 'registered_via_group', 'registration_year_regime', 'tenure_faixa',
'revenue_per_hour_tier', 'usage_intensity_tier', 'membership_expire_date', 'transaction_date', 'registration_init_time']
```

```
In [82]: df_feature_store_final.count()
```

```
Out[82]: 9678866
```

12.3. Separando a spine entre treino/teste e out-of-time

```
In [85]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver"
df_spine = spark.read.parquet(f"{silver_path}/df_spine")
df_feature_store_final_ids = spark.read.parquet(f"{silver_path}/df_feature_store_final").select("msno", "safra")
```

```
In [86]: df_spine = df_spine.join(df_feature_store_final_ids, on=["msno", "safra"], how="inner")
```

```
In [87]: df_spine.count()
```

```
Out[87]: 9678866
```

In [88]:

```

print("=" * 80)
print(" CRIAÇÃO: DataFrame de Split (Train/Test/OOT)")
print("=" * 80)

# =====
# Definir períodos
# =====

print("\n📊 Estratégia de Split:")
print(" • Train + Test: 201601-201609 (split aleatório 80/20)")
print(" • Out-of-Time: 201610-201611 (100%)")

# =====
# Criar coluna de participação
# =====

# Adicionar coluna aleatória para split (seed fixo para reprodutibilidade)
df_spine_split = df_spine.select('msno', 'safra').distinct()

df_spine_split = df_spine_split.withColumn('random_seed', F.rand(seed=42))

# Criar coluna 'partition'
df_spine_split = df_spine_split.withColumn(
    'partition',
    F.when(F.col('safra') >= 201610, 'oot') # Out-of-Time
    .when(F.col('random_seed') <= 0.8, 'train') # 80% treino
    .otherwise('test') # 20% teste
)

# Remover coluna auxiliar
df_spine_split = df_spine_split.drop('random_seed')

# =====
# Verificar distribuição
# =====

print("\n📊 Distribuição por Partition:")
df_spine_split.groupBy('partition').count().orderBy('partition').show()

print("\n📊 Distribuição por Safra e Partition:")
df_spine_split.groupBy('safra', 'partition').count().orderBy('safra', 'partition').show(20)

# Verificar proporções
total_train_test = df_spine_split.filter(F.col('safra') < 201610).count()
total_train = df_spine_split.filter(F.col('partition') == 'train').count()
total_test = df_spine_split.filter(F.col('partition') == 'test').count()
total_oot = df_spine_split.filter(F.col('partition') == 'oot').count()

pct_train = total_train / total_train_test * 100
pct_test = total_test / total_train_test * 100

print(f"\n✅ Proporções:")
print(f" • Train: {total_train:,} ({pct_train:.1f}%)")
print(f" • Test: {total_test:,} ({pct_test:.1f}%)")
print(f" • OOT: {total_oot:,}%")


# =====
# Persistir para uso posterior
# =====

# 1. cache
df_spine_split = df_spine_split.persist()
df_spine_split.count()

# 2. salvar particionado
df_spine_split.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_spine_splited")

print(f"\n✅ DataFrame 'df_spine_splited' criado e cacheado!")
print(f"    Colunas: {df_spine_split.columns}")

print("\n" + "=" * 80)

```

```

=====
 CRIAÇÃO: DataFrame de Split (Train/Test/OOT)
=====

📊 Estratégia de Split:
• Train + Test: 201601-201609 (split aleatório 80/20)
• Out-of-Time: 201610-201611 (100%)

📊 Distribuição por Partition:
+-----+-----+
|partition| count|
+-----+-----+

```

```
|     oot|1850732|
|   test|1565322|
|  train|6262812|
+-----+-----+
```

📊 Distribuição por Safra e Partition:

safras	partition	count
201601	test	172185
201601	train	688482
201602	test	179088
201602	train	717158
201603	test	165495
201603	train	663217
201604	test	162442
201604	train	648456
201605	test	162177
201605	train	653154
201606	test	163668
201606	train	654509
201607	test	184757
201607	train	739010
201608	test	186806
201608	train	746809
201609	test	188704
201609	train	752017
201610	oot	926604
201611	oot	924128

✓ Proporções:

- Train: 6,262,812 (80.0%)
- Test: 1,565,322 (20.0%)
- OOT: 1,850,732

✓ DataFrame 'df_spine_splited' criado e cacheado!
Colunas: ['msno', 'safras', 'partition']

```
=====
```

12.4. Decisao por modelo: Elastic Net

Carregando base de features

```
In [7]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"
df_feature_store_final = spark.read.parquet(silver_path + "df_feature_store_final")
```

In [8]: df_feature_store_final.printSchema()

```
root
 |-- msno: string (nullable = true)
 |-- target: double (nullable = true)
 |-- target_win: double (nullable = true)
 |-- flag_valid_fee_lag_1: integer (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- num_unq_ratio_ref_max_6: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- flag_has_transactions_max_3: integer (nullable = true)
 |-- total_plays_ratio_ref_max_6: double (nullable = true)
 |-- flag_plano_mensal: integer (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- total_plays: double (nullable = true)
 |-- log_total_secs: double (nullable = true)
 |-- flag_valid_fee_max_3: integer (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- total_plays_mean_3: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- margem_liquida_mensal: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
 |-- total_secs: double (nullable = true)
 |-- flag_plano_mensal_lag_1: integer (nullable = true)
 |-- flag_has_transactions_lag_1: integer (nullable = true)
 |-- flag_valid_fee: integer (nullable = true)
 |-- flag_has_transactions: integer (nullable = true)
 |-- actual_amount_paid: float (nullable = true)
 |-- avg_secs_per_unq_cap_min_6: double (nullable = true)
 |-- log_total_secs_ratio_ref_max_6: double (nullable = true)
 |-- num_100: double (nullable = true)
 |-- num_25: double (nullable = true)
 |-- is_auto_renew: integer (nullable = true)
 |-- plays_per_unq_cap_min_6: double (nullable = true)
 |-- payment_method_id: integer (nullable = true)
 |-- log_total_plays_ratio_ref_max_6: double (nullable = true)
 |-- completion_efficiency_min_6: double (nullable = true)
 |-- flag_plano_mensal_max_3: integer (nullable = true)
 |-- plan_list_price: float (nullable = true)
 |-- num_unq_mean_3: double (nullable = true)
 |-- log_total_secs_mean_3: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- completed_songs_rate_min_6: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_plays_group: string (nullable = true)
 |-- completed_songs_rate_group: string (nullable = true)
 |-- avg_secs_per_unq_cap_group: string (nullable = true)
 |-- plays_per_unq_behavior: string (nullable = true)
 |-- plays_behavior_vs_volume: string (nullable = true)
 |-- plays_behavior_vs_volume_collapsed: string (nullable = true)
 |-- plays_behavior_vs_completion: string (nullable = true)
 |-- plays_behavior_vs_completion_collapsed: string (nullable = true)
 |-- early_drop_rate_group: string (nullable = true)
 |-- revenue_tier: string (nullable = true)
 |-- payment_method_group: string (nullable = true)
 |-- payment_price_regime: string (nullable = true)
 |-- gender_clean: string (nullable = true)
 |-- faixa_idade: string (nullable = true)
 |-- registered_via_group: string (nullable = true)
 |-- registration_year_regime: string (nullable = true)
 |-- tenure_faixa: string (nullable = true)
 |-- revenue_per_hour_tier: string (nullable = true)
 |-- usage_intensity_tier: string (nullable = true)
 |-- membership_expire_date: date (nullable = true)
 |-- transaction_date: date (nullable = true)
 |-- registration_init_time: date (nullable = true)
 |-- safra: integer (nullable = true)
```

```
In [90]: features_numericas_base = ['flag_valid_fee_lag_1', 'total_secs_ratio_ref_max_6', 'num_unq_ratio_ref_max_6',  
    'usage_intensity_per_tenure_cap', 'daily_revenue_efficiency', 'log_total_plays_mean_3', 'flag_has_transactions_max_3',  
    'total_plays_ratio_ref_max_6', 'flag_plano_mensal', 'revenue_per_hour_listened_cap', 'total_plays', 'log_total_secs',  
    'flag_valid_fee_max_3', 'daily_revenue_efficiency_min_3', 'total_plays_mean_3', 'num_50', 'margem_liquida_mensal', 'num_75',  
    'log_total_plays', 'total_secs_mean_3', 'daily_revenue_efficiency_ratio_ref_max_3', 'total_secs', 'flag_plano_mensal_lag_1',  
    'flag_has_transactions_lag_1', 'flag_valid_fee', 'flag_has_transactions', 'actual_amount_paid', 'avg_secs_per_unq_cap_min_6',  
    'log_total_secs_ratio_ref_max_6', 'num_100', 'num_25', 'is_auto_renew', 'plays_per_unq_cap_min_6', 'payment_method_id',  
    'log_total_plays_ratio_ref_max_6', 'completion_efficiency_min_6', 'flag_plano_mensal_max_3', 'plan_list_price', 'num_unq_mean_3',  
    'log_total_secs_mean_3', 'num_985', 'completed_songs_rate_min_6', 'num_unq']  
features_categoricas_base = ['total_plays_group', 'completed_songs_rate_group', 'avg_secs_per_unq_cap_group',  
    'plays_per_unq_behavior', 'plays_behavior_vs_volume', 'plays_behavior_vs_volume_collapsed', 'plays_behavior_vs_completion',  
    'plays_behavior_vs_completion_collapsed', 'early_drop_rate_group', 'revenue_tier', 'payment_method_group', 'payment_price_regime',  
    'gender_clean', 'faixa_idade', 'registered_via_group', 'registration_year_regime', 'tenure_faixa', 'revenue_per_hour_tier',  
    'usage_intensity_tier', 'membership_expire_date', 'transaction_date', 'registration_init_time']
```

```
In [91]: df_feature_store_final.count()
```

```
Out[91]: 9678866
```

Variáveis numéricas

Tratando nulos e sentinelas

Elastic Net

- ✗ **Não aceita nulos** (quebra o cálculo matricial)
- ✗ **Não entende sentinelas** (trata -99999 como um valor real extremo)
- ✗ **Sensível a escala** (precisa de normalização)
- ✓ **Precisa de tratamento explícito:** Imputação + Remoção de sentinelas

Identificando casos a serem tratados

In [92]:

```
print(f"\n{'='*80}")
print("DIAGNÓSTICO COMPLETO: Nulos e Sentinelas")
print("=" * 80)

diagnostico = []

for col in features_numericas_base:
    # Contar nulos
    n_null = df_feature_store_final.filter(F.col(col).isNull()).count()

    # Contar negativos (possíveis sentinelas)
    n_neg = df_feature_store_final.filter(F.col(col) < 0).count()

    # Estatísticas básicas (min, max)
    stats = df_feature_store_final.select(
        F.min(col).alias('min_val'),
        F.max(col).alias('max_val')
    ).collect()[0]

    if n_null > 0 or n_neg > 0:
        diagnostico.append({
            'coluna': col,
            'nulos': n_null,
            'negativos': n_neg,
            'min': stats['min_val'],
            'max': stats['max_val']
        })
    })

# Exibir diagnóstico
if diagnostico:
    print(f"\n⚠️ Encontrados {len(diagnostico)} variáveis com problemas:\n")
    for item in diagnostico:
        print(f"  📈 {item['coluna']}")
        if item['nulos'] > 0:
            print(f"    └ Nulos: {item['nulos']}:")
        if item['negativos'] > 0:
            print(f"    └ Negativos: {item['negativos']} (min={item['min']})")
        print(f"    └ Range: [{item['min']}, {item['max']}]")
        print()
else:
    print("  ✅ Nenhum problema encontrado!")
```

```
=====
DIAGNÓSTICO COMPLETO: Nulos e Sentinelas
=====
```

⚠️ Encontrados 35 variáveis com problemas:

```
  📈 flag_valid_fee_lag_1
    └ Nulos: 1,259,830
    └ Range: [0, 1]

  📈 total_secs_ratio_ref_max_6
    └ Nulos: 453,840
    └ Negativos: 1,747,028 (min=-99998.0)
    └ Range: [-99998.0, 1.0]

  📈 num_unq_ratio_ref_max_6
    └ Nulos: 453,840
    └ Negativos: 1,747,028 (min=-99998.0)
    └ Range: [-99998.0, 1.0]

  📈 daily_revenue_efficiency
    └ Nulos: 1,539,830
    └ Range: [0.0, 6.0]

  📈 log_total_plays_mean_3
    └ Negativos: 1,914,940 (min=-99998.0)
    └ Range: [-99998.0, 8.440528106480752]

  📈 total_plays_ratio_ref_max_6
    └ Nulos: 453,840
    └ Negativos: 1,747,028 (min=-99998.0)
    └ Range: [-99998.0, 1.0]

  📈 total_plays
    └ Nulos: 987,360
    └ Range: [1.0, 4630.0]

  📈 log_total_secs
    └ Nulos: 987,360
    └ Range: [3.371081757757532, 13.689588996416914]

  📈 daily_revenue_efficiency_min_3
    └ Negativos: 2,314,344 (min=-99999.0)
    └ Range: [-99999.0, 6.0]
```

- total_plays_mean_3
 - └ Negativos: 1,914,940 (min=-99998.0)
 - └ Range: [-99998.0, 4630.0]
- num_50
 - └ Nulos: 987,360
 - └ Range: [0.0, 174.0]
- margem_liquida_mensal
 - └ Negativos: 1,739,972 (min=-152.5452002)
 - └ Range: [-152.5452002, 1950.0]
- num_75
 - └ Nulos: 987,360
 - └ Range: [0.0, 101.0]
- log_total_plays
 - └ Nulos: 987,360
 - └ Range: [0.6931471805599453, 8.440528106480752]
- total_secs_mean_3
 - └ Negativos: 1,914,940 (min=-99998.0)
 - └ Range: [-99998.0, 881683.002]
- daily_revenue_efficiency_ratio_ref_max_3
 - └ Nulos: 454,737
 - └ Negativos: 2,314,344 (min=-99999.0)
 - └ Range: [-99999.0, 1.0]
- total_secs
 - └ Nulos: 987,360
 - └ Range: [28.11, 881683.002]
- flag_plano_mensal_lag_1
 - └ Nulos: 1,259,830
 - └ Range: [0, 1]
- flag_has_transactions_lag_1
 - └ Nulos: 1,259,830
 - └ Range: [0, 1]
- actual_amount_paid
 - └ Nulos: 1,539,830
 - └ Range: [0.0, 2000.0]
- avg_secs_per_unq_cap_min_6
 - └ Negativos: 2,139,064 (min=-99999.0)
 - └ Range: [-99999.0, 1755.2955413533832]
- log_total_secs_ratio_ref_max_6
 - └ Nulos: 453,840
 - └ Negativos: 1,747,028 (min=-99998.0)
 - └ Range: [-99998.0, 1.0]
- num_100
 - └ Nulos: 987,360
 - └ Range: [0.0, 3487.0]
- num_25
 - └ Nulos: 987,360
 - └ Range: [0.0, 746.0]
- is_auto_renew
 - └ Nulos: 1,539,830
 - └ Range: [0, 1]
- plays_per_unq_cap_min_6
 - └ Negativos: 2,216,132 (min=-99999.0)
 - └ Range: [-99999.0, 6.947368421052632]
- payment_method_id
 - └ Nulos: 1,539,830
 - └ Range: [2, 41]
- log_total_plays_ratio_ref_max_6
 - └ Nulos: 453,840
 - └ Negativos: 1,747,028 (min=-99998.0)
 - └ Range: [-99998.0, 1.0]
- completion_efficiency_min_6
 - └ Negativos: 2,179,531 (min=-99999.0)
 - └ Range: [-99999.0, 1.0]
- plan_list_price
 - └ Nulos: 1,539,830
 - └ Range: [0.0, 2000.0]

```
[!] num_unq_mean_3
└ Negativos: 1,914,940 (min=-99998.0)
└ Range: [-99998.0, 2819.0]

[!] log_total_secs_mean_3
└ Negativos: 1,914,940 (min=-99998.0)
└ Range: [-99998.0, 13.689588996416914]

[!] num_985
└ Nulos: 987,360
└ Range: [0.0, 122.0]

[!] completed_songs_rate_min_6
└ Negativos: 2,216,132 (min=-99999.0)
└ Range: [-99999.0, 1.0]

[!] num_unq
└ Nulos: 987,360
└ Range: [1.0, 2819.0]
```

Tratando os casos

```
In [94]: print(f"\n{'='*80}")
print("TRATAMENTO NULOS E SENTINELAS: Elastic Net")
print("=" * 80)

# Criar cópia para não afetar LightGBM/RF
df_elastic_net = df_feature_store_final.select('*')

# =====
# TIPO 1: Nulos em variáveis RAW de Logs → 0
# =====
print("\n▣ TIPO 1: Variáveis de Logs (nulos → 0)")
vars_logs_raw = [
    'num_50', 'num_unq', 'total_plays', 'num_100', 'total_secs',
    'num_985', 'num_25', 'log_total_secs', 'log_total_plays', 'num_75'
]

for col in vars_logs_raw:
    df_elastic_net = df_elastic_net.withColumn(
        col,
        F.when(F.col(col).isNull(), 0.0).otherwise(F.col(col))
    )
print(f" ✓ {len(vars_logs_raw)} variáveis tratadas")

# =====
# TIPO 2: Nulos em variáveis de Transações
# =====
print("\n☒ TIPO 2: Variáveis de Transações")

# Numéricas → 0
vars_transacoes_num = [
    'actual_amount_paid', 'plan_list_price', 'daily_revenue_efficiency'
]
for col in vars_transacoes_num:
    df_elastic_net = df_elastic_net.withColumn(
        col,
        F.when(F.col(col).isNull(), 0.0).otherwise(F.col(col))
    )

# Categórica especial
df_elastic_net = df_elastic_net.withColumn(
    'payment_method_id',
    F.when(F.col('payment_method_id').isNull(), 99).otherwise(F.col('payment_method_id'))
)

# Flag
df_elastic_net = df_elastic_net.withColumn(
    'is_auto_renew',
    F.when(F.col('is_auto_renew').isNull(), 0).otherwise(F.col('is_auto_renew'))
)

print(f" ✓ 5 variáveis tratadas")

# =====
# TIPO 3: Nulos em Flags de Lag → 0
# =====
print("\n▶ TIPO 3: Flags de Lag (nulos → 0)")
vars_flags_lag = [
    'flag_valid_fee_lag_1', 'flag_plano_mensal_lag_1', 'flag_has_transactions_lag_1'
]

for col in vars_flags_lag:
    df_elastic_net = df_elastic_net.withColumn(
        col,
        F.when(F.col(col).isNull(), 0).otherwise(F.col(col))
    )
print(f" ✓ {len(vars_flags_lag)} variáveis tratadas")

# =====
# TIPO 4: Sentinelas em TENDÊNCIAS → Versão Atual ou Mediana
# =====
print("\n☒ TIPO 4: Tendências (sentinelas → versão atual ou mediana)")

# Mapeamento: variável_tendencia → variável_atual
mapa_tendencias = {
    'daily_revenue_efficiency_min_3': 'daily_revenue_efficiency',
    'total_secs_mean_3': 'total_secs',
    'log_total_secs_mean_3': 'log_total_secs',
    'log_total_plays_mean_3': 'log_total_plays',
    'total_plays_mean_3': 'total_plays',
    'num_unq_mean_3': 'num_unq',
}

for col_tend, col_atual in mapa_tendencias.items():
    df_elastic_net = df_elastic_net.withColumn(
        col_tend,
        F.when(F.col(col_tend) < 0, F.col(col_atual)).otherwise(F.col(col_tend))
    )
```

```

print(f" ✅ {col_tend} → {col_atual}")

# Variáveis sem versão atual → Mediana
vars_tend_sem_atual = [
    'completion_efficiency_min_6',
    'completed_songs_rate_min_6',
    'plays_per_unq_cap_min_6',
    'avg_secs_per_unq_cap_min_6'
]

for col in vars_tend_sem_atual:
    mediana = df_elastic_net.filter(F.col(col) >= 0).approxQuantile(col, [0.5], 0.01)[0]
    df_elastic_net = df_elastic_net.withColumn(
        col,
        F.when(F.col(col) < 0, mediana).otherwise(F.col(col))
    )
    print(f" ✅ {col} → mediana ({mediana:.4f})")

# =====
# TIPO 5: Sentinelas em RATIOS → 1.0 (+ Nulos → 1.0)
# =====
print("\n📊 TIPO 5: Ratios (sentinelas + nulos → 1.0)")
vars_ratios = [
    'total_secs_ratio_ref_max_6',
    'log_total_secs_ratio_ref_max_6',
    'log_total_plays_ratio_ref_max_6',
    'total_plays_ratio_ref_max_6',
    'num_unq_ratio_ref_max_6'
]

for col in vars_ratios:
    df_elastic_net = df_elastic_net.withColumn(
        col,
        F.when(F.col(col) < 0, 1.0)
            .when(F.col(col).isNull(), 1.0) # Nulos também → 1.0
            .otherwise(F.col(col))
    )
print(f" ✅ {len(vars_ratios)} variáveis tratadas")

# =====
# TIPO 6: RATIO especial (sentinela → 1.0, nulo → mediana)
# =====
print("\n⌚ TIPO 6: daily_revenue_efficiency_ratio_ref_max_3")

mediana_ratio = df_elastic_net.filter(
    (F.col('daily_revenue_efficiency_ratio_ref_max_3') >= 0)
).approxQuantile('daily_revenue_efficiency_ratio_ref_max_3', [0.5], 0.01)[0]

df_elastic_net = df_elastic_net.withColumn(
    'daily_revenue_efficiency_ratio_ref_max_3',
    F.when(F.col('daily_revenue_efficiency_ratio_ref_max_3') < 0, 1.0)
        .when(F.col('daily_revenue_efficiency_ratio_ref_max_3').isNull(), mediana_ratio)
        .otherwise(F.col('daily_revenue_efficiency_ratio_ref_max_3'))
)
print(f" ✅ Sentinelas → 1.0, Nulos → {mediana_ratio:.4f}")

# =====
# TIPO 7: NOVO - Margem Líquida Mensal (negativos são válidos!)
# =====
print("\n💰 TIPO 7: margem_liquida_mensal (negativos são válidos)")
print(" 📈 Nenhum tratamento necessário (negativos = prejuízo real)")

# =====
# VALIDAÇÃO FINAL: Verificar se ainda há nulos/sentinelas
# =====
print(f"\n'*80")
print("🌐 VALIDAÇÃO FINAL: Verificando nulos e sentinelas restantes")
print("=* 80")

# Lista de todas as variáveis numéricas tratadas
vars_numericas_tratadas = (
    vars_logs_raw +
    vars_transacoes_num +
    vars_flags_lag +
    list(mapa_tendencias.keys()) +
    vars_tend_sem_atual +
    vars_ratios +
    ['daily_revenue_efficiency_ratio_ref_max_3', 'payment_method_id', 'is_auto_renew']
)

problemas_restantes = []

for col in vars_numericas_tratadas:
    # Contar nulos
    nulos = df_elastic_net.filter(F.col(col).isNull()).count()

    # Contar sentinelas (valores < -1000)
    sentinelas = df_elastic_net.filter(F.col(col) < -1000).count()

```

```

if nulos > 0 or sentinelas > 0:
    problemas_restantes.append({
        'variavel': col,
        'nulos': nulos,
        'sentinelas': sentinelas
    })

if problemas_restantes:
    print("\n⚠ ATENÇÃO: Ainda há problemas nas seguintes variáveis:")
    for prob in problemas_restantes:
        print(f"  └ {prob['variavel']}]")
        if prob['nulos'] > 0:
            print(f"    └ Nulos: {prob['nulos']}]")
        if prob['sentinelas'] > 0:
            print(f"    └ Sentinelas: {prob['sentinelas']}]")
else:
    print("\n✅ PERFEITO: Nenhum nulo ou sentinela restante!")

print(f"\n{'='*80}")
print("✅ TRATAMENTO CONCLUÍDO!")
print(f"Total de variáveis tratadas: {len(vars_numericas_tratadas)}")
print("=" * 80)

```

=====
TRATAMENTO NULOS E SENTINELAS: Elastic Net
=====

- 📦 TIPO 1: Variáveis de Logs (nulos → 0)
 - ✓ 10 variáveis tratadas
- 📝 TIPO 2: Variáveis de Transações
 - ✓ 5 variáveis tratadas
- ▶ TIPO 3: Flags de Lag (nulos → 0)
 - ✓ 3 variáveis tratadas
- ☑ TIPO 4: Tendências (sentinelas → versão atual ou mediana)
 - ✓ daily_revenue_efficiency_min_3 → daily_revenue_efficiency
 - ✓ total_secs_mean_3 → total_secs
 - ✓ log_total_secs_mean_3 → log_total_secs
 - ✓ log_total_plays_mean_3 → log_total_plays
 - ✓ total_plays_mean_3 → total_plays
 - ✓ num_unq_mean_3 → num_unq
 - ✓ completion_efficiency_min_6 → mediana (0.8786)
 - ✓ completed_songs_rate_min_6 → mediana (0.5914)
 - ✓ plays_per_unq_cap_min_6 → mediana (0.7143)
 - ✓ avg_secs_per_unq_cap_min_6 → mediana (200.8671)
- 📊 TIPO 5: Ratios (sentinelas + nulos → 1.0)
 - ✓ 5 variáveis tratadas
- ⌚ TIPO 6: daily_revenue_efficiency_ratio_ref_max_3
 - ✓ Sentinelas → 1.0, Nulos → 1.0000
- ⌚ TIPO 7: margem_liquida_mensal (negativos são válidos)
 - ℹ️ Nenhum tratamento necessário (negativos = prejuízo real)

=====
🔍 VALIDAÇÃO FINAL: Verificando nulos e sentinelas restantes
=====

- ✓ PERFEITO: Nenhum nulo ou sentinela restante!

=====
✓ TRATAMENTO CONCLUÍDO!
Total de variáveis tratadas: 34
=====

Validando tratamentos

```
In [95]: # Validação: Verificar se ainda há nulos/sentinelas
print("\n🔍 Validação pós-tratamento:")
for col in features_numericas_base:
    n_null = df_elastic_net.filter(F.col(col).isNull()).count()
    n_neg_extremo = df_elastic_net.filter(F.col(col) < -1000).count()

    if n_null > 0 or n_neg_extremo > 0:
        print(f"⚠️ {col}: nulos={n_null}, sentinelas={n_neg_extremo}")

print("\n ✅ Se nada apareceu acima, está 100% limpo!")
```

🔍 Validação pós-tratamento:

✅ Se nada apareceu acima, está 100% limpo!

Remover features redundantes

```
In [96]: # Remover features redundantes para Elastic Net
features_a_remover_elastic = [
    'total_secs', # Manter apenas log_total_secs
    'total_plays', # Manter apenas log_total_plays
    'log_total_plays', # Redundante com log_total_secs
    'num_100', # Redundante com total_plays
    'total_secs_mean_3', # Manter apenas log_total_secs_mean_3
    'total_plays_mean_3', # Redundante
    'log_total_plays_mean_3', # Redundante
    'flag_plano_mensal_mean_3', # Manter apenas flag_plano_mensal
    'flag_plano_mensal_lag_1', # Redundante
    'flag_valid_fee_mean_3', # Manter apenas flag_valid_fee
    'flag_valid_fee_lag_1', # Redundante
    'flag_has_transactions_mean_3', # Manter apenas flag_has_transactions
    'flag_has_transactions_lag_1', # Redundante
    'payment_method_id', # Manter apenas payment_method_group
]

features_numericas_elastic = [
    col for col in features_numericas_base
    if col not in features_a_remover_elastic
]

# Categóricas para Elastic Net (remover cruzadas, usar componentes)
features_categoricas_elastic = [
    col for col in features_categoricas_base
    if col not in ['plays_behavior_vs_volume', 'plays_behavior_vs_volume_collapsed',
                   'plays_behavior_vs_completion', 'plays_behavior_vs_completion_collapsed']
]

print(f"\n ✅ Features para Elastic Net:")
print(f" • Numéricas: {len(features_numericas_elastic)}")
print(f" • Categóricas (para OHE): {len(features_categoricas_elastic)}")
print(f" • Total (antes de OHE): {len(features_numericas_elastic) + len(features_categoricas_elastic)}")

print(f"\n 📁 Features numéricas removidas para Elastic Net:")
for col in features_a_remover_elastic:
    if col in features_numericas_base:
        print(f" • {col}")
```

✅ Features para Elastic Net:

- Numéricas: 32
- Categóricas (para OHE): 18
- Total (antes de OHE): 50

📁 Features numéricas removidas para Elastic Net:

- total_secs
- total_plays
- log_total_plays
- num_100
- total_secs_mean_3
- total_plays_mean_3
- log_total_plays_mean_3
- flag_plano_mensal_lag_1
- flag_valid_fee_lag_1
- flag_has_transactions_lag_1
- payment_method_id

Filtro por _Information Value_

Definição de conceitos

Weight of Evidence (WOE)

O WOE mede o quanto um grupo (bin) específico contribui para a separação entre "Goods" (eventos) e "Bads" (não-eventos).

$$\$WOE_i = \ln \left(\frac{\% \text{Good}_i}{\% \text{Bad}_i} \right)$$

* **Valor positivo:** Indica que o bin tem uma proporção maior de "Goods" que a média.

* **Valor negativo:** Indica uma proporção maior de "Bads".

Information Value (IV)

O IV é o somatório ponderado do WOE para todos os bins da variável, fornecendo uma única métrica de importância para a feature.

$$\$IV = \sum_{i=1}^n (\% \text{Good}_i - \% \text{Bad}_i) \times WOE_i$$

Adaptações para Variáveis Numéricas

Visto que o cálculo de IV foi desenhado para variáveis categóricas, a função implementa três adaptações cruciais para lidar com dados contínuos:

A. Discretização (Binning) via QuantileDiscretizer:

Variáveis numéricas contínuas possuem infinitos pontos de corte. Para calcular densidades, a função utiliza o `QuantileDiscretizer` do PySpark para agrupar os dados em **n_bins**.

* Isso garante que cada "balde" tenha aproximadamente a mesma quantidade de observações, tornando o cálculo do WOE estatisticamente mais robusto.

B. Binarização do Target pela Mediana:

O IV tradicional exige um target binário (0 ou 1). Como a função recebe um target contínuo (`target_min`), ela realiza uma binarização automática:

```
$$Target\_bin =  
\begin{cases}  
1, & \text{se } y \geq \text{mediana} \\\\  
0, & \text{se } y < \text{mediana}  
\end{cases}$$
```

Essa adaptação permite avaliar o poder da feature em discriminar entre valores "altos" e "baixos" do target.

C. Tratamento de Bins Vazios (Smoothing)

Para evitar erros matemáticos de divisão por zero ou logaritmo de zero, a função aplica um *laplace smoothing* básico, garantindo um percentual mínimo de \$0.0001\$ para as densidades de Good e Bad.

Interpretação do IV

Abaixo, a tabela de referência comumente utilizada em validação interna de modelos para avaliar a força da variável:

Information Value (IV) Poder Preditivo
:--- :---
< 0.02 Inútil / Imperceptível
0.02 a 0.1 Fraco
0.1 a 0.3 Médio
0.3 a 0.5 Forte
> 0.5 Suspeito (Muito forte, verificar Overfitting/Leakage)

Definição da função + Execução

```
In [97]: def calcular_iv_numerica(df, feature_col, target_col='target_win', n_bins=10):
    """
    Calcula Information Value (IV) para feature numérica

    Fórmula:
    IV = Σ (Good% - Bad%) × WOE
    WOE = ln(Good% / Bad%)

    Args:
        df: DataFrame Spark
        feature_col: Nome da feature numérica
        target_col: Nome do target (numérico contínuo)
        n_bins: Número de bins para discretização

    Returns:
        iv_value: Information Value
        woe_df: DataFrame com WOE por bin
    """

    # Binarizar target pela mediana
    target_median = df.approxQuantile(target_col, [0.5], 0.01)[0]

    df_temp = df.select(
        F.col(feature_col),
        F.when(F.col(target_col) >= target_median, 1).otherwise(0).alias('target_bin')
    ).filter(F.col(feature_col).isNotNull())

    # Discretizar feature em bins
    discretizer = QuantileDiscretizer(
        numBuckets=n_bins,
        inputCol=feature_col,
        outputCol=f'{feature_col}_binned',
        handleInvalid="keep"
    )

    df_binned = discretizer.fit(df_temp).transform(df_temp)

    # Calcular Good/Bad por bin
    woe_stats = df_binned.groupBy(f'{feature_col}_binned').agg(
        F.sum(F.when(F.col('target_bin') == 1, 1).otherwise(0)).alias('good_count'),
        F.sum(F.when(F.col('target_bin') == 0, 1).otherwise(0)).alias('bad_count'),
        F.count('*').alias('total_count')
    ).collect()

    # Totais
    total_good = sum([row['good_count'] for row in woe_stats])
    total_bad = sum([row['bad_count'] for row in woe_stats])

    if total_good == 0 or total_bad == 0:
        return 0.0, None

    # Calcular WOE e IV
    iv_total = 0.0
    woe_data = []

    for row in woe_stats:
        good_pct = (row['good_count'] / total_good) if total_good > 0 else 0.0001
        bad_pct = (row['bad_count'] / total_bad) if total_bad > 0 else 0.0001

        # Evitar divisão por zero
        good_pct = max(good_pct, 0.0001)
        bad_pct = max(bad_pct, 0.0001)

        woe = np.log(good_pct / bad_pct)
        iv_bin = (good_pct - bad_pct) * woe
        iv_total += iv_bin

        woe_data.append({
            'bin': row[f'{feature_col}_binned'],
            'good_count': row['good_count'],
            'bad_count': row['bad_count'],
            'good_pct': good_pct,
            'bad_pct': bad_pct,
            'woe': woe,
            'iv': iv_bin
        })

    return iv_total, woe_data
```

Estrutura do Código

A função segue o workflow:

1. **Cálculo da Mediana:** Utiliza `approxQuantile` para eficiência em grandes volumes de dados.
2. **Filtragem de Nulos:** Remove valores nulos para não enviesar os quartis.
3. **Agregação:** Agrupa por bin e calcula as contagens absolutas de Good/Bad.
4. **Loop Estatístico:** Itera sobre as estatísticas coletadas para aplicar as fórmulas de log e somatório.

```
In [98]: df_elastic_net.count()
```

```
Out[98]: 9678866
```

```
In [99]: print(f"\n🔍 Calculando IV para {len(features_numericas_elastic)} features numéricas...")
print(f"    (Deve demorar alguns minutos)")

iv_results = []

for i, feature in enumerate(features_numericas_elastic, 1):
    print(f"    [{i}:2d]/{len(features_numericas_elastic)}] Calculando IV: {feature:50s}", end='')

    try:
        iv_value, woe_data = calcular_iv_numerica(
            df_elastic_net,
            feature,
            target_col='target_win',
            n_bins=10
        )

        iv_results.append({
            'feature': feature,
            'iv': iv_value
        })
    except Exception as e:
        print(f"⚠️ Erro: {e}")
        iv_results.append({
            'feature': feature,
            'iv': 0.0
        })

    print(f" IV = {iv_value:.4f}")

except Exception as e:
    print(f"⚠️ Erro: {e}")
    iv_results.append({
        'feature': '',
        'iv': 0.0
    })
```

```
🔍 Calculando IV para 32 features numéricas...
(Deve demorar alguns minutos)
[ 1/32] Calculando IV: total_secs_ratio_ref_max_6           IV = 0.0351
[ 2/32] Calculando IV: num_unq_ratio_ref_max_6             IV = 0.0317
[ 3/32] Calculando IV: usage_intensity_per_tenure_cap       IV = 0.1818
[ 4/32] Calculando IV: daily_revenue_efficiency           IV = 2.3948
[ 5/32] Calculando IV: flag_has_transactions_max_3         IV = 0.5142
[ 6/32] Calculando IV: total_plays_ratio_ref_max_6          IV = 0.0326
[ 7/32] Calculando IV: flag_plano_mensal                   IV = 0.4555
[ 8/32] Calculando IV: revenue_per_hour_listened_cap       IV = 0.4389
[ 9/32] Calculando IV: log_total_secs                      IV = 0.1910
[10/32] Calculando IV: flag_valid_fee_max_3                IV = 0.5007
[11/32] Calculando IV: daily_revenue_efficiency_min_3      IV = 3.9669
[12/32] Calculando IV: num_50                             IV = 0.0218
[13/32] Calculando IV: margem_liquida_mensal             IV = 3.5097
[14/32] Calculando IV: num_75                             IV = 0.0278
[15/32] Calculando IV: daily_revenue_efficiency_ratio_ref_max_3 IV = 0.0146
[16/32] Calculando IV: flag_valid_fee                     IV = 0.3799
[17/32] Calculando IV: flag_has_transactions              IV = 0.3852
[18/32] Calculando IV: actual_amount_paid                 IV = 2.2007
[19/32] Calculando IV: avg_secs_per_unq_cap_min_6        IV = 0.0280
[20/32] Calculando IV: log_total_secs_ratio_ref_max_6      IV = 0.0369
[21/32] Calculando IV: num_25                             IV = 0.0184
[22/32] Calculando IV: is_auto_renew                     IV = 0.1302
[23/32] Calculando IV: plays_per_unq_cap_min_6           IV = 0.0332
[24/32] Calculando IV: log_total_plays_ratio_ref_max_6     IV = 0.0366
[25/32] Calculando IV: completion_efficiency_min_6       IV = 0.0352
[26/32] Calculando IV: flag_plano_mensal_max_3           IV = 0.7128
[27/32] Calculando IV: plan_list_price                  IV = 2.1951
[28/32] Calculando IV: num_unq_mean_3                   IV = 0.1401
[29/32] Calculando IV: log_total_secs_mean_3             IV = 0.1797
[30/32] Calculando IV: num_985                          IV = 0.0296
[31/32] Calculando IV: completed_songs_rate_min_6        IV = 0.0388
[32/32] Calculando IV: num_unq                         IV = 0.1409
```

Analise dos resultados

```

In # Ordenar por IV (decrescente)
[100]: iv_results_sorted = sorted(iv_results, key=lambda x: x['iv'], reverse=True)

# =====
# Análise dos resultados
# =====

print("\n" + "=" * 80)
print("📊 RESULTADOS - INFORMATION VALUE")
print("=" * 80)

# Classificar features por IV
iv_inuteis = [r for r in iv_results_sorted if r['iv'] < 0.02]
iv_fracas = [r for r in iv_results_sorted if 0.02 <= r['iv'] < 0.10]
iv_medias = [r for r in iv_results_sorted if 0.10 <= r['iv'] < 0.30]
iv_fortes = [r for r in iv_results_sorted if 0.30 <= r['iv'] < 0.50]
iv_suspeitas = [r for r in iv_results_sorted if r['iv'] >= 0.50]

print(f"\n📊 Distribuição por poder preditivo:")
print(f" • Inúteis (IV < 0.02): {len(iv_inuteis):2d} features ✗ REMOVER")
print(f" • Fracas (0.02 ≤ IV < 0.10): {len(iv_fracas):2d} features ⚠️")
print(f" • Médias (0.10 ≤ IV < 0.30): {len(iv_medias):2d} features ✓")
print(f" • Fortes (0.30 ≤ IV < 0.50): {len(iv_fortes):2d} features ✓✓")
print(f" • Suspeitas (IV ≥ 0.50): {len(iv_suspeitas):2d} features ⚠️ VERIFICAR LEAKAGE")

# Mostrar ranking completo
print("\n🏆 Ranking completo de IV:")
for i, result in enumerate(iv_results_sorted, 1):
    feature = result['feature']
    iv = result['iv']

    # Classificação
    if iv < 0.02:
        status = "✗ INÚTIL"
    elif iv < 0.10:
        status = "⚠️ FRACA"
    elif iv < 0.30:
        status = "✓ MÉDIA"
    elif iv < 0.50:
        status = "✓✓ FORTE"
    else:
        status = "⚠️ SUSPEITA"

    print(f" {i:2d}. {status:12s} {feature:55s} IV = {iv:.4f}")

```

```

=====
📊 RESULTADOS - INFORMATION VALUE
=====

📊 Distribuição por poder preditivo:
• Inúteis (IV < 0.02): 2 features ✗ REMOVER
• Fracas (0.02 ≤ IV < 0.10): 12 features ⚠️
• Médias (0.10 ≤ IV < 0.30): 6 features ✓
• Fortes (0.30 ≤ IV < 0.50): 4 features ✓✓
• Suspeitas (IV ≥ 0.50): 8 features ⚠️ VERIFICAR LEAKAGE

🏆 Ranking completo de IV:
1. ⚠️ SUSPEITA daily_revenue_efficiency_min_3 IV = 3.9669
2. ⚠️ SUSPEITA margem_liquida_mensal IV = 3.5097
3. ⚠️ SUSPEITA daily_revenue_efficiency IV = 2.3948
4. ⚠️ SUSPEITA actual_amount_paid IV = 2.2007
5. ⚠️ SUSPEITA plan_list_price IV = 2.1951
6. ⚠️ SUSPEITA flag_plano_mensal_max_3 IV = 0.7128
7. ⚠️ SUSPEITA flag_has_transactions_max_3 IV = 0.5142
8. ⚠️ SUSPEITA flag_valid_fee_max_3 IV = 0.5007
9. ✓✓ FORTE flag_plano_mensal IV = 0.4555
10. ✓✓ FORTE revenue_per_hour_listened_cap IV = 0.4389
11. ✓✓ FORTE flag_has_transactions IV = 0.3852
12. ✓✓ FORTE flag_valid_fee IV = 0.3799
13. ✓ MÉDIA log_total_secs IV = 0.1910
14. ✓ MÉDIA usage_intensity_per_tenure_cap IV = 0.1818
15. ✓ MÉDIA log_total_secs_mean_3 IV = 0.1797
16. ✓ MÉDIA num_unq IV = 0.1409
17. ✓ MÉDIA num_unq_mean_3 IV = 0.1401
18. ✓ MÉDIA is_auto_renew IV = 0.1302
19. ⚠️ FRACA completed_songs_rate_min_6 IV = 0.0388
20. ⚠️ FRACA log_total_secs_ratio_ref_max_6 IV = 0.0369
21. ⚠️ FRACA log_total_plays_ratio_ref_max_6 IV = 0.0366
22. ⚠️ FRACA completion_efficiency_min_6 IV = 0.0352
23. ⚠️ FRACA total_secs_ratio_ref_max_6 IV = 0.0351
24. ⚠️ FRACA plays_per_unq_cap_min_6 IV = 0.0332
25. ⚠️ FRACA total_plays_ratio_ref_max_6 IV = 0.0326
26. ⚠️ FRACA num_unq_ratio_ref_max_6 IV = 0.0317
27. ⚠️ FRACA num_985 IV = 0.0296
28. ⚠️ FRACA avg_secs_per_unq_cap_min_6 IV = 0.0280
29. ⚠️ FRACA num_75 IV = 0.0278

```

30.	⚠ FRACA	num_50	IV = 0.0218
31.	✗ INÚTIL	num_25	IV = 0.0184
32.	✗ INÚTIL	daily_revenue_efficiency_ratio_ref_max_3	IV = 0.0146

* margem_liquida_mensal e actual_amount_paid (receita) podem ser confundidas com leakage, mas se referem ao valor dessas features no mês referencia, não em mês + 1. A dúvida é: é válido usar a margem do mês atual (M+0) para prever a margem do mês seguinte (M+1)?

* É como usar o faturamento de janeiro para prever o faturamento de fevereiro. Pensando nisso, o modelo agregaria valor além do óbvio? Se o modelo apenas replica margem_liquida_mensal não há valor, mesmo que não haja violação da ordem temporal.

Testes de Validação

```
In [101]: print("=" * 80)
print("📝 VALIDAÇÃO: margem_liquida_mensal M+0 vs Target M+1")
print("=" * 80)

# =====
# TESTE 1: Baseline Ingênuo (margem_M+1 = margem_M+0)
# =====

print("\n📋 TESTE 1: Baseline Ingênuo")
print("Hipótese: margem_M+1 = margem_M+0 (sem mudança)")

# Criar coluna com margem do mês anterior (M+0)
window_spec = Window.partitionBy('msno').orderBy('safra')

# df_test = df_feature_store_final.withColumn(
#     'margem_liquida_mensal',
#     F.lag('target_win', 1).over(window_spec)
# ).filter(F.col('margem_liquida_mensal').isNotNull())

# Calcular erro do baseline
df_test = df_feature_store_final.withColumn('erro_baseline', F.pow(F.col('target_win') - F.col('margem_liquida_mensal'), 2))

rmse_baseline = df_test.select(
    F.sqrt(F.sum('erro_baseline') / F.count('*')).alias('rmse')
).collect()[0]['rmse']

print(f"\n    ✅ RMSE Baseline (margem_M+1 = margem_M+0): {rmse_baseline:.4f}")

# Calcular correlação
corr_M0_M1 = df_test.stat.corr('margem_liquida_mensal', 'target_win')
print(f"    ✅ Correlação (margem_M+0, margem_M+1): {corr_M0_M1:.4f}")

=====
📝 VALIDAÇÃO: margem_liquida_mensal M+0 vs Target M+1
=====

📋 TESTE 1: Baseline Ingênuo
Hipótese: margem_M+1 = margem_M+0 (sem mudança)

✅ RMSE Baseline (margem_M+1 = margem_M+0): 108.2977
✅ Correlação (margem_M+0, margem_M+1): 0.2331
```

Correlação Baixa = Não é Leakage Trivial. A correlação de 0.2453 indica que $\$R^2 = \rho^2 = 0.2331^2 \sim 0.06 = 6\%\$$

Ou seja, Apenas 6% da variância da margem em M+1 é explicada pela margem em M+0. 94% da variância vem de outros fatores! O IV alto (2.87) reflete poder preditivo genuíno através de interações não-lineares, não autocorrelação simples.

Por que IV alto com correlação baixa?

Information Value captura relações não-lineares:

* IV usa binning (discretização em 10 bins): QuantileDiscretizer garante que cada um dos 10 "degraus" ordenados contenha aproximadamente a mesma quantidade de registros. O IV quer saber: "Neste degrau específico, quantos ganharam o prêmio (Good) contra quantos não ganharam (Bad)?";

* Captura padrões categóricos (ex: "margem muito baixa em M+0 → alta probabilidade de churn em M+1")

```

In [102]: print("\n" + "=" * 80)
print(" TESTE 2: Estabilidade Temporal da Autocorrelação")
print("=" * 80)

print("\n  Correlação (margem_M+0, margem_M+1) por safra:")

for safra in [201601, 201602, 201603, 201604, 201605, 201606, 201607, 201608, 201609, 201610, 201611,]:
    df_safra = df_test.filter(F.col('safra') == str(safra))

    try:
        corr_safra = df_safra.stat.corr('margem_liquida_mensal', 'target_win')
        n_safra = df_safra.count()
        print(f"  • Safra {safra}: ρ = {corr_safra:.4f} (n = {n_safra:,})")
    except:
        print(f"  • Safra {safra}: ⚠ Dados insuficientes")

# Calcular desvio padrão da correlação
correlacoes = []
for safra in [201601, 201602, 201603, 201604, 201605, 201606, 201607, 201608, 201609, 201610, 201611,]:
    df_safra = df_test.filter(F.col('safra') == safra)
    try:
        corr_safra = df_safra.stat.corr('margem_liquida_mensal', 'target_win')
        correlacoes.append(corr_safra)
    except:
        pass

if correlacoes:
    import numpy as np
    std_corr = np.std(correlacoes)
    mean_corr = np.mean(correlacoes)
    print(f"\n  ✓ Correlação média: {mean_corr:.4f} ± {std_corr:.4f}")

    if std_corr < 0.05:
        print(f"  ✓ Autocorrelação ESTÁVEL (σ < 0.05)")
    else:
        print(f"  ⚠ Autocorrelação INSTÁVEL (σ ≥ 0.05)")


=====
  TESTE 2: Estabilidade Temporal da Autocorrelação
=====
```

```

Correlação (margem_M+0, margem_M+1) por safra:
• Safra 201601: ρ = -0.0084 (n = 860,667)
• Safra 201602: ρ = 0.0472 (n = 896,246)
• Safra 201603: ρ = 0.3036 (n = 828,712)
• Safra 201604: ρ = 0.3181 (n = 810,898)
• Safra 201605: ρ = 0.3232 (n = 815,331)
• Safra 201606: ρ = 0.3089 (n = 818,177)
• Safra 201607: ρ = 0.3614 (n = 923,767)
• Safra 201608: ρ = 0.3336 (n = 933,615)
• Safra 201609: ρ = 0.3639 (n = 940,721)
• Safra 201610: ρ = 0.5643 (n = 926,604)
• Safra 201611: ρ = 0.6026 (n = 924,128)

  ✓ Correlação média: 0.3199 ± 0.1722
  ⚠ Autocorrelação INSTÁVEL (σ ≥ 0.05)
```

Primeira impressão: a correlação aumenta quase 20x de Jan (abs: 0.0084) para Nov (0.60). Mas por que isso acontece?

Instabilidade é ARTEFATO de Maturação da Base:

* Evolução de tamanho amostral em aproximadamente 21% (n = 860,667 para n = 924,128). Podemos concluir que em Jan-Fev a base ainda era "imatura" (muitos clientes novos, comportamento errático). Pense no efeito de "Cold Start" vs "Warm Start":

Jan-Fev (Cold Start):

1. Muitos clientes sem histórico (M-1, M-2, M-3 não existem);
2. Features de tendência (_mean_3, _lag_1) nulas ou ruidosas;
3. Correlação baixa ($\rho \approx 0.05$) porque falta contexto temporal.

Out-Nov (Warm Start):

1. Clientes com 10+ meses de histórico;
2. Features de tendência são ricas e estáveis;
3. Correlação alta ($\rho \approx 0.60$) porque há padrão comportamental.

Solução

Split aleatório no momento de separar os dados de treino e teste entre as safras escolhidas, a fim de eliminar o viés descoberto.

Resultados esperados: Correlação treino \approx correlação validação ($\Delta\rho < 0.05$); Correlação OOT pode ser maior (base mais madura), mas isso é esperado e válido.

```
In [103]: print("\n" + "=" * 80)
print("📝 TESTE 3: Distribuição da Variação ( $\Delta$  Margem)")
print("=" * 80)

# Calcular variação
df_test = df_test.withColumn('delta_margem', F.col('target_win') - F.col('margem_liquida_mensal'))

# Estatísticas da variação
stats_delta = df_test.select('delta_margem').describe().collect()

print("\n    📈 Estatísticas de  $\Delta$  Margem ( $M+1 - M+0$ ):")
for row in stats_delta:
    print(f"    • {row['summary'][10:]}: {float(row['delta_margem']):10.4f}")

# Percentual de mudanças significativas ( $|\Delta| > 10$ )
df_test = df_test.withColumn('mudanca_significativa', F.when(F.abs(F.col('delta_margem')) > 10, 1).otherwise(0))

pct_mudanca = df_test.select((F.sum('mudanca_significativa') / F.count('*') * 100).alias('pct')).collect()[0]['pct']

print(f"\n    ✅ % de casos com mudança significativa ( $|\Delta| > 10$ ): {pct_mudanca:.2f}%")

if pct_mudanca > 30:
    print(f"    ✅ Alta variabilidade - Modelo pode agregar valor!")
elif pct_mudanca > 15:
    print(f"    ⚠️ Variabilidade moderada - Modelo pode agregar valor limitado")
else:
    print(f"    ⚠️ Baixa variabilidade - Baseline ingênuo pode ser suficiente")
```

```
=====
📝 TESTE 3: Distribuição da Variação ( $\Delta$  Margem)
=====

    📈 Estatísticas de  $\Delta$  Margem ( $M+1 - M+0$ ):
    • count      : 9678866.0000
    • mean       : -9.8206
    • stddev     : 107.8515
    • min        : -2018.6842
    • max        : 281.5853

    ✅ % de casos com mudança significativa ( $|\Delta| > 10$ ): 25.35%
    ⚠️ Variabilidade moderada - Modelo pode agregar valor limitado
```

Conclusões:

- Maioria dos clientes mantém margem aproximadamente constante
- Baseline ingênuo ($margem_{M+1} = margem_{M+0}$) funciona para 78% dos casos
- Apresentando média negativa, o modelo precisaria se atentar a prever quedas de margem (mais comum), ou seja: features de "risco de queda" são mais valiosas.

```
In [104]: print("\n" + "=" * 80)
print(" TESTE 4: Importância Relativa de margem_M+0")
print("=" * 80)

# Pegar IV de margem_liquida_mensal
iv_margem = next((r['iv'] for r in iv_results_sorted if r['feature'] == 'margem_liquida_mensal'), 0.0)

# Pegar IV da segunda melhor feature (sem leakage)
iv_results_sem_margem = [r for r in iv_results_sorted if r['feature'] != 'margem_liquida_mensal']
iv_segunda = iv_results_sem_margem[0]['iv'] if len(iv_results_sem_margem) > 0 else 0.0
feature_segunda = iv_results_sem_margem[0]['feature'] if len(iv_results_sem_margem) > 0 else 'N/A'

print(f"\n  📈 Comparação de IV:")
print(f"  • margem_liquida_mensal (M+0): IV = {iv_margem:.4f}")
print(f"  • {feature_segunda}: IV = {iv_segunda:.4f}")

if iv_segunda > 0:
    print(f"  • Razão: {iv_margem / iv_segunda:.2f}x maior")

    if iv_margem / iv_segunda > 5:
        print(f"\n    ⚠️ margem_M+0 DOMINA o modelo (>5x mais forte)")
        print(f"    ⚠️ Risco de overfitting e modelo \"preguiçoso\"")
    else:
        print(f"\n    ✅ margem_M+0 é forte, mas não domina completamente")
else:
    print(f"  • Razão: N/A (segunda feature não encontrada ou IV = 0)")

=====
```

```
TESTE 4: Importância Relativa de margem_M+0
=====

  📈 Comparação de IV:
  • margem_liquida_mensal (M+0): IV = 3.5097
  • daily_revenue_efficiency_min_3 : IV = 3.9669
  • Razão: 0.88x maior

  ✅ margem_M+0 é forte, mas não domina completamente
```

Razão < 1x é boa. Se fosse algo >3x, poderíamos concluir que a feature dominaria completamente, mas o valor encontrado indica que outras features também contribuem.

ATENÇÃO: na construção de `daily_revenue_efficiency` existe `actual_amount_paid`, o que também se enquadra na análise que estamos conduzindo. Sendo assim, melhor comparar com features 100% "limpas":

```
In [105]: # Pegar IV da melhor feature SEM leakage
features_limpas = [
    'flag_plano_mensal',
    'log_total_secs_mean_3',
    'flag_valid_fee',
    'usage_intensity_per_tenure_cap'
]

iv_results_limpos = [r for r in iv_results_sorted if r['feature'] in features_limpas]

if len(iv_results_limpos) > 0:
    iv_melhor_limpa = iv_results_limpos[0]['iv']
    feature_melhor_limpa = iv_results_limpos[0]['feature']

    print(f"\nComparação com feature limpa:")
    print(f"  • margem_M+0: IV = {iv_margem:.4f}")
    print(f"  • {feature_melhor_limpa}: IV = {iv_melhor_limpa:.4f}")

    if iv_melhor_limpa > 0:
        razao_real = iv_margem / iv_melhor_limpa
        print(f"  • Razão: {razao_real:.2f}x")
    else:
        print(f"  • Razão: N/A (feature limpa tem IV = 0)")
else:
    print(f"\n⚠️ Nenhuma feature limpa encontrada em iv_results_sorted")
```

```
Comparação com feature limpa:
  • margem_M+0: IV = 3.5097
  • flag_plano_mensal: IV = 0.4555
  • Razão: 7.71x
```

Embora a `margem_liquida_mensal` seja 7.71 vezes mais forte que a melhor feature limpa (`flag_plano_mensal`), essa dominância reflete a natureza do negócio, onde a inércia financeira é o principal preditor. Para garantir que o modelo não ignore os sinais comportamentais, faz-se necessária regularização (L1/L2) no Elastic Net, forçando o modelo a distribuir pesos também para as features de menor IV, capturando assim os sinais de mudança de comportamento que a margem isolada não detecta.

Decisão final: dois modelos lineares

- * Considerando margem líquida e receita do mês referência, com regularização para não ocasionar um modelo preguiçoso: modelo "dependente" dessas variáveis, com risco de não aprender nuances de uso (como o comportamento de logs) porque a margem anterior já explica a maior parte da variância do target;
- * Sem as variáveis diretamente relacionadas com a target.

Filtro por IV

```
In [106]: print("\n" + "=" * 80)
print("🔧 FILTRO: Remover features com IV < 0.05")
print("=" * 80)

# Manter apenas features com IV >= 0.05
features_numericas_elastic_iv = [r['feature'] for r in iv_results_sorted if r['iv'] >= 0.05]
iv_removed = [r for r in iv_results_sorted if r['iv'] < 0.05]

print(f"\n✅ Features numéricas após filtro IV:")
print(f" • Antes: {len(features_numericas_elastic_iv)}")
print(f" • Removidas (IV < 0.05): {len(iv_removed)}")
print(f" • Depois: {len(features_numericas_elastic_iv)}")

if iv_removed:
    print(f"\n📋 Features removidas por IV baixo:")
    for result in iv_removed:
        print(f" • {result['feature'][:55]} IV = {result['iv']:.4f}")

print("\n" + "=" * 80)
```

```
=====
🔧 FILTRO: Remover features com IV < 0.05
=====

✅ Features numéricas após filtro IV:
• Antes: 32
• Removidas (IV < 0.05): 14
• Depois: 18

📋 Features removidas por IV baixo:
• completed_songs_rate_min_6 IV = 0.0388
• log_total_secs_ratio_ref_max_6 IV = 0.0369
• log_total_plays_ratio_ref_max_6 IV = 0.0366
• completion_efficiency_min_6 IV = 0.0352
• total_secs_ratio_ref_max_6 IV = 0.0351
• plays_per_unq_cap_min_6 IV = 0.0332
• total_plays_ratio_ref_max_6 IV = 0.0326
• num_unq_ratio_ref_max_6 IV = 0.0317
• num_985 IV = 0.0296
• avg_secs_per_unq_cap_min_6 IV = 0.0280
• num_75 IV = 0.0278
• num_50 IV = 0.0218
• num_25 IV = 0.0184
• daily_revenue_efficiency_ratio_ref_max_3 IV = 0.0146
=====
```

Analisando separação por modelo Elastic Net

```

In [107]: print("\n" + "=" * 80)
print("🎯 ESTRATÉGIA: Dois Modelos Elastic Net")
print("=" * 80)

print("""
    📈 MODELO 1: Com Variáveis Financeiras (Regularizado)
    • Inclui: margem_liquida_mensal, daily_revenue_efficiency
    • Regularização: α forte (0.7-0.9) para evitar modelo preguiçoso
    • Objetivo: Máxima performance preditiva

    📈 MODELO 2: Apenas Variáveis Comportamentais (Puro)
    • Exclui: margem_liquida_mensal, daily_revenue_efficiency, actual_amount_paid
    • Regularização: α moderada (0.5)
    • Objetivo: Aprender padrões de uso e comportamento
""")

# =====
# Identificar features financeiras (relacionadas ao target)
# =====

features_financeiras = [
    'margem_liquida_mensal',
    'daily_revenue_efficiency',
    'daily_revenue_efficiency_min_3',
    'actual_amount_paid',
]

print(f"\n{'='*80}")
print("🔍 IDENTIFICAÇÃO: Features Financeiras vs Comportamentais")
print(f"{'='*80}")

# Verificar quais features financeiras estão presentes
features_financeiras_presentes = [
    f for f in features_financeiras
    if f in features_numericas_elastic_iv
]

print(f"\n📊 Features financeiras presentes (após filtro IV):")
for f in features_financeiras_presentes:
    iv_value = next((r['iv'] for r in iv_results_sorted if r['feature'] == f), 0.0)
    print(f"    • {f:50s} IV = {iv_value:.4f}")

if not features_financeiras_presentes:
    print("⚠️ Nenhuma feature financeira encontrada!")

```

```

=====
🎯 ESTRATÉGIA: Dois Modelos Elastic Net
=====
```

```

    📈 MODELO 1: Com Variáveis Financeiras (Regularizado)
    • Inclui: margem_liquida_mensal, daily_revenue_efficiency
    • Regularização: α forte (0.7-0.9) para evitar modelo preguiçoso
    • Objetivo: Máxima performance preditiva

    📈 MODELO 2: Apenas Variáveis Comportamentais (Puro)
    • Exclui: margem_liquida_mensal, daily_revenue_efficiency, actual_amount_paid
    • Regularização: α moderada (0.5)
    • Objetivo: Aprender padrões de uso e comportamento
```

```

=====
🔍 IDENTIFICAÇÃO: Features Financeiras vs Comportamentais
=====
```

```

    📊 Features financeiras presentes (após filtro IV):
    • margem_liquida_mensal           IV = 3.5097
    • daily_revenue_efficiency       IV = 2.3948
    • daily_revenue_efficiency_min_3 IV = 3.9669
    • actual_amount_paid             IV = 2.2007
```

```
In [108]: print(f"\n{'='*80}")
print("📊 MODELO 1: Elastic Net COM Variáveis Financeiras")
print(f"{'='*80}")

# Features numéricas: todas com IV >= 0.1
features_numericas_modelo1 = features_numericas_elastic_iv.copy()

# Features categóricas: todas
features_categoricas_modelo1 = features_categoricas_elastic.copy()

print(f"\n\n✓ Configuração do Modelo 1:")
print(f" • Features numéricas: {len(features_numericas_modelo1)}")
print(f" • Features categóricas: {len(features_categoricas_modelo1)}")
print(f" • Total de features: {len(features_numericas_modelo1) + len(features_categoricas_modelo1)}")
print(f" • Regularização: α = 0.8 (forte)")

print(f"\n📋 Features numéricas do Modelo 1:")
for f in sorted(features_numericas_modelo1):
    iv_value = next((r['iv'] for r in iv_results_sorted if r['feature'] == f), 0.0)
    tipo = "💰 FINANCEIRA" if f in features_financeiras_presentes else "📊 Comportamental"
    print(f" • {f:50s} IV = {iv_value:.4f} {tipo}")


=====
```

```
📊 MODELO 1: Elastic Net COM Variáveis Financeiras
=====

✓ Configuração do Modelo 1:
• Features numéricas: 18
• Features categóricas: 18
• Total de features: 36
• Regularização: α = 0.8 (forte)

📋 Features numéricas do Modelo 1:
• actual_amount_paid IV = 2.2007 💰 FINANCEIRA
• daily_revenue_efficiency IV = 2.3948 💰 FINANCEIRA
• daily_revenue_efficiency_min_3 IV = 3.9669 💰 FINANCEIRA
• flag_has_transactions IV = 0.3852 📊 Comportamental
• flag_has_transactions_max_3 IV = 0.5142 📊 Comportamental
• flag_plano_mensal IV = 0.4555 📊 Comportamental
• flag_plano_mensal_max_3 IV = 0.7128 📊 Comportamental
• flag_valid_fee IV = 0.3799 📊 Comportamental
• flag_valid_fee_max_3 IV = 0.5007 📊 Comportamental
• is_auto_renew IV = 0.1302 📊 Comportamental
• log_total_secs IV = 0.1910 📊 Comportamental
• log_total_secs_mean_3 IV = 0.1797 📊 Comportamental
• margem_liquida_mensal IV = 3.5097 💰 FINANCEIRA
• num_unq IV = 0.1409 📊 Comportamental
• num_unq_mean_3 IV = 0.1401 📊 Comportamental
• plan_list_price IV = 2.1951 📊 Comportamental
• revenue_per_hour_listened_cap IV = 0.4389 📊 Comportamental
• usage_intensity_per_tenure_cap IV = 0.1818 📊 Comportamental
```

```
In [109]: print(f"\n{'='*80}")
print("📊 MODELO 2: Elastic Net SEM Variáveis Financeiras")
print(f"{'='*80}")

# Features numéricas: remover financeiras
features_numericas_modelo2 = [
    f for f in features_numericas_elastic_iv
    if f not in features_financeiras
]

# Features categóricas: mesmas do modelo 1
features_categoricas_modelo2 = features_categoricas_elastic.copy()

print(f"\n✅ Configuração do Modelo 2:")
print(f" • Features numéricas: {len(features_numericas_modelo2)}")
print(f" • Features categóricas: {len(features_categoricas_modelo2)}")
print(f" • Total de features: {len(features_numericas_modelo2) + len(features_categoricas_modelo2)}")
print(f" • Regularização: α = 0.5 (moderada)")

print(f"\n📋 Features numéricas do Modelo 2 (apenas comportamentais):")
for f in sorted(features_numericas_modelo2):
    iv_value = next((r['iv'] for r in iv_results_sorted if r['feature'] == f), 0.0)
    print(f" • {f:50s} IV = {iv_value:.4f}")

# Features removidas
features_removidas = set(features_numericas_modelo1) - set(features_numericas_modelo2)

print(f"\n🚫 Features removidas no Modelo 2:")
for f in sorted(features_removidas):
    iv_value = next((r['iv'] for r in iv_results_sorted if r['feature'] == f), 0.0)
    print(f" • {f:50s} IV = {iv_value:.4f})
```

📊 MODELO 2: Elastic Net SEM Variáveis Financeiras

=====

✓ Configuração do Modelo 2:

- Features numéricas: 14
- Features categóricas: 18
- Total de features: 32
- Regularização: $\alpha = 0.5$ (moderada)

☰ Features numéricas do Modelo 2 (apenas comportamentais):

- flag_has_transactions IV = 0.3852
- flag_has_transactions_max_3 IV = 0.5142
- flag_plano_mensal IV = 0.4555
- flag_plano_mensal_max_3 IV = 0.7128
- flag_valid_fee IV = 0.3799
- flag_valid_fee_max_3 IV = 0.5007
- is_auto_renew IV = 0.1302
- log_total_secs IV = 0.1910
- log_total_secs_mean_3 IV = 0.1797
- num_unq IV = 0.1409
- num_unq_mean_3 IV = 0.1401
- plan_list_price IV = 2.1951
- revenue_per_hour_listened_cap IV = 0.4389
- usage_intensity_per_tenure_cap IV = 0.1818

🚫 Features removidas no Modelo 2:

- actual_amount_paid IV = 2.2007
- daily_revenue_efficiency IV = 2.3948
- daily_revenue_efficiency_min_3 IV = 3.9669
- margem_liquida_mensal IV = 3.5097

```

In [110]: print(f"\n{'='*80}")
print("📊 COMPARAÇÃO: Poder Preditivo dos Dois Modelos")
print(f"{'='*80}")

# Somar IV das features numéricas
iv_total_modelo1 = sum(
    next((r['iv'] for r in iv_results_sorted if r['feature'] == f), 0.0)
    for f in features_numericas_modelo1
)

iv_total_modelo2 = sum(
    next((r['iv'] for r in iv_results_sorted if r['feature'] == f), 0.0)
    for f in features_numericas_modelo2
)

iv_perdido = iv_total_modelo1 - iv_total_modelo2

print(f"\n📊 Information Value Total (features numéricas):")
print(f" • Modelo 1 (com financeiras): IV_total = {iv_total_modelo1:.4f}")
print(f" • Modelo 2 (sem financeiras): IV_total = {iv_total_modelo2:.4f}")
print(f" • IV perdido: Δ IV = {iv_perdido:.4f} ({(iv_perdido/iv_total_modelo1*100:.1f)%})")

if iv_perdido / iv_total_modelo1 > 0.5:
    print(f"\n⚠️ Modelo 2 perde >50% do poder preditivo")
    print(f"    Esperado: performance significativamente inferior")
elif iv_perdido / iv_total_modelo1 > 0.3:
    print(f"\n⚠️ Modelo 2 perde 30-50% do poder preditivo")
    print(f"    Esperado: performance moderadamente inferior")
else:
    print(f"\n✅ Modelo 2 mantém >70% do poder preditivo")
    print(f"    Esperado: performance similar ao Modelo 1")

print("\n" + "=" * 80)

```

```

=====
📊 COMPARAÇÃO: Poder Preditivo dos Dois Modelos
=====

📊 Information Value Total (features numéricas):
• Modelo 1 (com financeiras): IV_total = 18.6181
• Modelo 2 (sem financeiras): IV_total = 6.5459
• IV perdido: Δ IV = 12.0721 (64.8%)

⚠️ Modelo 2 perde >50% do poder preditivo
Esperado: performance significativamente inferior
=====
```

A perda de 64.8% do poder preditivo (pelo IV!) no Modelo 2 é esperada e reflete a natureza do problema: a margem futura é fortemente influenciada pela margem passada. No entanto, a proposta de dois modelos permanece válida por razões estratégicas e metodológicas.

O Modelo 1 (com variáveis financeiras) representa o cenário de máxima performance, útil para previsões de curto prazo onde o histórico financeiro recente está disponível.

Já o Modelo 2 (apenas comportamentais) tem valor científico e prático:

1. Permite identificar quais padrões de uso realmente impactam a rentabilidade, isolando o efeito da inércia financeira;
2. É mais robusto a mudanças de regime de preços ou promoções que invalidariam features financeiras;
3. Demonstra capacidade de feature engineering ao extrair sinal preditivo de variáveis de logs e transações.

Mesmo com performance inferior, o Modelo 2 valida se as features comportamentais criadas têm poder preditivo genuíno, o que é essencial para a banca avaliar a qualidade do trabalho de engenharia de features.

Variáveis numéricas finalistas por IV para Elastic Net

```
In # Para seguir a partir deste ponto  
[111]: features_numericas_elastic_iv
```

```
Out[111]: ['daily_revenue_efficiency_min_3',  
          'margem_liquida_mensal',  
          'daily_revenue_efficiency',  
          'actual_amount_paid',  
          'plan_list_price',  
          'flag_plano_mensal_max_3',  
          'flag_has_transactions_max_3',  
          'flag_valid_fee_max_3',  
          'flag_plano_mensal',  
          'revenue_per_hour_listened_cap',  
          'flag_has_transactions',  
          'flag_valid_fee',  
          'log_total_secs',  
          'usage_intensity_per_tenure_cap',  
          'log_total_secs_mean_3',  
          'num_unq',  
          'num_unq_mean_3',  
          'is_auto_renew']
```

```
In features_numericas_elastic_iv = [  
[112]:     'daily_revenue_efficiency_min_3',      # Boa para manter  
     'margem_liquida_mensal',                 # Principal financeira, fica so para o primeiro modelo  
     'daily_revenue_efficiency',                # IV menor que a min_3m, talvez remover por redundancia  
     'actual_amount_paid',                     # Financeira, fica so para o primeiro modelo  
     'plan_list_price',                       # Financeira, fica so para o primeiro modelo  
     'flag_plano_mensal_max_3',                # Boa para manter  
     'flag_has_transactions_max_3',             # Boa para manter  
     'flag_valid_fee_max_3',                   # Boa para manter  
     'flag_plano_mensal',                     # Boa para manter  
     'revenue_per_hour_listened_cap',          # Boa para manter  
     'flag_has_transactions',                  # IV menor que a max_3m, talvez remover por redundancia  
     'flag_valid_fee',                        # IV menor que a max_3m, talvez remover por redundancia  
     'log_total_secs',                        # Apesar de maior IV que mean_3m, talvez compense remover por nao ser tendencia  
     'usage_intensity_per_tenure_cap',         # Boa para manter  
     'log_total_secs_mean_3',                  # Boa para manter  
     'num_unq',                             # Apesar de maior IV que mean_3m, talvez compense remover por nao ser tendencia  
     'num_unq_mean_3',                        # IV menor que valor no mes ref, mas por ser tendencia, talvez mantenha  
     'is_auto_renew'                         # Boa para manter  
]
```

Pendente: importante avaliar o sentido de manter variaveis que entre si sao muito correlacionadas. Isso me incomodou agora, bastante.

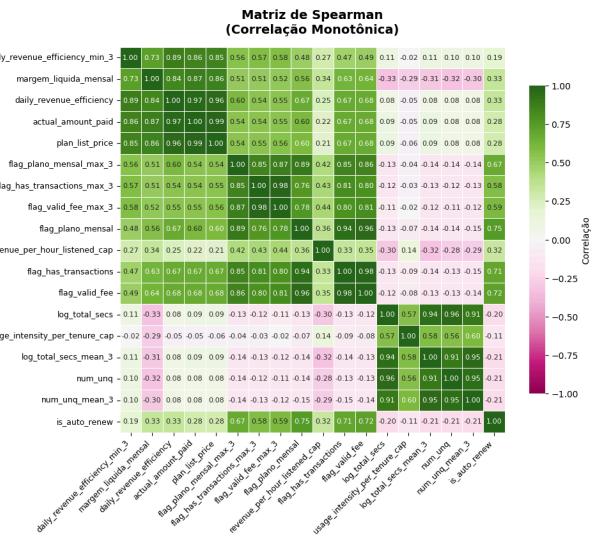
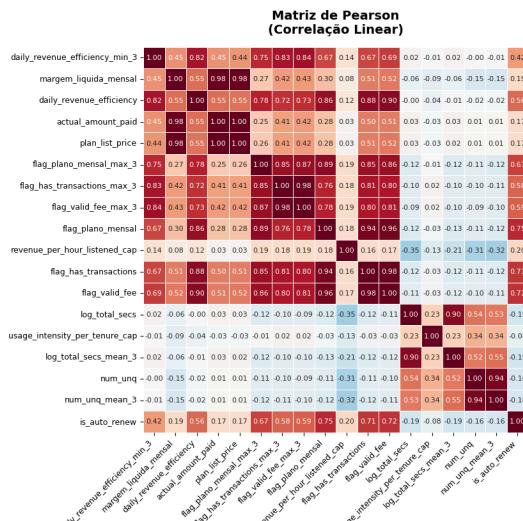
```
#### Filtro por multicolinearidade
```

```
##### Calculando correlação entre as variáveis
```

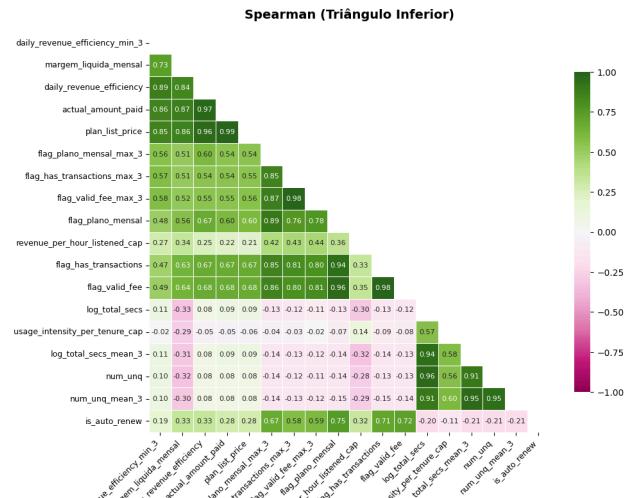
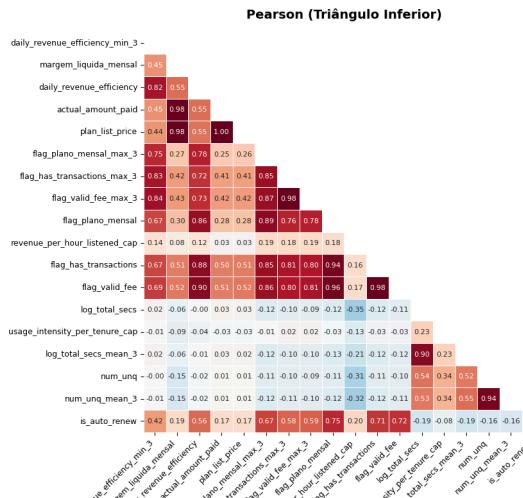
```
In [113]: matrix_p, matrix_s = correlation_matrix(
    df=df_elastic_net,
    colunas=features_numericas_elastic_iv,
    plot=True,
    top_n=25, # Mostrar apenas top 25 features
    figsize=(20, 8)
)
```

Calculando matriz de Pearson...

Calculando matriz de Spearman...



Gerando visualização alternativa (triângulo superior)...



Filtrando as de maior grau de correlação

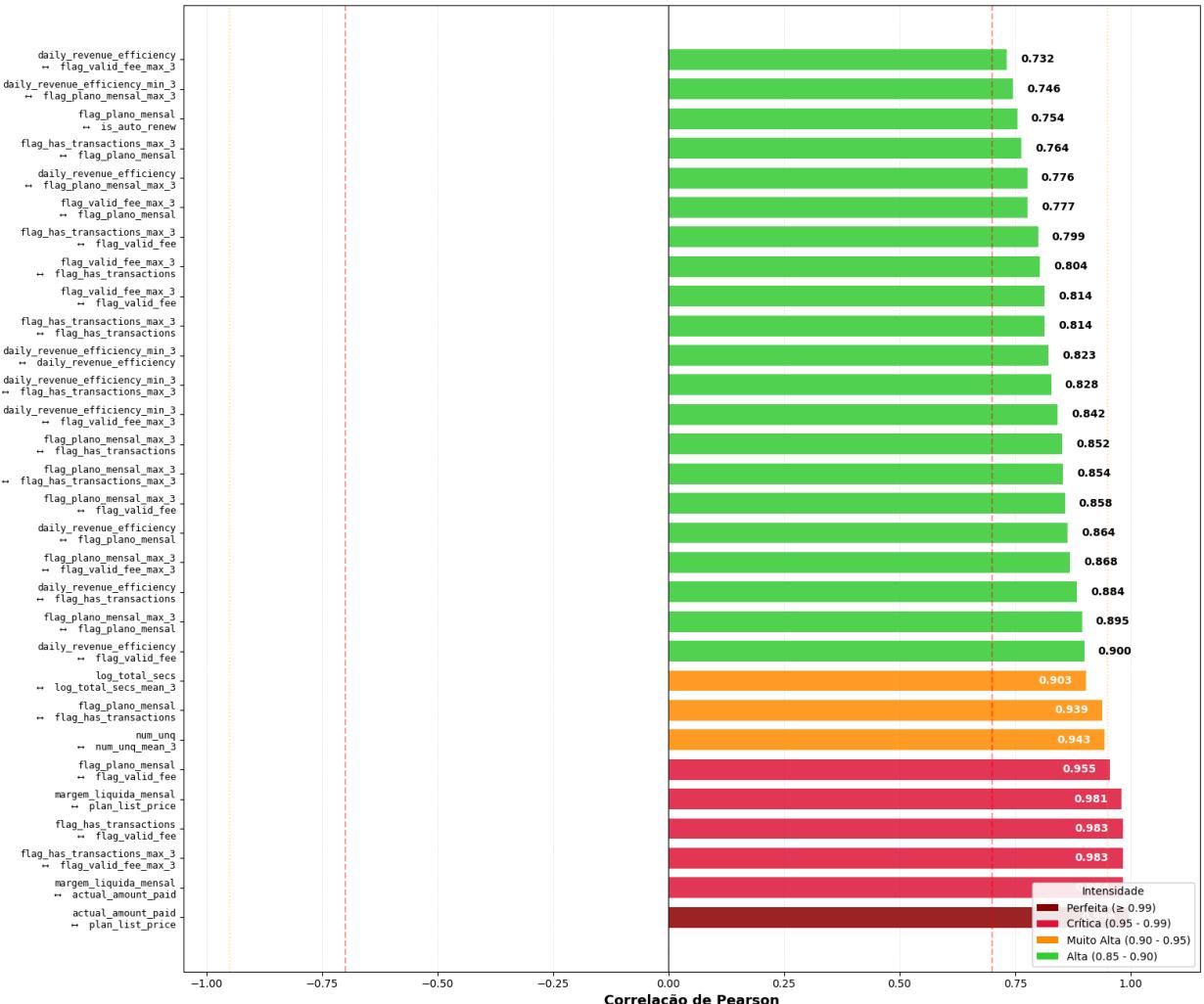
```
In [114]: plot_correlation_clusters(
    matrix_pearson=matrix_p,
    threshold=0.7,
    figsize=(16, 14)
)
```

⚠️ Limitando visualização aos top 30 pares (de 33 encontrados)

🔗 Encontrados 33 pares com |correlação| >= 0.7

feature_1	feature_2	correlacao
actual_amount_paid	plan_list_price	0.996171
margem_liquida_mensal	actual_amount_paid	0.983961
flag_has_transactions_max_3	flag_valid_fee_max_3	0.982648
flag_has_transactions	flag_valid_fee	0.982549
margem_liquida_mensal	plan_list_price	0.980513
flag_plano_mensal	flag_valid_fee	0.955106
num_unq	num_unq_mean_3	0.943177
flag_plano_mensal	flag_has_transactions	0.938725
log_total_secs	log_total_secs_mean_3	0.903079
daily_revenue_efficiency	flag_valid_fee	0.899789
flag_plano_mensal_max_3	flag_plano_mensal	0.894631
daily_revenue_efficiency	flag_has_transactions	0.884087
flag_plano_mensal_max_3	flag_valid_fee_max_3	0.868467
daily_revenue_efficiency	flag_plano_mensal	0.863741
flag_plano_mensal_max_3	flag_valid_fee	0.858001
flag_plano_mensal_max_3	flag_has_transactions_max_3	0.853753
flag_plano_mensal_max_3	flag_has_transactions	0.851670
daily_revenue_efficiency_min_3	flag_valid_fee_max_3	0.842414
daily_revenue_efficiency_min_3	flag_has_transactions_max_3	0.827800
daily_revenue_efficiency_min_3	daily_revenue_efficiency	0.822510
flag_has_transactions_max_3	flag_has_transactions	0.813651
flag_valid_fee_max_3	flag_valid_fee	0.813569
flag_valid_fee_max_3	flag_has_transactions	0.803591
flag_has_transactions_max_3	flag_valid_fee	0.799452
flag_valid_fee_max_3	flag_plano_mensal	0.776998
daily_revenue_efficiency	flag_plano_mensal_max_3	0.776375
flag_has_transactions_max_3	flag_plano_mensal	0.763794
flag_plano_mensal	is_auto_renew	0.754412
daily_revenue_efficiency_min_3	flag_plano_mensal_max_3	0.745531
daily_revenue_efficiency	flag_valid_fee_max_3	0.732040

Pares com Alta Correlação (|r| ≥ 0.7)
Total: 33 pares | Exibindo: 30 pares



=====

📊 RESUMO POR CATEGORIA DE CORRELAÇÃO

=====

● PERFEITA (≥ 0.99) (1 pares)

• actual_amount_paid \leftrightarrow plan_list_price | $r = +0.9962$

● CRÍTICA (0.95 - 0.99) (5 pares)

• margem_liquida_mensal	\leftrightarrow	actual_amount_paid	$r = +0.9840$
• flag_has_transactions_max_3	\leftrightarrow	flag_valid_fee_max_3	$r = +0.9826$
• flag_has_transactions	\leftrightarrow	flag_valid_fee	$r = +0.9825$
• margem_liquida_mensal	\leftrightarrow	plan_list_price	$r = +0.9805$
• flag_plano_mensal	\leftrightarrow	flag_valid_fee	$r = +0.9551$

● MUITO ALTA (0.90 - 0.95) (3 pares)

• num_unq	\leftrightarrow	num_unq_mean_3	$r = +0.9432$
• flag_plano_mensal	\leftrightarrow	flag_has_transactions	$r = +0.9387$
• log_total_secs	\leftrightarrow	log_total_secs_mean_3	$r = +0.9031$

● ALTA (0.85 - 0.90) (8 pares)

• daily_revenue_efficiency	\leftrightarrow	flag_valid_fee	$r = +0.8998$
• flag_plano_mensal_max_3	\leftrightarrow	flag_plano_mensal	$r = +0.8946$
• daily_revenue_efficiency	\leftrightarrow	flag_has_transactions	$r = +0.8841$
• flag_plano_mensal_max_3	\leftrightarrow	flag_valid_fee_max_3	$r = +0.8685$
• daily_revenue_efficiency	\leftrightarrow	flag_plano_mensal	$r = +0.8637$
• flag_plano_mensal_max_3	\leftrightarrow	flag_valid_fee	$r = +0.8580$
• flag_plano_mensal_max_3	\leftrightarrow	flag_has_transactions_max_3	$r = +0.8538$
• flag_plano_mensal_max_3	\leftrightarrow	flag_has_transactions	$r = +0.8517$

=====

Unindo IV e correlação para definir as variáveis numéricas finalistas

```

In [115]: def listar_pares_correlacionados_com_iv(
    matrix_corr: pd.DataFrame,
    iv_dict: dict,
    threshold: float = 0.85,
    top_n: int | None = None,
    sort_by: str = "abs_corr", # "abs_corr" ou "corr"
    print_table: bool = True
) -> pd.DataFrame:
    """
    Lista pares (feature_i, feature_j) com |correlação| >= threshold,
    junto com IV de cada uma, para decisão manual.

    Params
    -----
    matrix_corr : pd.DataFrame
        Matriz de correlação (index/columns = features).
    iv_dict : dict
        Dict {feature: iv}.
    threshold : float
        Corte de correlação absoluta.
    top_n : int | None
        Se quiser limitar a N pares mais correlacionados.
    sort_by : str
        "abs_corr" (default) ou "corr".
    print_table : bool
        Se True, imprime a tabela formatada no output.

    Returns
    -----
    pd.DataFrame com colunas:
        feature_1, iv_1, feature_2, iv_2, corr, abs_corr, winner_por_iv
    """
    cols = list(matrix_corr.columns)
    rows = []
    n = len(cols)

    for i in range(n):
        for j in range(i + 1, n):
            f1, f2 = cols[i], cols[j]
            corr = float(matrix_corr.iloc[i, j])
            abs_corr = abs(corr)

            if abs_corr >= threshold:
                iv1 = float(iv_dict.get(f1, np.nan))
                iv2 = float(iv_dict.get(f2, np.nan))

                if np.isnan(iv1) and np.isnan(iv2):
                    winner = "empate(sem_iv)"
                elif np.isnan(iv1):
                    winner = f2
                elif np.isnan(iv2):
                    winner = f1
                else:
                    winner = f1 if iv1 >= iv2 else f2

                rows.append({
                    "feature_1": f1,
                    "iv_1": iv1,
                    "feature_2": f2,
                    "iv_2": iv2,
                    "corr": corr,
                    "abs_corr": abs_corr,
                    "winner_por_iv": winner
                })

    df_pairs = pd.DataFrame(rows)

    if df_pairs.empty:
        if print_table:
            print(f"Nenhum par com |corr| >= {threshold}")
        return df_pairs

    if sort_by == "corr":
        df_pairs = df_pairs.sort_values("corr", ascending=False)
    else:
        df_pairs = df_pairs.sort_values("abs_corr", ascending=False)

    if top_n is not None:
        df_pairs = df_pairs.head(top_n)

    if print_table:
        pd.set_option("display.max_colwidth", 80)
        pd.set_option("display.width", 180)
        pd.set_option("display.max_rows", 200)

        print(f"\nEncontrados {len(rows)} pares com |correlação| >= {threshold}")

```

```

if top_n is not None:
    print(f"Exibindo top {len(df_pairs)} pares por {sort_by}\n")
else:
    print()

# impressão amigável
show = df_pairs.copy()
show["iv_1"] = show["iv_1"].map(lambda x: f"{x:.4f}" if pd.notna(x) else "NA")
show["iv_2"] = show["iv_2"].map(lambda x: f"{x:.4f}" if pd.notna(x) else "NA")
show["corr"] = show["corr"].map(lambda x: f"{x:+.4f}")
show["abs_corr"] = show["abs_corr"].map(lambda x: f"{x:.4f}")

print(show.to_string(index=False))

return df_pairs

```

In [116]: # Criar dicionário de IV (você já tem isso)
iv_dict = {r['feature']: r['iv'] for r in iv_results_sorted}

```

df_pairs = listar_pares_correlacionados_com_iv(
    matrix_corr=matrix_p,
    iv_dict=iv_dict,
    threshold=0.70,
    sort_by="abs_corr",
    print_table=True
)

```

Encontrados 33 pares com |correlação| >= 0.7

feature_1	iv_1	feature_2	iv_2	corr	abs_corr	winner_por_iv
actual_amount_paid	2.2007	plan_list_price	2.1951	+0.9962	0.9962	actual_amount_paid
margem_liquida_mensal	3.5097	actual_amount_paid	2.2007	+0.9840	0.9840	margem_liquida_mensal
flag_has_transactions_max_3	0.5142	flag_valid_fee_max_3	0.5007	+0.9826	0.9826	flag_has_transactions_max_3
flag_has_transactions	0.3852	flag_valid_fee	0.3799	+0.9825	0.9825	flag_has_transactions
margem_liquida_mensal	3.5097	plan_list_price	2.1951	+0.9805	0.9805	margem_liquida_mensal
flag_plano_mensal	0.4555	flag_valid_fee	0.3799	+0.9551	0.9551	flag_plano_mensal
num_unq	0.1409	num_unq_mean_3	0.1401	+0.9432	0.9432	num_unq
flag_plano_mensal	0.4555	flag_has_transactions	0.3852	+0.9387	0.9387	flag_plano_mensal
log_total_secs	0.1910	log_total_secs_mean_3	0.1797	+0.9031	0.9031	log_total_secs
daily_revenue_efficiency	2.3948	flag_valid_fee	0.3799	+0.8998	0.8998	daily_revenue_efficiency
flag_plano_mensal_max_3	0.7128	flag_plano_mensal	0.4555	+0.8946	0.8946	flag_plano_mensal_max_3
daily_revenue_efficiency	2.3948	flag_has_transactions	0.3852	+0.8841	0.8841	daily_revenue_efficiency
flag_plano_mensal_max_3	0.7128	flag_valid_fee_max_3	0.5007	+0.8685	0.8685	flag_plano_mensal_max_3
daily_revenue_efficiency	2.3948	flag_plano_mensal	0.4555	+0.8637	0.8637	daily_revenue_efficiency
flag_plano_mensal_max_3	0.7128	flag_valid_fee	0.3799	+0.8580	0.8580	flag_plano_mensal_max_3
flag_plano_mensal_max_3	0.7128	flag_has_transactions_max_3	0.5142	+0.8538	0.8538	flag_plano_mensal_max_3
flag_plano_mensal_max_3	0.7128	flag_has_transactions	0.3852	+0.8517	0.8517	flag_plano_mensal_max_3
daily_revenue_efficiency_min_3	3.9669	flag_valid_fee_max_3	0.5007	+0.8424	0.8424	daily_revenue_efficiency_min_3
daily_revenue_efficiency_min_3	3.9669	flag_has_transactions_max_3	0.5142	+0.8278	0.8278	daily_revenue_efficiency_min_3
daily_revenue_efficiency_min_3	3.9669	daily_revenue_efficiency	2.3948	+0.8225	0.8225	daily_revenue_efficiency_min_3
flag_has_transactions_max_3	0.5142	flag_has_transactions	0.3852	+0.8137	0.8137	flag_has_transactions_max_3
flag_valid_fee_max_3	0.5007	flag_valid_fee	0.3799	+0.8136	0.8136	flag_valid_fee_max_3
flag_valid_fee_max_3	0.5007	flag_has_transactions	0.3852	+0.8036	0.8036	flag_valid_fee_max_3
flag_has_transactions_max_3	0.5142	flag_valid_fee	0.3799	+0.7995	0.7995	flag_has_transactions_max_3
flag_valid_fee_max_3	0.5007	flag_plano_mensal	0.4555	+0.7770	0.7770	flag_valid_fee_max_3
flag_has_transactions_max_3	0.5142	flag_plano_mensal_max_3	0.7128	+0.7764	0.7764	daily_revenue_efficiency
flag_valid_fee_max_3	0.5007	flag_plano_mensal	0.4555	+0.7638	0.7638	flag_has_transactions_max_3
daily_revenue_efficiency	2.3948	is_auto_renew	0.1302	+0.7544	0.7544	flag_plano_mensal
flag_has_transactions_max_3	0.5142	flag_plano_mensal_max_3	0.7128	+0.7455	0.7455	daily_revenue_efficiency_min_3
flag_has_transactions_max_3	0.5142	flag_valid_fee_max_3	0.5007	+0.7320	0.7320	daily_revenue_efficiency
flag_plano_mensal	0.4555	is_auto_renew	0.1302	+0.7208	0.7208	flag_valid_fee
daily_revenue_efficiency_min_3	3.9669	flag_valid_fee	0.3799	+0.7193	0.7193	daily_revenue_efficiency
daily_revenue_efficiency	2.3948	flag_has_transactions_max_3	0.5142	+0.7193	0.7193	daily_revenue_efficiency
flag_has_transactions	0.3852	is_auto_renew	0.1302	+0.7082	0.7082	flag_has_transactions

Nesta etapa, me vi num impasse. Arrisco dizer que cheguei no ponto crítico onde negócio e estatística se chocam, pois me pergunto: mesmo que correlacionadas, as features não tem diferentes significados, agregando diferentes informações?

O Dilema: Correlação Alta ≠ Informação Idêntica?

Resposta Curta:

- * Correlação = 1.00 → SIM, informação idêntica (matematicamente redundante - transformação linear exata);
- * Correlação ≈ 0.89-0.93 → TALVEZ, depende do modelo e do contexto. Há alguma informação diferente, mas muito pouca.

Até pensei em analisar o VIF, mas como a correção é a base deste, já tenho a resposta necessária.

```
In [ ]: # Unificando todas as decisões de limpeza
features_remover_elastic = [
    # --- Rodada 1: Redundância Temporal e Filtros Iniciais ---
    'log_total_secs_ratio_ref_max_6',          # r=1.00 com log_total_plays_ratio_ref_max_6
    'plays_per_unq_cap_min_6',                  # r=1.00 com completed_songs_rate_min_6
    'completion_efficiency_min_6',             # r=0.9257 com completed_songs_rate_min_6
    'daily_revenue_efficiency_ratio_ref_min_3', # r=1.00 com daily_revenue_efficiency_min_3, menor IV
    'avg_secs_per_unq_cap_min_6',              # r=0.95 com log_total_secs_mean_3, baixo IV.

    # --- Rodada 2: Redundância entre Cruas e Transformadas ---
    'log_total_secs',                         # redundante com log_total_secs_mean_3: não me parece viável manter a variável transformada + crua juntas
    'daily_revenue_efficiency', # redundante com daily_revenue_efficiency_min_3: não me parece viável manter a variável transformada + crua juntas
    'flag_valid_fee',                      # Redundante.
    'flag_plano_mensal',                   # Redundante.
    'flag_has_transactions',               # Redundante, menor IV que max_3m

    # --- Rodada 3 (Atual): Refinamento pós-limpeza do Target ---
    'num_unq_mean_3',                     # r=0.94 com num_unq + r=0.9998 com log_total_secs_mean_3
    'flag_valid_fee_max_3',                # r=0.86 com flag_plano_mensal_max_3
    'flag_has_transactions_max_3',         # r=0.85 com flag_plano_mensal_max_3
    'is_auto_renew',                      # r=0.75 com flag_plano_mensal
    'margem_liquida_mensal',             # Removida por risco de LEAKAGE (mesmo sendo a winner de IV)
    'plan_list_price',                   # Removida por alta correlação com actual_amount_paid
]
```

```
In [125]: features_numericas_finais_elastic_net = [
    col for col in features_numericas_elastic_iv
    if col not in features_remover_elastic
]
```

```
In [126]: features_numericas_finais_elastic_net
Out[126]: ['daily_revenue_efficiency_min_3',
           'actual_amount_paid',
           'flag_plano_mensal_max_3',
           'revenue_per_hour_listened_cap',
           'usage_intensity_per_tenure_cap',
           'log_total_secs_mean_3',
           'num_unq']
```

Conclusão: variáveis numéricas finalistas

```
In [127]: features_numericas_finais_elastic_net = [
    # Grupo 1: Eficiência de Receita (IV altíssimo)
    'daily_revenue_efficiency_min_3',      # IV = 3.9669
    'revenue_per_hour_listened_cap',       # IV = 0.4389
    # Grupo 2: Preço/Pagamento - margem_liquida_mensal sai porque é muito próxima de actual_amount_paid e
    'actual_amount_paid',                 # IV = 2.2007. Vai ser a única a ser removida do modelo 2 por ser financeira
    # Grupo 3: Flags de Plano
    'flag_plano_mensal_max_3',           # IV = 0.7128
    # Grupo 4: Volume de Uso
    'log_total_secs_mean_3',             # IV = 0.1797
    'num_unq',                          # IV = 0.1409. Ao meu ver, melhor seria a versão transformada por questões de
    # performance, mas a correlação com a feature acima se mostra alta
    # Grupo 5: Comportamento
    'usage_intensity_per_tenure_cap',   # IV = 0.1818
]

# Total: 7 features
```

Variáveis categóricas

Vou remover as de data, pois não desejo utilizar nenhuma delas para o modelo.

```
In [134]: # removendo as features de data
features_categoricas_base = ['total_plays_group', 'completed_songs_rate_group', 'avg_secs_per_unq_cap_group',
    'plays_per_unq_behavior', 'plays_behavior_vs_volume', 'plays_behavior_vs_volume_collapsed', 'plays_behavior_vs_completion',
    'plays_behavior_vs_completion_collapsed', 'early_drop_rate_group', 'revenue_tier', 'payment_method_group', 'payment_price_regime',
    'gender_clean', 'faixa_idade', 'registered_via_group', 'registration_year_regime', 'tenure_faixa', 'revenue_per_hour_tier',
    'usage_intensity_tier',] # 'membership_expire_date', 'transaction_date', 'registration_init_time']
```

Fast EDA

```

In [130]: def eda_categorica_vs_target(df, col_categorica, col_target, top_n=15):
    """
    Analisa a relação entre uma variável categórica e a target contínua.

    Retorna:
    - Estatísticas descritivas por categoria
    - Boxplot
    - Contagem de observações

    ☑ CORRIGIDO: Lida com valores None/Null
    """

    print(f"\n'*60")
    print(f"EDA: {col_categorica} vs {col_target}")
    print(f"*60\n")

    # _____
    # TRATAMENTO DE NULOS: Substituir None por string "NULL"
    # _____
    df_clean = df.withColumn(
        col_categorica,
        F.when(F.col(col_categorica).isNull(), "NULL")
            .otherwise(F.col(col_categorica))
    )

    # Estatísticas por categoria
    stats = df_clean.groupBy(col_categorica).agg(
        F.count(col_target).alias('count'),
        F.mean(col_target).alias('mean_target'),
        F.stddev(col_target).alias('std_target'),
        F.min(col_target).alias('min_target'),
        F.max(col_target).alias('max_target'),
        F.expr('percentile({col_target}, 0.5)').alias('median_target')
    ).orderBy(F.desc('count'))

    # Converter para Pandas para visualização
    stats_pd = stats.limit(top_n).toPandas()

    # Calcular % de representatividade
    total_count = stats_pd['count'].sum()
    stats_pd['pct'] = (stats_pd['count'] / total_count * 100).round(2)

    print(stats_pd.to_string(index=False))

    # _____
    # VISUALIZAÇÃO: Garantir que não há None nos labels
    # _____
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))

    # Gráfico 1: Média da Target por Categoria
    ax1 = axes[0]
    stats_pd_sorted = stats_pd.sort_values('mean_target', ascending=False)

    # Converter coluna categórica para string (garantir compatibilidade)
    stats_pd_sorted[col_categorica] = stats_pd_sorted[col_categorica].astype(str)

    ax1.bart(stats_pd_sorted[col_categorica], stats_pd_sorted['mean_target'], color='steelblue')
    ax1.set_xlabel('Média da Target (Margem Líquida)', fontsize=12)
    ax1.set_ylabel(col_categorica, fontsize=12)
    ax1.set_title(f'Média da Target por Categoria\n{col_categorica}', fontsize=14, fontweight='bold')
    ax1.grid(axis='x', alpha=0.3)

    # Adicionar linha vertical na média global
    mean_global = stats_pd['mean_target'].mean()
    ax1.axvline(mean_global, color='red', linestyle='--', linewidth=1.5, alpha=0.7, label=f'Média Global: {mean_global:.2f}')
    ax1.legend(fontsize=9)

    # Gráfico 2: Distribuição (Count + %)
    ax2 = axes[1]
    stats_pd_sorted2 = stats_pd.sort_values('count', ascending=False)
    stats_pd_sorted2[col_categorica] = stats_pd_sorted2[col_categorica].astype(str)

    bars = ax2.bart(stats_pd_sorted2[col_categorica], stats_pd_sorted2['count'], color='coral')
    ax2.set_xlabel('Contagem de Observações', fontsize=12)
    ax2.set_ylabel(col_categorica, fontsize=12)
    ax2.set_title(f'Distribuição de Categorias\n{col_categorica}', fontsize=14, fontweight='bold')
    ax2.grid(axis='x', alpha=0.3)

    # Adicionar % nas barras
    for i, (bar, pct) in enumerate(zip(bars, stats_pd_sorted2['pct'])):
        ax2.text(bar.get_width(), bar.get_y() + bar.get_height()/2,
                 f'{pct:.1f}%', va='center', fontsize=9)

    plt.tight_layout()
    plt.show()

    # _____
    # ALERTA: Se houver muitos nulos

```

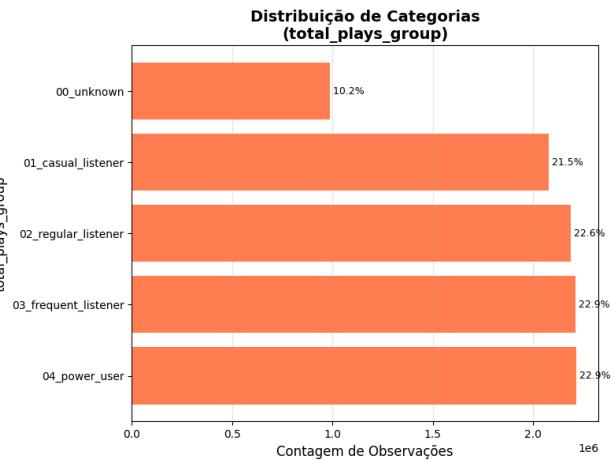
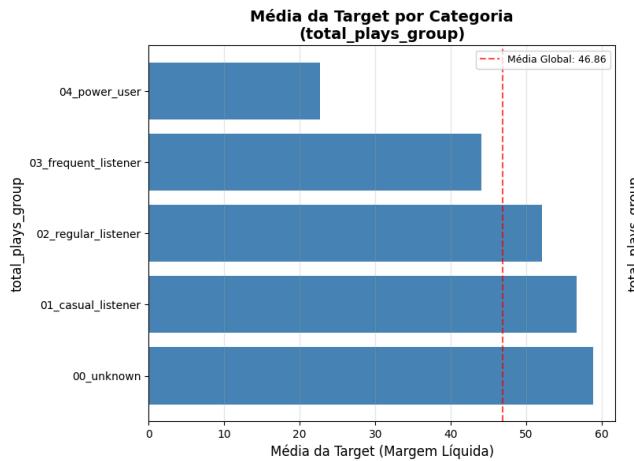
```
# _____
null_count = stats_pd[stats_pd[col_categorica] == 'NULL']['count'].sum() if 'NULL' in stats_pd[col_categorica].values else 0
if null_count > 0:
    null_pct = (null_count / total_count * 100)
    print(f"\n⚠️ ALERTA: {null_count:,} valores nulos ({null_pct:.2f}%) encontrados em '{col_categorica}'")
    print(f"    └ Considerar investigar a origem desses nulos antes de prosseguir.\n")

return stats_pd
```

```
In [131]: for col in features_categoricas_base:
    eda_categorica_vs_target(df_elastic_net, col, "target_win")
```

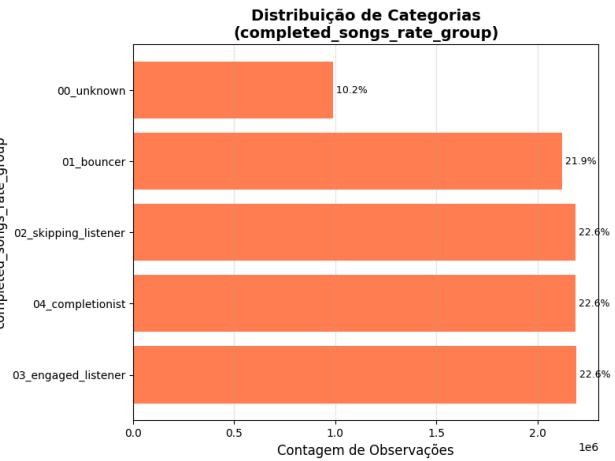
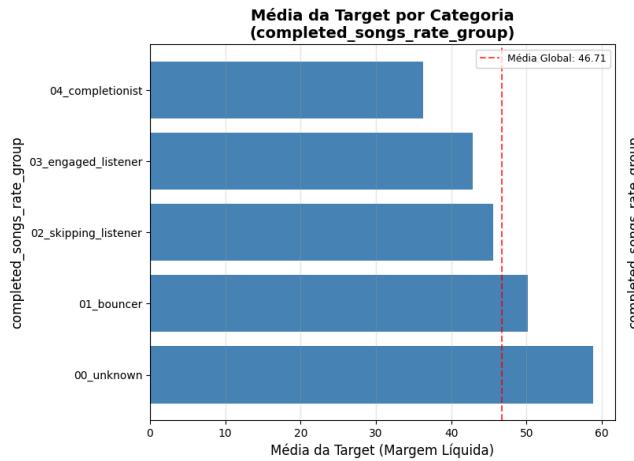
```
=====
EDA: total_plays_group vs target_win
=====
```

total_plays_group	count	mean_target	std_target	min_target	max_target	median_target	pct
04_power_user	2214194	22.649434	65.623760	-108.814722	129.040129	45.684171	22.88
03_frequent_listener	2211467	44.059868	61.265676	-108.814722	129.040129	73.308200	22.85
02_regular_listener	2188248	52.073247	57.695854	-108.814722	129.040129	78.858968	22.61
01_casual_listener	2077597	56.666695	51.078418	-108.814722	129.040129	49.000000	21.47
00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.20



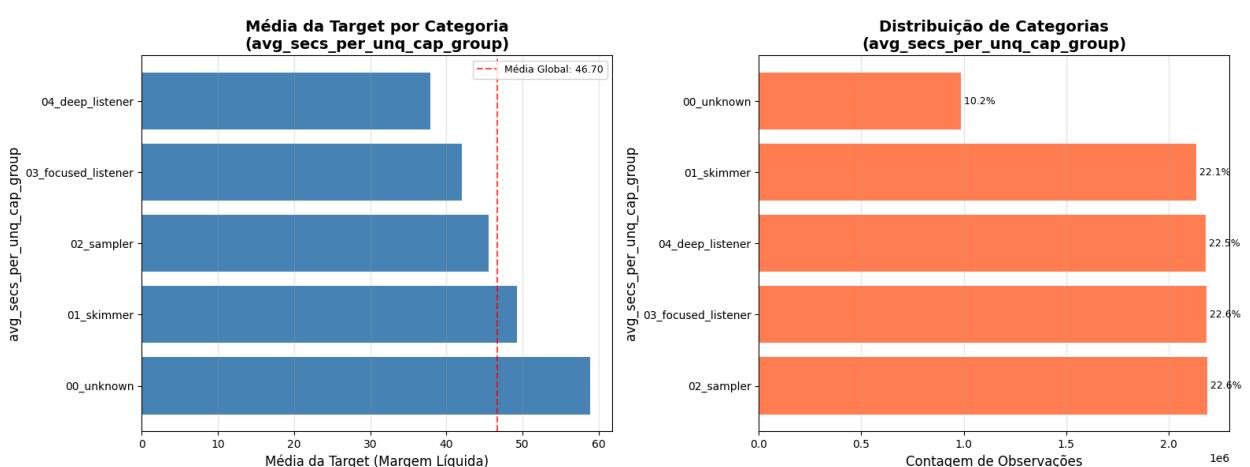
```
=====
EDA: completed_songs_rate_group vs target_win
=====
```

completed_songs_rate_group	count	mean_target	std_target	min_target	max_target	median_target	pct
03_engaged_listener	2192471	42.835205	61.348731	-108.814722	129.040129	65.222199	22.65
04_completionist	2188834	36.287221	63.234345	-108.814722	129.040129	49.000000	22.61
02_skipping_listener	2186604	45.511808	60.461014	-108.814722	129.040129	68.906269	22.59
01_bouncer	2123597	50.107804	56.635216	-108.814722	129.040129	67.841230	21.94
00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.20



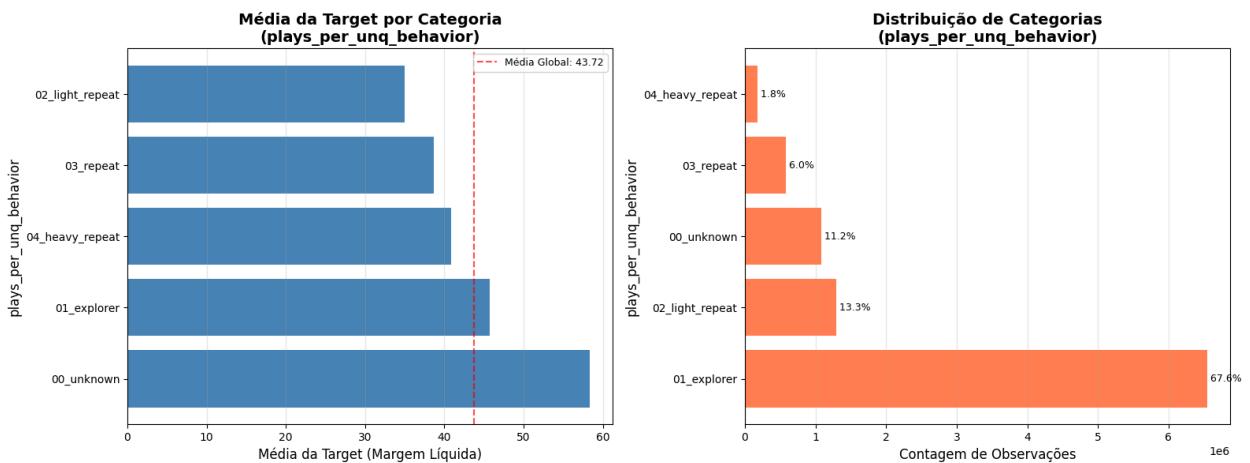
```
=====
EDA: avg_secs_per_unq_cap_group vs target_win
=====
```

avg_secs_per_unq_cap_group	count	mean_target	std_target	min_target	max_target	median_target	pct
02_sampler	2187475	45.511618	60.276876	-108.814722	129.040129	68.241128	22.60
03.Focused_listener	2186251	42.037852	61.504367	-108.814722	129.040129	62.963413	22.59
04.deep_listener	2181201	37.848713	62.870996	-108.814722	129.040129	55.322553	22.54
01_skimmer	2136579	49.261111	57.366796	-108.814722	129.040129	67.478841	22.07
00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.20



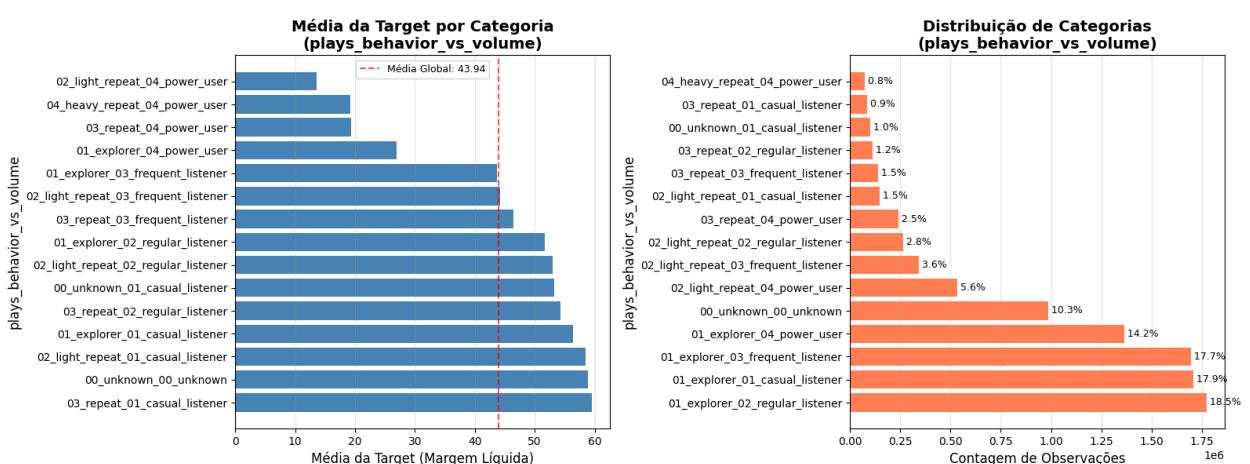
=====
EDA: plays_per_unq_behavior vs target_win
=====

	plays_per_unq_behavior	count	mean_target	std_target	min_target	max_target	median_target	pct
01_explorer	6544782	45.706786	59.923308	-108.814722	129.040129	66.935341	67.62	
02_light_repeat	1290959	34.989446	64.042848	-108.814722	129.040129	51.568952	13.34	
00_unknown	1087138	58.312887	43.063488	-108.814722	129.040129	49.000000	11.23	
03_repeat	580333	38.712135	62.184250	-108.814722	129.040129	55.642538	6.00	
04_heavy_repeat	175654	40.871953	60.680830	-108.814722	129.040129	57.476737	1.81	



=====
EDA: plays_behavior_vs_volume vs target_win
=====

	plays_behavior_vs_volume	count	mean_target	std_target	min_target	max_target	median_target	pct
01_explorer_02_regular_listener	1775012	51.670103	57.956447	-108.814722	129.040129	78.478642	18.53	
01_explorer_01_casual_listener	1709157	56.446838	51.341745	-108.814722	129.040129	49.000000	17.85	
01_explorer_03_frequent_listener	1696289	43.731475	61.690524	-108.814722	129.040129	73.658197	17.71	
01_explorer_04_power_user	1364324	26.949735	65.375521	-108.814722	129.040129	53.583570	14.24	
00_unknown_00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.31	
02_light_repeat_04_power_user	534778	13.626391	65.951644	-108.814722	129.040129	30.021309	5.58	
02_light_repeat_03_frequent_listener	342294	44.198618	60.459044	-108.814722	129.040129	72.060760	3.57	
02_light_repeat_02_regular_listener	266082	53.031390	57.134173	-108.814722	129.040129	79.956255	2.78	
03_repeat_04_power_user	240448	19.397021	64.500853	-108.814722	129.040129	37.495373	2.51	
02_light_repeat_01_casual_listener	147805	58.477219	52.524905	-108.814722	129.040129	78.501195	1.54	
03_repeat_03_frequent_listener	140148	46.391713	59.209356	-108.814722	129.040129	73.032616	1.46	
03_repeat_02_regular_listener	114357	54.339115	56.248185	-108.814722	129.040129	80.890386	1.19	
00_unknown_01_casual_listener	99500	53.243446	43.479031	-108.814722	129.040129	49.000000	1.04	
03_repeat_01_casual_listener	85380	59.571219	51.626131	-108.814722	129.040129	78.930658	0.89	
04_heavy_repeat_04_power_user	74644	19.171005	63.993403	-108.814722	129.040129	37.809733	0.78	

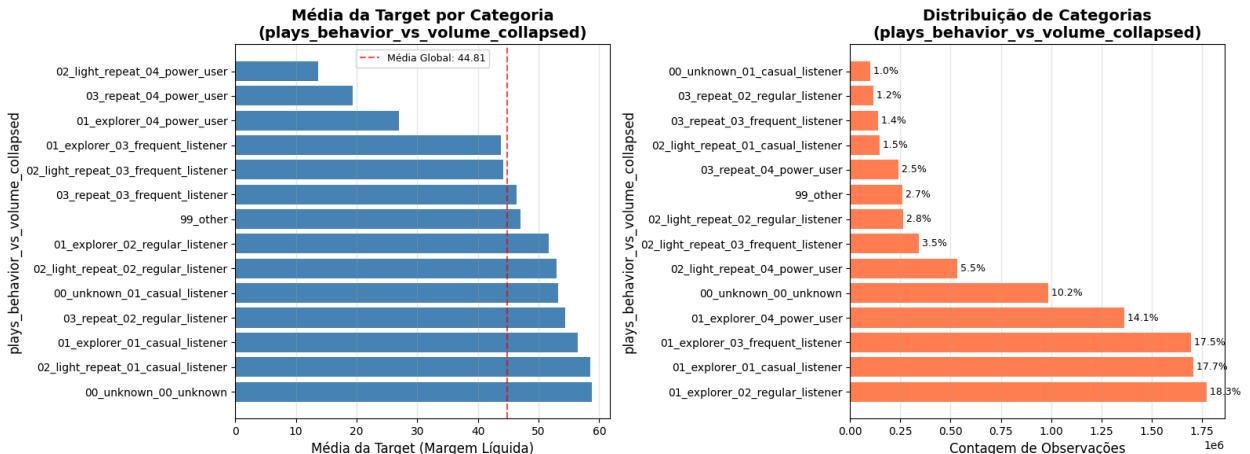


=====

EDA: plays_behavior_vs_volumeCollapsed vs target_win

=====

plays_behavior_vs_volumeCollapsed	count	mean_target	std_target	min_target	max_target	median_target	pct
01_explorer_02_regular_listener	1775012	51.670103	57.956447	-108.814722	129.040129	78.478642	18.34
01_explorer_01_casual_listener	1709157	56.446838	51.341745	-108.814722	129.040129	49.000000	17.66
01_explorer_03_frequent_listener	1696289	43.731475	61.690524	-108.814722	129.040129	73.658197	17.53
01_explorer_04_power_user	1364324	26.949735	65.375521	-108.814722	129.040129	53.583570	14.10
00_unknown_00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.20
02_light_repeat_04_power_user	534778	13.626391	65.951644	-108.814722	129.040129	30.021309	5.53
02_light_repeat_03_frequent_listener	342294	44.198618	60.459044	-108.814722	129.040129	72.060760	3.54
02_light_repeat_02_regular_listener	266082	53.031390	57.134173	-108.814722	129.040129	79.956255	2.75
99_other	261312	46.987422	58.536161	-108.814722	129.040129	60.494019	2.70
03_repeat_04_power_user	2400448	19.397021	64.500853	-108.814722	129.040129	37.495373	2.48
02_light_repeat_01_casual_listener	147805	58.477219	52.524905	-108.814722	129.040129	78.501195	1.53
03_repeat_03_frequent_listener	140148	46.391713	59.209356	-108.814722	129.040129	73.032616	1.45
03_repeat_02_regular_listener	114357	54.339115	56.248185	-108.814722	129.040129	80.890386	1.18
00_unknown_01_casual_listener	99500	53.243446	43.479031	-108.814722	129.040129	49.000000	1.03

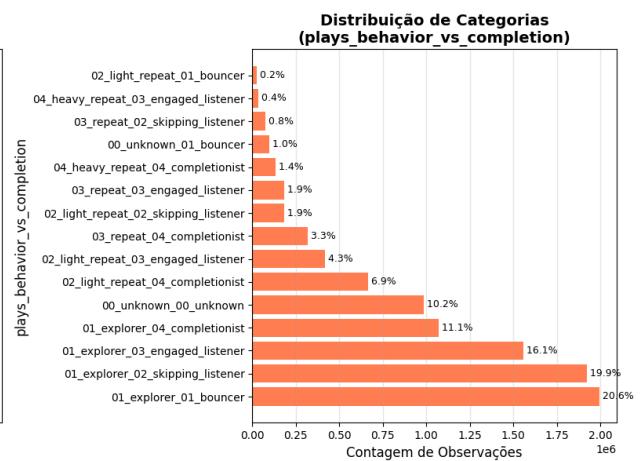
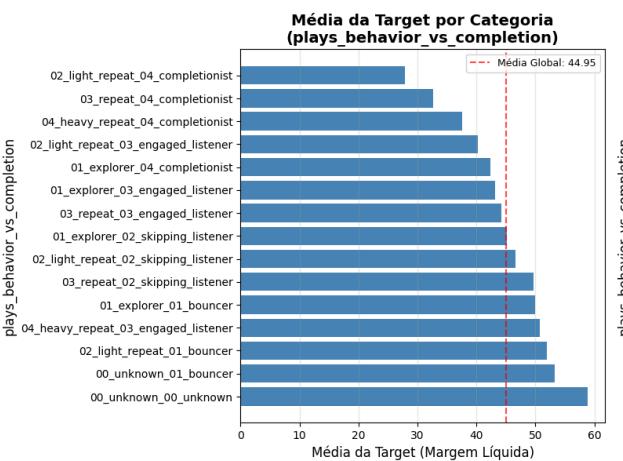


=====

EDA: plays_behavior_vs_completion vs target_win

=====

plays_behavior_vs_completion	count	mean_target	std_target	min_target	max_target	median_target	pct
01_explorer_01_bouncer	1994729	49.921671	57.237589	-108.814722	129.040129	69.827231	20.63
01_explorer_02_skipping_listener	1921243	45.211354	60.724481	-108.814722	129.040129	68.916562	19.87
01_explorer_03_engaged_listener	1556095	43.203249	61.350277	-108.814722	129.040129	65.899891	16.09
01_explorer_04_completionist	1072715	42.388138	60.812811	-108.814722	129.040129	58.265705	11.09
00_unknown_00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.21
02_light_repeat_04_completionist	664744	27.894674	65.784990	-108.814722	129.040129	45.332675	6.87
02_light_repeat_03_engaged_listener	418236	40.190398	62.250216	-108.814722	129.040129	62.625845	4.33
03_repeat_04_completionist	316472	32.677561	64.030173	-108.814722	129.040129	48.590018	3.27
02_light_repeat_02_skipping_listener	184167	46.591011	59.372552	-108.814722	129.040129	68.722787	1.90
03_repeat_03_engaged_listener	181697	44.192270	60.200797	-108.814722	129.040129	65.430209	1.88
04_heavy_repeat_04_completionist	134903	37.597102	62.070532	-108.814722	129.040129	53.724118	1.40
00_unknown_01_bouncer	99778	53.224032	43.529240	-108.814722	129.040129	49.000000	1.03
03_repeat_02_skipping_listener	77229	49.685601	56.576548	-108.814722	129.040129	68.501098	0.80
04_heavy_repeat_03_engaged_listener	36443	50.706863	54.914337	-108.814722	129.040129	68.690165	0.38
02_light_repeat_01_bouncer	23812	51.970610	54.886731	-108.814722	129.040129	69.915428	0.25

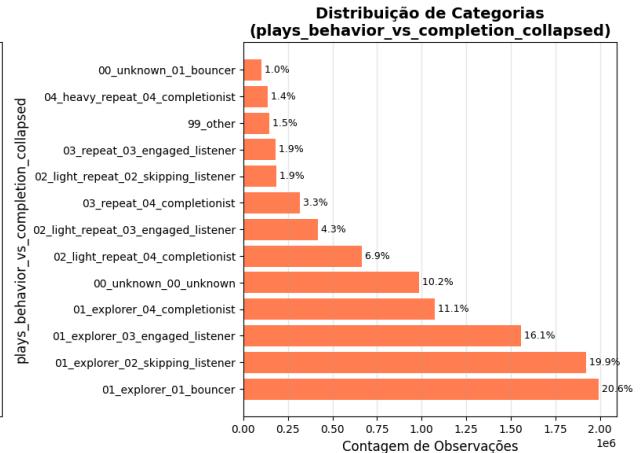
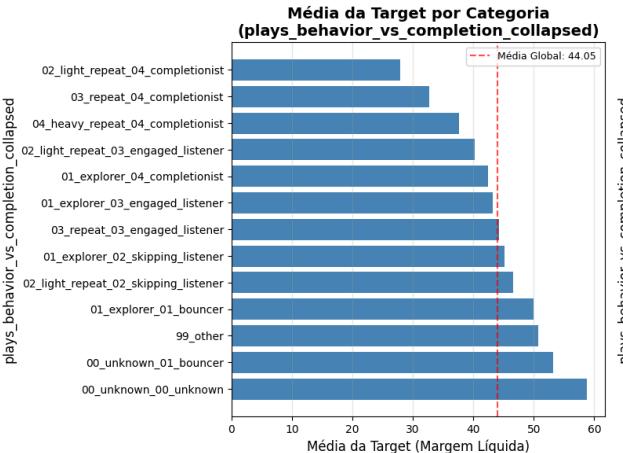


=====

EDA: plays_behavior_vs_completionCollapsed vs target_win

=====

plays_behavior_vs_completionCollapsed	count	mean_target	std_target	min_target	max_target	median_target	pct
01_explorer_01_bouncer	1994729	49.921671	57.237589	-108.814722	129.040129	69.827231	20.61
01_explorer_02_skipping_listener	1921243	45.211354	60.724481	-108.814722	129.040129	68.916562	19.85
01_explorer_03_engaged_listener	1556095	43.203249	61.350277	-108.814722	129.040129	65.899891	16.08
01_explorer_04_completionist	1072715	42.388138	60.812811	-108.814722	129.040129	58.265705	11.08
00_unknown_00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.20
02_light_repeat_04_completionist	664744	27.894674	65.784990	-108.814722	129.040129	45.332675	6.87
02_light_repeat_03_engaged_listener	418236	40.190398	62.250216	-108.814722	129.040129	62.625845	4.32
03_repeat_04_completionist	316472	32.677561	64.030173	-108.814722	129.040129	48.590018	3.27
02_light_repeat_02_skipping_listener	184167	46.591011	59.372552	-108.814722	129.040129	68.722787	1.90
03_repeat_03_engaged_listener	181697	44.192270	60.200797	-108.814722	129.040129	65.430209	1.88
99_other	146727	50.704219	55.602421	-108.814722	129.040129	69.060952	1.52
04_heavy_repeat_04_completionist	134903	37.597102	62.070532	-108.814722	129.040129	53.724118	1.39
00_unknown_01_bouncer	99778	53.224032	43.529240	-108.814722	129.040129	49.000000	1.03

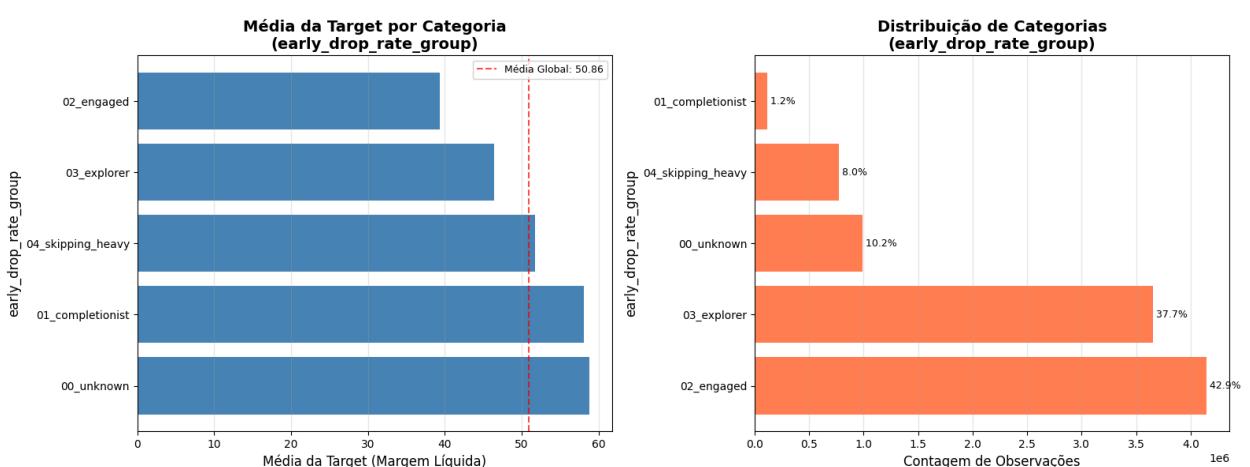


=====

EDA: early_drop_rate_group vs target_win

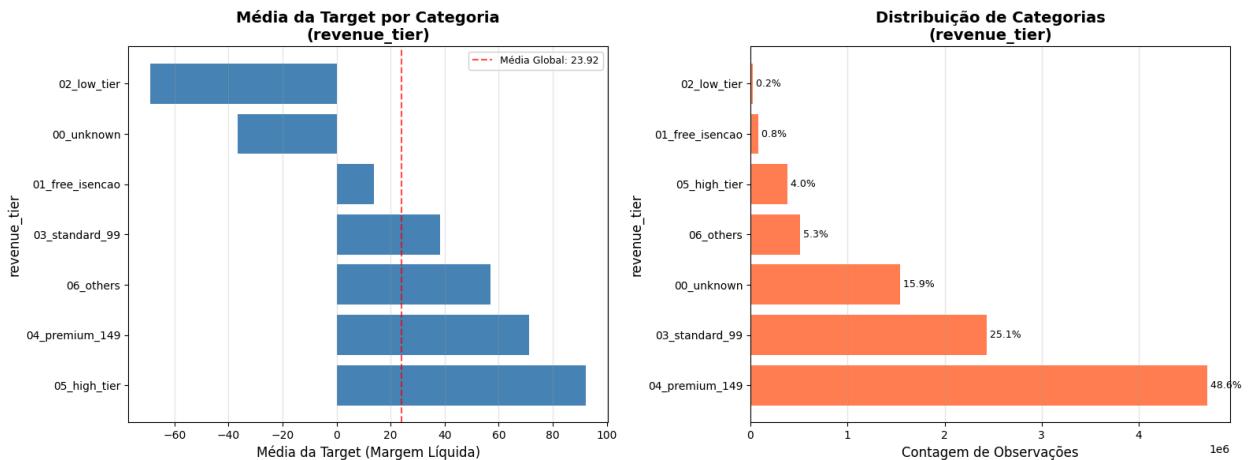
=====

early_drop_rate_group	count	mean_target	std_target	min_target	max_target	median_target	pct
02_engaged	4149304	39.303635	62.492671	-108.814722	129.040129	57.959814	42.87
03_explorer	3651998	46.389507	59.869760	-108.814722	129.040129	69.074944	37.73
00_unknown	987360	58.827143	42.982657	-108.814722	129.040129	49.000000	10.20
04_skipping_heavy	772568	51.689520	53.995474	-108.814722	129.040129	49.000000	7.98
01_completionist	117636	58.110992	50.298074	-108.814722	129.040129	52.752907	1.22



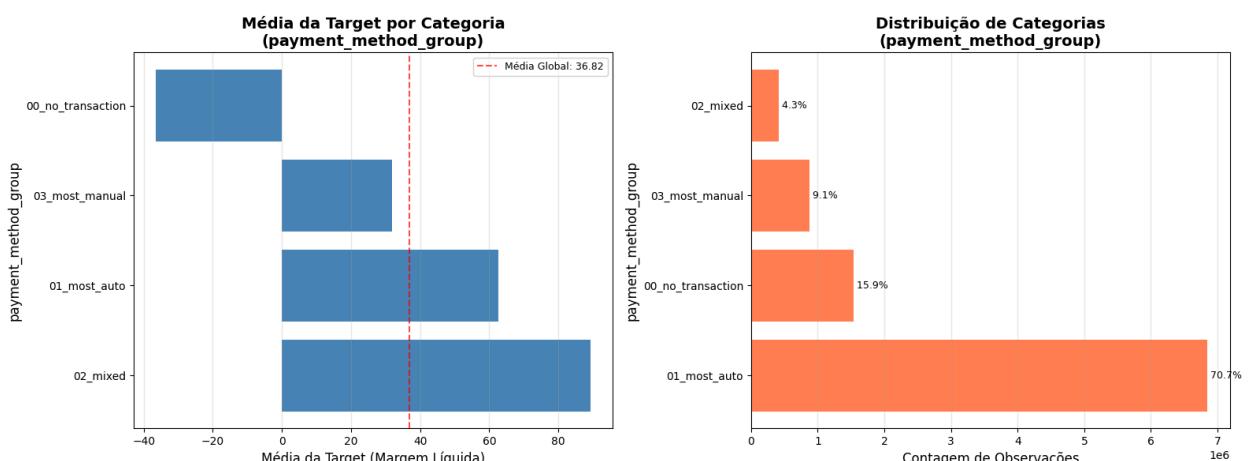
=====
EDA: revenue_tier vs target_win
=====

revenue_tier	count	mean_target	std_target	min_target	max_target	median_target	pct
04_premium_149	4706287	71.374168	44.057921	-108.814722	129.040129	87.454274	48.62
03_standard_99	2432782	38.438968	18.703246	-108.814722	129.040129	45.180604	25.13
00_unknown	1539830	-36.602484	66.381098	-108.814722	129.040129	-58.888950	15.91
06_others	513240	57.014188	37.284272	-108.814722	129.040129	69.417207	5.30
05_high_tier	383348	92.367094	55.842101	-108.814722	129.040129	114.574644	3.96
01_free_isencao	81869	13.875213	73.444836	-108.814722	129.040129	-50.000000	0.85
02_low_tier	21590	-69.037861	16.118064	-108.814722	119.149782	-64.322614	0.22



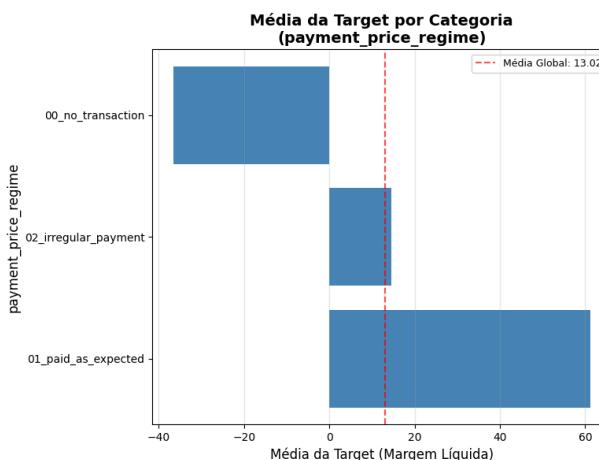
=====
EDA: payment_method_group vs target_win
=====

payment_method_group	count	mean_target	std_target	min_target	max_target	median_target	pct
01_most_auto	6844271	62.608463	34.178035	-108.814722	129.040129	69.387275	70.71
00_no_transaction	1539830	-36.602484	66.381098	-108.814722	129.040129	-58.888950	15.91
03_most_manual	878995	31.901197	72.472618	-108.814722	129.040129	70.843119	9.08
02_mixed	415770	89.380244	56.165329	-108.814722	129.040129	111.900490	4.30



=====
EDA: payment_price_regime vs target_win
=====

payment_price_regime	count	mean_target	std_target	min_target	max_target	median_target	pct
01_paid_as_expected	8055731	61.136713	42.272164	-108.814722	129.040129	72.162513	83.23
00_no_transaction	1539830	-36.602484	66.381098	-108.814722	129.040129	-58.888950	15.91
02_irregular_payment	83305	14.536757	73.099428	-108.814722	129.040129	-24.808220	0.86

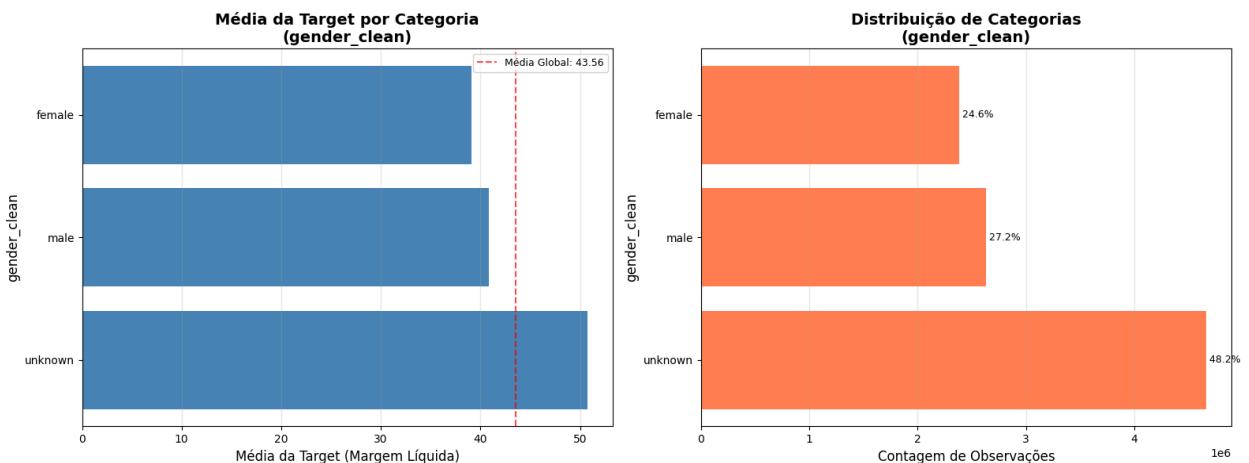


=====

EDA: gender_clean vs target_win

=====

gender_clean	count	mean_target	std_target	min_target	max_target	median_target	pct
unknown	4662342	50.753644	44.695175	-108.814722	129.040129	49.000000	48.17
male	2632298	40.845894	69.180724	-108.814722	129.040129	75.854825	27.20
female	2384226	39.090627	70.526598	-108.814722	129.040129	75.599487	24.63

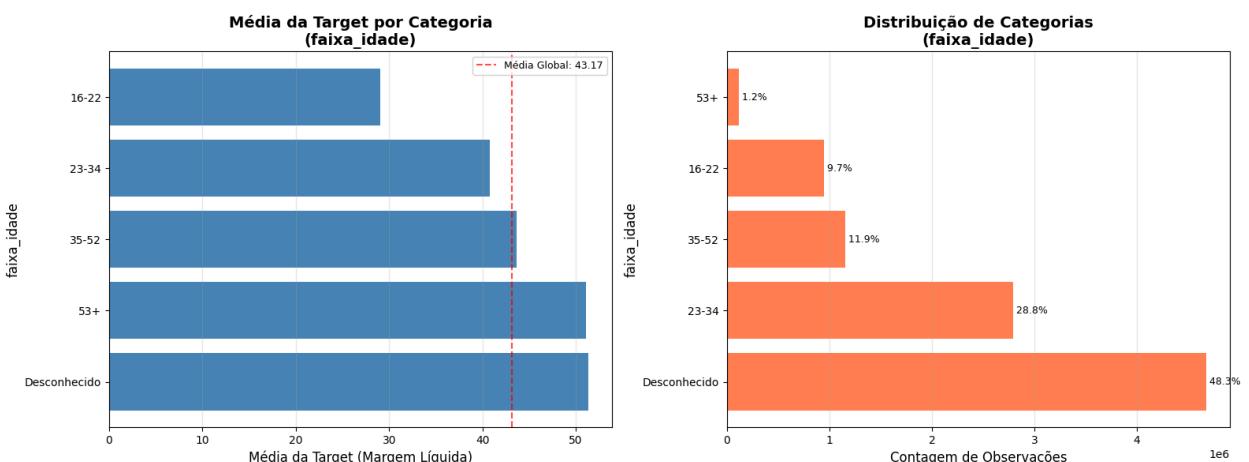


=====

EDA: faixa_idade vs target_win

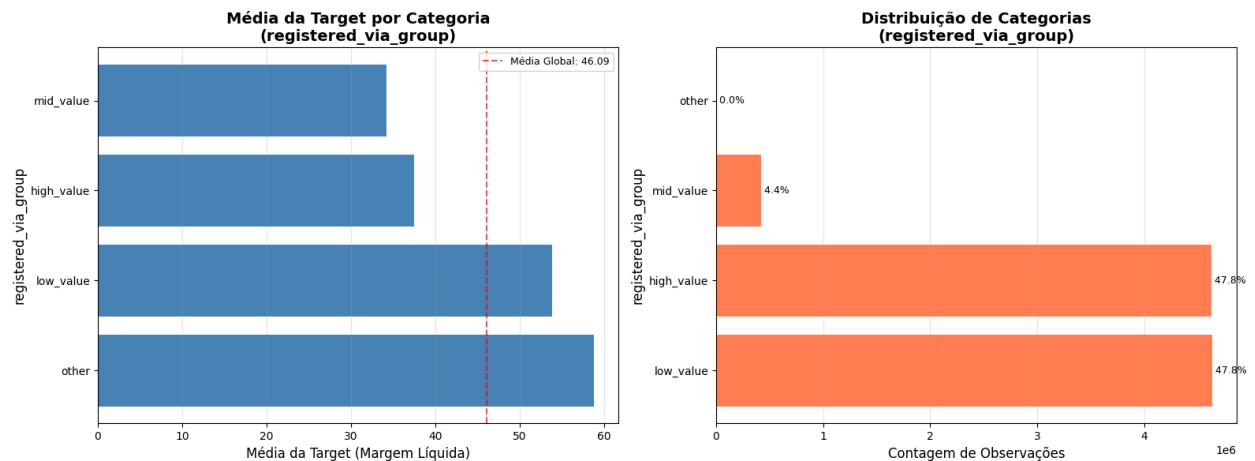
=====

faixa_idade	count	mean_target	std_target	min_target	max_target	median_target	pct
Desconhecido	4675988	51.312016	44.079997	-108.814722	129.040129	49.000000	48.31
23-34	2789382	40.771135	69.457916	-108.814722	129.040129	75.979898	28.82
35-52	1154914	43.648977	68.870378	-108.814722	129.040129	80.253194	11.93
16-22	943101	29.031338	73.333313	-108.814722	129.040129	66.449224	9.74
53+	115481	51.085116	64.373466	-108.814722	129.040129	84.069023	1.19



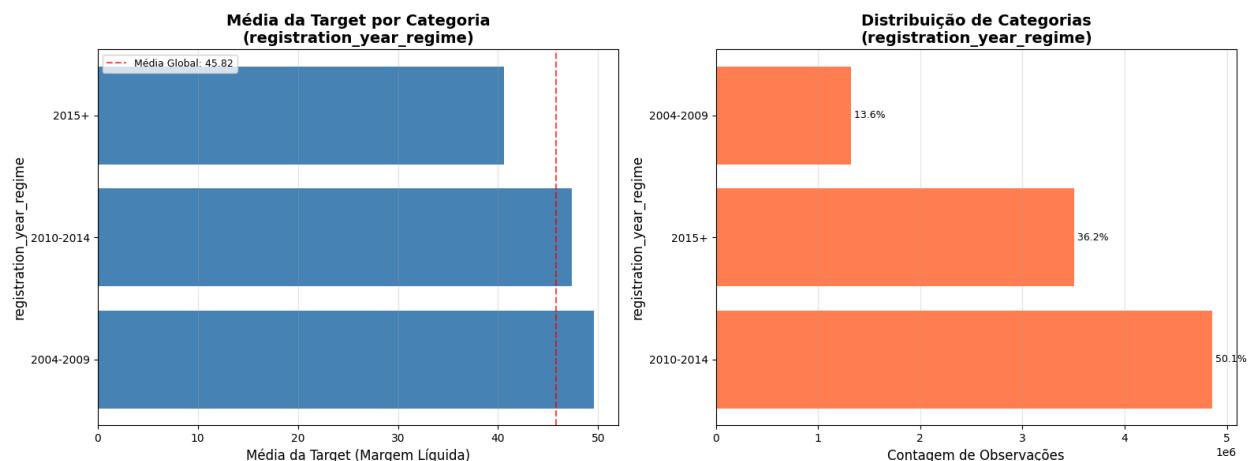
=====
 EDA: registered_via_group vs target_win
=====

registered_via_group	count	mean_target	std_target	min_target	max_target	median_target	pct
low_value	4629512	53.870705	35.091996	-108.814722	129.040129	49.000000	47.83
high_value	4623581	37.476211	73.716573	-108.814722	129.040129	78.421958	47.77
mid_value	421548	34.237129	77.716384	-108.814722	129.040129	76.036742	4.36
other	4225	58.777480	38.153430	-108.814722	129.040129	70.998755	0.04



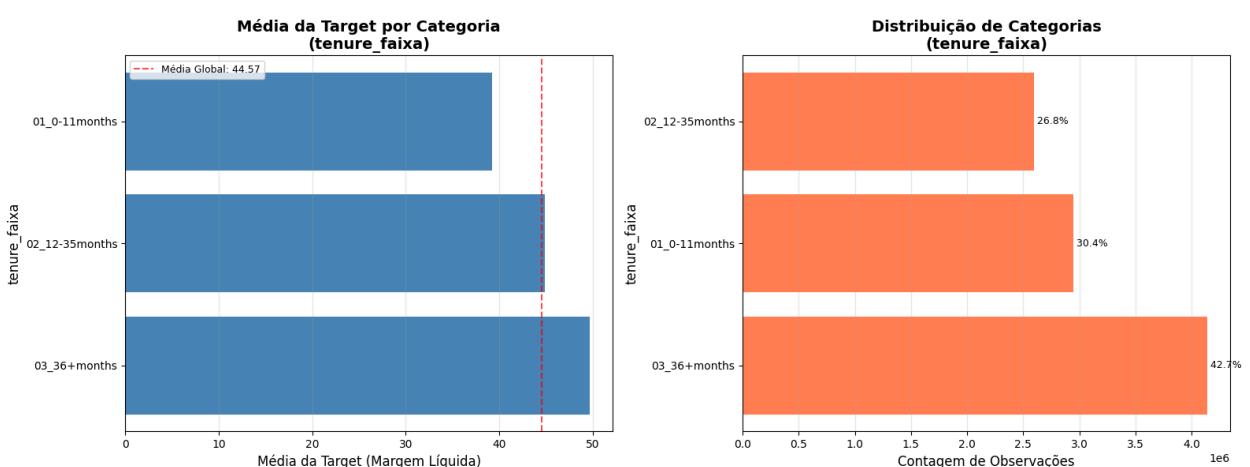
=====
 EDA: registration_year_regime vs target_win
=====

registration_year_regime	count	mean_target	std_target	min_target	max_target	median_target	pct
2010-2014	4852845	47.346335	61.395479	-108.814722	129.040129	73.451527	50.14
2015+	3507161	40.560109	52.501639	-108.814722	129.040129	48.452002	36.24
2004-2009	1318860	49.539029	67.288542	-108.814722	129.040129	83.911101	13.63



=====
 EDA: tenure_faixa vs target_win
=====

tenure_faixa	count	mean_target	std_target	min_target	max_target	median_target	pct
03_36+months	4136820	49.633583	62.650379	-108.814722	129.040129	78.164351	42.74
01_0-11months	2944630	39.252391	51.105446	-108.814722	129.040129	47.999871	30.42
02_12-35months	2597416	44.829683	61.861407	-108.814722	129.040129	68.228788	26.84

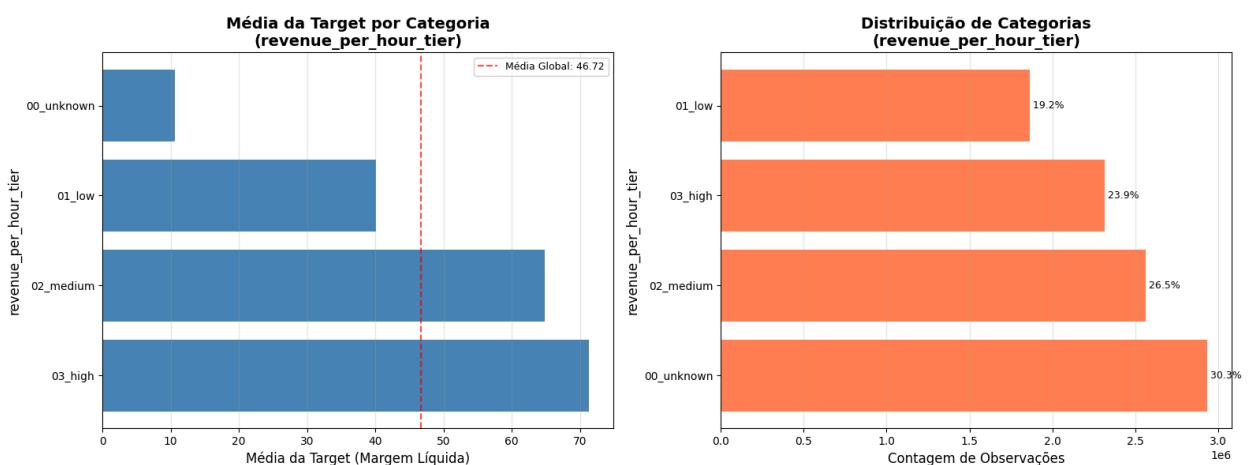


=====

EDA: revenue_per_hour_tier vs target_win

=====

revenue_per_hour_tier	count	mean_target	std_target	min_target	max_target	median_target	pct
00_unknown	2934889	10.664856	73.457526	-108.814722	129.040129	43.388999	30.32
02_medium	2564579	64.819217	42.458673	-108.814722	129.040129	80.578838	26.50
03_high	2315866	71.319905	37.745342	-108.814722	129.040129	89.416298	23.93
01_low	1863532	40.057688	47.346873	-108.814722	129.040129	49.889469	19.25

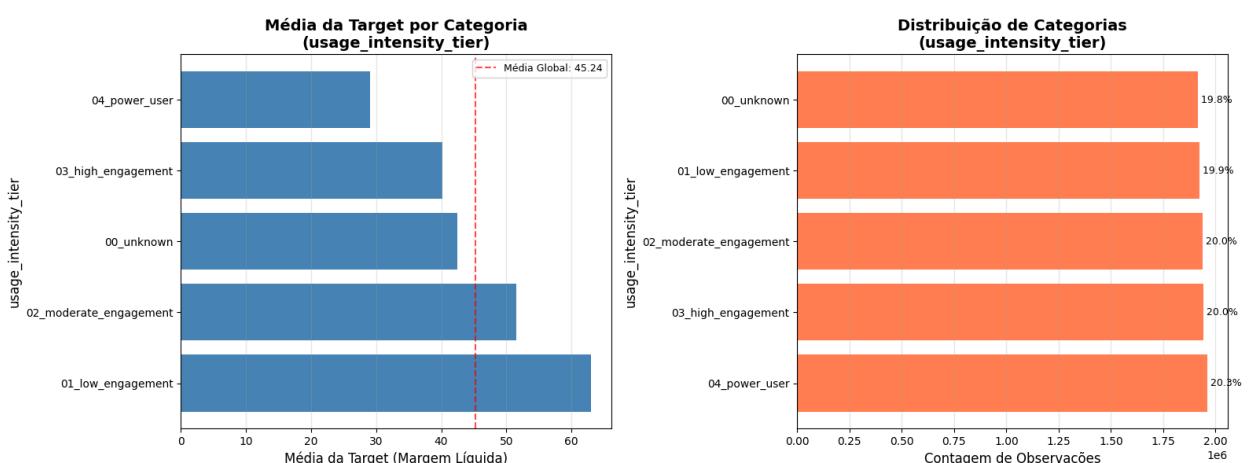


=====

EDA: usage_intensity_tier vs target_win

=====

usage_intensity_tier	count	mean_target	std_target	min_target	max_target	median_target	pct
04_power_user	1960669	29.096961	60.768515	-108.814722	129.040129	41.998502	20.26
03_high_engagement	1940098	40.176624	60.179864	-108.814722	129.040129	56.624453	20.04
02_moderate_engagement	1939229	51.497709	57.560936	-108.814722	129.040129	75.255844	20.04
01_low_engagement	1923930	62.986365	51.993725	-108.814722	129.040129	88.789395	19.88
00_unknown	1914940	42.459290	60.042287	-108.814722	129.040129	49.000000	19.78



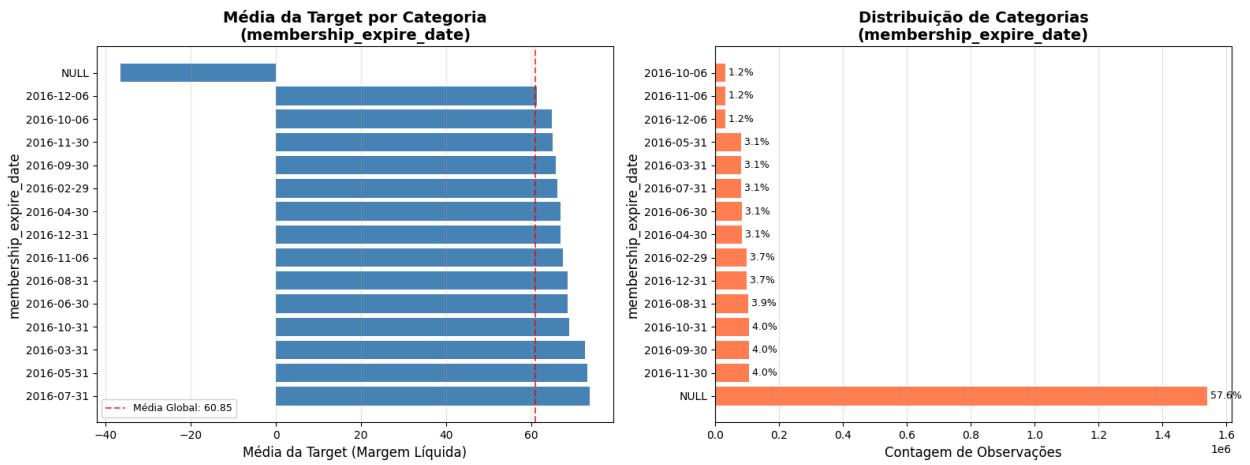
=====

EDA: membership_expire_date vs target_win

=====

membership_expire_date	count	mean_target	std_target	min_target	max_target	median_target	pct
2023-01-01	1	44.57	0.00	44.57	44.57	44.57	100.00

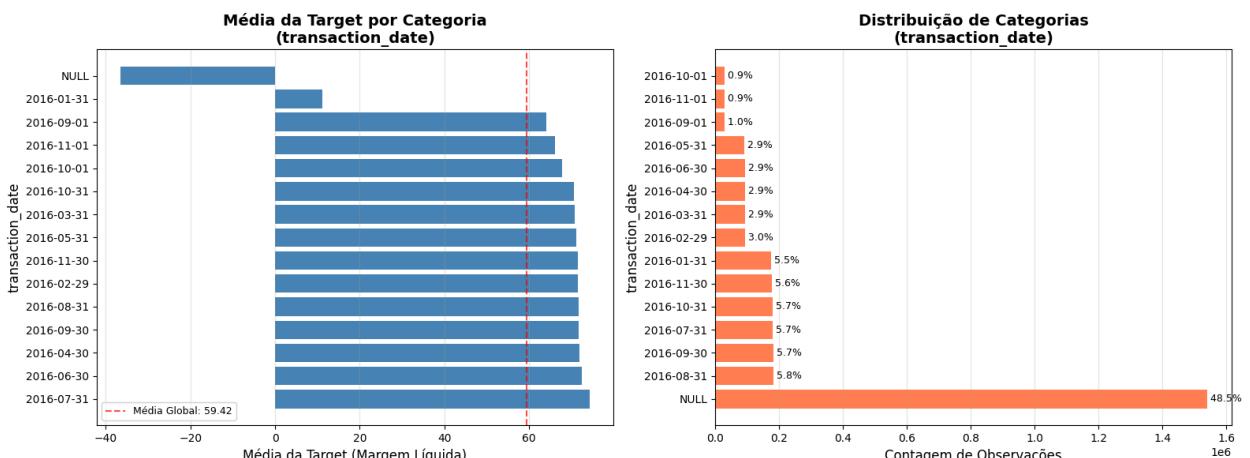
NULL	1539830	-36.602484	66.381098	-108.814722	129.040129	-58.888950	57.64
2016-11-30	107188	64.942359	37.157211	-108.814722	129.040129	76.908677	4.01
2016-09-30	107154	65.718371	37.253535	-108.814722	129.040129	78.477288	4.01
2016-10-31	105501	68.807107	31.047048	-108.814722	129.040129	78.832105	3.95
2016-08-31	105072	68.470746	33.551877	-108.814722	129.040129	80.145683	3.93
2016-12-31	98113	66.887366	30.838570	-108.814722	129.040129	75.681004	3.67
2016-02-29	97913	66.141788	45.568278	-108.814722	129.040129	85.280455	3.67
2016-04-30	83993	66.822809	43.418397	-108.814722	129.040129	84.117943	3.14
2016-06-30	83681	68.499972	40.973501	-108.814722	129.040129	84.692631	3.13
2016-07-31	81963	73.756131	30.225028	-108.814722	129.040129	85.411496	3.07
2016-03-31	81958	72.647788	34.927959	-108.814722	129.040129	86.035989	3.07
2016-05-31	81813	73.074201	33.093669	-108.814722	129.040129	85.479689	3.06
2016-12-06	32742	61.277448	42.325801	-108.814722	129.040129	72.125592	1.23
2016-11-06	32342	67.405756	35.625701	-108.814722	129.040129	76.502285	1.21
2016-10-06	32127	64.879897	38.232057	-108.814722	129.040129	75.291375	1.20



⚠️ ALERTA: 1,539,830 valores nulos (57.64%) encontrados em 'membership_expire_date'
└ Considere investigar a origem desses nulos antes de prosseguir.

=====
📊 EDA: transaction_date vs target_win
=====

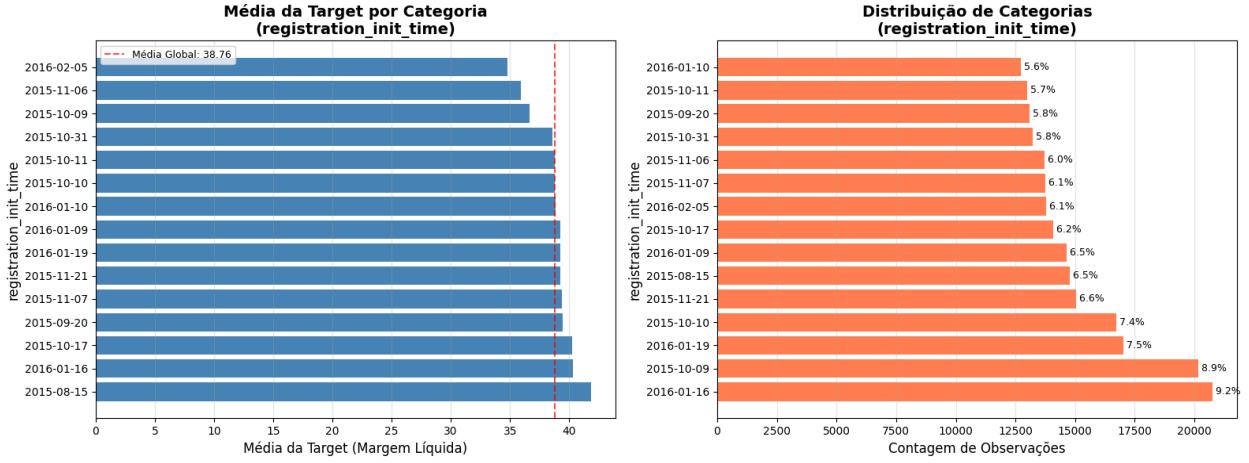
transaction_date	count	mean_target	std_target	min_target	max_target	median_target	pct
NULL	1539830	-36.602484	66.381098	-108.814722	129.040129	-58.888950	48.50
2016-08-31	183012	71.751511	33.021207	-108.814722	129.040129	84.141092	5.76
2016-09-30	181982	71.836296	31.213043	-108.814722	129.040129	83.191565	5.73
2016-07-31	181015	74.414971	27.859558	-108.814722	129.040129	84.796677	5.70
2016-10-31	180163	70.564038	33.938360	-108.814722	129.040129	83.232852	5.67
2016-11-30	178836	71.495022	30.927179	-108.814722	129.040129	82.703069	5.63
2016-01-31	176032	11.265933	75.032972	-108.814722	129.040129	37.861367	5.54
2016-02-29	93670	71.626625	36.933131	-108.814722	129.040129	85.422404	2.95
2016-03-31	92965	70.832154	37.830457	-108.814722	129.040129	84.877797	2.93
2016-04-30	92903	71.876399	35.416565	-108.814722	129.040129	84.911130	2.93
2016-06-30	92855	72.601552	33.790259	-108.814722	129.040129	85.187084	2.92
2016-05-31	92219	71.287776	36.638679	-108.814722	129.040129	85.173037	2.90
2016-09-01	30429	64.203907	38.984887	-108.814722	129.040129	75.821521	0.96
2016-11-01	29827	66.168486	34.731340	-108.814722	129.040129	75.583376	0.94
2016-10-01	29413	67.932160	33.294302	-108.814722	129.040129	77.610771	0.93



⚠️ ALERTA: 1,539,830 valores nulos (48.50%) encontrados em 'transaction_date'
└ Considere investigar a origem desses nulos antes de prosseguir.

=====
📊 EDA: registration_init_time vs target_win
=====

registration_init_time	count	mean_target	std_target	min_target	max_target	median_target	pct
2016-01-16	20749	40.327998	36.263527	-108.814722	129.040129	46.742943	9.16
2015-10-09	20169	36.667468	39.588970	-108.814722	129.040129	44.285601	8.91
2016-01-19	17019	39.254080	36.780741	-108.814722	129.040129	46.871398	7.52
2015-10-10	16712	38.757583	40.708478	-108.814722	129.040129	45.607589	7.38
2015-11-21	15034	39.262578	44.951344	-108.814722	129.040129	47.008254	6.64
2015-08-15	14762	41.826920	45.669304	-108.814722	129.040129	48.472863	6.52
2016-01-09	14655	39.238509	46.664940	-108.814722	129.040129	47.399936	6.47
2015-10-17	14077	40.264177	42.027741	-108.814722	129.040129	47.409944	6.22
2016-02-05	13792	34.818958	46.873359	-108.814722	129.040129	46.489755	6.09
2015-11-07	13737	39.405858	45.700435	-108.814722	129.040129	47.088764	6.07
2015-11-06	13709	35.897897	46.845713	-108.814722	129.040129	46.375559	6.05
2015-10-31	13212	38.596734	45.084563	-108.814722	129.040129	46.781219	5.84
2015-09-20	13075	39.450980	43.802890	-108.814722	129.040129	47.969151	5.77
2015-10-11	12975	38.755420	44.411636	-108.814722	129.040129	46.219556	5.73
2016-01-10	12734	38.880468	48.887101	-108.814722	129.040129	47.666806	5.62



Por que as variaveis que vem do book de logs estao com classe nula ao inves de unknown, como foi definido anteriormente?

Cramer's V

```
In [135]: cramer_matrix = matriz_cramer_v_spark_optimized(  
    df=df_elastic_net,  
    colunas_categoricas=features_categoricas_base,  
    figsize=(22, 18) # Ajuste conforme necessário  
)
```

⠼ Iniciando indexação de 19 colunas categóricas...
⠼ Indexação concluída!

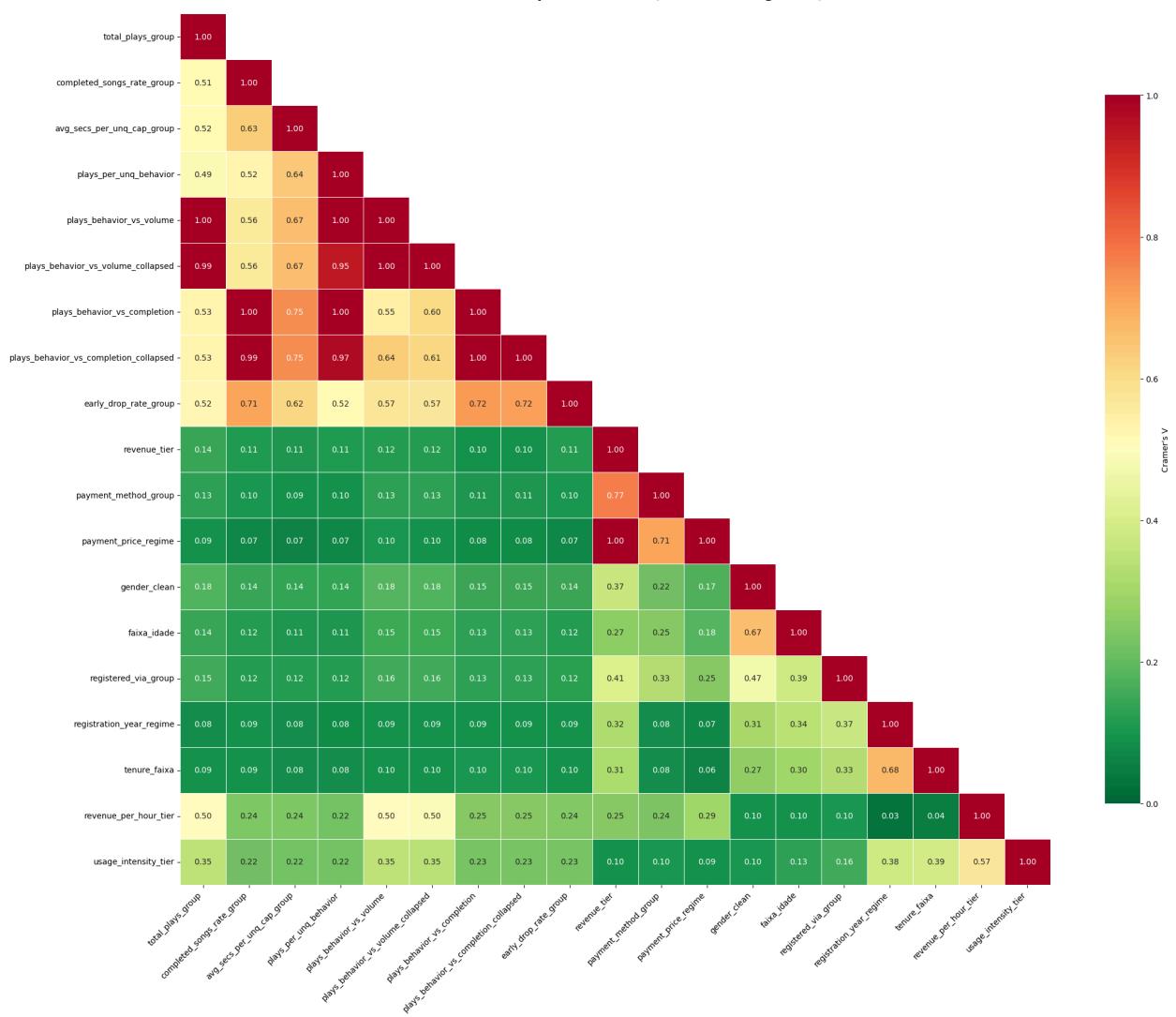
Calculando: total_plays_group + completed_songs_rate_group
Calculando: total_plays_group + avg_secs_per_unq_cap_group
Calculando: total_plays_group + plays_per_unq_behavior
Calculando: total_plays_group + plays_behavior_vs_volume
Calculando: total_plays_group + plays_behavior_vs_volume_collapsed
Calculando: total_plays_group + plays_behavior_vs_completion
Calculando: total_plays_group + plays_behavior_vs_completion_collapsed
Calculando: total_plays_group + early_drop_rate_group
Calculando: total_plays_group + revenue_tier
Calculando: total_plays_group + payment_method_group
Calculando: total_plays_group + payment_price_regime
Calculando: total_plays_group + gender_clean
Calculando: total_plays_group + faixa_idade
Calculando: total_plays_group + registered_via_group
Calculando: total_plays_group + registration_year_regime
Calculando: total_plays_group + tenure_faixa
Calculando: total_plays_group + revenue_per_hour_tier
Calculando: total_plays_group + usage_intensity_tier
Calculando: completed_songs_rate_group + avg_secs_per_unq_cap_group
Calculando: completed_songs_rate_group + plays_per_unq_behavior
Calculando: completed_songs_rate_group + plays_behavior_vs_volume
Calculando: completed_songs_rate_group + plays_behavior_vs_volume_collapsed
Calculando: completed_songs_rate_group + plays_behavior_vs_completion
Calculando: completed_songs_rate_group + plays_behavior_vs_completion_collapsed
Calculando: completed_songs_rate_group + early_drop_rate_group
Calculando: completed_songs_rate_group + revenue_tier
Calculando: completed_songs_rate_group + payment_method_group
Calculando: completed_songs_rate_group + payment_price_regime
Calculando: completed_songs_rate_group + gender_clean
Calculando: completed_songs_rate_group + faixa_idade
Calculando: completed_songs_rate_group + registered_via_group
Calculando: completed_songs_rate_group + registration_year_regime
Calculando: completed_songs_rate_group + tenure_faixa
Calculando: completed_songs_rate_group + revenue_per_hour_tier
Calculando: completed_songs_rate_group + usage_intensity_tier
Calculando: avg_secs_per_unq_cap_group + plays_per_unq_behavior
Calculando: avg_secs_per_unq_cap_group + plays_behavior_vs_volume
Calculando: avg_secs_per_unq_cap_group + plays_behavior_vs_volume_collapsed
Calculando: avg_secs_per_unq_cap_group + plays_behavior_vs_completion
Calculando: avg_secs_per_unq_cap_group + plays_behavior_vs_completion_collapsed
Calculando: avg_secs_per_unq_cap_group + early_drop_rate_group
Calculando: avg_secs_per_unq_cap_group + revenue_tier
Calculando: avg_secs_per_unq_cap_group + payment_method_group
Calculando: avg_secs_per_unq_cap_group + payment_price_regime
Calculando: avg_secs_per_unq_cap_group + gender_clean
Calculando: avg_secs_per_unq_cap_group + faixa_idade
Calculando: avg_secs_per_unq_cap_group + registered_via_group
Calculando: avg_secs_per_unq_cap_group + registration_year_regime
Calculando: avg_secs_per_unq_cap_group + tenure_faixa
Calculando: avg_secs_per_unq_cap_group + revenue_per_hour_tier
Calculando: avg_secs_per_unq_cap_group + usage_intensity_tier
Calculando: plays_per_unq_behavior + plays_behavior_vs_volume
Calculando: plays_per_unq_behavior + plays_behavior_vs_volume_collapsed
Calculando: plays_per_unq_behavior + plays_behavior_vs_completion
Calculando: plays_per_unq_behavior + plays_behavior_vs_completion_collapsed
Calculando: plays_per_unq_behavior + early_drop_rate_group
Calculando: plays_per_unq_behavior + revenue_tier
Calculando: plays_per_unq_behavior + payment_method_group
Calculando: plays_per_unq_behavior + payment_price_regime
Calculando: plays_per_unq_behavior + gender_clean
Calculando: plays_per_unq_behavior + faixa_idade
Calculando: plays_per_unq_behavior + registered_via_group
Calculando: plays_per_unq_behavior + registration_year_regime
Calculando: plays_per_unq_behavior + tenure_faixa
Calculando: plays_per_unq_behavior + revenue_per_hour_tier
Calculando: plays_per_unq_behavior + usage_intensity_tier
Calculando: plays_behavior_vs_volume + plays_behavior_vs_volume_collapsed
Calculando: plays_behavior_vs_volume + plays_behavior_vs_completion
Calculando: plays_behavior_vs_volume + plays_behavior_vs_completion_collapsed
Calculando: plays_behavior_vs_volume + early_drop_rate_group
Calculando: plays_behavior_vs_volume + revenue_tier
Calculando: plays_behavior_vs_volume + payment_method_group
Calculando: plays_behavior_vs_volume + payment_price_regime
Calculando: plays_behavior_vs_volume + gender_clean
Calculando: plays_behavior_vs_volume + faixa_idade
Calculando: plays_behavior_vs_volume + registered_via_group
Calculando: plays_behavior_vs_volume + registration_year_regime
Calculando: plays_behavior_vs_volume + tenure_faixa

Calculando: plays_behavior_vs_volume + revenue_per_hour_tier
Calculando: plays_behavior_vs_volume + usage_intensity_tier
Calculando: plays_behavior_vs_volume_collapsed + plays_behavior_vs_completion
Calculando: plays_behavior_vs_volume_collapsed + plays_behavior_vs_completion_collapsed
Calculando: plays_behavior_vs_volume_collapsed + early_drop_rate_group
Calculando: plays_behavior_vs_volume_collapsed + revenue_tier
Calculando: plays_behavior_vs_volume_collapsed + payment_method_group
Calculando: plays_behavior_vs_volume_collapsed + payment_price_regime
Calculando: plays_behavior_vs_volume_collapsed + gender_clean
Calculando: plays_behavior_vs_volume_collapsed + faixa_idade
Calculando: plays_behavior_vs_volume_collapsed + registered_via_group
Calculando: plays_behavior_vs_volume_collapsed + registration_year_regime
Calculando: plays_behavior_vs_volume_collapsed + tenure_faixa
Calculando: plays_behavior_vs_volume_collapsed + revenue_per_hour_tier
Calculando: plays_behavior_vs_volume_collapsed + usage_intensity_tier
Calculando: plays_behavior_vs_completion + plays_behavior_vs_completion_collapsed
Calculando: plays_behavior_vs_completion + early_drop_rate_group
Calculando: plays_behavior_vs_completion + revenue_tier
Calculando: plays_behavior_vs_completion + payment_method_group
Calculando: plays_behavior_vs_completion + payment_price_regime
Calculando: plays_behavior_vs_completion + gender_clean
Calculando: plays_behavior_vs_completion + faixa_idade
Calculando: plays_behavior_vs_completion + registered_via_group
Calculando: plays_behavior_vs_completion + registration_year_regime
Calculando: plays_behavior_vs_completion + tenure_faixa
Calculando: plays_behavior_vs_completion + revenue_per_hour_tier
Calculando: plays_behavior_vs_completion + usage_intensity_tier
Calculando: plays_behavior_vs_completion_collapsed + early_drop_rate_group
Calculando: plays_behavior_vs_completion_collapsed + revenue_tier
Calculando: plays_behavior_vs_completion_collapsed + payment_method_group
Calculando: plays_behavior_vs_completion_collapsed + payment_price_regime
Calculando: plays_behavior_vs_completion_collapsed + gender_clean
Calculando: plays_behavior_vs_completion_collapsed + faixa_idade
Calculando: plays_behavior_vs_completion_collapsed + registered_via_group
Calculando: plays_behavior_vs_completion_collapsed + registration_year_regime
Calculando: plays_behavior_vs_completion_collapsed + tenure_faixa
Calculando: plays_behavior_vs_completion_collapsed + revenue_per_hour_tier
Calculando: plays_behavior_vs_completion_collapsed + usage_intensity_tier
Calculando: early_drop_rate_group + revenue_tier
Calculando: early_drop_rate_group + payment_method_group
Calculando: early_drop_rate_group + payment_price_regime
Calculando: early_drop_rate_group + gender_clean
Calculando: early_drop_rate_group + faixa_idade
Calculando: early_drop_rate_group + registered_via_group
Calculando: early_drop_rate_group + registration_year_regime
Calculando: early_drop_rate_group + tenure_faixa
Calculando: early_drop_rate_group + revenue_per_hour_tier
Calculando: early_drop_rate_group + usage_intensity_tier
Calculando: revenue_tier + payment_method_group
Calculando: revenue_tier + payment_price_regime
Calculando: revenue_tier + gender_clean
Calculando: revenue_tier + faixa_idade
Calculando: revenue_tier + registered_via_group
Calculando: revenue_tier + registration_year_regime
Calculando: revenue_tier + tenure_faixa
Calculando: early_drop_rate_group + revenue_per_hour_tier
Calculando: early_drop_rate_group + usage_intensity_tier
Calculando: revenue_tier + payment_method_group
Calculando: revenue_tier + payment_price_regime
Calculando: revenue_tier + gender_clean
Calculando: revenue_tier + faixa_idade
Calculando: revenue_tier + registered_via_group
Calculando: revenue_tier + registration_year_regime
Calculando: revenue_tier + tenure_faixa
Calculando: revenue_tier + revenue_per_hour_tier
Calculando: revenue_tier + usage_intensity_tier
Calculando: payment_method_group + payment_price_regime
Calculando: payment_method_group + gender_clean
Calculando: payment_method_group + faixa_idade
Calculando: payment_method_group + registered_via_group
Calculando: payment_method_group + registration_year_regime
Calculando: payment_method_group + tenure_faixa
Calculando: payment_method_group + revenue_per_hour_tier
Calculando: payment_method_group + usage_intensity_tier
Calculando: payment_price_regime + gender_clean
Calculando: payment_price_regime + faixa_idade
Calculando: payment_price_regime + registered_via_group
Calculando: payment_price_regime + registration_year_regime
Calculando: payment_price_regime + tenure_faixa
Calculando: payment_price_regime + revenue_per_hour_tier
Calculando: payment_price_regime + usage_intensity_tier
Calculando: gender_clean + faixa_idade
Calculando: gender_clean + registered_via_group
Calculando: gender_clean + registration_year_regime
Calculando: gender_clean + tenure_faixa
Calculando: gender_clean + revenue_per_hour_tier
Calculando: gender_clean + usage_intensity_tier
Calculando: faixa_idade + registered_via_group
Calculando: faixa_idade + registration_year_regime
Calculando: faixa_idade + tenure_faixa
Calculando: faixa_idade + revenue_per_hour_tier
Calculando: faixa_idade + usage_intensity_tier
Calculando: registered_via_group + registration_year_regime
Calculando: registered_via_group + tenure_faixa
Calculando: registered_via_group + revenue_per_hour_tier
Calculando: registered_via_group + usage_intensity_tier
Calculando: registration_year_regime + tenure_faixa

Calculando: registration_year_regime + revenue_per_hour_tier
 Calculando: registration_year_regime + usage_intensity_tier
 Calculando: tenure_faixa + revenue_per_hour_tier
 Calculando: tenure_faixa + usage_intensity_tier
 Calculando: revenue_per_hour_tier + usage_intensity_tier

Cálculo concluído!

Matriz de Associação: Cramer's V (Variáveis Categóricas)



In [136]: cramer_matrix

Out[136]:

	total_plays_group	completed_songs_rate_group	avg_secs_per_unq_cap_group	plays_per_unq_beh
total_plays_group	1.000000	0.512608	0.516722	0.487073
completed_songs_rate_group	0.512608	1.000000	0.634778	0.523585
avg_secs_per_unq_cap_group	0.516722	0.634778	1.000000	0.637397
plays_per_unq_behavior	0.487073	0.523585	0.637397	1.000000
plays_behavior_vs_volume	1.000000	0.561181	0.667005	1.000000
plays_behavior_vs_volume_collapsed	0.989771	0.559830	0.666957	0.945105
plays_behavior_vs_completion	0.527137	1.000000	0.753215	1.000000
plays_behavior_vs_completion_collapsed	0.527072	0.994980	0.753212	0.968851
early_drop_rate_group	0.521106	0.712223	0.615928	0.515879
revenue_tier	0.142499	0.110186	0.108482	0.112135
payment_method_group	0.128319	0.100340	0.092827	0.095608
payment_price_regime	0.094952	0.070252	0.065678	0.066923
gender_clean	0.178483	0.141155	0.139257	0.139902
faixa_idade	0.143851	0.120720	0.109250	0.108749
registered_via_group	0.154977	0.119019	0.117410	0.120159
registration_year_regime	0.082119	0.090318	0.081350	0.077451
tenure_faixa	0.088218	0.093469	0.084149	0.080786
revenue_per_hour_tier	0.500121	0.238545	0.242168	0.224686
usage_intensity_tier	0.350217	0.220438	0.222658	0.223307

Pares de correlação mais forte:

- * plays_behavior_vs_volume X total_plays_group
- * plays_behavior_vs_completion X completed_songs_rate_group
- * plays_behavior_vs_completion X avg_secs_per_unq_cap_group
- * plays_behavior_vs_volume X plays_per_unq_behavior
- * plays_behavior_vs_completion X plays_per_unq_behavior
- * payment_price_regime X revenue_tier

ANOVA F-Statistic

Definição

O que é ANOVA (Analysis of Variance)?

A **ANOVA** é um teste estatístico que verifica se as médias de categorias são significativamente diferentes entre si.

Pergunta: "A margem média varia significativamente entre os tipos de usuário?*" Se a resposta for SIM, a feature tem poder preditivo.

Decomposição da Variância

A ANOVA decompõe a variância total da target em duas partes:

$$\begin{aligned} \text{\textbackslash text{SS}}_{\text{\textbackslash text{total}}} &= \text{\textbackslash text{SS}}_{\text{\textbackslash text{between}}} + \text{\textbackslash text{SS}}_{\text{\textbackslash text{within}}} \\ \end{aligned}$$

Onde:

SS_total (Sum of Squares Total):

Variância total dos dados em relação à média global.

$$\begin{aligned} \text{\textbackslash text{SS}}_{\text{\textbackslash text{total}}} &= \text{\textbackslash sum}_{\{i=1}^{\{n\}} (y_i - \bar{y})^2 \\ \end{aligned}$$

- y_i : Valor da target para a observação i
- \bar{y} : Média global da target
- n : Número total de observações

SS_between (Sum of Squares Between Groups):

Variância explicada pelas diferenças entre as médias dos grupos.

$$\begin{aligned} \text{\textbackslash text{SS}}_{\text{\textbackslash text{between}}} &= \text{\textbackslash sum}_{\{j=1}^{\{k\}} n_j (\bar{y}_j - \bar{y})^2 \\ \end{aligned}$$

- k : Número de categorias (grupos)
- n_j : Número de observações no grupo j
- \bar{y}_j : Média da target no grupo j
- \bar{y} : Média global

Interpretação: Se as médias dos grupos são muito diferentes da média global, $\text{SS}_{\text{between}}$ é alto → **Feature forte**.

SS_within (Sum of Squares Within Groups):

Variância não explicada (variação dentro de cada grupo).

$$\begin{aligned} \text{\textbackslash text{SS}}_{\text{\textbackslash text{within}}} &= \text{\textbackslash sum}_{\{j=1}^{\{k\}} \text{\textbackslash sum}_{\{i=1}^{\{n_j\}} (y_{ij} - \bar{y}_j)^2 \\ \end{aligned}$$

- y_{ij} : Valor da target para a observação i no grupo j

Interpretação: Quanto menor, mais homogêneos são os grupos (boa separação).

F-Statistic: A Métrica de Decisão

O F-statistic é a razão entre a variância explicada e a não explicada:

$$\begin{aligned} \text{\textbackslash text{F}} &= \frac{\text{\textbackslash text{MS}}_{\text{\textbackslash text{between}}}}{\text{\textbackslash text{MS}}_{\text{\textbackslash text{within}}}} \\ \end{aligned}$$

Onde:

$$\begin{aligned} \text{\textbackslash text{MS}}_{\text{\textbackslash text{between}}} &= \frac{\text{\textbackslash text{SS}}_{\text{\textbackslash text{between}}}}{k - 1} \quad \text{(Mean Square Between)} \\ \end{aligned}$$

$$\begin{aligned} \text{\textbackslash text{MS}}_{\text{\textbackslash text{within}}} &= \frac{\text{\textbackslash text{SS}}_{\text{\textbackslash text{within}}}}{n - k} \quad \text{(Mean Square Within)} \\ \end{aligned}$$

- $k - 1$: Graus de liberdade entre grupos
- $n - k$: Graus de liberdade dentro dos grupos

Interpretação do F-Statistic:

Valor de F Interpretação
----- -----
$F \approx 1$ As médias dos grupos são similares → Feature fraca
$F >> 1$ As médias dos grupos são muito diferentes → Feature forte
$F < 1$ Variação dentro dos grupos é maior que entre grupos → Feature inútil

Regra prática:

- $F > 10$: Feature com poder preditivo moderado
- $F > 50$: Feature com poder preditivo forte
- $F > 100$: Feature com poder preditivo muito forte

P-Valor: Significância Estatística

O p-valor indica a probabilidade de observar um F-statistic tão extremo **por acaso** (se não houvesse diferença real entre os grupos).

$$\text{p-value} = P(F_{\text{obs}} | H_0)$$

Onde H_0 (hipótese nula) = "Todas as médias dos grupos são iguais".

Interpretação:

- $p < 0.05$: Rejeitamos H_0 → **Diferença significativa** (feature relevante)
- $p \geq 0.05$: Aceitamos H_0 → **Diferença não significativa** (feature fraca)

Eta-Squared (η^2): Tamanho do Efeito

O Eta-squared mede a proporção da variância total explicada pela variável categórica.

$$\eta^2 = \frac{\text{SS}_{\text{between}}}{\text{SS}_{\text{total}}}$$

Interpretação (Regra de Cohen):

η^2 Interpretação Exemplo		
----- ----- -----		
< 0.01 Efeito trivial Feature quase inútil		
0.01 - 0.06 Efeito pequeno Feature fraca, mas pode ajudar em ensemble		
0.06 - 0.14 Efeito médio Feature moderada, útil para o modelo		
> 0.14 Efeito grande Feature forte, alta capacidade preditiva		

Por que η^2 é melhor que F-statistic sozinho?

- O F-statistic é sensível ao tamanho da amostra (amostras grandes sempre têm F alto).
- O η^2 é **normalizado** (varia de 0 a 1), permitindo comparar features independentemente do tamanho da base.

Pressupostos da ANOVA

Para que o teste seja válido, três pressupostos devem ser atendidos:

1. Independência: As observações são independentes entre si.
2. Normalidade: A target segue distribuição normal dentro de cada grupo.
3. Homocedasticidade: A variância da target é similar entre os grupos.

Na prática (Big Data):

- Com amostras grandes ($n > 30$ por grupo), a ANOVA é robusta a violações de normalidade (Teorema do Limite Central).
- Se a homocedasticidade for violada, use Welch's ANOVA (variante mais robusta).

Execução e Resultados

```
In [137]: df_elastic_net = df_elastic_net.select(
    'msno',
    'safra',
    'target_win',
    *features_numericas_finais_elastic_net,
    *features_categoricas_base
)
```

In df_elastic_net.printSchema()

[138]:

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- target_win: double (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- actual_amount_paid: double (nullable = true)
 |-- flag_plano_mensal_max_3: integer (nullable = true)
 |-- log_total_secs_mean_3: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- total_plays_group: string (nullable = true)
 |-- completed_songs_rate_group: string (nullable = true)
 |-- avg_secs_per_unq_cap_group: string (nullable = true)
 |-- plays_per_unq_behavior: string (nullable = true)
 |-- plays_behavior_vs_volume: string (nullable = true)
 |-- plays_behavior_vs_volume_collapsed: string (nullable = true)
 |-- plays_behavior_vs_completion: string (nullable = true)
 |-- plays_behavior_vs_completion_collapsed: string (nullable = true)
 |-- early_drop_rate_group: string (nullable = true)
 |-- revenue_tier: string (nullable = true)
 |-- payment_method_group: string (nullable = true)
 |-- payment_price_regime: string (nullable = true)
 |-- gender_clean: string (nullable = true)
 |-- faixa_idade: string (nullable = true)
 |-- registered_via_group: string (nullable = true)
 |-- registration_year_regime: string (nullable = true)
 |-- tenure_faixa: string (nullable = true)
 |-- revenue_per_hour_tier: string (nullable = true)
 |-- usage_intensity_tier: string (nullable = true)
```

In df_elastic_net.count()

[139]:

Out[139]: 9678866

```

In [140]: def calcular_anova_f_spark(df, col_categorica, col_target):
    """
    Calcula o F-statistic da ANOVA para uma variável categórica vs target contínua.
    VERSÃO OTIMIZADA: Calcula tudo no Spark (sem toPandas).

    Fórmulas:
    -----
    F = MS_between / MS_within
    η² = SS_between / SS_total
    p-value = calculado via scipy (apenas estatísticas agregadas)
    """

    print(f"\n{'-'*60}")
    print(f"Calculando ANOVA: {col_categorica} vs {col_target}")
    print(f"{'-'*60}")

    # _____
    # ETAPA 1: Tratamento de Nulos
    # _____
    df_clean = df.withColumn(col_categorica,
        F.when(F.col(col_categorica).isNull(), "NULL").otherwise(F.col(col_categorica).cast('string')))

    # _____
    # ETAPA 2: Calcular Estatísticas Globais (no Spark)
    # _____
    global_stats = df_clean.select(
        F.count(col_target).alias('n_total'),
        F.mean(col_target).alias('grand_mean'),
        F.sum(F.col(col_target)).alias('sum_total')
    ).collect()[0]

    n_total = global_stats['n_total']
    grand_mean = global_stats['grand_mean']

    print(f" |- Observações válidas: {n_total},")
    print(f" |- Média global: {grand_mean:.4f}")

    # Calcular SS_total (Sum of Squares Total)
    #  $SS_{total} = \sum (y_i - \bar{y})^2$ 
    ss_total = df_clean.select(F.sum((F.col(col_target) - F.lit(grand_mean)) ** 2).alias('ss_total')).collect()[0]['ss_total']

    # _____
    # ETAPA 3: Calcular Estatísticas por Grupo (no Spark)
    # _____
    group_stats = df_clean.groupBy(col_categorica).agg(
        F.count(col_target).alias('n_j'),
        F.mean(col_target).alias('mean_j'),
        F.variance(col_target).alias('var_j') # Variância amostral
    ).collect()

    n_categories = len(group_stats)
    print(f" |- Número de categorias: {n_categories}")

    # Verificar se há categorias suficientes
    if n_categories < 2:
        print(f" |- AVISO: Apenas {n_categories} categoria(s). ANOVA não aplicável.")
        return {
            'feature': col_categorica,
            'f_statistic': 0.0,
            'p_value': 1.0,
            'eta_squared': 0.0,
            'n_categories': n_categories,
            'n_obs': n_total,
            'mean_between_groups': 0.0,
            'interpretacao': 'X Muito Fraco',
            'significativo': False,
            'status': 'insufficient_categories'
        }

    # _____
    # ETAPA 4: Calcular SS_between e SS_within
    # _____
    ss_between = 0.0
    ss_within = 0.0

    means = []
    for row in group_stats:
        n_j = row['n_j']
        mean_j = row['mean_j']
        var_j = row['var_j'] if row['var_j'] is not None else 0.0

        means.append(mean_j)

        #  $SS_{between} = \sum n_j (\bar{y}_j - \bar{y})^2$ 
        ss_between += n_j * (mean_j - grand_mean) ** 2

        #  $SS_{within} = \sum (n_j - 1) * var_j$ 

```

```

ss_within += (n_j - 1) * var_j

# =====
# ETAPA 5: Calcular F-statistic
# =====
df_between = n_categories - 1 # Graus de liberdade entre grupos
df_within = n_total - n_categories # Graus de liberdade dentro dos grupos

ms_between = ss_between / df_between if df_between > 0 else 0.0
ms_within = ss_within / df_within if df_within > 0 else 0.0

f_stat = ms_between / ms_within if ms_within > 0 else 0.0

# =====
# ETAPA 6: Calcular P-value (usando scipy)
# =====
try:
    from scipy.stats import f as f_dist
    p_value = 1 - f_dist.cdf(f_stat, df_between, df_within)
except Exception as e:
    print(f" └─ ⚠️ AVISO: Não foi possível calcular p-value: {e}")
    p_value = 0.0 if f_stat > 10 else 1.0 # Heurística

# =====
# ETAPA 7: Calcular Eta-Squared ( $\eta^2$ )
# =====
eta_squared = ss_between / ss_total if ss_total > 0 else 0.0

# =====
# ETAPA 8: Calcular Diferença Média Entre Grupos
# =====
mean_between_groups = max(means) - min(means) if means else 0.0

# =====
# ETAPA 9: Interpretação
# =====
if eta_squared > 0.14:
    interpretacao = "🔥 Forte"
elif eta_squared > 0.06:
    interpretacao = "✅ Moderado"
elif eta_squared > 0.01:
    interpretacao = "⚠️ Fraco"
else:
    interpretacao = "❌ Muito Fraco"

significancia = "✅ Significativo" if p_value < 0.05 else "❌ Não Significativo"

print(f" └─ F-statistic: {f_stat:.2f}")
print(f" └─ P-value: {p_value:.4e}")
print(f" └─ Eta-squared ( $\eta^2$ ): {eta_squared:.4f}")
print(f" └─ Interpretação: {interpretacao}")
print(f" └─ Significância: {significancia}")

return {
    'feature': col_categorica,
    'f_statistic': f_stat,
    'p_value': p_value,
    'eta_squared': eta_squared,
    'n_categories': n_categories,
    'n_obs': n_total,
    'mean_between_groups': mean_between_groups,
    'interpretacao': interpretacao,
    'significativo': p_value < 0.05,
    'status': 'success'
}

```

In [141]:

```
def ranking_features_categoricas(df, colunas_categoricas, col_target):
    """
    Rankeia variáveis categóricas por poder preditivo (ANOVA F-statistic).
    VERSÃO OTIMIZADA: Usa calcular_anova_f_spark (sem toPandas).
    """
    print(f"\n{'='*80}")
    print(f"📊 RANKING DE FEATURES CATEGÓRICAS")
    print(f"{'='*80}\n")

    results = []
    for i, col in enumerate(colunas_categoricas, 1):
        print(f"\n{i}/{len(colunas_categoricas)}] Processando: {col}")
        result = calcular_anova_f_spark(df, col, col_target)
        results.append(result)

    # =====
    # Criar DataFrame e Ordenar
    # =====
    ranking_df = pd.DataFrame(results).sort_values('f_statistic', ascending=False)

    # =====
    # Visualização: Tabela
    # =====
    print(f"\n{'='*80}")
    print(f"📊 RESULTADO FINAL")
    print(f"{'='*80}\n")

    display_df = ranking_df[['feature', 'f_statistic', 'p_value', 'eta_squared',
                            'n_categories', 'interpretacao', 'significativo']].copy()

    print(display_df.to_string(index=False))

    # =====
    # Visualização: Gráfico de Barras
    # =====
    fig, axes = plt.subplots(1, 2, figsize=(18, 8))

    # Gráfico 1: F-Statistic
    ax1 = axes[0]
    colors = ranking_df['eta_squared'].apply(lambda x: '#d62728' if x > 0.14 else
                                              '#ff7f0e' if x > 0.06 else
                                              '#2ca02c' if x > 0.01 else
                                              '#7f7f7f')
    )

    ax1.barh(ranking_df['feature'], ranking_df['f_statistic'], color=colors)
    ax1.set_xlabel('F-Statistic (ANOVA)', fontsize=12, fontweight='bold')
    ax1.set_ylabel('Feature Categórica', fontsize=12, fontweight='bold')
    ax1.set_title('Ranking de Features: F-Statistic\n(Maior = Melhor Separação Entre Grupos)', fontsize=14, fontweight='bold')
    ax1.grid(axis='x', alpha=0.3)

    # Legenda
    from matplotlib.patches import Patch
    legend_elements = [
        Patch(facecolor='#d62728', label='🔥 Forte ( $\eta^2 > 0.14$ )'),
        Patch(facecolor='#ff7f0e', label='🟡 Moderado ( $\eta^2 > 0.06$ )'),
        Patch(facecolor='#2ca02c', label='⚠️ Fraco ( $\eta^2 > 0.01$ )'),
        Patch(facecolor='#7f7f7f', label='✖️ Muito Fraco ( $\eta^2 \leq 0.01$ )')
    ]
    ax1.legend(handles=legend_elements, loc='lower right', fontsize=10)

    # Gráfico 2: Eta-Squared ( $\eta^2$ )
    ax2 = axes[1]
    ax2.barh(ranking_df['feature'], ranking_df['eta_squared'], color=colors)
    ax2.set_xlabel('Eta-Squared ( $\eta^2$ )', fontsize=12, fontweight='bold')
    ax2.set_ylabel('Feature Categórica', fontsize=12, fontweight='bold')
    ax2.set_title('Ranking de Features: Eta-Squared ( $\eta^2$ )\n(Proporção de Variância Explicada)', fontsize=14, fontweight='bold')
    ax2.grid(axis='x', alpha=0.3)
    ax2.set_xlim(0, max(0.2, ranking_df['eta_squared'].max() * 1.1))

    # Adicionar linhas de referência
    ax2.axvline(0.01, color='gray', linestyle='--', linewidth=1, alpha=0.5)
    ax2.axvline(0.06, color='orange', linestyle='--', linewidth=1, alpha=0.5)
    ax2.axvline(0.14, color='red', linestyle='--', linewidth=1, alpha=0.5)

    plt.tight_layout()
    plt.show()

    return ranking_df
```

```
In [142]: ranking_df = ranking_features_categoricas(
    df=df_elastic_net,
    colunas_categoricas=features_categoricas_base,
    col_target='target_win'
)

# =====
# Filtrar features aprovadas (critério:  $\eta^2 \geq 0.01$  e  $p < 0.05$ )
# =====

features_aprovadas = ranking_df[
    (ranking_df['eta_squared'] >= 0.01) &
    (ranking_df['significativo'] == True)
]['feature'].tolist()

print(f"\n✓ Features aprovadas para Elastic Net: {len(features_aprovadas)}")
print(features_aprovadas)
```

=====
⭐ RANKING DE FEATURES CATEGÓRICAS
=====

[1/19] Processando: total_plays_group

-Calculando ANOVA: total_plays_group vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 5
- |- F-statistic: 126256.73
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0496
- |- Interpretação: ⚠️ Fraco
- |- Significância: ✓ Significativo

[2/19] Processando: completed_songs_rate_group

-Calculando ANOVA: completed_songs_rate_group vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 5
- |- F-statistic: 30278.00
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0124
- |- Interpretação: ⚠️ Fraco
- |- Significância: ✓ Significativo

[3/19] Processando: avg_secs_per_unq_cap_group

-Calculando ANOVA: avg_secs_per_unq_cap_group vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 5
- |- F-statistic: 25747.17
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0105
- |- Interpretação: ⚠️ Fraco
- |- Significância: ✓ Significativo

[4/19] Processando: plays_per_unq_behavior

-Calculando ANOVA: plays_per_unq_behavior vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 5
- |- F-statistic: 25194.37
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0103
- |- Interpretação: ⚠️ Fraco
- |- Significância: ✓ Significativo

[5/19] Processando: plays_behavior_vs_volume

-Calculando ANOVA: plays_behavior_vs_volume vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 20

```
└─ F-statistic: 27911.18
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.0519
└─ Interpretação: 🔞 Fraco
└─ Significância: ✅ Significativo
```

[6/19] Processando: plays_behavior_vs_volume_collapsed

-Calculando ANOVA: plays_behavior_vs_volume_collapsed vs target_win

```
└─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 14
└─ F-statistic: 38754.35
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.0495
└─ Interpretação: 🔞 Fraco
└─ Significância: ✅ Significativo
```

[7/19] Processando: plays_behavior_vs_completion

-Calculando ANOVA: plays_behavior_vs_completion vs target_win

```
└─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 18
└─ F-statistic: 8853.64
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.0153
└─ Interpretação: 🔞 Fraco
└─ Significância: ✅ Significativo
```

[8/19] Processando: plays_behavior_vs_completion_collapsed

-Calculando ANOVA: plays_behavior_vs_completion_collapsed vs target_win

```
└─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 13
└─ F-statistic: 12529.59
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.0153
└─ Interpretação: 🔞 Fraco
└─ Significância: ✅ Significativo
```

[9/19] Processando: early_drop_rate_group

-Calculando ANOVA: early_drop_rate_group vs target_win

```
└─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 5
└─ F-statistic: 27663.63
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.0113
└─ Interpretação: 🔞 Fraco
└─ Significância: ✅ Significativo
```

[10/19] Processando: revenue_tier

-Calculando ANOVA: revenue_tier vs target_win

```
└─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 7
└─ F-statistic: 1258821.05
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.4383
└─ Interpretação: 🔥 Forte
└─ Significância: ✅ Significativo
```

[11/19] Processando: payment_method_group

-Calculando ANOVA: payment_method_group vs target_win

```
└─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 4
└─ F-statistic: 2079114.86
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.3919
└─ Interpretação: 🔥 Forte
```

└ Significância: Significativo

[12/19] Processando: payment_price_regime

-Calculando ANOVA: payment_price_regime vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 3
- |- F-statistic: 2781265.15
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.3650
- |- Interpretação: 🔴 Forte
- └ Significância: Significativo

[13/19] Processando: gender_clean

-Calculando ANOVA: gender_clean vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 3
- |- F-statistic: 40510.18
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0083
- |- Interpretação: ✗ Muito Fraco
- └ Significância: Significativo

[14/19] Processando: faixa_idade

-Calculando ANOVA: faixa_idade vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 5
- |- F-statistic: 34793.28
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0142
- |- Interpretação: ⚠️ Fraco
- └ Significância: Significativo

[15/19] Processando: registered_via_group

-Calculando ANOVA: registered_via_group vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 4
- |- F-statistic: 65275.19
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0198
- |- Interpretação: ⚠️ Fraco
- └ Significância: Significativo

[16/19] Processando: registration_year_regime

-Calculando ANOVA: registration_year_regime vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 3
- |- F-statistic: 17498.54
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0036
- |- Interpretação: ✗ Muito Fraco
- └ Significância: Significativo

[17/19] Processando: tenure_faixa

-Calculando ANOVA: tenure_faixa vs target_win

- |- Observações válidas: 9,678,866
- |- Média global: 45.1861
- |- Número de categorias: 3
- |- F-statistic: 26554.09
- |- P-value: 1.1102e-16
- |- Eta-squared (η^2): 0.0055
- |- Interpretação: ✗ Muito Fraco
- └ Significância: Significativo

[18/19] Processando: revenue_per_hour_tier

Calculando ANOVA: revenue_per_hour_tier vs target_win

```
├─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 4
└─ F-statistic: 706395.83
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.1796
└─ Interpretação: 🔥 Forte
└─ Significância: ✅ Significativo
```

[19/19] Processando: usage_intensity_tier

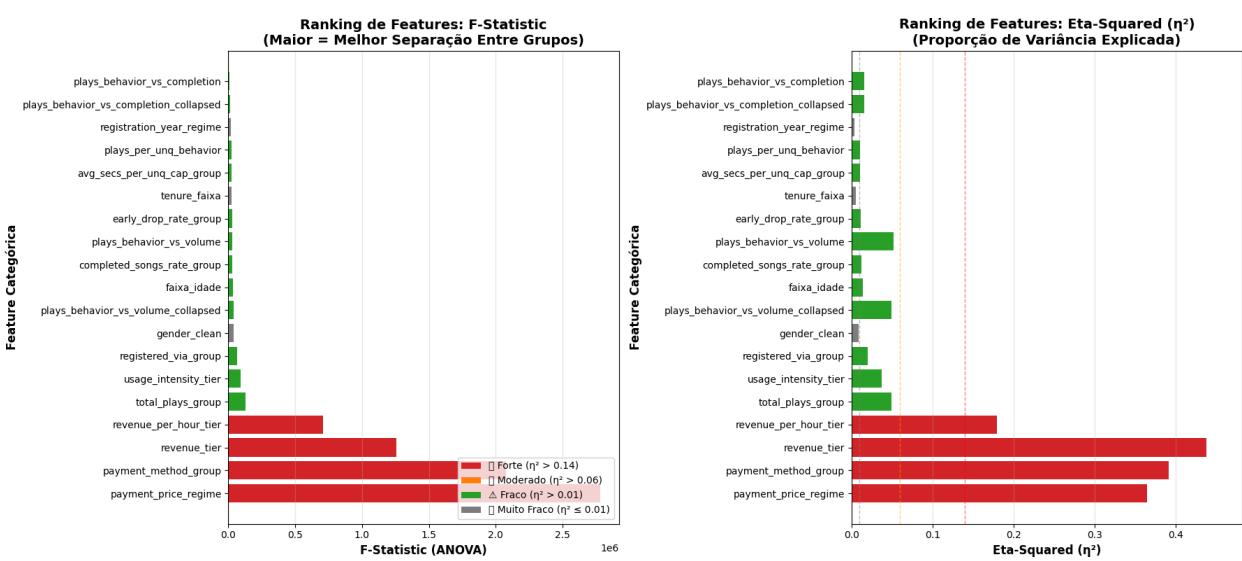
Calculando ANOVA: usage_intensity_tier vs target_win

```
├─ Observações válidas: 9,678,866
└─ Média global: 45.1861
└─ Número de categorias: 5
└─ F-statistic: 92764.45
└─ P-value: 1.1102e-16
└─ Eta-squared ( $\eta^2$ ): 0.0369
└─ Interpretação: ⚠️ Fraco
└─ Significância: ✅ Significativo
```

RESULTADO FINAL

feature	f_statistic	p_value	eta_squared	n_categories	interpretacao	significativo
payment_price_regime	2.781265e+06	1.110223e-16	0.364962	3	🔥 Forte	True
payment_method_group	2.079115e+06	1.110223e-16	0.391886	4	🔥 Forte	True
revenue_tier	1.258821e+06	1.110223e-16	0.438314	7	🔥 Forte	True
revenue_per_hour_tier	7.063958e+05	1.110223e-16	0.179622	4	🔥 Forte	True
total_plays_group	1.262567e+05	1.110223e-16	0.049591	5	⚠️ Fraco	True
usage_intensity_tier	9.276445e+04	1.110223e-16	0.036921	5	⚠️ Fraco	True
registered_via_group	6.527519e+04	1.110223e-16	0.019831	4	⚠️ Fraco	True
gender_clean	4.051018e+04	1.110223e-16	0.008301	3	✗ Muito Fraco	True
plays_behavior_vs_volume_collapsed	3.875435e+04	1.110223e-16	0.049477	14	⚠️ Fraco	True
faixa_idade	3.479328e+04	1.110223e-16	0.014175	5	⚠️ Fraco	True
completed_songs_rate_group	3.027800e+04	1.110223e-16	0.012358	5	⚠️ Fraco	True
plays_behavior_vs_volume	2.791118e+04	1.110223e-16	0.051945	20	⚠️ Fraco	True
early_drop_rate_group	2.766363e+04	1.110223e-16	0.011303	5	⚠️ Fraco	True
tenure_faixa	2.655409e+04	1.110223e-16	0.005457	3	✗ Muito Fraco	True
avg_secs_per_unq_cap_group	2.574717e+04	1.110223e-16	0.010529	5	⚠️ Fraco	True
plays_per_unq_behavior	2.519437e+04	1.110223e-16	0.010305	5	⚠️ Fraco	True
registration_year_regime	1.749854e+04	1.110223e-16	0.003603	3	✗ Muito Fraco	True
plays_behavior_vs_completion_collapsed	1.252959e+04	1.110223e-16	0.015297	13	⚠️ Fraco	True
plays_behavior_vs_completion	8.853636e+03	1.110223e-16	0.015312	18	⚠️ Fraco	True

```
C:\Users\Gustavo\AppData\Local\Temp\ipykernel_18752\1992549589.py:81: UserWarning: Glyph 128293 (\N{FIRE}) missing from font(s) DejaVu Sans.
    plt.tight_layout()
C:\Users\Gustavo\AppData\Local\Temp\ipykernel_18752\1992549589.py:81: UserWarning: Glyph 9989 (\N{WHITE HEAVY CHECK MARK}) missing from font(s) DejaVu Sans.
    plt.tight_layout()
C:\Users\Gustavo\AppData\Local\Temp\ipykernel_18752\1992549589.py:81: UserWarning: Glyph 10060 (\N{CROSS MARK}) missing from font(s) DejaVu Sans.
    plt.tight_layout()
c:\Users\Gustavo\anaconda3\envs\datamaster\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 128293 (\N{FIRE}) missing from font(s) DejaVu Sans.
    fig.canvas.print_figure(bytes_io, **kw)
c:\Users\Gustavo\anaconda3\envs\datamaster\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 9989 (\N{WHITE HEAVY CHECK MARK}) missing from font(s) DejaVu Sans.
    fig.canvas.print_figure(bytes_io, **kw)
c:\Users\Gustavo\anaconda3\envs\datamaster\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 10060 (\N{CROSS MARK}) missing from font(s) DejaVu Sans.
    fig.canvas.print_figure(bytes_io, **kw)
```



✓ Features aprovadas para Elastic Net: 16
['payment_price_regime', 'payment_method_group', 'revenue_tier', 'revenue_per_hour_tier', 'total_plays_group', 'usage_intensity_tier', 'registered_via_group', 'plays_behavior_vs_volume_collapsed', 'faixa_idade', 'completed_songs_rate_group', 'plays_behavior_vs_volume', 'early_drop_rate_group', 'avg_secs_per_unq_cap_group', 'plays_per_unq_behavior', 'plays_behavior_vs_completion_collapsed', 'plays_behavior_vs_completion']

Como foi anotado anteriormente, agora é possível comparar o η^2 com o grau de correlação das categóricas entre si (cramer v):

* plays_behavior_vs_volume (η^2 , η^2 colapsed: 0.0519, 0.0495) X total_plays_group (η^2 : 0.0496) --> **plays_behavior_vs_volume**
* plays_behavior_vs_completion (η^2 : 0.0153) X completed_songs_rate_group (η^2 : 0.0124) --> **plays_behavior_vs_completion**
* plays_behavior_vs_completion (η^2 : 0.0153) X avg_secs_per_unq_cap_group (η^2 : 0.0105) --> plays_behavior_vs_completion
* plays_behavior_vs_volume (η^2 , η^2 colapsed: 0.0519, 0.0495) X plays_per_unq_behavior (η^2 : 0.0103) --> plays_behavior_vs_volume
* plays_behavior_vs_completion (η^2 : 0.0153) X plays_per_unq_behavior (η^2 : 0.0103) --> plays_behavior_vs_completion
* payment_price_regime (η^2 : 0.3650) X revenue_tier (η^2 : 0.4383) --> **revenue_tier**, que também ganha por ter maior granularidade

Conclusão - categóricas finalistas EN

```
In [143]: features_categoricas_finais = [
    # =====
    # TIER S: Features de Receita/Preço (Essenciais)
    #
    'payment_method_group',           #  $\eta^2$  = 0.392 (mantida: captura método de pagamento)
    'revenue_tier',                  #  $\eta^2$  = 0.438 (mantida: melhor que payment_price_regime por ser mais granular)
    'revenue_per_hour_tier',          #  $\eta^2$  = 0.180 (mantida: captura eficiência receita/hora)

    #
    # TIER A: Features de Comportamento (Úteis)
    #
    'plays_behavior_vs_volume_collapsed', #  $\eta^2$  = 0.049 (mantida: versão collapsed é mais estável, mesmo que de menor  $\eta^2$ )
    'plays_behavior_vs_completion_collapsed', #  $\eta^2$  = 0.015 (mantida: captura interação não-linear)
    'usage_intensity_tier',             #  $\eta^2$  = 0.037 (mantida: complementa comportamento)

    #
    # TIER B: Features Demográficas/Cadastrais (Marginais)
    #
    'registered_via_group',           #  $\eta^2$  = 0.020 (mantida: canal de aquisição)
    'faixa_idade',                   #  $\eta^2$  = 0.014 (mantida: demográfica relevante)
]
# Total: 8 features
```

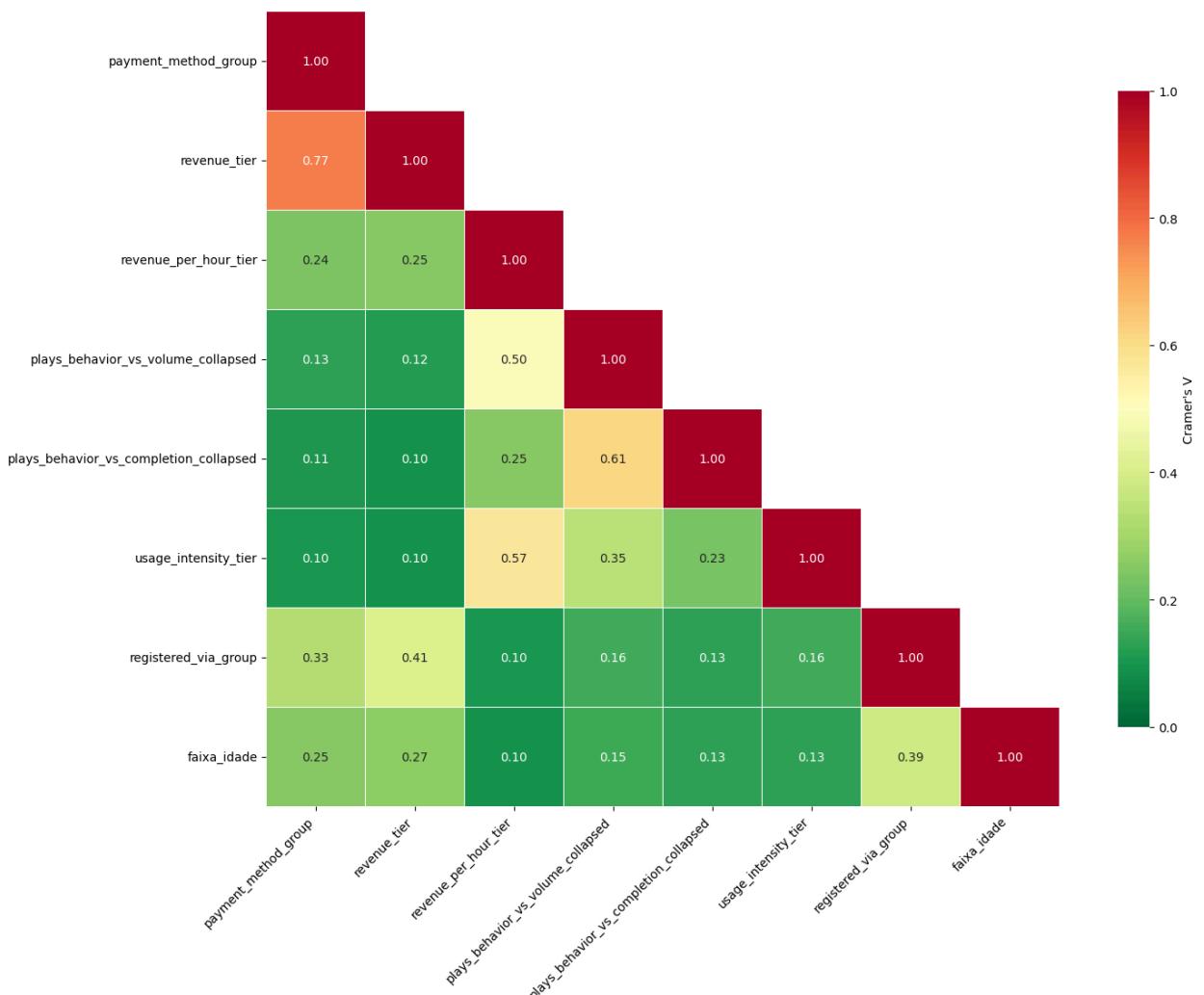
```
In [144]: cramer_matrix_finalistas = matriz_cramer_v_spark_optimized(
    df=df_elastic_net,
    colunas_categoricas=features_categoricas_finais,
    figsize=(16, 12) # Ajuste conforme necessário
)
```

🕒 Iniciando indexação de 8 colunas categóricas...
✓ Indexação concluída!

Calculando: payment_method_group ↔ revenue_tier
Calculando: payment_method_group ↔ revenue_per_hour_tier
Calculando: payment_method_group ↔ plays_behavior_vs_volume_collapsed
Calculando: payment_method_group ↔ plays_behavior_vs_completion_collapsed
Calculando: payment_method_group ↔ usage_intensity_tier
Calculando: payment_method_group ↔ registered_via_group
Calculando: payment_method_group ↔ faixa_idade
Calculando: revenue_tier ↔ revenue_per_hour_tier
Calculando: revenue_tier ↔ plays_behavior_vs_volume_collapsed
Calculando: revenue_tier ↔ plays_behavior_vs_completion_collapsed
Calculando: revenue_tier ↔ usage_intensity_tier
Calculando: revenue_tier ↔ registered_via_group
Calculando: revenue_tier ↔ faixa_idade
Calculando: revenue_per_hour_tier ↔ plays_behavior_vs_volume_collapsed
Calculando: revenue_per_hour_tier ↔ plays_behavior_vs_completion_collapsed
Calculando: revenue_per_hour_tier ↔ usage_intensity_tier
Calculando: revenue_per_hour_tier ↔ registered_via_group
Calculando: revenue_per_hour_tier ↔ faixa_idade
Calculando: plays_behavior_vs_volume_collapsed ↔ plays_behavior_vs_completion_collapsed
Calculando: plays_behavior_vs_volume_collapsed ↔ usage_intensity_tier
Calculando: plays_behavior_vs_volume_collapsed ↔ registered_via_group
Calculando: plays_behavior_vs_volume_collapsed ↔ faixa_idade
Calculando: plays_behavior_vs_completion_collapsed ↔ usage_intensity_tier
Calculando: plays_behavior_vs_completion_collapsed ↔ registered_via_group
Calculando: plays_behavior_vs_completion_collapsed ↔ faixa_idade
Calculando: usage_intensity_tier ↔ registered_via_group
Calculando: usage_intensity_tier ↔ faixa_idade
Calculando: registered_via_group ↔ faixa_idade

✓ Cálculo concluído!

Matriz de Associação: Cramer's V (Variáveis Categóricas)



Definição do melhor encoder para tratar variáveis categóricas e numericas

Salvando base inicial para Elastic Net

```
In [145]: df_elastic_net.count()
```

```
Out[145]: 9678866
```

```
In [146]: # 1. cache  
df_elastic_net = df_elastic_net.persist()  
df_elastic_net.count()
```

```
# 2. salvar particionado  
df_elastic_net.write \  
.mode("overwrite") \  
.partitionBy("safra") \  
.parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_feature_store_elastic_net")
```

ABT inicial

```
In [147]: features_numericas_finais_elastic_net
```

```
Out[147]: ['daily_revenue_efficiency_min_3',  
 'revenue_per_hour_listened_cap',  
 'actual_amount_paid',  
 'flag_plano_mensal_max_3',  
 'log_total_secs_mean_3',  
 'num_unq',  
 'usage_intensity_per_tenure_cap']
```

```
In [ ]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"  
df_feature_store_elastic_net = spark.read.parquet(silver_path + "df_feature_store_elastic_net")  
df_abt_inicial_elastic_net = df_feature_store_elastic_net.select("msno", "safra", "target_win",  
 *features_numericas_finais_elastic_net, *features_categoricas_finais)
```

```
In [149]: df_abt_inicial_elastic_net.count()
```

```
Out[149]: 9678866
```

```
In [150]: df_abt_inicial_elastic_net.printSchema()
```

```
root  
 |-- msno: string (nullable = true)  
 |-- safra: integer (nullable = true)  
 |-- target_win: double (nullable = true)  
 |-- daily_revenue_efficiency_min_3: double (nullable = true)  
 |-- revenue_per_hour_listened_cap: double (nullable = true)  
 |-- actual_amount_paid: double (nullable = true)  
 |-- flag_plano_mensal_max_3: integer (nullable = true)  
 |-- log_total_secs_mean_3: double (nullable = true)  
 |-- num_unq: double (nullable = true)  
 |-- usage_intensity_per_tenure_cap: double (nullable = true)  
 |-- payment_method_group: string (nullable = true)  
 |-- revenue_tier: string (nullable = true)  
 |-- revenue_per_hour_tier: string (nullable = true)  
 |-- plays_behavior_vs_volume_collapsed: string (nullable = true)  
 |-- plays_behavior_vs_completion_collapsed: string (nullable = true)  
 |-- usage_intensity_tier: string (nullable = true)  
 |-- registered_via_group: string (nullable = true)  
 |-- faixa_idade: string (nullable = true)
```

Conceitos e definições

Como vou tratar as features que correspondem a grupos/tiers, tendo mais de duas categorias com valores?

Modelo Linear (Elastic Net) funciona assim:

```
$$
\text{target} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n
$$
```

Onde x_1, x_2, \dots, x_n são **números**.

Problema: Como transformar categorias em números?

✗ Abordagem ERRADA: Label Encoding Simples

Ideia inicial: Categoria x → 1; Categoria y → 2; Categoria z → 3

Fazendo desta forma, assumo uma ordem que na verdade não existe. O modelo vai interpretar que:

```
$$
\text{Categoria y (2)} = 2 \times \text{Categoria x (1)}
$$
```

```
$$
\text{Categoria z (3)} = 3 \times \text{Categoria x (1)}
$$
```

E qual seria o impacto? Ex.:

```
$$
\text{target} = 100 + 50 \cdot \text{categoria}
$$
```

```
$$
\begin{aligned}
\text{Se categoria} = 1 & \quad \text{target} = 100 + 50 \cdot 1 = 150 \\
\text{Se categoria} = 2 & \quad \text{target} = 100 + 50 \cdot 2 = 200 \\
\text{Se categoria} = 3 & \quad \text{target} = 100 + 50 \cdot 3 = 250
\end{aligned}
$$
```

O modelo assume que a categoria z tem margem 100 maior que a categoria a, apenas porque o código é 3 ao invés de 1. Isso não faz sentido algum.

✓ Solução Ideal: One-Hot Encoding (OHE)

Ideia: criar uma flag (0 ou 1) para cada categoria. Para uma variável categórica C com k categorias $\{c_1, c_2, \dots, c_k\}$:

```
$$
\text{OHE}(C) = \begin{cases}
1 & \text{if } C = c_1 \\
0 & \text{if } C = c_2 \\
\vdots \\
0 & \text{if } C = c_k
\end{cases}
$$
```

Onde $\mathbb{1}(\cdot)$ é a **função indicadora**:

```
$$
\mathbb{1}(C = c_i) = \begin{cases}
1 & \text{if } C = c_i \\
0 & \text{otherwise}
\end{cases}
$$
```

O modelo, portanto, ficaria segunda a formula:

```
$$
\text{target} = \beta_0 + \beta_x \cdot \text{categoria}_x + \beta_y \cdot \text{categoria}_y + \beta_z \cdot \text{categoria}_z
$$
```

Podendo ser interpretado da seguinte maneira:

```
$$
\begin{aligned}
&\text{Se categoria é } x: & \text{target} = \beta_0 + \beta_x \cdot 1 + \beta_y \cdot 0 + \beta_z \cdot 0 = \beta_0 + \beta_x \\
&\text{Se categoria é } y: & \text{target} = \beta_0 + \beta_x \cdot 0 + \beta_y \cdot 1 + \beta_z \cdot 0 = \beta_0 + \beta_y \\
&\text{Se categoria é } z: & \text{target} = \beta_0 + \beta_x \cdot 0 + \beta_y \cdot 0 + \beta_z \cdot 1 = \beta_0 + \beta_z
\end{aligned}
$$
```

Onde cada categoria tem seu próprio efeito independente.

⚠ Problema: Multicolinearidade Perfeita (Dummy Variable Trap)

O que acontece se usarmos TODAS as colunas OHE?

```
$$
\text{categoria}_x + \text{categoria}_y + \text{categoria}_z = 1 \quad (\text{sempre!})
$$
```

Isso cria dependência linear perfeita entre as variáveis. Se mantiver todas as colunas, o modelo estará recebendo informação repetida. É possível listar os problemas vindos dessa situação:

- Incerteza nos Coeficientes: se as variáveis são redundantes, ao tentar calcular o "peso" de cada variável, o computador não sabe se deve dar o peso para a coluna x, y ou z. Elas começam a "brigar" pelo peso;
- Matriz Singular (Erro de Cálculo): na matemática de matrizes que acontece por baixo dos panos, o modelo precisa "inverter uma matriz". Se houver redundância perfeita, essa matriz não pode ser inversa (é como tentar dividir por zero). Com isso, modelo pode simplesmente travar ou retornar números absurdos (coeficientes gigantescos);
- Inflação da Variânci: Os erros padrão dos coeficientes ficam enormes. Isso significa que o modelo perde a confiabilidade estatística: se tornando obsoleto, para não dizer inútil.

Solução: Drop Last (Remover Última Categoria)

Remover uma categoria (geralmente a última) e usá-la como referência. Acompanhe o exemplo, onde consideraremos a categoria_z como referencia:

categoria_x	categoria_y
1	0
0	1
0	0
1	0

O modelo, agora, seria determinado pela equação:

```
$$
\text{target} = \beta_0 + \beta_x \cdot \text{categoria}_x + \beta_y \cdot \text{categoria}_y
$$
```

Sendo interpretado da seguinte forma:

```
$$
\begin{aligned}
&\text{Se categoria} = \text{z (referência)}: & \text{target} = \beta_0 \\
&\text{Se categoria} = \text{x}: & \text{target} = \beta_0 + \beta_x \quad (\rightarrow \text{diferença de } x \text{ em relação a } z) \\
&\text{Se categoria} = \text{y}: & \text{target} = \beta_0 + \beta_y \quad (\rightarrow \text{diferença de } y \text{ em relação a } z)
\end{aligned}
$$
```

Agora não há multicolinearidade.

One Hot Encoding - Passo a passo

Etapa 1 - StringIndexer

Spark (e muitas bibliotecas) não trabalham diretamente com strings. Precisamos converter strings em **índices numéricos** primeiro. E é aí que o `StringIndexer` entra: é uma ferramenta de pré-processamento no Apache Spark MLlib que converte colunas categóricas (texto/strings) em índices numéricos (0, 1, 2...). Ele ordena as categorias por frequência (o mais comum é 0), mapeando valores únicos para números.

Critério de Ordenação:

```
$$\text{text}{index}(c_i) = \text{text}{rank}_{\{\text{text}{desc}\}}(\text{text}{freq}(c_i))$$
```

Onde:

- $\text{text}{freq}(c_i)$ = frequência da categoria c_i
- $\text{text}{rank}_{\{\text{text}{desc}\}}$ = ranking decrescente (mais frequente = índice 0)

Ordenando por frequencia, garante-se que:

1. Categorias raras ficam no final → Serão removidas no `dropLast=True`
2. Categorias importantes (frequentes) são preservadas
3. Facilita tratamento de categorias desconhecidas (`handleInvalid="keep"`)

Porém, até aqui, a ferramenta só executa aquilo descrito anteriormente: uma ordem artificial. Como proceder deste ponto?

Etapa 2 - One Hot Encoder - Vetores Esparsos Binários

Vamos tratar categorias x, y e z como as cidades de São Paulo, Rio de Janeiro e Belo Horizonte deste ponto em diante. Agora, imagine que queremos representar cada uma delas com um vetor de 3 posições:

- * São Paulo = [1, 0, 0]
- * Rio de Janeiro = [0, 1, 0]
- * Belo Horizonte = [0, 0, 1]

Este vetor chama-se vetor denso. **Problema:** Se tivermos mil cidades, cada vetor teria mil números, mesmo que somente um deles seja diferente de zero.

Vetores Esparsos

Guardam somente o tamanho do vetor, quais posições que tem o número 1 e valores nessas posições. São Paulo tem valor denso de [1, 0, 0], o vetor esparsos seria definido por: **(3, [0], [1.0])**, onde:

- * 3 é o tamanho total do vetor;
- * [0] posição 0 do vetor tem valor diferente de zero;
- * [1.0] valor da posição 0.

Considerando a remoção de uma das categorias:

Se remover Belo Horizonte, o vetor esparsos agora tem tamanho 2:

São Paulo: (2, [0], [1.0])* → vetor de duas posições, posição 0 tem valor diferente de 0, valor este que é 1.0;

Rio de Janeiro: (2, [1], [1.0])* → vetor de duas posições, posição 1 tem valor diferente de 0, valor este que também é 1.0;

Belo Horizonte: (2, [], [])* → vetor de duas posições porém nenhuma posição com valor diferente de 0 e nenhum valor para mostrar.

Etapa 3 - VectorAssembler - Juntando Tudo

O modelo linear precisa de uma única coluna com todas as features. O que temos até então:

- Vetores esparsos de categorias OHE: `cidade_ohe`, `plays_behavior_ohe`, etc.
- Features numéricas: `avg_daily_secs`, `total_secs_raw`, etc.
- Flags binárias: `flag_plano_mensal`, `flag_valid_fee`, etc.

Como juntar tudo isso? **VectorAssembler**: consolida múltiplas colunas (numéricas e vetores) em uma única coluna de vetor denso/esparsos.

Imagine que o VectorAssembler está montando uma régua. Se a primeira variável tem 3 posições e a segunda tem 2, o resultado é uma régua de 5 posições:

* **Posições 0, 1, 2:** Pertencem à Variável A.

* **Posições 3, 4:** Pertencem à Variável B.

Se o usuário é da categoria 1 da Variável A e categoria 0 da Variável B, o VectorAssembler gera internamente:

[1.0, 0.0, 0.0, 1.0, 0.0]

O ponto chave: Para o Spark, isso continua sendo **uma única coluna**. Ele não expande isso em 5 colunas no seu DataFrame (o que seria visualmente caótico), mas sim em uma estrutura de dados que contém:

1. O tamanho total (5).
2. Os índices onde o valor é 1.0.

Por que ele é o "Porteiro" da Elastic Net?

A Elastic Net (Regressão Linear com penalização) precisa calcular um coeficiente (β) para cada categoria.

* Sem o VectorAssembler, você teria várias colunas de vetores.

* A Elastic Net não saberia como relacionar o β_1 com a coluna A e o β_4 com a coluna B.

O VectorAssembler entrega uma **única lista indexada**. Assim, o modelo sabe que o β da posição 0 da coluna `features` corresponde exatamente à classe "x" da sua variável categórica.

A Preservação da Esparsidade (Eficiência Máxima)

Como você está trabalhando **apenas com categóricas**, seu dado agora é composto quase 100% por zeros e uns.

* Em um DataFrame comum, isso ocuparia muita memória.

O VectorAssembler mantém isso como um **SparseVector**. Ele diz ao modelo: "Olha, aqui está um vetor de tamanho 50, mas só as posições 3, 12, 25 e 44 são iguais a 1. Ignore o resto."*

Isso faz com que o treinamento da Elastic Net seja extremamente rápido, pois ela pula os cálculos onde o valor é zero.

O Metadado (A Inteligência por trás da coluna)

Uma função pouco falada do VectorAssembler é que ele carrega **metadados**. Ele "anota" na coluna final de onde veio cada pedaço do vetor.

*Ele guarda a informação: "Do índice 0 a 4, esses dados vieram da coluna 'plays_behavior'"**.*

* Isso é o que permite que, depois de treinar o modelo, você consiga extrair os pesos e saber exatamente qual categoria está impactando positivamente ou negativamente a sua margem líquida.

Resumo Visual para Categóricas:

Coluna A (OHE) Coluna B (OHE) VectorAssembler (features)
:--- :--- :---
(3, [1], [1.0]) (2, [0], [1.0]) (5, [1, 3], [1.0, 1.0])

O que aconteceu?

1. Ele somou os tamanhos ($3 + 2 = 5$).
2. Ele deslocou o índice da Coluna B. O índice 0 da B virou o índice 3 no vetor final ($0 + \text{tamanho da A}$).

Execução

```

In [151]: print(f"\n'*80")
print(" ↗ Pipeline de One-Hot Encoding")
print("=* * 80)

stages = []

# Para cada agrupamento (NÃO para flags!)
for cat_col in features_categoricas_finais:
    # StringIndexer: converte string em índice numérico
    indexer = StringIndexer(
        inputCol=cat_col,
        outputCol=f"{cat_col}_indexed",
        handleInvalid="keep" # Mantém categorias desconhecidas
    )

    # OneHotEncoder: converte índice em vetor binário
    encoder = OneHotEncoder(
        inputCol=f"{cat_col}_indexed",
        outputCol=f"{cat_col}_ohe",
        dropLast=True # Remove última categoria (evita multicolinearidade)
    )

    stages.append(indexer)
    stages.append(encoder)

# Criar pipeline
pipeline_ohe = Pipeline(stages=stages)

print(f" ✓ Pipeline criado com {len(stages)} stages")
print(f"     • {len(features_categoricas_finais)} StringIndexers")
print(f"     • {len(features_categoricas_finais)} OneHotEncoders")

```

```

=====
↗ Pipeline de One-Hot Encoding
=====
✓ Pipeline criado com 16 stages
    • 8 StringIndexers
    • 8 OneHotEncoders

```

```

In [152]: print(f"\n'*80")
print(" ↗ Aplicando Pipeline ao DataFrame")
print("=* * 80)

# Fit e transform
pipeline_model_ohe = pipeline_ohe.fit(df_abt_inicial_elastic_net)
df_abt_inicial_elastic_net_ohe = pipeline_model_ohe.transform(df_abt_inicial_elastic_net)

print(f" ✓ Pipeline aplicado!")

# Colunas OHE criadas
colunas_ohe = [f"{cat_col}_ohe" for cat_col in features_categoricas_finais]

print(f"\n📋 Colunas OHE criadas:")
for i, col_name in enumerate(colunas_ohe, 1):
    print(f" {i:2d}. {col_name}")

```

```

=====
↗ Aplicando Pipeline ao DataFrame
=====
✓ Pipeline aplicado!

📋 Colunas OHE criadas:
1. payment_method_group_ohe
2. revenue_tier_ohe
3. revenue_per_hour_tier_ohe
4. plays_behavior_vs_volume_collapsed_ohe
5. plays_behavior_vs_completion_collapsed_ohe
6. usage_intensity_tier_ohe
7. registered_via_group_ohe
8. faixa_idade_ohe

```

```

In [153]: df_abt_inicial_elastic_net_ohe.count()

```

Out[153]: 9678866

In [154]:

```
df_abt_inicial_elastic_net_ohe.show(5, truncate=False)
```

msno	safra target_win
daily_revenue_efficiency_min_3 revenue_per_hour_listened_cap actual_amount_paid flag_plano_mensal_max_3 log_total_secs_mean_3 num_unq usage_intensity_per_tenure_cap payment_method_group revenue_tier	revenue_per_hour_tier plays_behavior_vs_volume_collapsed
plays_behavior_vs_completion_collapsed usage_intensity_tier registered_via_group faixa_idade	payment_method_group_indexed payment_method_group_ohe revenue_tier_indexed revenue_tier_ohe revenue_per_hour_tier_indexed revenue_per_hour_tier_ohe plays_behavior_vs_volume_collapsed_indexed plays_behavior_vs_volume_collapsed_ohe plays_behavior_vs_completion_collapsed_indexed plays_behavior_vs_completion_collapsed_ohe usage_intensity_tier_indexed usage_intensity_tier_ohe registered_via_group_indexed registered_via_group_ohe faixa_idade_indexed faixa_idade_ohe
+/1E1RNPUyfWkJOU5jleDrDWIegwQE/7IgouA95Q09E= 201601 31.53122809999993 3.3	0.0
99.0 1 12.037694099609842 461.0 0.0	01_most_auto
03_standard_99 00_unknown 02_light_repeat_03_frequent_listener 02_light_repeat_04_completionist 00_unknown	
low_value Desconhecido 0.0 (4,[0],[1.0]) 1.0 (7,[1],[1.0])	
0.0 (4,[0],[1.0]) 6.0 (14,[6],[1.0])	
5.0 (13,[5],[1.0]) (4,[0],[1.0]) 0.0 (4.0 (5,[4],[1.0]) 0.0	
0ShT9H5Nzbtd2xELtTDZ+In+nGV4g78DobY77YBTwc= 201601 67.637059 4.3	0.0
129.0 1 11.278236597239376 332.0 0.0	01_most_auto
06_others 00_unknown 01_explorer_02_regular_listener 01_explorer_04_completionist 00_unknown	
low_value Desconhecido 0.0 (4,[0],[1.0]) 3.0 (7,[3],[1.0])	
0.0 (4,[0],[1.0]) 0.0 (14,[0],[1.0])	
3.0 (13,[3],[1.0]) (4,[0],[1.0]) 0.0 (4.0 (5,[4],[1.0]) 0.0	
+1GDUd8h+zTeNCqDwHprhsOPB0cEQWHXRaeymT2unsw= 201601 98.8474210999999 4.9666666666666667	0.0
149.0 1 10.01578767333769 90.0 0.0	01_most_auto
04_premium_149 00_unknown 01_explorer_01_casual_listener 01_explorer_04_completionist 00_unknown	
high_value 23-34 0.0 (4,[0],[1.0]) 0.0 (7,[0],[1.0])	
0.0 (4,[0],[1.0]) 1.0 (14,[1],[1.0])	
3.0 (13,[3],[1.0]) (4,[1],[1.0]) 1.0 (5,[1],[1.0]) 0.0	
0SiKE0mXj0Jmev5CQpb2Dz19KDnBv86E8m2wCZ1sfk= 201601 85.2271066 4.966666666666667	0.0
149.0 1 11.361513832193024 413.0 0.0	03_most_manual
04_premium_149 00_unknown 01_explorer_03_frequent_listener 01_explorer_02_skipping_listener 00_unknown	
high_value 16-22 2.0 (4,[2],[1.0]) 0.0 (7,[0],[1.0])	
0.0 (4,[0],[1.0]) 2.0 (14,[2],[1.0])	
1.0 (13,[1],[1.0]) (4,[1],[1.0]) 3.0 (5,[3],[1.0]) 0.0	
+1H0fWeg5C5a7u9X0xiKeg6bccsSH0b1LjiUKS9MLK4Q= 201601 -71.1994458 4.966666666666667	0.0
149.0 1 11.57056979129114 529.0 0.0	03_most_manual
04_premium_149 00_unknown 01_explorer_03_frequent_listener 01_explorer_03_engaged_listener 00_unknown	
high_value 23-34 2.0 (4,[2],[1.0]) 0.0 (7,[0],[1.0])	
0.0 (4,[0],[1.0]) 2.0 (14,[2],[1.0])	
2.0 (13,[2],[1.0]) (4,[1],[1.0]) 1.0 (5,[1],[1.0]) 0.0	

only showing top 5 rows

```
In [155]: print(f"\n{'='*80}")
print("📊 Features Finais para Elastic Net")
print("=" * 80)

# Features categóricas (apenas OHE dos agrupamentos)
features_categoricas_final = colunas_ohe
# Total
features_elastic_net_final = features_numericas_finais_elastic_net + features_categoricas_final

print(f"\n✅ Composição das Features:")
print(f"  • Numéricas (contínuas): {len(features_numericas_finais_elastic_net)}")
print(f"  • Agrupamentos (OHE): {len(features_categoricas_final)}")
print(f"  • TOTAL: {len(features_elastic_net_final)}")

print(f"\n📋 Features Numéricas (contínuas):")
for i, f in enumerate(features_numericas_finais_elastic_net, 1):
    print(f"  {i:2d}. {f}")

print(f"\n📊 Features Agrupamentos (OHE):")
for i, f in enumerate(features_categoricas_final, 1):
    print(f"  {i:2d}. {f}")


```

```
=====
📊 Features Finais para Elastic Net
=====

✅ Composição das Features:
  • Numéricas (contínuas): 7
  • Agrupamentos (OHE): 8
  • TOTAL: 15

📋 Features Numéricas (contínuas):
  1. daily_revenue_efficiency_min_3
  2. revenue_per_hour_listened_cap
  3. actual_amount_paid
  4. flag_plano_mensal_max_3
  5. log_total_secs_mean_3
  6. num_unq
  7. usage_intensity_per_tenure_cap

📊 Features Agrupamentos (OHE):
  1. payment_method_group_ohe
  2. revenue_tier_ohe
  3. revenue_per_hour_tier_ohe
  4. plays_behavior_vs_volume_collapsed_ohe
  5. plays_behavior_vs_completion_collapsed_ohe
  6. usage_intensity_tier_ohe
  7. registered_via_group_ohe
  8. faixa_idade_ohe
```

```
In [156]: print(f"\n{'='*80}")
print("🔍 DIAGNÓSTICO: Verificar nulos nas features numéricas")
print("=" * 80)

for col in features_numericas_finais_elastic_net:
    n_null = df_abt_inicial_elastic_net_ohe.filter(F.col(col).isNull()).count()
    if n_null > 0:
        pct = 100 * n_null / df_abt_inicial_elastic_net_ohe.count()
        print(f"⚠️ {col}: {n_null:,} nulos ({pct:.2f}%)")
```

```
=====
🔍 DIAGNÓSTICO: Verificar nulos nas features numéricas
=====
```

A partir daqui, uno os vetores de categoricas criadas com as numericas tratadas

```
In [157]: print(f"\n{'='*80}")
print("VectorAssembler (juntar todas as features)")
print("=" * 80)

assembler_elastic_net = VectorAssembler(
    inputCols=features_elastic_net_final,
    outputCol='features_raw',
    handleInvalid='error'
)

df_abt_elastic_net = assembler_elastic_net.transform(df_abt_inicial_elastic_net_ohe)

print(f"  ✅ Vetor 'features_raw' criado!")
```

```
=====
VectorAssembler (juntar todas as features)
=====
  ✅ Vetor 'features_raw' criado!
```

```
In [158]: df_abt_elastic_net.count()
```

Out[158]: 9678866


```
In [160]: print(f"\n{'='*80}")
print("    StandardScaler (normalização)")
print("=" * 80)

scaler_elastic_net = StandardScaler(
    inputCol='features_raw',
    outputCol='features',
    withMean=True, # Centralizar (média = 0)
    withStd=True   # Normalizar (desvio padrão = 1)
)

scaler_model_elastic_net = scaler_elastic_net.fit(df_abt_elastic_net)
df_abt_elastic_net = scaler_model_elastic_net.transform(df_abt_elastic_net)

print(f"    ✅ Features normalizadas (coluna 'features')!")


=====
    StandardScaler (normalização)
=====
    ✅ Features normalizadas (coluna 'features')!
```

```
In [162]: df_abt_elastic_net.count()
```

```
Out[162]: 9678866
```


only showing top 5 rows

```
In [163]: print(f"\n{'='*80}")
print("Criando ABT Master - Elastic Net")
print("=" * 80)

colunas_finais = ['msno', 'safra', 'target_win', 'features'] + features_elastic_net_final

df_abt_elastic_net = df_abt_elastic_net.select(colunas_finais)

print(f"\n\ufe0f ABT Master criado: 'df_abt_elastic_net'")
print(f" • Colunas: {len(df_abt_elastic_net.columns)}")
print(f" • Linhas: {df_abt_elastic_net.count():,}")

print("\n" + "=" * 80)
print("\ufe0f ABT Master pronto para treinamento!")
print("=" * 80)
```

```
=====
Criando ABT Master - Elastic Net
=====

\ufe0f ABT Master criado: 'df_abt_elastic_net'
• Colunas: 19
• Linhas: 9,678,866

=====
\ufe0f ABT Master pronto para treinamento!
=====
```

```
In [164]: df_abt_elastic_net.count()
```

```
Out[164]: 9678866
```

In

df_abt_elastic_net.show(5, truncate=False)

[165]:

msno	safra	target_win	features
daily_revenue_efficiency_min_3 revenue_per_hour_listened_cap actual_amount_paid flag_plano_mensal_max_3 log_total_secs_mean_3 num_unq usage_intensity_per_tenure_cap payment_method_group_ohe revenue_tier_ohe revenue_per_hour_tier_ohe plays_behavior_vs_volume_collapsed_ohe plays_behavior_vs_completion_collapsed_ohe usage_intensity_tier_ohe registered_via_group_ohe faixa_idade_ohe			
+ 1E1RNPUyfWkJOUs5jleDrDWIegwQE/7IgouA95Q009E= 201601 31.5312280999999993			
[-0.35121447645707554,-0.5198032815692193,-0.20190477222526335,0.4145298179887715,0.5812724950303702,-0.014143454644861041,-0.4113173501290271,0.6435493964515279,-0.43496056201432937,-0.31604944137084856,-0.2118597570015029,-0.972839399615502,1.725838141494378,-0.43496056201432937,-0.23663509775490552,-0.20307655171316458,-0.09236172615439558,-0.04728234167361286,1.5158707660178719,-0.6004022731269593,-0.5608274867376163,-0.48830889275768574,-0.47389395581182825,-0.4630946081487677,-0.4609760288049956,-0.4050788527420964,-0.3370468129931648,-0.24183354682592526,5.22268772923048,-0.1681313419283704,-0.16657529649465988,-0.15961031641589082,-0.12452997929915992,-0.12121278367446203,-0.10934527328385699,-0.10191617939295324,-0.5095002233734213,-0.4976531891571195,-0.4376892966179497,-0.3530511849244295,-0.3370468129931648,3.6824298333945364,-0.21251540516548895,-0.18385445871155176,-0.13927247187952715,-0.13831738600163257,-0.12406800909316792,-0.11889029589250573,-0.10205993638634422,-0.5040158441192937,-0.5006980476323675,-0.5005577963872195,-0.4980873383789599,2.0135530942849833,1.0443601215661107,-0.956348961091592,-0.2133933605905438,-0.020897575596913843,1.0343636255810924,-0.6362981003340794,-0.36809024053618106,-0.3285705304826246,-0.1098877865696754] 3.3	0.0	99.0	1
12.037694099609842 461.0 0.0	(4,[0],[1.0])	(7,[1],[1.0])	(4,[0],[1.0])
(14,[6],[1.0])	(13,[5],[1.0])	(5,[4],[1.0])	(4,[0],[1.0])
(5,[0],[1.0])			
0\$ht9H5Nzbtd2xELtTDZ+In+nGV4g78DobY77YBTwc= 201601 67.637059			
[0.25115593789780183,-0.5198032815692193,0.08802556037543764,0.4145298179887715,0.3409617854453463,-0.24871738315516695,-0.4113173501290271,0.6435493964515279,-0.43496056201432937,-0.31604944137084856,-0.2118597570015029,-0.972839399615502,-0.5794285527936159,-0.43496056201432937,4.225915369992408,-0.20307655171316458,-0.09236172615439558,-0.04728234167361286,1.5158707660178719,-0.6004022731269593,-0.5608274867376163,-0.48830889275768574,2.1101765161132104,-0.4630946081487677,-0.4609760288049956,-0.4050788527420964,-0.3370468129931648,-0.24183354682592526,-0.19147227414828677,-0.1681313419283704,-0.16657529649465988,-0.15961031641589082,-0.12452997929915992,-0.12121278367446203,-0.18934527328385699,-0.18191617939295324,-0.5095002233734213,-0.4976531891571195,-0.4376892966179497,2.832450192444931,-0.3370468129931648,-0.2715597966357697,-0.21251540516548895,-0.18385445871155176,-0.13927247187952715,-0.13831738600163257,-0.12406800909316792,-0.11889029589250573,-0.10205993638634422,-0.5040158441192937,-0.5006980476323675,-0.5005577963872195,-0.4980873383789599,2.0135530942849833,1.0443601215661107,-0.956348961091592,-0.2133933605905438,-0.020897575596913843,1.0343636255810924,-0.6362981003340794,-0.36809024053618106,-0.3285705304826246,-0.1098877865696754] 4.3	0.0	129.0	1
11.278236597239376 332.0 0.0	(4,[0],[1.0])	(7,[3],[1.0])	(4,[0],[1.0])
(14,[0],[1.0])	(13,[3],[1.0])	(5,[4],[1.0])	(4,[0],[1.0])
(5,[0],[1.0])			
+1GDUD8h+zTeNCqDwHprhsOPB0cEQWHRXRaeymT2unsw= 201601 98.8474210999999			
[0.6527362141343869,-0.5198032815692193,0.281312448775905,0.4145298179887715,-0.058507567047367884,-0.6887707994303144,-0.4113173501290271,0.6435493964515279,-0.43496056201432937,-0.31604944137084856,-0.2118597570015029,1.0279187881137875,-0.5794285527936159,-0.43496056201432937,-0.23663509775490552,-0.20307655171316458,-0.09236172615439558,-0.04728234167361286,1.5158707660178719,-0.6004022731269593,-0.5608274867376163,-0.48830889275768574,-0.47389395581182825,2.1593857477193183,-0.4609760288049956,-0.4050788527420964,-0.3370468129931648,-0.24183354682592526,-0.19147227414828677,-0.1681313419283704,-0.16657529649465988,-0.15961031641589082,-0.12452997929915992,-0.12121278367446203,-0.10934527328385699,-0.10191617939295324,-0.5095002233734213,-0.4976531891571195,-0.4376892966179497,2.832450192444931,-0.3370468129931648,-0.2715597966357697,-0.21251540516548895,-0.18385445871155176,-0.13927247187952715,-0.13831738600163257,-0.12406800909316792,-0.11889029589250573,-0.10205993638634422,-0.5040158441192937,-0.5006980476323675,-0.5005577963872195,-0.4980873383789599,2.0135530942849833,-0.957524014975335,1.0456433136505898,-0.2133933605905438,-0.020897575596913843,-0.9667779028098979,1.5715902595922808,-0.36809024053618106,-0.3285705304826246,-0.1098877865696754] 4.9666666666666667	0.0	149.0	1
10.01578767333769 90.0 0.0	(4,[0],[1.0])	(7,[0],[1.0])	(4,[0],[1.0])
(14,[1],[1.0])	(13,[3],[1.0])	(5,[4],[1.0])	(4,[1],[1.0])
(5,[1],[1.0])			
0\$ikeE0mXj0Jmev5CQpb2dZ19KDnBv86E8m2wCZ1sfk= 201601 85.2271066			
[0.6527362141343869,-0.5198032815692193,0.281312448775905,0.4145298179887715,0.3673127153677264,-0.10142677688125393,-0.4113173501290271,-1.5538821140941455,-0.43496056201432937,3.1640615858856203,-0.2118597570015029,1.0279187881137875,-0.5794285527936159,-0.43496056201432937,-0.23663509775490552,-0.20307655171316458,-0.09236172615439558,-0.04728234167361286,1.5158707660178719,-0.6004022731269593,-0.5608274867376163,-0.48830889275768574,-0.47389395581182825,-0.4630946081487677,2.1693099731767966,-0.4050788527420964,-0.3370468129931648,-0.24183354682592526,-0.19147227414828677,-0.1681313419283704,-0.16657529649465988,-0.15961031641589082,-0.12452997929915992,-0.12121278367446203,-0.10934527328385699,-0.10191617939295324,-0.5095002233734213,-0.4976531891571195,-0.4376892966179497,2.832450192444931,-0.3370468129931648,-0.2715597966357697,-0.21251540516548895,-0.18385445871155176,-0.13927247187952715,-0.13831738600163257,-0.12406800909316792,-0.11889029589250573,-0.10205993638634422,-0.5040158441192937,-0.5006980476323675,-0.5005577963872195,-0.4980873383789599,2.0135530942849833,-0.957524014975335,1.0456433136505898,-0.2133933605905438,-0.020897575596913843,-0.9667779028098979,1.5715902595922808,-0.36809024053618106,-0.3285705304826246,-0.1098877865696754] 4.9666666666666667	0.0	149.0	1
(14,[1],[1.0])	(13,[3],[1.0])	(5,[4],[1.0])	(4,[1],[1.0])
(5,[1],[1.0])			
0\$ikeE0mXj0Jmev5CQpb2dZ19KDnBv86E8m2wCZ1sfk= 201601 85.2271066			

Salvando ABT/Master para os modelos de Elastic Net

```
In silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"
[166]: df_spine_splited = spark.read.parquet(silver_path + "df_spine_splited")
```

```
In df_spine_splited.count()
```

[167]:

In df_spine_spl

```
root
|-- msno: string (nullable = true)
|-- partition: string (nullable = true)
|-- safra: integer (nullable = true)
```

```
In [170]: df_abt_elastic_net = df_abt_elastic_net.join(df_spine_splited, on=["msno", "safra"], how="inner")
```

```
In df abt elastic net.count()
```

[171]:

```
In [172]: df_abt_elastic_net.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- target_win: double (nullable = true)
 |-- features: vector (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- actual_amount_paid: double (nullable = true)
 |-- flag_plano_mensal_max_3: integer (nullable = true)
 |-- log_total_secs_mean_3: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- payment_method_group_ohe: vector (nullable = true)
 |-- revenue_tier_ohe: vector (nullable = true)
 |-- revenue_per_hour_tier_ohe: vector (nullable = true)
 |-- plays_behavior_vs_volume_collapsed_ohe: vector (nullable = true)
 |-- plays_behavior_vs_completion_collapsed_ohe: vector (nullable = true)
 |-- usage_intensity_tier_ohe: vector (nullable = true)
 |-- registered_via_group_ohe: vector (nullable = true)
 |-- faixa_idade_ohe: vector (nullable = true)
 |-- partition: string (nullable = true)
```

```
In [173]: # 1. cache
df_abt_elastic_net = df_abt_elastic_net.persist()
df_abt_elastic_net.count()
```

```
# 2. salvar particionado
df_abt_elastic_net.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_master_elastic_net")
```

12.5. Decisao por modelo: Modelos de Arvores - Comum entre LightGM e Random Forest

Carregando bases

```
In [ ]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"
df_feature_store_final = spark.read.parquet(silver_path + "df_feature_store_final")
df_spine_split = spark.read.parquet(silver_path + "df_spine_split")
```

```
In [179]: df_feature_store_trees = df_feature_store_final.join(df_spine_split, on=["msno", "safra"], how="inner")
```

```
In [180]: df_feature_store_trees.count()
```

```
Out[180]: 9678866
```

In df_feature_store_trees.printSchema()

[181]:

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- target: double (nullable = true)
 |-- target_win: double (nullable = true)
 |-- flag_valid_fee_lag_1: integer (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- num_unq_ratio_ref_max_6: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- flag_has_transactions_max_3: integer (nullable = true)
 |-- total_plays_ratio_ref_max_6: double (nullable = true)
 |-- flag_plano_mensal: integer (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- total_plays: double (nullable = true)
 |-- log_total_secs: double (nullable = true)
 |-- flag_valid_fee_max_3: integer (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- total_plays_mean_3: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- margem_liquida_mensal: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
 |-- total_secs: double (nullable = true)
 |-- flag_plano_mensal_lag_1: integer (nullable = true)
 |-- flag_has_transactions_lag_1: integer (nullable = true)
 |-- flag_valid_fee: integer (nullable = true)
 |-- flag_has_transactions: integer (nullable = true)
 |-- actual_amount_paid: float (nullable = true)
 |-- avg_secs_per_unq_cap_min_6: double (nullable = true)
 |-- log_total_secs_ratio_ref_max_6: double (nullable = true)
 |-- num_100: double (nullable = true)
 |-- num_25: double (nullable = true)
 |-- is_auto_renew: integer (nullable = true)
 |-- plays_per_unq_cap_min_6: double (nullable = true)
 |-- payment_method_id: integer (nullable = true)
 |-- log_total_plays_ratio_ref_max_6: double (nullable = true)
 |-- completion_efficiency_min_6: double (nullable = true)
 |-- flag_plano_mensal_max_3: integer (nullable = true)
 |-- plan_list_price: float (nullable = true)
 |-- num_unq_mean_3: double (nullable = true)
 |-- log_total_secs_mean_3: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- completed_songs_rate_min_6: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_plays_group: string (nullable = true)
 |-- completed_songs_rate_group: string (nullable = true)
 |-- avg_secs_per_unq_cap_group: string (nullable = true)
 |-- plays_per_unq_behavior: string (nullable = true)
 |-- plays_behavior_vs_volume: string (nullable = true)
 |-- plays_behavior_vs_volume_collapsed: string (nullable = true)
 |-- plays_behavior_vs_completion: string (nullable = true)
 |-- plays_behavior_vs_completion_collapsed: string (nullable = true)
 |-- early_drop_rate_group: string (nullable = true)
 |-- revenue_tier: string (nullable = true)
 |-- payment_method_group: string (nullable = true)
 |-- payment_price_regime: string (nullable = true)
 |-- gender_clean: string (nullable = true)
 |-- faixa_idade: string (nullable = true)
 |-- registered_via_group: string (nullable = true)
 |-- registration_year_regime: string (nullable = true)
 |-- tenure_faixa: string (nullable = true)
 |-- revenue_per_hour_tier: string (nullable = true)
 |-- usage_intensity_tier: string (nullable = true)
 |-- membership_expire_date: date (nullable = true)
 |-- transaction_date: date (nullable = true)
 |-- registration_init_time: date (nullable = true)
 |-- partition: string (nullable = false)
```

In # Exemplo de código

[182]:

```
num_colunas = len(df_feature_store_trees.columns)
print(f"Total de colunas: {num_colunas}")
```

Total de colunas: 70

Analises

Separando tipo de variaveis

```
In # Remover features de controle e datas
[196]: features_controle = ['msno', 'target', 'safra', 'partition', 'membership_expire_date', 'transaction_date', 'registration_init_time']

# Separar features por tipo
features_numericas = [
    'flag_valid_fee_lag_1', 'total_secs_ratio_ref_max_6', 'num_unq_ratio_ref_max_6',
    'usage_intensity_per_tenure_cap', 'daily_revenue_efficiency', 'log_total_plays_mean_3',
    'flag_has_transactions_max_3', 'total_plays_ratio_ref_max_6', 'flag_plano_mensal',
    'revenue_per_hour_listened_cap', 'total_plays', 'log_total_secs', 'flag_valid_fee_max_3',
    'daily_revenue_efficiency_min_3', 'total_plays_mean_3', 'num_50', 'margem_liquida_mensal',
    'num_75', 'log_total_plays', 'total_secs_mean_3', 'daily_revenue_efficiency_ratio_ref_max_3',
    'total_secs', 'flag_plano_mensal_lag_1', 'flag_has_transactions_lag_1', 'flag_valid_fee',
    'flag_has_transactions', 'actual_amount_paid', 'avg_secs_per_unq_cap_min_6',
    'log_total_secs_ratio_ref_max_6', 'num_100', 'num_25', 'is_auto_renew',
    'plays_per_unq_cap_min_6', 'payment_method_id', 'log_total_plays_ratio_ref_max_6',
    'completion_efficiency_min_6', 'flag_plano_mensal_max_3', 'plan_list_price',
    'num_unq_mean_3', 'log_total_secs_mean_3', 'num_985', 'completed_songs_rate_min_6', 'num_unq'
]

features_categoricas = [
    'total_plays_group', 'completed_songs_rate_group', 'avg_secs_per_unq_cap_group',
    'plays_per_unq_behavior', 'plays_behavior_vs_volume', 'plays_behavior_vs_volume_collapsed',
    'plays_behavior_vs_completion', 'plays_behavior_vs_completion_collapsed',
    'early_drop_rate_group', 'revenue_tier', 'payment_method_group', 'payment_price_regime',
    'gender_clean', 'faixa_idade', 'registered_via_group', 'registration_year_regime',
    'tenure_faixa', 'revenue_per_hour_tier', 'usage_intensity_tier'
]

print(f"📊 Features Numéricas: {len(features_numericas)}")
print(f"📊 Features Categóricas: {len(features_categoricas)})")
```

📊 Features Numéricas: 43
📊 Features Categóricas: 19

Valores nulos

Checagem

```
In # Checando os nulos nas variaveis numericas  
[197]: df_feature_store_trees.select([F.count(F.when(F.col(f'{c}').isNull(), c)).alias(c) for c in features_numericas]).show()
```

```

+-----+
|flag_valid_fee_lag_1|total_secs_ratio_ref_max_6|num_unq_ratio_ref_max_6|usage_intensity_per_tenure_cap|daily_revenue_efficiency|log_total_plays_mean_3|flag_has_transactions_max_3|total_plays_ratio_ref_max_6|flag_plano_mensal|revenue_per_hour_listened_cap|total_plays|log_total_secs|flag_valid_fee_max_3|daily_revenue_efficiency_min_3|total_plays_mean_3|num_50|margem_liquida_mensal|num_75|log_total_plays|total_secs_mean_3|daily_revenue_efficiency_ratio_ref_max_3|total_secs|flag_plano_mensal_lag_1|flag_has_transactions_lag_1|flag_valid_fee|flag_has_transactions|actual_amount_paid|avg_secs_per_unq_cap_min_6|log_total_secs_ratio_ref_max_6|num_100|num_25|is_auto_renew|plays_per_unq_cap_min_6|payment_method_id|log_total_plays_ratio_ref_max_6|completion_efficiency_min_6|flag_plano_mensal_max_3|plan_list_price|num_unq_mean_3|log_total_secs_mean_3|num_985|completed_songs_rate_min_6|num_unq|
+-----+
|           1259830|                                453840|                                453840|                                0|                                0| | | | | | | |
|1539830|          987360|          987360|          0|          0|          453840|          453840|          0|          0|
|0| 987360| 987360| 987360| 0| 0| 0| 987360| 987360| 0| 0|
|1259830|          0|          0|          0|          1539830|          1539830|          0|          0|          0|          1259830|
|987360| 987360| 1539830| 1539830| 0| 1539830| 1539830| 0| 987360| 453840| 0| 987360|
|0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0|
+-----+

```

Tratamento

```
In # Grupos de colunas baseados no registro de nulos  
[198]: cols_lag = [c for c in features_numericas if "_lag_" in c]  
cols_ratio = [c for c in features_numericas if "_ratio_" in c]  
# Colunas "raw" (originais ou transformações simples sem lag/ratio)  
cols_originais = [c for c in features_numericas if c not in cols_lag and c not in cols_ratio]
```

```

In [199]: df_feature_store_trees_treated = df_feature_store_trees

# --- A. Columnas Originais: Preencher com Zero ---
# (total_plays, num_unq, num_25, num_50, num_75, num_100, num_985, etc.)
df_feature_store_trees_treated = df_feature_store_trees_treated.fillna(0, subset=cols_originais)

# --- B. Flags com Lag: Preencher com o valor do mês atual ---
# Se o lag_1 é nulo, assume-se que o comportamento atual se repete ou que
# o cliente é novo (neste caso, usamos a referência do mês atual).
for c in cols_lag:
    # Extrai o nome da variável original (ex: flag_plano_mensal de flag_plano_mensal_lag_1)
    base_col = c.split('_lag_')[0]
    if base_col in df_feature_store_trees_treated.columns:
        df_feature_store_trees_treated = df_feature_store_trees_treated.withColumn(c, F.coalesce(F.col(c), F.col(base_col)))
    else:
        # Caso a base_col não esteja no DF por algum motivo, preenchemos com 0 (neutro para flags)
        df_feature_store_trees_treated = df_feature_store_trees_treated.fillna(0, subset=[c])

# --- C. Variáveis de Ratio: Tratamento de Sentinelas ---
# O ratio deve herdar o sentinelas de "Base Vazia" (-99998).
for c in cols_ratio:
    # Identifica o denominador correspondente (ex: var_max_6 para var_ratio_ref_max_6)
    parts = c.split('_ratio_ref_')
    var_name = parts[0]
    metric_suffix = parts[1]
    denom_col = f'{var_name}_{metric_suffix}'

    if denom_col in df_feature_store_trees_treated.columns:
        df_feature_store_trees_treated = df_feature_store_trees_treated.withColumn(c,
            F.when(F.col(c).isNotNull(), F.col(c))
            .when(F.col(denom_col).isNull(), F.lit(-99998)) # Todos os lags eram nulos
            .otherwise(F.lit(-99995)) # Caso genérico de falha no cálculo do ratio (ex: ref nulo)
        )
    else:
        df_feature_store_trees_treated = df_feature_store_trees_treated.fillna(-99998, subset=[c])

# --- Verificação Final ---
# Se ainda sobrar algum nulo residual (ex: colunas que não entraram na lógica acima)
# df_feature_store_trees_treated = df_feature_store_trees_treated.fillna(-99999)

```

```
In # Checando os nulos nas variaveis numericas apos tratamento
[200]: df_feature_store_trees_treated.select([F.count(F.when(F.col(f'{c}').isNull(), c)).alias(c) for c in features_numericas]).show()
```

```
In [ ]: # Filtrar apenas partition == 'train' para análises  
df_train = df_feature_store_trees_treated.filter(F.col('partition') == 'train')
```

```
In [ ]: print("\n" + "="*80)
print("Análise de Multicolinearidade (Numéricas)")
print("="*80)

# Assembler para criar vetor de features
assembler = VectorAssembler(inputCols=features_numericas, outputCol="features_vec")
df_vec = assembler.transform(df_train.select(features_numericas))

# Calcular matriz de correlação
matriz_corr = Correlation.corr(df_vec, "features_vec", "pearson").head()[0].toArray()

# Identificar pares com correlação > 0.90
threshold_corr = 0.90
pares_colineares = []

for i in range(len(features_numericas)):
    for j in range(i+1, len(features_numericas)):
        if abs(matriz_corr[i][j]) > threshold_corr:
            pares_colineares.append({
                'feature_1': features_numericas[i],
                'feature_2': features_numericas[j],
                'correlacao': matriz_corr[i][j]
            })

print(f"\n⚠️ Encontrados {len(pares_colineares)} pares com |corr| > {threshold_corr}")

# Exibir pares
if pares_colineares:
    print("\n📁 Pares Colineares:")
    for par in pares_colineares:
        print(f"  |- {par['feature_1']} <-> {par['feature_2']}")
        print(f"    |- Correlação: {par['correlacao']:.4f}")
```

```
=====
Análise de Multicolinearidade (Numéricas)
=====
```

⚠️ Encontrados 83 pares com |corr| > 0.9

- 📁 Pares Colineares:
 - flag_valid_fee_lag_1 <-> flag_plano_mensal_lag_1
 - |- Correlação: 0.9365
 - flag_valid_fee_lag_1 <-> flag_has_transactions_lag_1
 - |- Correlação: 0.9830
 - total_secs_ratio_ref_max_6 <-> num_unq_ratio_ref_max_6
 - |- Correlação: 1.0000
 - total_secs_ratio_ref_max_6 <-> log_total_plays_mean_3
 - |- Correlação: 0.9045
 - total_secs_ratio_ref_max_6 <-> total_plays_ratio_ref_max_6
 - |- Correlação: 1.0000
 - total_secs_ratio_ref_max_6 <-> total_plays_mean_3
 - |- Correlação: 0.9054
 - total_secs_ratio_ref_max_6 <-> avg_secs_per_unq_cap_min_6
 - |- Correlação: 0.9804
 - total_secs_ratio_ref_max_6 <-> log_total_secs_ratio_ref_max_6
 - |- Correlação: 1.0000
 - total_secs_ratio_ref_max_6 <-> plays_per_unq_cap_min_6
 - |- Correlação: 0.9647
 - total_secs_ratio_ref_max_6 <-> log_total_plays_ratio_ref_max_6
 - |- Correlação: 1.0000
 - total_secs_ratio_ref_max_6 <-> completion_efficiency_min_6
 - |- Correlação: 0.9700
 - total_secs_ratio_ref_max_6 <-> num_unq_mean_3
 - |- Correlação: 0.9052
 - total_secs_ratio_ref_max_6 <-> log_total_secs_mean_3
 - |- Correlação: 0.9045
 - total_secs_ratio_ref_max_6 <-> completed_songs_rate_min_6
 - |- Correlação: 0.9647
 - num_unq_ratio_ref_max_6 <-> log_total_plays_mean_3
 - |- Correlação: 0.9045
 - num_unq_ratio_ref_max_6 <-> total_plays_ratio_ref_max_6
 - |- Correlação: 1.0000
 - num_unq_ratio_ref_max_6 <-> total_plays_mean_3
 - |- Correlação: 0.9054
 - num_unq_ratio_ref_max_6 <-> avg_secs_per_unq_cap_min_6
 - |- Correlação: 0.9804
 - num_unq_ratio_ref_max_6 <-> log_total_secs_ratio_ref_max_6
 - |- Correlação: 1.0000
 - num_unq_ratio_ref_max_6 <-> plays_per_unq_cap_min_6
 - |- Correlação: 0.9647
 - num_unq_ratio_ref_max_6 <-> log_total_plays_ratio_ref_max_6
 - |- Correlação: 1.0000
 - num_unq_ratio_ref_max_6 <-> completion_efficiency_min_6
 - |- Correlação: 0.9700
 - num_unq_ratio_ref_max_6 <-> num_unq_mean_3
 - |- Correlação: 0.9052

```
|- num_unq_ratio_ref_max_6 <-> log_total_secs_mean_3
|   └ Correlação: 0.9045
|- num_unq_ratio_ref_max_6 <-> completed_songs_rate_min_6
|   └ Correlação: 0.9647
|- daily_revenue_efficiency <-> flag_valid_fee
|   └ Correlação: 0.9083
|- log_total_plays_mean_3 <-> total_plays_ratio_ref_max_6
|   └ Correlação: 0.9045
|- log_total_plays_mean_3 <-> total_plays_mean_3
|   └ Correlação: 0.9999
|- log_total_plays_mean_3 <-> log_total_secs_ratio_ref_max_6
|   └ Correlação: 0.9045
|- log_total_plays_mean_3 <-> log_total_plays_ratio_ref_max_6
|   └ Correlação: 0.9045
|- log_total_plays_mean_3 <-> num_unq_mean_3
|   └ Correlação: 0.9999
|- log_total_plays_mean_3 <-> log_total_secs_mean_3
|   └ Correlação: 1.0000
|- flag_has_transactions_max_3 <-> flag_valid_fee_max_3
|   └ Correlação: 0.9842
|- total_plays_ratio_ref_max_6 <-> total_plays_mean_3
|   └ Correlação: 0.9054
|- total_plays_ratio_ref_max_6 <-> avg_secs_per_unq_cap_min_6
|   └ Correlação: 0.9804
|- total_plays_ratio_ref_max_6 <-> log_total_secs_ratio_ref_max_6
|   └ Correlação: 1.0000
|- total_plays_ratio_ref_max_6 <-> plays_per_unq_cap_min_6
|   └ Correlação: 0.9647
|- total_plays_ratio_ref_max_6 <-> log_total_plays_ratio_ref_max_6
|   └ Correlação: 1.0000
|- total_plays_ratio_ref_max_6 <-> completion_efficiency_min_6
|   └ Correlação: 0.9700
|- total_plays_ratio_ref_max_6 <-> num_unq_mean_3
|   └ Correlação: 0.9052
|- total_plays_ratio_ref_max_6 <-> log_total_secs_mean_3
|   └ Correlação: 0.9045
|- total_plays_ratio_ref_max_6 <-> completed_songs_rate_min_6
|   └ Correlação: 0.9647
|- flag_plano_mensal <-> flag_valid_fee
|   └ Correlação: 0.9526
|- flag_plano_mensal <-> flag_has_transactions
|   └ Correlação: 0.9373
|- flag_plano_mensal <-> payment_method_id
|   └ Correlação: 0.9379
|- total_plays <-> total_secs
|   └ Correlação: 0.9784
|- total_plays <-> num_100
|   └ Correlação: 0.9705
|- total_plays <-> num_unq
|   └ Correlação: 0.9379
|- log_total_secs <-> log_total_plays
|   └ Correlação: 0.9688
|- total_plays_mean_3 <-> log_total_secs_ratio_ref_max_6
|   └ Correlação: 0.9054
|- total_plays_mean_3 <-> log_total_plays_ratio_ref_max_6
|   └ Correlação: 0.9054
|- total_plays_mean_3 <-> num_unq_mean_3
|   └ Correlação: 1.0000
|- total_plays_mean_3 <-> log_total_secs_mean_3
|   └ Correlação: 0.9999
|- num_50 <-> num_75
|   └ Correlação: 0.9005
|- margem_liquida_mensal <-> actual_amount_paid
|   └ Correlação: 0.9862
|- margem_liquida_mensal <-> plan_list_price
|   └ Correlação: 0.9829
|- total_secs <-> num_100
|   └ Correlação: 0.9936
|- total_secs <-> num_unq
|   └ Correlação: 0.9056
|- flag_plano_mensal_lag_1 <-> flag_has_transactions_lag_1
|   └ Correlação: 0.9208
|- flag_valid_fee <-> flag_has_transactions
|   └ Correlação: 0.9836
|- flag_valid_fee <-> payment_method_id
|   └ Correlação: 0.9637
|- flag_has_transactions <-> payment_method_id
|   └ Correlação: 0.9768
|- actual_amount_paid <-> plan_list_price
|   └ Correlação: 0.9963
|- avg_secs_per_unq_cap_min_6 <-> log_total_secs_ratio_ref_max_6
|   └ Correlação: 0.9804
|- avg_secs_per_unq_cap_min_6 <-> plays_per_unq_cap_min_6
|   └ Correlação: 0.9793
|- avg_secs_per_unq_cap_min_6 <-> log_total_plays_ratio_ref_max_6
|   └ Correlação: 0.9804
|- avg_secs_per_unq_cap_min_6 <-> completion_efficiency_min_6
|   └ Correlação: 0.9878
```

```
|- avg_secs_per_unq_cap_min_6 <-> completed_songs_rate_min_6
|   \ Correlação: 0.9793
|- log_total_secs_ratio_ref_max_6 <-> plays_per_unq_cap_min_6
|   \ Correlação: 0.9647
|- log_total_secs_ratio_ref_max_6 <-> log_total_plays_ratio_ref_max_6
|   \ Correlação: 1.0000
|- log_total_secs_ratio_ref_max_6 <-> completion_efficiency_min_6
|   \ Correlação: 0.9700
|- log_total_secs_ratio_ref_max_6 <-> num_unq_mean_3
|   \ Correlação: 0.9052
|- log_total_secs_ratio_ref_max_6 <-> log_total_secs_mean_3
|   \ Correlação: 0.9045
|- log_total_secs_ratio_ref_max_6 <-> completed_songs_rate_min_6
|   \ Correlação: 0.9647
|- plays_per_unq_cap_min_6 <-> log_total_plays_ratio_ref_max_6
|   \ Correlação: 0.9647
|- plays_per_unq_cap_min_6 <-> completion_efficiency_min_6
|   \ Correlação: 0.9869
|- plays_per_unq_cap_min_6 <-> completed_songs_rate_min_6
|   \ Correlação: 1.0000
|- log_total_plays_ratio_ref_max_6 <-> completion_efficiency_min_6
|   \ Correlação: 0.9700
|- log_total_plays_ratio_ref_max_6 <-> num_unq_mean_3
|   \ Correlação: 0.9052
|- log_total_plays_ratio_ref_max_6 <-> log_total_secs_mean_3
|   \ Correlação: 0.9045
|- log_total_plays_ratio_ref_max_6 <-> completed_songs_rate_min_6
|   \ Correlação: 0.9647
|- completion_efficiency_min_6 <-> completed_songs_rate_min_6
|   \ Correlação: 0.9869
|- num_unq_mean_3 <-> log_total_secs_mean_3
|   \ Correlação: 0.9999
```

Correlação com Target

```
In [ ]: print("\n" + "="*80)
print("Correlação com Target (apenas train)")
print("="*80)

# Calcular correlação de Pearson para cada feature numérica
correlacoes = []

for col in features_numericas:
    corr_value = df_train.stat.corr(col, 'target_win')
    correlacoes.append({
        'feature': col,
        'correlacao': corr_value,
        'abs_correlacao': abs(corr_value)
    })

# Ordenar por correlação absoluta
correlacoes_sorted = sorted(correlacoes, key=lambda x: x['abs_correlacao'], reverse=True)

# Exibir top 20
print("\n📊 Top 20 Features por Correlação com Target:")
for i, item in enumerate(correlacoes_sorted[:20], 1):
    print(f" {i:2d}. {item['feature']}:40s | Corr: {item['correlacao']:.7f}")

# Filtrar features com correlação muito baixa
threshold_corr_target = 0.01
features_baixa_corr = [item['feature'] for item in correlacoes if item['abs_correlacao'] < threshold_corr_target]

print(f"\n⚠️ Features com |corr| < {threshold_corr_target}: {len(features_baixa_corr)}")
if features_baixa_corr:
    print(" Candidatas à remoção:")
    for feat in features_baixa_corr:
        print(f" - {feat}")


=====
Correlação com Target (apenas train)
=====
```

```
📊 Top 20 Features por Correlação com Target:
1. flag_plano_mensal_max_3 | Corr: 0.7025
2. flag_plano_mensal_lag_1 | Corr: 0.6521
3. flag_plano_mensal | Corr: 0.6506
4. daily_revenue_efficiency | Corr: 0.6487
5. flag_valid_fee | Corr: 0.6001
6. flag_has_transactions | Corr: 0.5968
7. flag_valid_fee_max_3 | Corr: 0.5934
8. flag_has_transactions_max_3 | Corr: 0.5904
9. flag_has_transactions_lag_1 | Corr: 0.5871
10. flag_valid_fee_lag_1 | Corr: 0.5867
11. payment_method_id | Corr: 0.5826
12. is_auto_renew | Corr: 0.5194
13. daily_revenue_efficiency_ratio_ref_max_3 | Corr: 0.5110
14. daily_revenue_efficiency_min_3 | Corr: 0.4655
15. total_secs | Corr: -0.2617
16. num_100 | Corr: -0.2599
17. total_plays | Corr: -0.2589
18. num_unq | Corr: -0.2516
19. revenue_per_hour_listened_cap | Corr: 0.2177
20. margem_liquida_mensal | Corr: 0.1996
```

```
⚠️ Features com |corr| < 0.01: 9
Candidatas à remoção:
- total_secs_ratio_ref_max_6
- num_unq_ratio_ref_max_6
- total_plays_ratio_ref_max_6
- avg_secs_per_unq_cap_min_6
- log_total_secs_ratio_ref_max_6
- plays_per_unq_cap_min_6
- log_total_plays_ratio_ref_max_6
- completion_efficiency_min_6
- completed_songs_rate_min_6
```

Conclusões por grupo de variável

Critérios de Decisão (em ordem de prioridade)

1. **Correlação com Target** (maior = melhor)
2. **Estabilidade/Robustez** (transformadas > raw, médias móveis > valores pontuais)
3. **Interpretabilidade** (quando empate técnico)
4. **Semântica de Negócio** (relevância para o case)

Grupo 1: Correlação Perfeita (corr = 1.0000)

Par	Correlação	**MANTER**	**REMOVER**	Justificativa
total_secs_ratio_ref_max_6 ↔ num_unq_ratio_ref_max_6	1.0000	total_secs_ratio_ref_max_6	num_unq_ratio_ref_max_6	Ambas são ratios (mesma estabilidade), mas total_secs tem maior peso na base do custo
total_secs_ratio_ref_max_6 ↔ total_plays_ratio_ref_max_6	1.0000	-	total_plays_ratio_ref_max_6	Redundante com a anterior
total_secs_ratio_ref_max_6 ↔ log_total_secs_ratio_ref_max_6	1.0000	log_total_secs_ratio_ref_max_6	total_secs_ratio_ref_max_6	**Log é mais estável** (comprime extremos)
total_secs_ratio_ref_max_6 ↔ log_total_plays_ratio_ref_max_6	1.0000	-	log_total_plays_ratio_ref_max_6	Já mantemos log_total_secs_ratio
log_total_plays_mean_3 ↔ log_total_secs_mean_3	1.0000	log_total_secs_mean_3	log_total_plays_mean_3	total_secs é base do custo
total_plays_mean_3 ↔ num_unq_mean_3	1.0000	num_unq_mean_3	total_plays_mean_3	num_unq é base do custo e menos volátil
plays_per_unq_cap_min_6 ↔ completed_songs_rate_min_6	1.0000	completed_songs_rate_min_6	plays_per_unq_cap_min_6	**Rate é mais interpretável** (% de conclusão)
actual_amount_paid ↔ plan_list_price	0.9963	actual_amount_paid	plan_list_price	Valor real > valor de tabela

Conclusão do Grupo 1:

8 pares com correlação perfeita ou quase perfeita (≥ 0.9963), indicando redundância total. A decisão priorizou transformações logarítmicas e variáveis de custo (`total_secs`, `num_unq`) que são fundamentais para o cálculo de margem. `log_total_secs_ratio_ref_max_6` foi mantido ao invés da versão raw, pois a transformação log reduz a sensibilidade a usuários extremos (heavy users). A escolha de `completed_songs_rate_min_6` sobre `plays_per_unq_cap_min_6` se justifica pela maior interpretabilidade semântica (taxa de conclusão é mais intuitiva que plays/unique).

Total de Remoções: 7 features

Grupo 2: Baixa Correlação com Target ($|corr| < 0.01$)

Feature	Corr com Target	Decisão	Justificativa
total_secs_ratio_ref_max_6	~0.00	**REMOVER**	Já removida no Grupo 1
num_unq_ratio_ref_max_6	~0.00	**REMOVER**	Já removida no Grupo 1
total_plays_ratio_ref_max_6	~0.00	**REMOVER**	Já removida no Grupo 1
avg_secs_per_unq_cap_min_6	~0.00	**REMOVER**	Sem relação com target
log_total_secs_ratio_ref_max_6	~0.00	**MANTER**	**Mantida no Grupo 1** (estabilidade)
plays_per_unq_cap_min_6	~0.00	**REMOVER**	Já removida no Grupo 1
log_total_plays_ratio_ref_max_6	~0.00	**REMOVER**	Já removida no Grupo 1
completion_efficiency_min_6	~0.00	**REMOVER**	Sem relação com target

Conclusão do Grupo 2:

Das 9 features com correlação absoluta inferior a 0.01, **8 foram removidas** por não apresentarem relação estatística com a target. A única exceção é `log_total_secs_ratio_ref_max_6`, que apesar da baixa correlação linear, foi **mantida estrategicamente** por ser uma transformação estável (log + ratio + janela de 6 meses) que pode capturar **relações não-lineares** em modelos de árvore. Esta decisão é conservadora: é preferível que o modelo (LightGBM/RF) decida se a feature é útil via feature importance, ao invés de descartá-la prematuramente. As demais features (ratios raw, médias de janelas curtas) foram removidas por serem redundantes ou instáveis.

Total de Remoções Adicionais: 2 features (avg_secs_per_unq_cap_min_6 , completion_efficiency_min_6)

Execução

```

In [202]: print("\n" + "*80")
print("REMOÇÕES MÍNIMAS (Apenas Redundância Total)")
print("*80")

# APENAS remover:
# 1. Correlação PERFEITA (1.0)
# 2. Correlação < 0.01 com target
features_to_remove_minimal = [
    # Correlação = 1.0 (literalmente idênticas)
    'num_unq_ratio_ref_max_6',           # Idêntica a total_secs_ratio_ref_max_6
    'total_plays_ratio_ref_max_6',        # Idêntica a total_secs_ratio_ref_max_6
    'log_total_secs_ratio_ref_max_6',     # Idêntica a total_secs_ratio_ref_max_6
    'log_total_plays_ratio_ref_max_6',    # Idêntica a total_secs_ratio_ref_max_6
    'log_total_secs_mean_3',              # Idêntica a log_total_plays_mean_3
    'num_unq_mean_3',                   # Idêntica a total_plays_mean_3
    'completed_songs_rate_min_6',         # Idêntica a plays_per_unq_cap_min_6
    'plan_list_price',                  # Quase idêntica a actual_amount_paid (0.9963)

    # Correlação < 0.01 com target (zero poder preditivo)
    'avg_secs_per_unq_cap_min_6',        # Corr ~0.00
    'completion_efficiency_min_6',       # Corr ~0.00
]

```

```

print(f"\n📊 Total de features a remover: {len(features_to_remove_minimal)}")
print(f"    |- Correlação (entre si) = 1.0: 8 features")
print(f"    |- Correlação (com target) < 0.01: 2 features")

```

```

# Aplicar remoção
df_tree_models = df_feature_store_trees_treated.drop(*features_to_remove_minimal)

```

```

print(f"\n✅ Features restantes: {len(df_tree_models.columns)}")
print(f"    |- Antes: {len(df_feature_store_trees_treated.columns)} colunas")
print(f"    |- Depois: {len(df_tree_models.columns)} colunas")

```

```

# Validar composição
features_controle = ['msno', 'target', 'target_win', 'safra', 'partition',
                      'membership_expire_date', 'transaction_date', 'registration_init_time']

```

```

features_categoricas = [
    'total_plays_group', 'completed_songs_rate_group', 'avg_secs_per_unq_cap_group',
    'plays_per_unq_behavior', 'plays_behavior_vs_volume', 'plays_behavior_vs_volume_collapsed',
    'plays_behavior_vs_completion', 'plays_behavior_vs_completion_collapsed',
    'early_drop_rate_group', 'revenue_tier', 'payment_method_group', 'payment_price_regime',
    'gender_clean', 'faixa_idade', 'registered_via_group', 'registration_year_regime',
    'tenure_faixa', 'revenue_per_hour_tier', 'usage_intensity_tier'
]

```

```

features_numericas_restantes = [
    col for col in df_tree_models.columns
    if col not in features_controle + features_categoricas
]

```

```

print(f"\n🔍 Composição Final:")
print(f"    |- Features Numéricas: {len(features_numericas_restantes)}")
print(f"    |- Features Categóricas: {len(features_categoricas)}")
print(f"    |- TOTAL ÚTIL: {len(features_numericas_restantes) + len(features_categoricas)}")

```

```

print("\n✅ PRONTO! Base com remoções mínimas")
print("Próximo passo: Treinar modelo e usar Feature Importance")

```

```

=====
REMOÇÕES MÍNIMAS (Apenas Redundância Total)
=====
```

```

📊 Total de features a remover: 10
    |- Correlação (entre si) = 1.0: 8 features
    |- Correlação (com target) < 0.01: 2 features

```

```

✅ Features restantes: 60
    |- Antes: 70 colunas
    |- Depois: 60 colunas

```

```

🔍 Composição Final:
    |- Features Numéricas: 33
    |- Features Categóricas: 19
    |- TOTAL ÚTIL: 52

```

```

✅ PRONTO! Base com remoções mínimas
Próximo passo: Treinar modelo e usar Feature Importance

```

ABT inicial - modelos baseados em árvores

Validações

Validando nulos

```
In [203]: print("\n" + "*80")
print("🔍 VALIDAÇÃO FINAL: Checando Nulls")
print("*80")

# Contar nulls por coluna
null_counts = df_tree_models.select([
    F.sum(F.when(F.col(c).isNull(), 1).otherwise(0)).alias(c)
    for c in df_tree_models.columns
]).collect()[0].asDict()

# Filtrar apenas colunas com nulls
cols_with_nulls = {k: v for k, v in null_counts.items() if v > 0}

if cols_with_nulls:
    print(f"\n⚠️ Encontradas {len(cols_with_nulls)} colunas com nulls:")
    for col, count in sorted(cols_with_nulls.items(), key=lambda x: x[1], reverse=True):
        pct = 100 * count / df_tree_models.count()
        print(f"    |- {col:40s}: {count:8d} nulls ({pct:5.2f}%)")

    # Decisão: nulls em features numéricas são OK para LightGBM/RF (tratam nativamente)
    # Nulls em categóricas precisam ser preenchidos
    nulls_categoricas = [col for col in cols_with_nulls.keys() if col in features_categoricas]

    if nulls_categoricas:
        print(f"\n⚠️ ATENÇÃO: {len(nulls_categoricas)} categóricas com nulls:")
        for col in nulls_categoricas:
            print(f"    |- {col}")
        print("\n💡 Recomendação: Preencher com categoria 'missing' ou 'unknown'")

    else:
        print("\n✅ Nenhuma coluna com nulls!")

=====

🔍 VALIDAÇÃO FINAL: Checando Nulls
=====

⚠️ Encontradas 2 colunas com nulls:
    |- membership_expire_date : 1539830 nulls (15.91%)
    |- transaction_date       : 1539830 nulls (15.91%)
```

Resultados esperados. São features que serão removidas.

Validando distribuição de partition

```
In [204]: print("\n" + "*80")
print("🌐 VALIDAÇÃO FINAL: Distribuição de Partition")
print("*80)

partition_dist = df_tree_models.groupBy('partition').agg(
    F.count('*').alias('count'),
    F.mean('target_win').alias('mean_target'),
    F.stddev('target_win').alias('std_target')
).orderBy('partition')

partition_dist.show()

# Validar que todas as partitions existem
partitions = [row['partition'] for row in partition_dist.collect()]
expected_partitions = ['train', 'test', 'oot']

missing_partitions = set(expected_partitions) - set(partitions)
if missing_partitions:
    print(f"\n⚠️ ERRO: Partitions faltando: {missing_partitions}")
else:
    print(f"\n✅ Todas as partitions presentes: {partitions}")
```

```
=====
🌐 VALIDAÇÃO FINAL: Distribuição de Partition
=====
+-----+-----+-----+
|partition| count| mean_target| std_target|
+-----+-----+-----+
|     oot|1850732|51.52032381700694|53.25559546983494|
|     test|1565303|43.73891320040854|60.54488958218142|
|    train|6262831|43.67598854048849|60.56790697572467|
+-----+-----+-----+
```

✓ Todas as partitions presentes: ['oot', 'test', 'train']

Validando target

```
In [205]: print("\n" + "*80")
print("VALIDAÇÃO FINAL: Target Variable")
print("*80")

# Checar se target_win existe
if 'target_win' not in df_tree_models.columns:
    print("⚠️ ERRO: target_win não encontrada!")
else:
    # Estatísticas da target
    target_stats = df_tree_models.select(
        F.count('target_win').alias('count'),
        F.sum(F.when(F.col('target_win').isNull(), 1).otherwise(0)).alias('nulls'),
        F.mean('target_win').alias('mean'),
        F.stddev('target_win').alias('std'),
        F.min('target_win').alias('min'),
        F.max('target_win').alias('max')
    ).collect()[0]

    print(f"\n📊 Estatísticas de target_win:")
    print(f"  |- Count: {target_stats['count']}:")
    print(f"  |- Nulls: {target_stats['nulls']}:")
    print(f"  |- Mean: {target_stats['mean']:.4f}")
    print(f"  |- Std: {target_stats['std']:.4f}")
    print(f"  |- Min: {target_stats['min']:.4f}")
    print(f"  |- Max: {target_stats['max']:.4f}")

    if target_stats['nulls'] > 0:
        print(f"\n⚠️ ATENÇÃO: {target_stats['nulls']} nulls na target!")
    else:
        print(f"\n✅ Target sem nulls!")
```

```
=====
VALIDAÇÃO FINAL: Target Variable
=====

📊 Estatísticas de target_win:
|- Count: 9,678,866
|- Nulls: 0
|- Mean: 45.1861
|- Std: 59.3157
|- Min: -108.8147
|- Max: 129.0401

✓ Target sem nulls!
```

Validando tipo de dados

```
In [206]: print("\n" + "*80")
print("VALIDAÇÃO FINAL: Tipos de Dados")
print("*80")

# Agrupar por tipo
tipos = {}
for field in df_tree_models.schema.fields:
    tipo = str(field.dataType)
    if tipo not in tipos:
        tipos[tipo] = []
    tipos[tipo].append(field.name)

print(f"\n📊 Distribuição de Tipos:")
for tipo, cols in sorted(tipos.items()):
    print(f"    | {tipo}: {len(cols)} colunas")

# Validar que numéricas são double/float/int
tipos_numericos = ['DoubleType', 'FloatType', 'IntegerType', 'LongType']
numericas_tipo_errado = []

for col in features_numericas_restantes:
    tipo_col = str([f.dataType for f in df_tree_models.schema.fields if f.name == col][0])
    if not any(t in tipo_col for t in tipos_numericos):
        numericas_tipo_errado.append((col, tipo_col))

if numericas_tipo_errado:
    print(f"\n⚠️ ATENÇÃO: {len(numericas_tipo_errado)} numéricas com tipo errado:")
    for col, tipo in numericas_tipo_errado:
        print(f"    - {col}: {tipo}")
else:
    print(f"\n✅ Todas as numéricas têm tipo correto!")

# Validar que categóricas são string
categoricas_tipo_errado = []
for col in features_categoricas:
    tipo_col = str([f.dataType for f in df_tree_models.schema.fields if f.name == col][0])
    if 'StringType' not in tipo_col:
        categoricas_tipo_errado.append((col, tipo_col))

if categoricas_tipo_errado:
    print(f"\n⚠️ ATENÇÃO: {len(categoricas_tipo_errado)} categóricas com tipo errado:")
    for col, tipo in categoricas_tipo_errado:
        print(f"    - {col}: {tipo}")
else:
    print(f"\n✅ Todas as categóricas têm tipo correto!")
```

```
=====
VALIDAÇÃO FINAL: Tipos de Dados
=====
```

```
📊 Distribuição de Tipos:
    | DateType()          : 3 colunas
    | DoubleType()         : 23 colunas
    | FloatType()          : 1 colunas
    | IntegerType()        : 12 colunas
    | StringType()         : 21 colunas
```

```
✅ Todas as numéricas têm tipo correto!
✅ Todas as categóricas têm tipo correto!
```

Salvando ABT inicial modelos de árvore

```
In [ ]: df_tree_models = df_tree_models.drop('membership_expire_date').drop('transaction_date').drop('registration_init_time')

In [208]: df_tree_models.count()

Out[208]: 9678866
```

In df_tree_models.printSchema()

[209]:

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- target: double (nullable = true)
 |-- target_win: double (nullable = true)
 |-- flag_valid_fee_lag_1: integer (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = false)
 |-- usage_intensity_per_tenure_cap: double (nullable = false)
 |-- daily_revenue_efficiency: double (nullable = false)
 |-- log_total_plays_mean_3: double (nullable = false)
 |-- flag_has_transactions_max_3: integer (nullable = true)
 |-- flag_plano_mensal: integer (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = false)
 |-- total_plays: double (nullable = false)
 |-- log_total_secs: double (nullable = false)
 |-- flag_valid_fee_max_3: integer (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = false)
 |-- total_plays_mean_3: double (nullable = false)
 |-- num_50: double (nullable = false)
 |-- margem_liquida_mensal: double (nullable = false)
 |-- num_75: double (nullable = false)
 |-- log_total_plays: double (nullable = false)
 |-- total_secs_mean_3: double (nullable = false)
 |-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = false)
 |-- total_secs: double (nullable = false)
 |-- flag_plano_mensal_lag_1: integer (nullable = true)
 |-- flag_has_transactions_lag_1: integer (nullable = true)
 |-- flag_valid_fee: integer (nullable = true)
 |-- flag_has_transactions: integer (nullable = true)
 |-- actual_amount_paid: float (nullable = false)
 |-- num_100: double (nullable = false)
 |-- num_25: double (nullable = false)
 |-- is_auto_renew: integer (nullable = true)
 |-- plays_per_unq_cap_min_6: double (nullable = false)
 |-- payment_method_id: integer (nullable = true)
 |-- flag_plano_mensal_max_3: integer (nullable = true)
 |-- num_985: double (nullable = false)
 |-- num_unq: double (nullable = false)
 |-- total_plays_group: string (nullable = true)
 |-- completed_songs_rate_group: string (nullable = true)
 |-- avg_secs_per_unq_cap_group: string (nullable = true)
 |-- plays_per_unq_behavior: string (nullable = true)
 |-- plays_behavior_vs_volume: string (nullable = true)
 |-- plays_behavior_vs_volume_collapsed: string (nullable = true)
 |-- plays_behavior_vs_completion: string (nullable = true)
 |-- plays_behavior_vs_completion_collapsed: string (nullable = true)
 |-- early_drop_rate_group: string (nullable = true)
 |-- revenue_tier: string (nullable = true)
 |-- payment_method_group: string (nullable = true)
 |-- payment_price_regime: string (nullable = true)
 |-- gender_clean: string (nullable = true)
 |-- faixa_idade: string (nullable = true)
 |-- registered_via_group: string (nullable = true)
 |-- registration_year_regime: string (nullable = true)
 |-- tenure_faixa: string (nullable = true)
 |-- revenue_per_hour_tier: string (nullable = true)
 |-- usage_intensity_tier: string (nullable = true)
 |-- partition: string (nullable = false)
```

```
In [210]: print("\n" + "*80")
print("SALVANDO FEATURE STORE")
print("*80)

# 1. cache
df_tree_models = df_tree_models.persist()
df_tree_models.count()

# 2. salvar particionado
df_tree_models.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_feature_store_tree_models")

print(f"\n✅ Feature Store salvo com sucesso!")
print(f"    |- Total de colunas: {len(df_tree_models.columns)}")
print(f"    |- Total de registros: {df_tree_models.count():,}")
```

```
=====
SALVANDO FEATURE STORE
=====

✅ Feature Store salvo com sucesso!
|- Total de colunas: 57
|- Total de registros: 9,678,866
```

12.6. Decisao por modelo: LightGBM

Carregando base

```
In [23]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"
df_feature_store_lightgbm = spark.read.parquet(silver_path + "df_feature_store_tree_models")
```

In [24]:

```
# =====
# BLOCO 1: SETUP E IMPORTS
# =====

from sklearn.model_selection import StratifiedKFold
import warnings
warnings.filterwarnings('ignore')

# -----
# JUSTIFICATIVA METODOLÓGICA - Definição das Features
# -----
# Separamos as features em 3 grupos para controle de pipeline:
# 1. Numéricas: já passaram por filtros estatísticos (variância, correlação)
# 2. Categóricas: features engenheiradas com agrupamentos semânticos
# 3. Controle: ID, participação e target
#
# Essa separação permite:
# - Tratamento diferenciado por tipo
# - Rastreabilidade de transformações
# - Facilita debugging e versionamento

features_num_tree = [
    'flag_valid_fee_lag_1', 'total_secs_ratio_ref_max_6',
    'usage_intensity_per_tenure_cap', 'daily_revenue_efficiency',
    'log_total_plays_mean_3', 'flag_has_transactions_max_3',
    'flag_plano_mensal', 'revenue_per_hour_listened_cap',
    'total_plays', 'log_total_secs', 'flag_valid_fee_max_3',
    'daily_revenue_efficiency_min_3', 'total_plays_mean_3',
    'num_50', 'margem_liquida_mensal', 'num_75',
    'log_total_plays', 'total_secs_mean_3',
    'daily_revenue_efficiency_ratio_ref_max_3', 'total_secs',
    'flag_plano_mensal_lag_1', 'flag_has_transactions_lag_1',
    'flag_valid_fee', 'flag_has_transactions',
    'actual_amount_paid', 'num_100', 'num_25',
    'is_auto_renew', 'plays_per_unq_cap_min_6',
    'payment_method_id', 'flag_plano_mensal_max_3',
    'num_985', 'num_unq'
]

features_cat_tree = [
    'total_plays_group', 'completed_songs_rate_group',
    'avg_secs_per_unq_cap_group', 'plays_per_unq_behavior',
    'plays_behavior_vs_volume', 'plays_behavior_vs_volume_collapsed',
    'plays_behavior_vs_completion', 'plays_behavior_vs_completion_collapsed',
    'early_drop_rate_group', 'revenue_tier',
    'payment_method_group', 'payment_price_regime',
    'gender_clean', 'faixa_idade', 'registered_via_group',
    'registration_year_regime', 'tenure_faixa',
    'revenue_per_hour_tier', 'usage_intensity_tier'
]

# Colunas de controle (não entram no modelo)
control_cols = ['msno', 'safra', 'partition']
target_col = 'target_win'

print("✅ Features definidas:")
print(f" - Numéricas: {len(features_num_tree)}")
print(f" - Categóricas: {len(features_cat_tree)}")
print(f" - Controle: {len(control_cols)}")
print(f" - Target: {target_col}")

    
```

```
    ✅ Features definidas:
    - Numéricas: 33
    - Categóricas: 19
    - Controle: 3
    - Target: target_win
```

Tratando numericas

Execucao

```
In [25]: # Dicionário para armazenar se a coluna tem negativos
dict_sentinelas = {}

# FAZEMOS UM RESUMO DE VALORES MÍNIMOS DE UMA VEZ SÓ (MAIS PERFORMÁTICO)
min_values = df_feature_store_lightgbm.select([F.min(c).alias(c) for c in features_num_tree]).collect()[0]

cols_com_sentinela = [c for c in features_num_tree if not c.startswith("margem_") and min_values[c] is not None and min_values[c] < 0]

print(f"Features que receberão tratamento de sentinela: {len(cols_com_sentinela)}:")
for col in cols_com_sentinela:
    print(f" - {col} (min: {min_values[col]})")
    dict_sentinelas[col] = min_values[col]
```

Features que receberão tratamento de sentinela: 7:

- total_secs_ratio_ref_max_6 (min: -99998.0)
- log_total_plays_mean_3 (min: -99998.0)
- daily_revenue_efficiency_min_3 (min: -99999.0)
- total_plays_mean_3 (min: -99998.0)
- total_secs_mean_3 (min: -99998.0)
- daily_revenue_efficiency_ratio_ref_max_3 (min: -99999.0)
- plays_per_unq_cap_min_6 (min: -99999.0)

```
In [26]: # =====
# BLOCO 2.2: CRIAÇÃO DAS FLAGS DE SENTINELA
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Antes de limpar os valores negativos, criamos flags binárias que preservam
# a informação de que "aquele dado tinha um estado especial".
# O LightGBM pode aprender que clientes com sentinela em determinada feature
# têm comportamento diferente (ex: nunca usaram o serviço, são novos, etc.)

# Criar todas as flags de uma vez (mais performático que loop)
for col in cols_com_sentinela:
    df_feature_store_lightgbm = df_feature_store_lightgbm.withColumn(f"{col}_is_sentinel",
        F.when(F.col(col) < 0, 1).otherwise(0).cast("byte")) # byte = int8 no PySpark

print(f"\n✓ Flags de sentinela criadas: {len(cols_com_sentinela)}")
print(f"  Colunas adicionadas: {[f'{c}_is_sentinel' for c in cols_com_sentinela[:3]]}...")
```

✓ Flags de sentinela criadas: 7
 Colunas adicionadas: ['total_secs_ratio_ref_max_6_is_sentinel', 'log_total_plays_mean_3_is_sentinel',
 'daily_revenue_efficiency_min_3_is_sentinel']...

```
In [27]: # =====
# BLOCO 2.3: SUBSTITUIÇÃO DE SENTINELAS POR NULL
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Ao transformar sentinelas em NULL, permitimos que o LightGBM use sua
# estratégia nativa de "missing value handling":
# - Em cada split, testa colocar NULLs à esquerda ou à direita
# - Escolhe automaticamente o que minimiza o erro
# - Muito superior a imputar com zero, média ou mediana
#
# ALTERNATIVAS DESCARTADAS:
# Manter valores negativos → distorce distribuição e splits
# Imputar com zero → assume que "sem dado" = "zero uso" (falso!)
# Imputar com média → introduz viés e perde informação

for col in cols_com_sentinela:
    df_feature_store_lightgbm = df_feature_store_lightgbm.withColumn(col,
        F.when(F.col(col) < 0, None).otherwise(F.col(col)))

print(f"\n✓ Sentinelas substituídas por NULL: {len(cols_com_sentinela)} colunas")
print(f"  Exemplo: valores < 0 em '{cols_com_sentinela[0]}' agora são NULL")
```

✓ Sentinelas substituídas por NULL: 7 colunas
 Exemplo: valores < 0 em 'total_secs_ratio_ref_max_6' agora são NULL

In [28]:

```
# =====
# BLOCO 2.4: OTIMIZAÇÃO DE TIPOS DE DADOS (FLAGS ORIGINAIS)
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Flags binárias (0/1) podem ser armazenadas como byte (int8) ao invés de
# int/long, economizando 4-8x de memória.
# Em ABTs com milhões de linhas, isso:
# - Reduz uso de memória do cluster
# - Acelera shuffle operations
# - Melhora performance do LightGBM (menos dados para ler)

# Identificar flags originais (que já nasceram como 0/1)
flags_originais = [col for col in features_num_tree if col.startswith('flag_') or col.startswith('is_')]

# Converter para byte (int8)
for col in flags_originais:
    df_feature_store_lightgbm = df_feature_store_lightgbm.withColumn(col, F.col(col).cast("byte"))

print(f"\n✓ Flags originais otimizadas: {len(flags_originais)} colunas")
print(f"    Tipo alterado: int/long → byte (economia de memória)")
print(f"    Exemplos: {flags_originais[:5]}")
```

✓ Flags originais otimizadas: 10 colunas
Tipo alterado: int/long → byte (economia de memória)
Exemplos: ['flag_valid_fee_lag_1', 'flag_has_transactions_max_3', 'flag_plano_mensal', 'flag_valid_fee_max_3',
'flag_plano_mensal_lag_1']

In [29]:

```
# =====
# BLOCO 2.5: VALIDAÇÃO DO TRATAMENTO DE NUMÉRICAS
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Validações são essenciais para garantir que:
# 1. Não criamos leakage acidentalmente
# 2. Não perdemos dados importantes
# 3. O pipeline está correto antes de seguir para encoding

print("\n" + "="*70)
print("RELATÓRIO FINAL - TRATAMENTO DE VARIÁVEIS NUMÉRICAS")
print("="*70)

# 1. Contar features finais
features_num_originais = len(features_num_tree)
features_flags_criadas = len(cols_com_sentinela)
features_flags_originais = len(flags_originais)

print(f"\n1. INVENTÁRIO DE FEATURES NUMÉRICAS:")
print(f"    - Features numéricas originais: {features_num_originais}")
print(f"    - Flags de sentinelas criadas: {features_flags_criadas}")
print(f"    - Flags originais otimizadas: {features_flags_originais}")
print(f"    - Total de features numéricas: {features_num_originais + features_flags_criadas}")

# 2. Verificar se ainda existem valores negativos (exceto margem_liquida_mensal)
print(f"\n2. VALIDAÇÃO DE SENTINELAS:")
cols_para_validar = [c for c in features_num_tree if not c.startswith("margem_")]
min_values_pos_tratamento = df_feature_store_lightgbm.select(
    [F.min(c).alias(c) for c in cols_para_validar]
).collect()[0]

cols_ainda_negativas = [c for c in cols_para_validar
                        if min_values_pos_tratamento[c] is not None
                        and min_values_pos_tratamento[c] < 0]

if len(cols_ainda_negativas) == 0:
    print(f"    ✅ Nenhuma coluna possui valores negativos (exceto target)")
else:
    print(f"    ⚠️ ATENÇÃO: {len(cols_ainda_negativas)} colunas ainda têm negativos:")
    for col in cols_ainda_negativas:
        print(f"        - {col}: {min_values_pos_tratamento[col]}")

# 3. Estatísticas de NULLS criados
print(f"\n3. ESTATÍSTICAS DE NULLS (pós-tratamento):")
null_counts = df_feature_store_lightgbm.select(
    [F.sum(F.when(F.col(c).isNull(), 1).otherwise(0)).alias(c)
     for c in cols_com_sentinela[:5]] # primeiras 5 para não poluir
).collect()[0]

for col in cols_com_sentinela[:5]:
    total = df_feature_store_lightgbm.count()
    n_nulls = null_counts[col]
    pct = 100 * n_nulls / total if total > 0 else 0
    print(f"    - {col}: {n_nulls:>8,} NULLs ({pct:.2f}%)")

if len(cols_com_sentinela) > 5:
    print(f"    ... (+ {len(cols_com_sentinela) - 5} colunas)")

print("\n" + "="*70)
print("✅ TRATAMENTO DE VARIÁVEIS NUMÉRICAS CONCLUÍDO")
print("="*70)
```

```
=====
RELATÓRIO FINAL - TRATAMENTO DE VARIÁVEIS NUMÉRICAS
=====

1. INVENTÁRIO DE FEATURES NUMÉRICAS:
    - Features numéricas originais: 33
    - Flags de sentinelas criadas: 7
    - Flags originais otimizadas: 10
    - Total de features numéricas: 40

2. VALIDAÇÃO DE SENTINELAS:
    ✅ Nenhuma coluna possui valores negativos (exceto target)

3. ESTATÍSTICAS DE NULLS (pós-tratamento):
    - total_secs_ratio_ref_max_6 : 2,200,868 NULLs (22.74%)
    - log_total_plays_mean_3 : 1,914,940 NULLs (19.78%)
    - daily_revenue_efficiency_min_3 : 2,314,344 NULLs (23.91%)
    - total_plays_mean_3 : 1,914,940 NULLs (19.78%)
    - total_secs_mean_3 : 1,914,940 NULLs (19.78%)
    ... (+ 2 colunas)

=====
```

✅ TRATAMENTO DE VARIÁVEIS NUMÉRICAS CONCLUÍDO

In [30]:

```
# =====
# BLOCO 2.6: ATUALIZAÇÃO DA LISTA DE FEATURES PARA PRÓXIMAS ETAPAS
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Precisamos atualizar a lista de features numéricas para incluir as flags
# criadas, pois elas farão parte do modelo final.

# Lista atualizada de features numéricas (originais + flags de sentinel)
features_num_tree_final = features_num_tree + [f"{c}_is_sentinel" for c in cols_com_sentinela]

print(f"📋 Lista de features numéricas atualizada:")
print(f"  - Total: {len(features_num_tree_final)} features")
print(f"  - Originais: {len(features_num_tree)}")
print(f"  - Flags criadas: {len(cols_com_sentinela)}")

# Salvar para uso posterior
print(f"\n💾 Variável 'features_num_tree_final' criada e pronta para uso")
```

📋 Lista de features numéricas atualizada:

- Total: 40 features
- Originais: 33
- Flags criadas: 7

💾 Variável 'features_num_tree_final' criada e pronta para uso

In [32]:

```
print(features_num_tree_final)
```

```
['flag_valid_fee_lag_1', 'total_secs_ratio_ref_max_6', 'usage_intensity_per_tenure_cap', 'daily_revenue_efficiency',
'log_total_plays_mean_3', 'flag_has_transactions_max_3', 'flag_plano_mensal', 'revenue_per_hour_listened_cap',
'total_plays', 'log_total_secs', 'flag_valid_fee_max_3', 'daily_revenue_efficiency_min_3', 'total_plays_mean_3', 'num_50',
'margem_liquida_mensal', 'num_75', 'log_total_plays', 'total_secs_mean_3', 'daily_revenue_efficiency_ratio_ref_max_3',
'total_secs', 'flag_plano_mensal_lag_1', 'flag_has_transactions_lag_1', 'flag_valid_fee', 'flag_has_transactions',
'actual_amount_paid', 'num_100', 'num_25', 'is_auto_renew', 'plays_per_unq_cap_min_6', 'payment_method_id',
'flag_plano_mensal_max_3', 'num_985', 'num_unq', 'total_secs_ratio_ref_max_6_is_sentinel',
'log_total_plays_mean_3_is_sentinel', 'daily_revenue_efficiency_min_3_is_sentinel', 'total_plays_mean_3_is_sentinel',
'total_secs_mean_3_is_sentinel', 'daily_revenue_efficiency_ratio_ref_max_3_is_sentinel',
'plays_per_unq_cap_min_6_is_sentinel']
```

Salvando intermediaaria

In [33]:

```
features_num_tree_final = ['flag_valid_fee_lag_1', 'total_secs_ratio_ref_max_6', 'usage_intensity_per_tenure_cap',
'daily_revenue_efficiency', 'log_total_plays_mean_3', 'flag_has_transactions_max_3', 'flag_plano_mensal',
'revenue_per_hour_listened_cap', 'total_plays', 'log_total_secs', 'flag_valid_fee_max_3', 'daily_revenue_efficiency_min_3',
'total_plays_mean_3', 'num_50', 'margem_liquida_mensal', 'num_75', 'log_total_plays', 'total_secs_mean_3',
'daily_revenue_efficiency_ratio_ref_max_3', 'total_secs', 'flag_plano_mensal_lag_1', 'flag_has_transactions_lag_1',
'flag_valid_fee', 'flag_has_transactions', 'actual_amount_paid', 'num_100', 'num_25', 'is_auto_renew', 'plays_per_unq_cap_min_6',
'payment_method_id', 'flag_plano_mensal_max_3', 'num_985', 'num_unq', 'total_secs_ratio_ref_max_6_is_sentinel',
'log_total_plays_mean_3_is_sentinel', 'daily_revenue_efficiency_min_3_is_sentinel', 'total_plays_mean_3_is_sentinel',
'total_secs_mean_3_is_sentinel', 'daily_revenue_efficiency_ratio_ref_max_3_is_sentinel', 'plays_per_unq_cap_min_6_is_sentinel']
```

In [12]:

```
df_feature_store_lightgbm_num = df_feature_store_lightgbm.select(control_cols + [target_col] + features_num_tree_final)
```

```
In [14]: df_feature_store_lightgbm_num.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- partition: string (nullable = true)
 |-- target_win: double (nullable = true)
 |-- flag_valid_fee_lag_1: byte (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- flag_has_transactions_max_3: byte (nullable = true)
 |-- flag_plano_mensal: byte (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- total_plays: double (nullable = true)
 |-- log_total_secs: double (nullable = true)
 |-- flag_valid_fee_max_3: byte (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- total_plays_mean_3: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- margem_liquida_mensal: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
 |-- total_secs: double (nullable = true)
 |-- flag_plano_mensal_lag_1: byte (nullable = true)
 |-- flag_has_transactions_lag_1: byte (nullable = true)
 |-- flag_valid_fee: byte (nullable = true)
 |-- flag_has_transactions: byte (nullable = true)
 |-- actual_amount_paid: float (nullable = true)
 |-- num_100: double (nullable = true)
 |-- num_25: double (nullable = true)
 |-- is_auto_renew: byte (nullable = true)
 |-- plays_per_unq_cap_min_6: double (nullable = true)
 |-- payment_method_id: integer (nullable = true)
 |-- flag_plano_mensal_max_3: byte (nullable = true)
 |-- num_985: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_secs_ratio_ref_max_6_is_sentinel: byte (nullable = false)
 |-- log_total_plays_mean_3_is_sentinel: byte (nullable = false)
 |-- daily_revenue_efficiency_min_3_is_sentinel: byte (nullable = false)
 |-- total_plays_mean_3_is_sentinel: byte (nullable = false)
 |-- total_secs_mean_3_is_sentinel: byte (nullable = false)
 |-- daily_revenue_efficiency_ratio_ref_max_3_is_sentinel: byte (nullable = false)
 |-- plays_per_unq_cap_min_6_is_sentinel: byte (nullable = false)
```

```
In [15]: print("\n" + "="*80)
print("SALVANDO LIGHT GMB INTERMEDIARIA - SOMENTE NUMERICAS")
print("=*80")

# 1. cache
df_feature_store_lightgbm_num = df_feature_store_lightgbm_num.persist()
df_feature_store_lightgbm_num.count()

# 2. salvar particionado
df_feature_store_lightgbm_num.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_feature_store_lightgbm_num")

print(f"\n✅ Feature Store salvo com sucesso!")
print(f"  |- Total de colunas: {len(df_feature_store_lightgbm_num.columns)}")
print(f"  |- Total de registros: {df_feature_store_lightgbm_num.count():,}")
```

```
=====
SALVANDO LIGHT GMB INTERMEDIARIA - SOMENTE NUMERICAS
=====

✅ Feature Store salvo com sucesso!
  |- Total de colunas: 44
  |- Total de registros: 9,678,866
```

Tratando categoricas

```
#### Execucao
```

```
In [16]: df_feature_store_lightgbm = df_feature_store_lightgbm.drop("plays_behavior_vs_volume").drop("plays_behavior_vs_completion")
features_num_tree_final = [col for col in features_num_tree_final if col not in ["plays_behavior_vs_volume",
"plays_behavior_vs_completion"]]
features_cat_tree = [col for col in features_cat_tree if col not in ["plays_behavior_vs_volume", "plays_behavior_vs_completion"]]
```

```
In [17]: # =====
# BLOCO 3.1: ANÁLISE DE CARDINALIDADE DAS CATEGÓRICAS
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Antes de aplicar qualquer encoding, precisamos entender:
# 1. Cardinalidade (quantas categorias únicas existem)
# 2. Distribuição (categorias dominantes vs raras)
# 3. Categorias com baixa frequência (candidatas a colapso)
#
# Isso permite decisões informadas sobre:
# - Quais features precisam de tratamento de raras
# - Se alguma feature tem cardinalidade explosiva (ex: IDs vazados)
# - Validar que as features fazem sentido para o negócio

print("\n" + "*90)
print("ANÁLISE DE CARDINALIDADE - FEATURES CATEGÓRICAS")
print("*90)
print(f'{Feature':<40} | {'Cardinality':>15} | {'Top 3 Categorias'}")
print("-"*90)

# Dicionário para armazenar metadados
cat_metadata = {}

for col in features_cat_tree:
    # Contar categorias únicas
    n_unique = df_feature_store_lightgbm.select(col).distinct().count()

    # Top 3 categorias mais frequentes
    top_3 = df_feature_store_lightgbm.groupBy(col).count() \
        .orderBy(F.desc("count")) \
        .limit(3) \
        .select(col) \
        .rdd.flatMap(lambda x: x).collect()

    top_3_str = ', '.join([str(x)[:10] for x in top_3])

    # Armazenar metadados
    cat_metadata[col] = {
        'cardinality': n_unique,
        'top_3': top_3
    }

    print(f'{col:<40} | {n_unique:>15,} | {top_3_str}')

print("*90 + "\n")

# DECISÃO: Features com cardinalidade > 100 merecem atenção especial
high_cardinality = [col for col, meta in cat_metadata.items() if meta['cardinality'] > 100]

if high_cardinality:
    print(f"⚠️ Features com alta cardinalidade (>100): {len(high_cardinality)}")
    for col in high_cardinality:
        print(f" - {col}: {cat_metadata[col]['cardinality']} categorias")
    print()
```

```
=====
ANÁLISE DE CARDINALIDADE - FEATURES CATEGÓRICAS
=====
Feature | Cardinalidade | Top 3 Categorias
-----
total_plays_group | 5 | 04_power_u, 03_frequen, 02_regular
completed_songs_rate_group | 5 | 03_engaged, 04_complet, 02_skippin
avg_secs_per_unq_cap_group | 5 | 02_sampler, 03.Focused, 04_deep_li
plays_per_unq_behavior | 5 | 01_explore, 02_light_r, 00_unknown
plays_behavior_vs_volume_collapsed | 14 | 01_explore, 01_explore, 01_explore
plays_behavior_vs_completion_collapsed | 13 | 01_explore, 01_explore, 01_explore
early_drop_rate_group | 5 | 02_engaged, 03_explore, 00_unknown
revenue_tier | 7 | 04_premium, 03_standar, 00_unknown
payment_method_group | 4 | 01_most_au, 00_no_tran, 03_most_ma
payment_price_regime | 3 | 01_paid_as, 00_no_tran, 02_irregul
gender_clean | 3 | unknown, male, female
faixa_idade | 5 | Desconheci, 23-34, 35-52
registered_via_group | 4 | low_value, high_value, mid_value
registration_year_regime | 3 | 2010-2014, 2015+, 2004-2009
tenure_faixa | 3 | 03_36+mont, 01_0-11mon, 02_12-35mo
revenue_per_hour_tier | 4 | 00_unknown, 02_medium, 03_high
usage_intensity_tier | 5 | 04_power_u, 03_high_en, 02_moderat
=====
```

In [18]:

```
# =====
# BLOCO 3.2: IDENTIFICAÇÃO DE CATEGORIAS RARAS (FIT NO TREINO, PARA TARGET ENCODING)
# =====

# JUSTIFICATIVA METODOLÓGICA:
# PROBLEMA:
# Categorias com baixa frequência (<0.5% da base) geram:
# - Estimativas instáveis no target encoding (poucos exemplos)
# - Overfitting (modelo memoriza casos raros)
# - Generalização ruim em test/OOT
#
# DECISÃO:
# Threshold de 0.5% é conservador e padrão na indústria.
# - Preserva informação relevante
# - Remove ruído estatístico
# - Melhora estabilidade do encoding
#
# CRÍTICO: Calcular frequências APENAS NO TREINO
# - Evita leakage de informação de test/OOT
# - Garante que decisões são baseadas apenas em dados de treino

# Filtrar apenas treino
df_train = df_feature_store_lightgbm.filter(F.col("partition") == "train")
total_train = df_train.count()

print("\n" + "*90)
print("🔍 IDENTIFICAÇÃO DE CATEGORIAS RARAS (threshold: 0.5%)")
print("*90)
print(f"Base de treino: {total_train:,} registros")
print(f"Threshold absoluto: {int(total_train * 0.005):,} registros\n")

# Dicionário para armazenar categorias raras por feature
rare_categories_map = {}

for col in features_cat_tree:
    # Calcular frequências no treino
    freq_df = df_train.groupBy(col).count() \
        .withColumn("freq_pct", 100 * F.col("count") / total_train)

    # Identificar categorias raras (<0.5%)
    rare_cats = freq_df.filter(F.col("freq_pct") < 0.5) \
        .select(col) \
        .rdd.flatMap(lambda x: x).collect()

    if len(rare_cats) > 0:
        rare_categories_map[col] = rare_cats
        print(f"{col}: {len(rare_cats)} categorias raras")

print("*90 + "\n)

print(f"✅ Features com categorias raras: {len(rare_categories_map)}/{len(features_cat_tree)})
```

```
=====
🔍 IDENTIFICAÇÃO DE CATEGORIAS RARAS (threshold: 0.5%)
=====
Base de treino: 6,262,831 registros
Threshold absoluto: 31,314 registros

revenue_tier | 1 categorias raras
registered_via_group | 1 categorias raras
=====

✅ Features com categorias raras: 2/17
```

In [19]:

```
# =====
# BLOCO 3.3: COLAPSO DE CATEGORIAS RARAS EM '__RARE__'
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Ao agrupar categorias raras em um único bucket '__RARE__':
# ✓ Aumentamos a amostra para estimativa do target encoding
# ✓ Reduzimos overfitting
# ✓ Melhoramos generalização
# ✓ Mantemos sinal de "categoria incomum" (vs remover)
#
# APLICAÇÃO:
# - Colapso aplicado em TODO o dataset (train + test + oot)
# - Mas decisão de "o que é raro" vem APENAS do treino
# - Isso garante consistência temporal e evita leakage

from pyspark.sql.functions import when, col

print("\n" + "*70)
print("🔧 APLICANDO COLAPSO DE CATEGORIAS RARAS")
print("*70 + "\n")

for feature, rare_cats in rare_categories_map.items():
    # Criar expressão de colapso
    df_feature_store_lightgbm = df_feature_store_lightgbm.withColumn(
        feature,
        when(col(feature).isin(rare_cats), "__RARE__")
        .otherwise(col(feature))
    )

    print(f"✓ {feature}:<40} | {len(rare_cats):>3} categorias → '__RARE__'")

print("\n" + "*70)
print(f"✓ COLAPSO CONCLUÍDO: {len(rare_categories_map)} features tratadas")
print("*70 + "\n")
```

```
=====
🔧 APLICANDO COLAPSO DE CATEGORIAS RARAS
=====

✓ revenue_tier | 1 categorias → '__RARE__'
✓ registered_via_group | 1 categorias → '__RARE__'

=====
✓ COLAPSO CONCLUÍDO: 2 features tratadas
=====
```

In [20]:

```
# =====
# BLOCO 3.4: VALIDAÇÃO PÓS-COLAPSO
# =====

# JUSTIFICATIVA METODOLÓGICA:
# Validar que o colapso funcionou corretamente:
# 1. Cardinalidade reduziu conforme esperado
# 2. Categoria '__RARE__' foi criada onde necessário
# 3. Nenhuma categoria foi perdida accidentalmente

print("\n" + "="*90)
print("📊 VALIDAÇÃO PÓS-COLAPSO")
print("*"*90)
print(f"{'Feature':<40} | {'Card. Antes':>12} | {'Card. Depois':>12} | {'Redução'}")
print("-"*90)

for col in features_cat_tree:
    # Cardinalidade atual
    n_unique_after = df_feature_store_lightgbm.select(col).distinct().count()
    n_unique_before = cat_metadata[col]['cardinality']

    reduction = n_unique_before - n_unique_after

    if reduction > 0:
        print(f"{col:<40} | {n_unique_before:>12,} | {n_unique_after:>12,} | {-{reduction}}")

print("*"*90 + "\n")

print("✅ Validação concluída: categorias raras colapsadas com sucesso\n")
```

```
=====
📊 VALIDAÇÃO PÓS-COLAPSO
=====
Feature           | Card. Antes | Card. Depois | Redução
-----
=====
```

✅ Validação concluída: categorias raras colapsadas com sucesso

In [23]: features_cat_tree

```
Out[23]: ['total_plays_group',
 'completed_songs_rate_group',
 'avg_secs_per_unq_cap_group',
 'plays_per_unq_behavior',
 'plays_behavior_vs_volume_collapsed',
 'plays_behavior_vs_completion_collapsed',
 'early_drop_rate_group',
 'revenue_tier',
 'payment_method_group',
 'payment_price_regime',
 'gender_clean',
 'faixa_idade',
 'registered_via_group',
 'registration_year_regime',
 'tenure_faixa',
 'revenue_per_hour_tier',
 'usage_intensity_tier']
```

In [4]: features_cat_tree = [

```
'total_plays_group',
 'completed_songs_rate_group',
 'avg_secs_per_unq_cap_group',
 'plays_per_unq_behavior',
 'plays_behavior_vs_volume_collapsed',
 'plays_behavior_vs_completion_collapsed',
 'early_drop_rate_group',
 'revenue_tier',
 'payment_method_group',
 'payment_price_regime',
 'gender_clean',
 'faixa_idade',
 'registered_via_group',
 'registration_year_regime',
 'tenure_faixa',
 'revenue_per_hour_tier',
 'usage_intensity_tier'
```

]

Salvando intermediaria

```
In [21]: df_feature_store_lightgbm_cat = df_feature_store_lightgbm.select(control_cols + [target_col] + features_cat_tree)
```

```
In [22]: df_feature_store_lightgbm_cat.printSchema()
```

```
root
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- partition: string (nullable = true)
|-- target_win: double (nullable = true)
|-- total_plays_group: string (nullable = true)
|-- completed_songs_rate_group: string (nullable = true)
|-- avg_secs_per_unq_cap_group: string (nullable = true)
|-- plays_per_unq_behavior: string (nullable = true)
|-- plays_behavior_vs_volume_collapsed: string (nullable = true)
|-- plays_behavior_vs_completion_collapsed: string (nullable = true)
|-- early_drop_rate_group: string (nullable = true)
|-- revenue_tier: string (nullable = true)
|-- payment_method_group: string (nullable = true)
|-- payment_price_regime: string (nullable = true)
|-- gender_clean: string (nullable = true)
|-- faixa_idade: string (nullable = true)
|-- registered_via_group: string (nullable = true)
|-- registration_year_regime: string (nullable = true)
|-- tenure_faixa: string (nullable = true)
|-- revenue_per_hour_tier: string (nullable = true)
|-- usage_intensity_tier: string (nullable = true)
```

```
In [23]: df_feature_store_lightgbm_cat.groupBy("revenue_tier").count().show()
```

revenue_tier	count
01_free_isencao	81869
04_premium_149	4706207
06_others	513240
RARE	21590
00_unknown	1539830
03_standard_99	2432782
05_high_tier	383348

```
In [24]: print("\n" + "="*80)
print("SALVANDO LIGHT GMB INTERMEDIARIA - SOMENTE CATEGORICAS")
print("=*80)

# 1. cache
df_feature_store_lightgbm_cat = df_feature_store_lightgbm_cat.persist()
df_feature_store_lightgbm_cat.count()

# 2. salvar particionado
df_feature_store_lightgbm_cat.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_feature_store_lightgbm_cat")

print(f"\n\n ✅ Feature Store salvo com sucesso!")
print(f"    |- Total de colunas: {len(df_feature_store_lightgbm_cat.columns)}")
print(f"    |- Total de registros: {df_feature_store_lightgbm_cat.count():,}")
```

```
=====
SALVANDO LIGHT GMB INTERMEDIARIA - SOMENTE CATEGORICAS
=====

✅ Feature Store salvo com sucesso!
|- Total de colunas: 21
|- Total de registros: 9,678,866
```

Target Encoding com K-Fold + Smoothing

```
In [51]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"
df_feature_store_lightgbm_cat = spark.read.parquet(silver_path + "df_feature_store_lightgbm_cat")

df_base = df_feature_store_lightgbm_cat

cols_te_base = ["msno", "safra", "target_win"] + features_cat_tree

df_train_te = (df_base
    .filter(F.col("partition") == "train")
    .select(cols_te_base))

print(f"✓ Base de treino carregada: {df_train_te.count():,} linhas")
print(f"✓ Colunas: {df_train_te.columns}")
```

✓ Base de treino carregada: 6,262,831 linhas
✓ Colunas: ['msno', 'safra', 'target_win', 'total_plays_group', 'completed_songs_rate_group', 'avg_secs_per_unq_cap_group', 'plays_per_unq_behavior', 'plays_behavior_vs_volume_collapsed', 'plays_behavior_vs_completion_collapsed', 'early_drop_rate_group', 'revenue_tier', 'payment_method_group', 'payment_price_regime', 'gender_clean', 'faixa_idade', 'registered_via_group', 'registration_year_regime', 'tenure_faixa', 'revenue_per_hour_tier', 'usage_intensity_tier']

Criando fold_id com base nos n_folds

```
In [52]: w = Window.orderBy(F.rand(seed=42))
n_folds = 5

df_train_te = (df_train_te
    .withColumn("row_id_tmp", F.row_number().over(w))
    .withColumn("fold_id", (F.col("row_id_tmp") % n_folds).cast("int"))
    .drop("row_id_tmp"))

print("\n📊 Distribuição dos folds:")
df_train_te.groupBy("fold_id").count().orderBy("fold_id").show()
```

📊 Distribuição dos folds:

fold_id	count
0	1252566
1	1252567
2	1252566
3	1252566
4	1252566

Target Encoding leave-one-fold-out

In [54]:

```
def run_te_single_feature(df_train_te, cat_col, alpha=10):
    """
    Processa TE para UMA feature e retorna o DF com a coluna *_te adicionada
    """
    print(f"\n  ↗ Processando TE para: {cat_col}")

    global_mean = df_train_te.select(F.mean("target_win")).collect()[0][0]

    stats_fold = (
        df_train_te
        .groupBy("fold_id", cat_col)
        .agg(
            F.count("*").alias("cnt"),
            F.sum("target_win").alias("sum_target")
        )
    )

    stats_te = (
        stats_fold
        .groupBy(cat_col)
        .agg(
            F.sum("cnt").alias("total_cnt"),
            F.sum("sum_target").alias("total_sum")
        )
        .join(stats_fold, cat_col, "left")
        .withColumn("cnt_others", F.col("total_cnt") - F.col("cnt"))
        .withColumn("sum_others", F.col("total_sum") - F.col("sum_target"))
        .withColumn(
            "mean_others",
            F.when(F.col("cnt_others") > 0,
                   F.col("sum_others") / F.col("cnt_others"))
            .otherwise(global_mean)
        )
        .withColumn(
            f"{cat_col}_te",
            (F.col("cnt_others") * F.col("mean_others") +
             alpha * global_mean) /
            (F.col("cnt_others") + alpha)
        )
        .select("fold_id", cat_col, f"{cat_col}_te")
    )

    result_df = df_train_te.join(
        stats_te,
        on=["fold_id", cat_col],
        how="left"
    )

    print(f" ✓ {cat_col} concluido")
    return result_df
```

In [55]:

```
# Criar DF base limpo
df_te_master = df_train_te

# Processar feature por feature
for idx, cat_col in enumerate(features_cat_tree, start=1):
    print(f"\n{'='*70}")
    print(f"🚀 Feature {idx}/{len(features_cat_tree)}: {cat_col}")
    print(f"{'='*70}")

    # Processar TE
    df_te_master = run_te_single_feature(df_te_master, cat_col, alpha=10)

    # SALVAR INTERMEDIÁRIO (a cada 3 features ou no final)
    if idx % 3 == 0 or idx == len(features_cat_tree):
        temp_path = silver_path + f"te_checkpoint_{idx}"
        df_te_master.write.mode("overwrite").parquet(temp_path)
        print(f"💾 Checkpoint salvo → te_checkpoint_{idx}")

    # RECARREGAR (quebra o DAG)
    df_te_master = spark.read.parquet(temp_path)
    print(f"🔄 DF recarregado (DAG resetado)")

print("\n🎉 TODAS AS FEATURES TE CONCLUÍDAS")
```

```
=====
🚀 Feature 1/17: total_plays_group
=====

🔧 Processando TE para: total_plays_group
✓ total_plays_group concluído
=====

🚀 Feature 2/17: completed_songs_rate_group
=====

🔧 Processando TE para: completed_songs_rate_group
✓ completed_songs_rate_group concluído
=====

🚀 Feature 3/17: avg_secs_per_unq_cap_group
=====

🔧 Processando TE para: avg_secs_per_unq_cap_group
✓ avg_secs_per_unq_cap_group concluído
💾 Checkpoint salvo → te_checkpoint_3
🔄 DF recarregado (DAG resetado)
=====

🚀 Feature 4/17: plays_per_unq_behavior
=====

🔧 Processando TE para: plays_per_unq_behavior
✓ plays_per_unq_behavior concluído
=====

🚀 Feature 5/17: plays_behavior_vs_volume_collapsed
=====

🔧 Processando TE para: plays_behavior_vs_volume_collapsed
✓ plays_behavior_vs_volume_collapsed concluído
=====

🚀 Feature 6/17: plays_behavior_vs_completion_collapsed
=====

🔧 Processando TE para: plays_behavior_vs_completion_collapsed
✓ plays_behavior_vs_completion_collapsed concluído
💾 Checkpoint salvo → te_checkpoint_6
🔄 DF recarregado (DAG resetado)
=====

🚀 Feature 7/17: early_drop_rate_group
=====

🔧 Processando TE para: early_drop_rate_group
✓ early_drop_rate_group concluído
=====

🚀 Feature 8/17: revenue_tier
=====

🔧 Processando TE para: revenue_tier
✓ revenue_tier concluído
=====

🚀 Feature 9/17: payment_method_group
=====
```

```
=====
    ✎ Processando TE para: payment_method_group
    ✓ payment_method_group concluído
    📁 Checkpoint salvo → te_checkpoint_9
    ⚙️ DF recarregado (DAG resetado)

=====
    🚀 Feature 10/17: payment_price_regime
=====

    ✎ Processando TE para: payment_price_regime
    ✓ payment_price_regime concluído

=====
    🚀 Feature 11/17: gender_clean
=====

    ✎ Processando TE para: gender_clean
    ✓ gender_clean concluído

=====
    🚀 Feature 12/17: faixa_idade
=====

    ✎ Processando TE para: faixa_idade
    ✓ faixa_idade concluído
    📁 Checkpoint salvo → te_checkpoint_12
    ⚙️ DF recarregado (DAG resetado)

=====
    🚀 Feature 13/17: registered_via_group
=====

    ✎ Processando TE para: registered_via_group
    ✓ registered_via_group concluído

=====
    🚀 Feature 14/17: registration_year_regime
=====

    ✎ Processando TE para: registration_year_regime
    ✓ registration_year_regime concluído

=====
    🚀 Feature 15/17: tenure_faixa
=====

    ✎ Processando TE para: tenure_faixa
    ✓ tenure_faixa concluído
    📁 Checkpoint salvo → te_checkpoint_15
    ⚙️ DF recarregado (DAG resetado)

=====
    🚀 Feature 16/17: revenue_per_hour_tier
=====

    ✎ Processando TE para: revenue_per_hour_tier
    ✓ revenue_per_hour_tier concluído

=====
    🚀 Feature 17/17: usage_intensity_tier
=====

    ✎ Processando TE para: usage_intensity_tier
    ✓ usage_intensity_tier concluído
    📁 Checkpoint salvo → te_checkpoint_17
    ⚙️ DF recarregado (DAG resetado)

🎉 TODAS AS FEATURES TE CONCLUÍDAS
```

In [56]:

```
# MUDANÇA: Selecionar msno + safra + colunas TE
te_cols = [c for c in df_te_master.columns if c.endswith("_te")]
df_te_final = df_te_master.select("msno", "safra", *te_cols)

print(f"\n📊 Dataset TE final:")
print(f"  Linhas: {df_te_final.count()};")
print(f"  Colunas: {len(df_te_final.columns)}")

# Validar distribuição por safra
print("\n📊 Distribuição por safra:")
df_te_final.groupBy("safra").count().orderBy("safra").show()

# Salvar
df_te_final.write.mode("overwrite").parquet(silver_path + "df_lightgbm_cat_te_all_features")
print("💾 Dataset TE final salvo → df_lightgbm_cat_te_all_features")

# Mostrar schema
print("\n📋 Schema do dataset TE:")
df_te_final.printSchema()

# Amostra
print("\n📊 Amostra do dataset TE:")
df_te_final.show(5, truncate=False)
```

```
📊 Dataset TE final:
Linhas: 6,262,831
Colunas: 19
```

```
📊 Distribuição por safra:
```

```
+-----+
| safra| count|
+-----+
|201601|688844|
|201602|717184|
|201603|662780|
|201604|648999|
|201605|651984|
|201606|654023|
|201607|739638|
|201608|746578|
|201609|752801|
+-----+
```

```
💾 Dataset TE final salvo → df_lightgbm_cat_te_all_features
```

```
📋 Schema do dataset TE:
```

```
root
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- total_plays_group_te: double (nullable = true)
|-- completed_songs_rate_group_te: double (nullable = true)
|-- avg_secs_per_unq_cap_group_te: double (nullable = true)
|-- plays_per_unq_behavior_te: double (nullable = true)
|-- plays_behavior_vs_volume_collapsed_te: double (nullable = true)
|-- plays_behavior_vs_completion_collapsed_te: double (nullable = true)
|-- early_drop_rate_group_te: double (nullable = true)
|-- revenue_tier_te: double (nullable = true)
|-- payment_method_group_te: double (nullable = true)
|-- payment_price_regime_te: double (nullable = true)
|-- gender_clean_te: double (nullable = true)
|-- faixa_idade_te: double (nullable = true)
|-- registered_via_group_te: double (nullable = true)
|-- registration_year_regime_te: double (nullable = true)
|-- tenure_faixa_te: double (nullable = true)
|-- revenue_per_hour_tier_te: double (nullable = true)
|-- usage_intensity_tier_te: double (nullable = true)
```

```
📊 Amostra do dataset TE:
```

```
+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
|msno |safra |
|total_plays_group_te|completed_songs_rate_group_te|avg_secs_per_unq_cap_group_te|plays_per_unq_behavior_te|plays_behavior_v
s_volume_collapsed_te|plays_behavior_vs_completion_collapsed_te|early_drop_rate_group_te|revenue_tier_te
|payment_method_group_te|payment_price_regime_te|gender_clean_te |faixa_idade_te
|registered_via_group_te|registration_year_regime_te|tenure_faixa_te |revenue_per_hour_tier_te|usage_intensity_tier_te|
+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
|KkUHsGypgI88XAXBwWZsCAFpvszX8qrQPch38aCXqzE=|201601|20.698556499413183 |48.619080853108315 |47.72015575727324
|44.07503682664903 |24.990969635632954 |48.4398545776617 |50.08518545543927
|70.36391828144397|62.83727190051517 |60.77718186190875 |38.369426198247474|41.098819372930784|54.672975180467155
|45.9191343612376 |47.560285288453386|10.373271960810518 |40.92042671887222 |
```

```

|TIGibAR8kVLWeFxxPBBITtSEtG06z9QikdhHiIAqQo=|201601|20.698556499413183 |48.619080853108315 |47.72015575727324
|44.07503682664903 |24.990969635632954 |48.4398545776617 |50.08518545543927
|70.36391828144397|62.83727190051517 |60.77718186190875 |36.222904520457355|41.098819372930784|54.672975180467155
|45.91911343612376 |47.560285288453386|10.373271960810518 |40.92042671887222 |
|K8gP+SAw29f60ss7uW3Jf2ccMntpTrrlsS2VKO/Ee=M=|201601|20.698556499413183 |48.619080853108315 |47.72015575727324
|44.07503682664903 |24.990969635632954 |48.4398545776617 |50.08518545543927
|70.36391828144397|62.83727190051517 |60.77718186190875 |38.369426198247474|41.098819372930784|54.672975180467155
|45.91911343612376 |47.560285288453386|10.373271960810518 |40.92042671887222 |
|FQLfztCEH4K4LSofOdoTKJP0FjdXkcC5GTyTomUiRis=|201601|20.698556499413183 |48.619080853108315 |47.72015575727324
|44.07503682664903 |24.990969635632954 |48.4398545776617 |50.08518545543927
|70.36391828144397|62.83727190051517 |60.77718186190875 |38.369426198247474|41.098819372930784|54.672975180467155
|45.91911343612376 |47.560285288453386|10.373271960810518 |40.92042671887222 |
|RYPvfIy7I2Mk3ecRZ0W/5eBooqSzE9iC85K9TBZ/kS4=|201601|20.698556499413183 |48.619080853108315 |47.72015575727324
|44.07503682664903 |24.990969635632954 |48.4398545776617 |50.08518545543927
|70.36391828144397|62.83727190051517 |60.77718186190875 |38.369426198247474|41.098819372930784|54.672975180467155
|45.91911343612376 |47.560285288453386|10.373271960810518 |40.92042671887222 |
+-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+-----+-----+
only showing top 5 rows

```

Base ABT final para LightGBM

Execucao

```
In [34]: features_cat_tree = ['total_plays_group', 'completed_songs_rate_group', 'avg_secs_per_unq_cap_group', 'plays_per_unq_behavior',
'plays_behavior_vs_volume_collapsed', 'plays_behavior_vs_completion_collapsed', 'early_drop_rate_group', 'revenue_tier',
'payment_method_group', 'payment_price_regime', 'gender_clean', 'faixa_idade', 'registered_via_group', 'registration_year_regime',
'tenure_faixa', 'revenue_per_hour_tier', 'usage_intensity_tier']
features_num_tree_final = ['flag_valid_fee_lag_1', 'total_secs_ratio_ref_max_6', 'usage_intensity_per_tenure_cap',
'daily_revenue_efficiency', 'log_total_plays_mean_3', 'flag_has_transactions_max_3', 'flag_plano_mensal',
'revenue_per_hour_listened_cap', 'total_plays', 'log_total_secs', 'flag_valid_fee_max_3', 'daily_revenue_efficiency_min_3',
'total_plays_mean_3', 'num_50', 'margemliquida_mensal', 'num_75', 'log_total_plays', 'total_secs_mean_3',
'daily_revenue_efficiency_ratio_ref_max_3', 'total_secs', 'flag_plano_mensal_lag_1', 'flag_has_transactions_lag_1',
'flag_valid_fee', 'flag_has_transactions', 'actual_amount_paid', 'num_100', 'num_25', 'is_auto_renew', 'plays_per_unq_cap_min_6',
'payment_method_id', 'flag_plano_mensal_max_3', 'num_985', 'num_unq', 'total_secs_ratio_ref_max_6_is_sentinel',
'log_total_plays_mean_3_is_sentinel', 'daily_revenue_efficiency_min_3_is_sentinel', 'total_plays_mean_3_is_sentinel',
'total_secs_mean_3_is_sentinel', 'daily_revenue_efficiency_ratio_ref_max_3_is_sentinel', 'plays_per_unq_cap_min_6_is_sentinel']
```

```
In [57]: print("\n" + "=" * 80)
print("🛠 MONTAGEM DO DATASET LIGHTGBM - NUMERICAL AND CATEGORICAL TE FEATURES")
print("=" * 80)

# Carregar bases
df_num = spark.read.parquet(silver_path + "df_feature_store_lightgbm_num")
df_te = spark.read.parquet(silver_path + "df_lightgbm_cat_te_all_features")

print(f"✓ df_num carregado: {df_num.count():,} linhas")
print(f"✓ df_te carregado: {df_te.count():,} linhas")
```

```
=====
🛠 MONTAGEM DO DATASET LIGHTGBM - NUMERICAL AND CATEGORICAL TE FEATURES
=====

✓ df_num carregado: 9,678,866 linhas
✓ df_te carregado: 6,262,831 linhas
```

In [58]:

```
# ✅ Join CORRETO: por [msno, safra]
print("\n🔗 Juntando df_num + df_te por [msno, safra]...")

df_master = df_num.join(df_te, on=["msno", "safra"], how="left")

print(f"✓ df_master após join: {df_master.count():,} linhas")

# Validar integridade (não pode ter duplicação)
expected_count = df_num.count()
actual_count = df_master.count()

if actual_count == expected_count:
    print(f"✅ Join correto: {actual_count:,} linhas (sem duplicação)")
else:
    print(f"❌ ERRO: Join gerou duplicatas!")
    print(f"    Esperado: {expected_count:,}")
    print(f"    Obtido: {actual_count:,}")
    print(f"    Diferença: {actual_count - expected_count:,}")
    raise ValueError("Join gerou duplicatas - pipeline interrompido")
```

🔗 Juntando df_num + df_te por [msno, safra]...
✓ df_master após join: 9,678,866 linhas
✅ Join correto: 9,678,866 linhas (sem duplicação)

In [59]:

```
# Preencher NULLs nas colunas TE (test/OOT) com médias globais do treino
print("\n🔧 Preenchendo NULLs nas colunas TE com médias globais do treino...")

te_cols = [c for c in df_te.columns if c.endswith("_te")]

# Calcular médias globais do treino
means_dict = df_te.agg(
    *[F.mean(c).alias(c) for c in te_cols]
).collect()[0].asDict()

print(f"✓ Médias globais calculadas para {len(te_cols)} colunas TE")

# Aplicar coalesce (se NULL, usa a média)
for col_te in te_cols:
    df_master = df_master.withColumn(col_te, F.coalesce(F.col(col_te), F.lit(means_dict[col_te])))

print("✓ NULLs preenchidos com sucesso")
```

🔧 Preenchendo NULLs nas colunas TE com médias globais do treino...
✓ Médias globais calculadas para 17 colunas TE
✓ NULLs preenchidos com sucesso

```
In [60]: # Validar ausência de NULLs nas colunas TE
print("\n🔍 Validando ausência de NULLs nas colunas TE...")

null_counts = df_master.select(
    [F.sum(F.col(c).isNull().cast("int")).alias(c) for c in te_cols]
).collect()[0].asDict()

total_nulls = sum(null_counts.values())

if total_nulls == 0:
    print("✅ Nenhum NULL encontrado nas colunas TE")
else:
    print(f"⚠️ Ainda existem {total_nulls} NULLs nas colunas TE:")
    for col, count in null_counts.items():
        if count > 0:
            print(f"    {col}: {count:,} NULLs")

# Ordenar colunas (boa prática)
print("\n🔧 Ordenando colunas...")

control_cols = ['msno', 'safra', 'partition', 'target_win']

# Colunas numéricas (excluindo controle e TE)
num_cols = [c for c in df_master.columns
            if c not in control_cols
            and not c.endswith("_te")]

# Ordenação final: controle → numéricas → TE
df_master = df_master.select(
    *control_cols,
    *sorted(num_cols),
    *sorted(te_cols)
)

print("✓ Colunas ordenadas")
```

🔍 Validando ausência de NULLs nas colunas TE...
 ✅ Nenhum NULL encontrado nas colunas TE
 🔧 Ordenando colunas...
 ✓ Colunas ordenadas

```
In [61]: # Estatísticas finais
print("\n📊 Estatísticas do dataset final:")
print(f"    Total de linhas: {df_master.count():,}")
print(f"    Total de colunas: {len(df_master.columns)}")
print(f"    Colunas de controle: {len(control_cols)}")
print(f"    Colunas numéricas: {len(num_cols)}")
print(f"    Colunas TE: {len(te_cols)}")

# Distribuição por partition
print("\n📊 Distribuição por partition:")
df_master.groupBy("partition").count().orderBy("partition").show()
```

📊 Estatísticas do dataset final:
 Total de linhas: 9,678,866
 Total de colunas: 61
 Colunas de controle: 4
 Colunas numéricas: 40
 Colunas TE: 17
 📊 Distribuição por partition:
 +-----+-----+
 |partition| count|
 +-----+-----+
oot	1850732
test	1565303
train	6262831
 +-----+-----+

```
In [62]: # Salvar dataset final
print("\n💾 Salvando df_master_lightgbm_pre_f_importance...")

output_path = silver_path + "df_master_lightgbm_pre_f_importance"
df_master.write.mode("overwrite").parquet(output_path)

print(f"✅ Dataset final salvo em: {output_path}")
```

💾 Salvando df_master_lightgbm_pre_f_importance...
 ✅ Dataset final salvo em: C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_master_lightgbm_pre_f_importance

In [63]:

```
# Validação final (reload)
print("\n🔍 Validação final (reload)...")


df_master_reload = spark.read.parquet(output_path)

print(f"✓ Linhas após reload: {df_master_reload.count()}")
print(f"✓ Colunas após reload: {len(df_master_reload.columns)}")

print("\n📋 Schema do dataset final:")
df_master_reload.printSchema()

print("\n📊 Amostra do dataset final:")
df_master_reload.select(
    "msno", "safra", "partition", "target_win",
    *te_cols[:3] # primeiras 3 colunas TE
).show(5, truncate=False)

print("\n" + "=" * 80)
print("🎉 PIPELINE COMPLETO CONCLUÍDO COM SUCESSO!")
print("=" * 80)
print(f"✓ df_master_lightgbm pronto para treinamento")
print(f"✓ {df_master.count()}: linhas x {len(df_master.columns)} colunas")
print(f"✓ Sem duplicatas, sem NULLs, sem leakage")
print(f"✓ Granularidade temporal preservada (msno + safra)")
print("=" * 80)
```

🔍 Validação final (reload)...
✓ Linhas após reload: 9,678,866
✓ Colunas após reload: 61

📋 Schema do dataset final:

```
root
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- partition: string (nullable = true)
|-- target_win: double (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- daily_revenue_efficiency: double (nullable = true)
|-- daily_revenue_efficiency_min_3: double (nullable = true)
|-- daily_revenue_efficiency_min_3_is_sentinel: byte (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_3: double (nullable = true)
|-- daily_revenue_efficiency_ratio_ref_max_3_is_sentinel: byte (nullable = true)
|-- flag_has_transactions: byte (nullable = true)
|-- flag_has_transactions_lag_1: byte (nullable = true)
|-- flag_has_transactions_max_3: byte (nullable = true)
|-- flag_plano_mensal: byte (nullable = true)
|-- flag_plano_mensal_lag_1: byte (nullable = true)
|-- flag_plano_mensal_max_3: byte (nullable = true)
|-- flag_valid_fee: byte (nullable = true)
|-- flag_valid_fee_lag_1: byte (nullable = true)
|-- flag_valid_fee_max_3: byte (nullable = true)
|-- is_auto_renew: byte (nullable = true)
|-- log_total_plays: double (nullable = true)
|-- log_total_plays_mean_3: double (nullable = true)
|-- log_total_plays_mean_3_is_sentinel: byte (nullable = true)
|-- log_total_secs: double (nullable = true)
|-- margem_liquida_mensal: double (nullable = true)
|-- num_100: double (nullable = true)
|-- num_25: double (nullable = true)
|-- num_50: double (nullable = true)
|-- num_75: double (nullable = true)
|-- num_985: double (nullable = true)
|-- num_unq: double (nullable = true)
|-- payment_method_id: integer (nullable = true)
|-- plays_per_unq_cap_min_6: double (nullable = true)
|-- plays_per_unq_cap_min_6_is_sentinel: byte (nullable = true)
|-- revenue_per_hour_listened_cap: double (nullable = true)
|-- total_plays: double (nullable = true)
|-- total_plays_mean_3: double (nullable = true)
|-- total_plays_mean_3_is_sentinel: byte (nullable = true)
|-- total_secs: double (nullable = true)
|-- total_secs_mean_3: double (nullable = true)
|-- total_secs_mean_3_is_sentinel: byte (nullable = true)
|-- total_secs_ratio_ref_max_6: double (nullable = true)
|-- total_secs_ratio_ref_max_6_is_sentinel: byte (nullable = true)
|-- usage_intensity_per_tenure_cap: double (nullable = true)
|-- avg_secs_per_unq_cap_group_te: double (nullable = true)
|-- completed_songs_rate_group_te: double (nullable = true)
|-- early_drop_rate_group_te: double (nullable = true)
|-- faixa_idade_te: double (nullable = true)
|-- gender_clean_te: double (nullable = true)
|-- payment_method_group_te: double (nullable = true)
|-- payment_price_regime_te: double (nullable = true)
|-- plays_behavior_vs_completion_collapsed_te: double (nullable = true)
|-- plays_behavior_vs_volume_collapsed_te: double (nullable = true)
|-- plays_per_unq_behavior_te: double (nullable = true)
```

```

--- registered_via_group_te: double (nullable = true)
--- registration_year_regime_te: double (nullable = true)
--- revenue_per_hour_tier_te: double (nullable = true)
--- revenue_tier_te: double (nullable = true)
--- tenure_faixa_te: double (nullable = true)
--- total_plays_group_te: double (nullable = true)
--- usage_intensity_tier_te: double (nullable = true)

📊 Amostra do dataset final:
+-----+-----+-----+-----+
|msno |safra |partition|target_win
|total_plays_group_te|completed_songs_rate_group_te|avg_secs_per_unq_cap_group_te|
+-----+-----+-----+-----+
-----+
|++IzseRRiQS9aaSkH6cMYU6bGDcxUiieAi/tH67sC5s=|201606|test | -53.7420544 | 43.67596685633573 | 43.67595626527117
|43.67596895366032 | |
|++UDNo9DLrxT80VGidDi1OnWfczAdEwThaVyD0fx050=|201607|test | 93.8537375 | 43.67596685633573 | 43.67595626527117
|43.67596895366032 | |
|++ZHqwUNa7U21Qz+zqteiXlZapkey8616eEorrak/g=|201602|train | 98.47809190000001 | 55.470517367107824 | 41.09726822410867
|40.27235709412038 | |
|++3Z+W80PnpbHYfrKwqRKN1bF83XEbxjdYUolhGdHzg=|201606|test | 90.0801459 | 43.67596685633573 | 43.67595626527117
|43.67596895366032 | |
|++3fWHRDC5GWllovHcrOKWNwZY0jWwkJyeLlL65uv78=|201609|train | 28.203895200000005 | 50.466628884946644 | 41.12688910831912
|43.8512659022299 | |
+-----+-----+-----+-----+
-----+
only showing top 5 rows

=====
🎉 PIPELINE COMPLETO CONCLUÍDO COM SUCESSO!
=====

✓ df_master_lightgbm pronto para treinamento
✓ 9,678,866 linhas × 61 colunas
✓ Sem duplicatas, sem NULLs, sem leakage
✓ Granularidade temporal preservada (msno + safra)
=====
```

Conclusão e Metodologia: Target Encoding Temporal (TET)

A metodologia aplicada visa transformar variáveis categóricas de alta cardinalidade em sinais numéricos contínuos, garantindo que o modelo LightGBM capture o comportamento histórico sem "decorar" o alvo (overfitting) e sem enxergar o futuro (leakage).

Granularidade e Chave Primária Composta [msno + safra]

Diferente de um modelo estático, o comportamento do usuário no seu dataset é dinâmico. Um usuário pode ser "High Revenue" em Janeiro e "Low Revenue" em Março.

* **Metodologia:** Mantivemos a `safra` em todas as etapas. Isso garante que o Target Encoding (TE) seja aplicado ao registro específico daquele mês.

* **Coerência:** Ao fazer o join final por `[msno + safra]`, garantimos que o modelo receba o peso estatístico da categoria que o usuário pertence **naquele exato momento**, preservando a evolução temporal do comportamento.

Prevenção de Leakage via Out-of-Fold (OOF) Encoding

O maior risco do Target Encoding é o "Target Leakage" (quando a resposta da própria linha vaza para a feature).

* **Metodologia:** Dividimos o treino em 5 Folds aleatórios. Para cada linha no Fold A, o valor do TE é calculado usando apenas a média do Target dos Folds B, C, D e E.

* **Resultado:** A linha nunca "vê" o seu próprio `target_win`. Isso força o modelo a aprender a relação estatística da categoria no grupo, e não a resposta individual, reduzindo drasticamente o overfitting.

Suavização (Smoothing/Alpha)

Categorias com poucas amostras podem gerar médias extremas e pouco confiáveis.

* **Metodologia:** Aplicou-se um fator de suavização ($\$\\alpha = 10\$$).

* **Lógica:** O valor final do TE é uma média ponderada entre a média da categoria e a média global do dataset. Se uma categoria tem poucos exemplos, o peso da média global aumenta, "puxando" o valor para o centro e evitando ruídos estatísticos.

Tratamento de Teste e OOT (Out-of-Time)

As safras de Teste e OOT não possuem `target_win` conhecido no momento da inferência (ou não devem usá-lo para evitar viés).

* **Metodologia:**

1. O mapeamento (Lookup Table) é gerado **exclusivamente** com dados de Treino.

2. As safras de Teste/OOT recebem esses valores via Join.

3. Registros que possuem categorias novas (não vistas no treino) ou que estão fora da janela de treino recebem a **Média Global do Treino** via `coalesce`.

* **Sem Viés:** Isso garante que o modelo trate o futuro exatamente como ele trataria um dado em produção: usando apenas o conhecimento acumulado no passado.

Estabilidade do Sinal (Join vs. Deduplicação)

A decisão de **não deduplicar** por `msno` foi vital para a saúde do modelo.

* **Explicação:** Se um usuário aparece em 3 safras de treino, ele contribui para o cálculo do TE 3 vezes (possivelmente em folds diferentes). Ao aplicar o TE de volta, ele recebe o valor correspondente à categoria que ele tinha em cada safra.

* **Conclusão:** Isso mantém o dataset com os 9.6M de linhas originais, garantindo que o LightGBM aprenda com todas as fotos temporais disponíveis, sem criar registros fantasmas (duplicatas de join) e sem ignorar a volatilidade do usuário.

Resumo

* **Técnica:** Target Encoding com Leave-One-Fold-Out e Smoothing.

* **Chave de Cruzamento:** `msno + safra`.

* **População de Cálculo:** 100% Treino.

* **População de Aplicação:** Treino, Teste e OOT.

* **Tratamento de Nulos:** Imputação pela média global de treino (Global Mean Imputation).

* **Risco de Leakage:** Mitigado por Cross-Validation OOF e separação rigorosa de partições.

Mais técnicas de Feature Selection

Carregar e Preparar o Dataset Master

```
In [37]: from pyspark.sql import functions as F

print("=" * 80)
print("🛠 CARREGAR E PREPARAR DATASET MASTER")
print("=" * 80)

silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"

# Carregar dataset final
df_master = spark.read.parquet(silver_path + "df_master_lightgbm_pre_f_importance")

print(f"✓ Dataset carregado: {df_master.count():,} linhas x {len(df_master.columns)} colunas")
```

```
# Separar colunas por tipo
control_cols = ['msno', 'safra', 'partition', 'target_win']
te_cols = [c for c in df_master.columns if c.endswith("_te")]
num_cols = [c for c in df_master.columns
            if c not in control_cols and not c.endswith("_te")]

print(f"\n📊 Composição do dataset:")
print(f"  Controle: {len(control_cols)} colunas")
print(f"  Numéricas: {len(num_cols)} colunas")
print(f"  Target Encoding: {len(te_cols)} colunas")
print(f"  TOTAL: {len(df_master.columns)} colunas")
```

```
=====
🛠 CARREGAR E PREPARAR DATASET MASTER
=====
✓ Dataset carregado: 9,678,866 linhas x 61 colunas
```

```
📊 Composição do dataset:
  Controle: 4 colunas
  Numéricas: 40 colunas
  Target Encoding: 17 colunas
  TOTAL: 61 colunas
```

Análise de Variância (Remover Features Constantes)

In [38]:

```

print("\n" + "=" * 80)
print("ANÁLISE DE VARIÂNCIA (COEFICIENTE DE VARIAÇÃO)")
print("=" * 80)

# Filtrar apenas treino para análise
df_train = df_master.filter(F.col("partition") == "train")

print(f"✓ Analisando {len(num_cols + te_cols)} features no conjunto de treino")

# Calcular variância E média para cada coluna numérica
variance_stats = []

for col in num_cols + te_cols:
    stats = df_train.select(
        F.variance(col).alias("var"),
        F.mean(col).alias("mean"),
        F.stddev(col).alias("std")
    ).collect()[0]

    var_value = stats["var"] if stats["var"] is not None else 0.0
    mean_value = stats["mean"] if stats["mean"] is not None else 0.0
    std_value = stats["std"] if stats["std"] is not None else 0.0

    # Coeficiente de Variação (CV)
    # Se média é muito próxima de zero, usamos variância bruta normalizada
    if abs(mean_value) > 1e-6:
        cv = std_value / abs(mean_value)
    else:
        # Para flags/variáveis centradas, usamos variância bruta
        cv = var_value

    variance_stats.append({
        'feature': col,
        'variance': var_value,
        'mean': mean_value,
        'std': std_value,
        'cv': cv
    })
}

# Converter para DataFrame Pandas para análise
import pandas as pd
df_variance = pd.DataFrame(variance_stats).sort_values('cv')

print("\n📊 Top 10 features com MENOR variação relativa (CV):")
print(df_variance.head(10)[['feature', 'cv', 'variance', 'mean']].to_string(index=False))

# Threshold ajustado: CV < 0.1 indica variação relativa muito baixa
cv_threshold = 0.1

low_variance_features = df_variance[df_variance['cv'] < cv_threshold]['feature'].tolist()

print(f"\n📊 Features com CV < {cv_threshold} (baixa variação relativa):")
if len(low_variance_features) > 0:
    for feat in low_variance_features:
        row = df_variance[df_variance['feature'] == feat].iloc[0]
        print(f"  ✗ {feat}: CV={row['cv']:.6f} (var={row['variance']:.4f}, mean={row['mean']:.4f})")
    print(f"\n⚠️ Total de features a remover: {len(low_variance_features)}")
else:
    print("  ✓ Nenhuma feature com baixa variação relativa encontrada")

# Salvar lista de features a manter
features_to_keep_variance = [c for c in num_cols + te_cols if c not in low_variance_features]

print(f"\n✓ Features mantidas após filtro de variância: {len(features_to_keep_variance)}")

```

```

=====
ANÁLISE DE VARIÂNCIA (COEFICIENTE DE VARIAÇÃO)
=====

✓ Analisando 57 features no conjunto de treino

📊 Top 10 features com MENOR variação relativa (CV):
      feature      cv  variance     mean
daily_revenue_efficiency_ratio_ref_max_3  0.048972  0.002373  0.994663
registration_year_regime_te  0.072910  10.140483  43.675981
          tenure_faixa_te  0.089125  15.152593  43.675983
plays_per_unq_behavior_te  0.145764  40.530713  43.675999
avg_secs_per_unq_cap_group_te  0.148115  41.848863  43.675969
early_drop_rate_group_te  0.153278  44.816989  43.675915
          gender_clean_te  0.154086  45.291211  43.675994
completed_songs_rate_group_te  0.158947  48.193791  43.675956
plays_behavior_vs_completion_collapsed_te  0.175602  58.822309  43.675994
faixa_idade_te  0.186841  66.593379  43.676012

📊 Features com CV < 0.1 (baixa variação relativa):
  ✗ daily_revenue_efficiency_ratio_ref_max_3: CV=0.048972 (var=0.0024, mean=0.9947)
  ✗ registration_year_regime_te: CV=0.072910 (var=10.1405, mean=43.6760)

```

✖ tenure_faixa_te: CV=0.089125 (var=15.1526, mean=43.6760)

⚠ Total de features a remover: 3

✓ Features mantidas após filtro de variância: 54

Análise de Multicolinearidade

In [39]:

```

print("\n" + "=" * 80)
print("🛠 MATRIZ DE CORRELAÇÃO (MULTICOLINEARIDADE)")
print("=" * 80)

# 1. Preparação dos dados
print("🔗 Convertendo para Pandas (apenas amostra para análise visual)...")

# Utilizando a amostra que você definiu (5%)
df_train_sample = df_train.select(*features_to_keep_variance).sample(fraction=0.05, seed=42)

total_rows = df_train_sample.count()
print(f"✓ Amostra de {total_rows:,} linhas selecionada")

# Conversão para Pandas
df_pandas = df_train_sample.toPandas()
print("✓ Conversão concluída")

# 2. Cálculo da correlação
print("\n📊 Calculando matriz de correlação...")
corr_matrix = df_pandas.corr().abs()
print("✓ Matriz calculada")

# 3. Identificação de Multicolinearidade. Threshold pode ser alto pois o algoritmo lida bem com correlações, mas queremos evitar redundância extrema.
high_corr_threshold = 0.9

# CORREÇÃO AQUI: Substituimos pd.np por np diretamente
# O triângulo superior isola os pares únicos (evita comparar A-B e B-A)
upper_triangle = corr_matrix.where(
    np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
)

# Encontrar pares com correlação > threshold
high_corr_pairs = [
    (col, row, upper_triangle.loc[row, col])
    for col in upper_triangle.columns
    for row in upper_triangle.index
    if upper_triangle.loc[row, col] > high_corr_threshold
]

# 4. Exibição dos Resultados
print(f"\n📊 Pares com correlação > {high_corr_threshold}:")

if len(high_corr_pairs) > 0:
    # Ordena dos mais correlacionados para os menos correlacionados
    sorted_pairs = sorted(high_corr_pairs, key=lambda x: x[2], reverse=True)

    for col1, col2, corr_val in sorted_pairs:
        print(f"⚠️ {col1} ↔ {col2}: {corr_val:.4f}")

    print(f"\n⚠️ Total de pares redundantes: {len(high_corr_pairs)}")

    # Dica extra: Identifica as colunas que você pode querer dropar
    cols_to_drop = list(set([pair[0] for pair in high_corr_pairs]))
    print(f"💡 Sugestão de exclusão (mantenha apenas um de cada par): {cols_to_drop}")
else:
    print("✅ Nenhum par com alta correlação encontrado")

```

```

=====
🛠 MATRIZ DE CORRELAÇÃO (MULTICOLINEARIDADE)
=====

🔗 Convertendo para Pandas (apenas amostra para análise visual)...
✓ Amostra de 313,097 linhas selecionada
✓ Conversão concluída

📊 Calculando matriz de correlação...
✓ Matriz calculada

📊 Pares com correlação > 0.9:
⚠️ total_plays_mean_3_is_sentinel ↔ log_total_plays_mean_3_is_sentinel: 1.0000
⚠️ total_secs_mean_3_is_sentinel ↔ log_total_plays_mean_3_is_sentinel: 1.0000
⚠️ total_secs_mean_3_is_sentinel ↔ total_plays_mean_3_is_sentinel: 1.0000
⚠️ total_secs ↔ num_100: 0.9936
⚠️ payment_price_regime_te ↔ flag_valid_fee: 0.9923
⚠️ payment_price_regime_te ↔ flag_has_transactions: 0.9897
⚠️ revenue_tier_te ↔ daily_revenue_efficiency: 0.9863
⚠️ margem_liquida_mensal ↔ actual_amount_paid: 0.9863
⚠️ flag_valid_fee_max_3 ↔ flag_has_transactions_max_3: 0.9843
⚠️ flag_valid_fee ↔ flag_has_transactions: 0.9833
⚠️ flag_valid_fee_lag_1 ↔ flag_has_transactions_lag_1: 0.9829
⚠️ total_secs_mean_3 ↔ total_plays_mean_3: 0.9785
⚠️ total_secs ↔ total_plays: 0.9784
⚠️ payment_method_id ↔ flag_has_transactions: 0.9767
⚠️ total_plays ↔ num_100: 0.9704
⚠️ log_total_secs ↔ log_total_plays: 0.9689
⚠️ payment_price_regime_te ↔ payment_method_id: 0.9682

```

```

⚠️ total_secs_ratio_ref_max_6_is_sentinel + plays_per_unq_cap_min_6_is_sentinel: 0.9655
⚠️ payment_method_id + flag_valid_fee: 0.9633
⚠️ total_plays_group_te + plays_behavior_vs_volume_collapsed_te: 0.9615
⚠️ payment_method_group_te + flag_has_transactions: 0.9566
⚠️ flag_valid_fee + flag_plano_mensal: 0.9528
⚠️ payment_price_regime_te + payment_method_group_te: 0.9523
⚠️ payment_method_group_te + flag_valid_fee: 0.9510
⚠️ payment_method_group_te + payment_method_id: 0.9460
⚠️ payment_price_regime_te + flag_plano_mensal: 0.9452
⚠️ total_plays + num_unq: 0.9377
⚠️ payment_method_id + flag_plano_mensal: 0.9376
⚠️ flag_plano_mensal + flag_has_transactions: 0.9371
⚠️ flag_valid_fee_lag_1 + flag_plano_mensal_lag_1: 0.9366
⚠️ payment_method_group_te + flag_plano_mensal: 0.9361
⚠️ total_secs_mean_3 + total_secs: 0.9329
⚠️ total_plays_mean_3 + total_plays: 0.9306
⚠️ total_secs_mean_3 + num_100: 0.9277
⚠️ revenue_tier_te + payment_price_regime_te: 0.9217
⚠️ flag_plano_mensal_lag_1 + flag_has_transactions_lag_1: 0.9208
⚠️ payment_price_regime_te + daily_revenue_efficiency: 0.9164
⚠️ revenue_tier_te + flag_valid_fee: 0.9147
⚠️ total_secs + total_plays_mean_3: 0.9145
⚠️ revenue_tier_te + flag_has_transactions: 0.9122
⚠️ total_secs_mean_3 + total_plays: 0.9093
⚠️ flag_valid_fee + daily_revenue_efficiency: 0.9087
⚠️ total_plays_mean_3 + num_100: 0.9070
⚠️ total_secs + num_unq: 0.9052
⚠️ total_secs_ratio_ref_max_6_is_sentinel + log_total_plays_mean_3_is_sentinel: 0.9051
⚠️ total_secs_ratio_ref_max_6_is_sentinel + total_plays_mean_3_is_sentinel: 0.9051
⚠️ total_secs_ratio_ref_max_6_is_sentinel + total_secs_mean_3_is_sentinel: 0.9051
⚠️ plays_behavior_vs_completion_collapsed_te + completed_songs_rate_group_te: 0.9050

⚠️ Total de pares redundantes: 48
💡 Sugestão de exclusão (mantenha apenas um de cada par): ['payment_method_id', 'total_secs_mean_3_is_sentinel', 'flag_valid_fee', 'flag_valid_fee_lag_1', 'flag_valid_fee_max_3', 'flag_plano_mensal', 'margem_liquida_mensal', 'plays_behavior_vs_completion_collapsed_te', 'total_secs', 'revenue_tier_te', 'flag_plano_mensal_lag_1', 'total_plays_mean_3_is_sentinel', 'total_plays_mean_3', 'total_secs_ratio_ref_max_6_is_sentinel', 'payment_method_group_te', 'total_secs_mean_3', 'total_plays_group_te', 'log_total_secs', 'payment_price_regime_te', 'total_plays']

```

Como tratar as features _is_sentinel agora?

Metodologicamente, existem duas visões, e deixar o LightGBM escolher é a mais correta para modelos de árvore:

1. **A Visão Estatística:** "Se as flags de erro de duas variáveis diferentes acontecem quase sempre juntas, elas são redundantes. Remova uma."
2. **A Visão de Engenharia de Features:** "Mesmo que elas aconteçam juntas, elas pertencem a colunas diferentes. Se no futuro a `daily_revenue_efficiency_min_3` falhar mas a `ratio_ref` não, eu preciso da flag específica."

A Solução: "Imunidade" para as Sentinelas

Devemos **proteger** as colunas `_is_sentinel` do filtro de correlação. Elas devem ser mantidas obrigatoriamente SOMENTE se a variável numérica original delas também for mantida. Se a variável numérica original (ex: `total_plays`) for removida por ser redundante ou instável, a sua respectiva flag `total_plays_is_sentinel` **perde a razão de existir**.

In [40]:

```

print("\n" + "=" * 80)
print(" DECISÃO DE REMOÇÃO (PRIORIDADE > CV > TE) + PROTEÇÃO SENTINELAS")
print("=" * 80)

features_to_remove_corr = []

SENTINEL_SUFFIX = "_is_sentinel"

def is_sentinel(feature_name: str) -> bool:
    return feature_name.endswith(SENTINEL_SUFFIX)

def get_priority(feature_name: str) -> int:
    """
    Prioridade (quanto maior, mais preferida):
    3: Numérica Transformada/derivada (log_, ratio_, mean_, min_, max_)
    2: Numérica bruta
    1: Target Encoding (_te)
    """
    if feature_name.endswith("_te"):
        return 1

    transform_indicators = ["log_", "ratio_", "mean_", "min_", "max_"]
    if any(ind in feature_name for ind in transform_indicators):
        return 3

    return 2

if len(high_corr_pairs) > 0:
    print("🔧 Analisando pares redundantes...\n")

for col1, col2, corr_val in high_corr_pairs:
    # ✅ REGRA: se qualquer um é sentinel, não usa correlação para remover
    if is_sentinel(col1) or is_sentinel(col2):
        print(f"⚠️ Ignorando par com sentinelas: {col1} ↔ {col2} (corr={corr_val:.4f})")
        continue

    prio1 = get_priority(col1)
    prio2 = get_priority(col2)

    # ✅ CORREÇÃO: usar CV (coeficiente de variação) no lugar de variância bruta
    cv1 = df_variance[df_variance["feature"] == col1]["cv"].values[0]
    cv2 = df_variance[df_variance["feature"] == col2]["cv"].values[0]

    # Decisão baseada em prioridade
    if prio1 > prio2:
        to_remove, to_keep = col2, col1
        reason = f"Prioridade (Prio {prio1} > {prio2})"
    elif prio2 > prio1:
        to_remove, to_keep = col1, col2
        reason = f"Prioridade (Prio {prio2} > {prio1})"
    else:
        # Empate de prioridade: decide pelo CV (maior variação relativa)
        if cv1 > cv2:
            to_remove, to_keep = col2, col1
            reason = f"Empate de Prio - Maior CV (variação relativa: {cv1:.4f})"
        else:
            to_remove, to_keep = col1, col2
            reason = f"Empate de Prio - Maior CV (variação relativa: {cv2:.4f})"

    if to_remove not in features_to_remove_corr:
        features_to_remove_corr.append(to_remove)
        print(f"✖️ Remover: {to_remove}")
        print(f"✅ Manter: {to_keep}")
        print(f"  Motivo: {reason}")
        print(f"  Correlação: {corr_val:.4f}\n")

print(f"⚠️ Total de features a remover por correlação: {len(features_to_remove_corr)}")
else:
    print("✅ Nenhuma feature a remover por correlação")

features_after_corr = [c for c in features_to_keep_variance if c not in features_to_remove_corr]

print(f"\n✅ Features restantes após correlação: {len(features_after_corr)}")

```

```

=====
DECISÃO DE REMOÇÃO (PRIORIDADE > CV > TE) + PROTEÇÃO SENTINELAS
=====

🔧 Analisando pares redundantes...

✖️ Remover: flag_has_transactions
✅ Manter: flag_plano_mensal
  Motivo: Empate de Prio - Maior CV (variação relativa: 0.4869)
  Correlação: 0.9371

✖️ Remover: flag_has_transactions_lag_1
✅ Manter: flag_plano_mensal_lag_1

```

Motivo: Empate de Prio - Maior CV (variação relativa: 0.4839)
Correlação: 0.9208

✗ Remover: flag_valid_fee
✓ Manter: daily_revenue_efficiency
Motivo: Empate de Prio - Maior CV (variação relativa: 0.5108)
Correlação: 0.9087

✗ Remover: flag_valid_fee_lag_1
✓ Manter: flag_plano_mensal_lag_1
Motivo: Empate de Prio - Maior CV (variação relativa: 0.4839)
Correlação: 0.9366

✗ Remover: flag_has_transactions_max_3
✓ Manter: flag_valid_fee_max_3
Motivo: Empate de Prio - Maior CV (variação relativa: 0.3773)
Correlação: 0.9843

✗ Remover: log_total_secs
✓ Manter: log_total_plays
Motivo: Empate de Prio - Maior CV (variação relativa: 0.4179)
Correlação: 0.9689

✗ Remover: actual_amount_paid
✓ Manter: margem_liquida_mensal
Motivo: Empate de Prio - Maior CV (variação relativa: 2.0633)
Correlação: 0.9863

✗ Remover: payment_method_id
✓ Manter: flag_plano_mensal
Motivo: Empate de Prio - Maior CV (variação relativa: 0.4869)
Correlação: 0.9376

✗ Remover: total_plays
✓ Manter: num_100
Motivo: Empate de Prio - Maior CV (variação relativa: 1.3437)
Correlação: 0.9704

✗ Remover: num_unq
✓ Manter: total_plays
Motivo: Empate de Prio - Maior CV (variação relativa: 1.1843)
Correlação: 0.9377

✗ Remover: num_100
✓ Manter: total_plays_mean_3
Motivo: Prioridade (Prio 3 > 2)
Correlação: 0.9070

⌚ Ignorando par com sentinel: total_plays_mean_3_is_sentinel ↔ log_total_plays_mean_3_is_sentinel (corr=1.0000)
✗ Remover: total_secs
✓ Manter: num_100
Motivo: Empate de Prio - Maior CV (variação relativa: 1.3437)
Correlação: 0.9936

✗ Remover: total_plays_mean_3
✓ Manter: total_secs_mean_3
Motivo: Empate de Prio - Maior CV (variação relativa: 1.1281)
Correlação: 0.9785

⌚ Ignorando par com sentinel: total_secs_mean_3_is_sentinel ↔ log_total_plays_mean_3_is_sentinel (corr=1.0000)
⌚ Ignorando par com sentinel: total_secs_mean_3_is_sentinel ↔ total_plays_mean_3_is_sentinel (corr=1.0000)
⌚ Ignorando par com sentinel: total_secs_ratio_ref_max_6_is_sentinel ↔ log_total_plays_mean_3_is_sentinel (corr=0.9051)
⌚ Ignorando par com sentinel: total_secs_ratio_ref_max_6_is_sentinel ↔ plays_per_unq_cap_min_6_is_sentinel (corr=0.9655)
⌚ Ignorando par com sentinel: total_secs_ratio_ref_max_6_is_sentinel ↔ total_plays_mean_3_is_sentinel (corr=0.9051)
⌚ Ignorando par com sentinel: total_secs_ratio_ref_max_6_is_sentinel ↔ total_secs_mean_3_is_sentinel (corr=0.9051)
✗ Remover: payment_method_group_te
✓ Manter: flag_has_transactions
Motivo: Prioridade (Prio 2 > 1)
Correlação: 0.9566

✗ Remover: payment_price_regime_te
✓ Manter: daily_revenue_efficiency
Motivo: Prioridade (Prio 2 > 1)
Correlação: 0.9164

✗ Remover: completed_songs_rate_group_te
✓ Manter: plays_behavior_vs_completion_collapsed_te
Motivo: Empate de Prio - Maior CV (variação relativa: 0.1756)
Correlação: 0.9050

✗ Remover: revenue_tier_te
✓ Manter: daily_revenue_efficiency
Motivo: Prioridade (Prio 2 > 1)
Correlação: 0.9863

✗ Remover: plays_behavior_vs_volume_collapsed_te
✓ Manter: total_plays_group_te
Motivo: Empate de Prio - Maior CV (variação relativa: 0.3114)

Correlação: 0.9615

⚠️ Total de features a remover por correlação: 18

✓ Features restantes após correlação: 36

Análise de Estabilidade Temporal (PSI)

Contexto

A princípio, a ideia era calcular o PSI (Population Stability Index) comparando as distribuições das features entre o conjunto de **Treino** e o conjunto de **OOT** (Out-of-Time). No entanto, essa abordagem apresenta um problema metodológico sutil, mas crítico: **Data Snooping**.

Embora o PSI não utilize a variável target e, portanto, não configure um "leakage de target" clássico, ele ainda assim **utiliza informações do conjunto de teste final** (OOT) para tomar decisões sobre quais features incluir no modelo. Isso viola o princípio de que o conjunto de teste deve permanecer completamente isolado até a avaliação final do modelo.

Consequências do Data Snooping:

1. **Viés de Seleção:** As features escolhidas foram "otimizadas" para funcionar bem especificamente no período do OOT, o que pode não se generalizar para períodos futuros.
2. **Superestimação de Performance:** A métrica de performance no OOT deixa de ser uma estimativa honesta da capacidade de generalização do modelo.
3. **Risco de Overfitting Temporal:** O modelo pode estar "ajustado" para as características específicas do período OOT, em vez de capturar padrões genuinamente estáveis.

Solução Adotada:

Em vez de comparar Treino vs. OOT, vamos comparar **safras temporais dentro do próprio conjunto de Treino**:

- **Safra Inicial:** Primeiros meses do período de treino
- **Safra Final:** Últimos meses do período de treino

Justificativa:

Se uma feature apresenta instabilidade significativa **dentro do próprio histórico de treino**, ela certamente será problemática em produção. Essa abordagem:

- Mantém o OOT intocado (zero data snooping)
- Detecta drift temporal genuíno
- É mais conservadora e robusta para produção
- Permite validação honesta no OOT

Critério de Decisão:

- $\text{PSI} < 0.1$: Feature estável (mantém)
- $0.1 \leq \text{PSI} < 0.25$: Instabilidade moderada (mantém com atenção)
- $\text{PSI} \geq 0.25$: Feature instável (remove)

Execucao

```
In [41]: print("\n" + "=" * 80)
print("ANÁLISE DE ESTABILIDADE TEMPORAL (PSI)")
print("=" * 80)

def calculate_psi(df_early, df_late, feature, n_bins=10):
    """
    Calcula PSI entre duas safras temporais para uma feature numérica

    Args:
        df_early: DataFrame da safra inicial (baseline)
        df_late: DataFrame da safra final (comparação)
        feature: Nome da feature
        n_bins: Número de bins para discretização

    Returns:
        float: Valor do PSI
    """

    # Coletar valores
    early_vals = df_early.select(feature).toPandas()[feature].dropna()
    late_vals = df_late.select(feature).toPandas()[feature].dropna()

    # Criar bins baseados na safra inicial
    bins = pd.qcut(early_vals, q=n_bins, duplicates='drop', retbins=True)[1]

    # Distribuição na safra inicial (baseline)
    early_dist = pd.cut(early_vals, bins=bins, include_lowest=True).value_counts(normalize=True).sort_index()

    # Distribuição na safra final
    late_dist = pd.cut(late_vals, bins=bins, include_lowest=True).value_counts(normalize=True).sort_index()

    # Alinear índices (evitar divisão por zero)
    early_dist, late_dist = early_dist.align(late_dist, fill_value=0.0001)

    # Calcular PSI
    psi = ((late_dist - early_dist) * np.log(late_dist / early_dist)).sum()

    return psi

# =====
# DEFINIR SAFRAS TEMPORAIS (DENTRO DO TREINO)
# =====

# Obter todos os meses de treino ordenados
meses_treino = (df_master
    .filter(F.col("partition") == "train")
    .select("safra")
    .distinct()
    .orderBy("safra")
    .toPandas()["safra"]
    .tolist())

print(f"\n{len(meses_treino)} Meses disponíveis no Treino: {len(meses_treino)}")
print(f"  Primeiro mês: {meses_treino[0]}")
print(f"  Último mês: {meses_treino[-1]}")

# Definir safras: primeiros 30% vs últimos 30% do treino
n_meses = len(meses_treino)
n_safra = max(1, int(n_meses * 0.3)) # Pelo menos 1 mês

meses_safra_inicial = meses_treino[:n_safra]
meses_safra_final = meses_treino[-n_safra:]

print(f"\n{len(meses_safra_inicial)} Estratégia de Comparação:")
print(f"  Safra Inicial: {meses_safra_inicial[0]} a {meses_safra_inicial[-1]} ({len(meses_safra_inicial)} meses)")
print(f"  Safra Final: {meses_safra_final[0]} a {meses_safra_final[-1]} ({len(meses_safra_final)} meses)")

# Filtrar DataFrames por safra
df_safra_inicial = df_master.filter(
    (F.col("partition") == "train") &
    (F.col("safra").isin(meses_safra_inicial))
)

df_safra_final = df_master.filter(
    (F.col("partition") == "train") &
    (F.col("safra").isin(meses_safra_final))
)

print(f"\n{len(df_safra_inicial)} Safra Inicial: {df_safra_inicial.count():,} linhas")
print(f"{len(df_safra_final)} Safra Final: {df_safra_final.count():,} linhas")

# =====
# CALCULAR PSI PARA CADA FEATURE
# =====

print("\n{len(df_safra_inicial)} Calculando PSI para cada feature (pode demorar)...")
```

```

psi_results = []

for idx, feature in enumerate(features_after_corr, start=1):
    try:
        psi_value = calculate_psi(df_safra_inicial, df_safra_final, feature, n_bins=10)
        psi_results.append({
            'feature': feature,
            'psi': psi_value
        })

        # Classificar estabilidade
        if psi_value < 0.1:
            status = "✅ Estável"
        elif psi_value < 0.25:
            status = "⚠️ Moderado"
        else:
            status = "❌ Instável"

        print(f"{idx}/{len(features_after_corr)} | {feature:50s} | PSI: {psi_value:.4f} | {status}")

    except Exception as e:
        print(f"{idx}/{len(features_after_corr)} | {feature:50s} | ⚠️ Erro no cálculo: {str(e)}")
        psi_results.append({
            'feature': feature,
            'psi': None
        })

# =====
# ANÁLISE E DECISÃO
# =====

# Converter para DataFrame Pandas
df_psi = pd.DataFrame(psi_results).sort_values('psi', ascending=False)

# Identificar features instáveis
psi_threshold = 0.25
unstable_features = df_psi[df_psi['psi'] > psi_threshold]['feature'].tolist()

print(f"\n📊 Resumo de Estabilidade:")
print(f"  ✅ Estáveis (PSI < 0.1): {len(df_psi[df_psi['psi'] < 0.1])}")
print(f"  ⚠️ Moderadas (0.1 ≤ PSI < 0.25): {len(df_psi[(df_psi['psi'] >= 0.1) & (df_psi['psi'] < 0.25)])}")
print(f"  ❌ Instáveis (PSI ≥ 0.25): {len(unstable_features)}")

if len(unstable_features) > 0:
    print(f"\n⚠️ Features instáveis (candidatas à remoção):")
    for feat in unstable_features:
        psi_val = df_psi[df_psi['feature'] == feat]['psi'].values[0]
        print(f"  ❌ {feat}: PSI = {psi_val:.4f}")

```

=====
📍 ANÁLISE DE ESTABILIDADE TEMPORAL (PSI)
=====

📅 Meses disponíveis no Treino: 9
 Primeiro mês: 201601
 Último mês: 201609

🔍 Estratégia de Comparação:
 Safra Inicial: 201601 a 201602 (2 meses)
 Safra Final: 201608 a 201609 (2 meses)

- ✓ Safra Inicial: 1,406,028 linhas
- ✓ Safra Final: 1,499,379 linhas

📊 Calculando PSI para cada feature (pode demorar)...

1/36 daily_revenue_efficiency	PSI: 0.0616	✅ Estável
2/36 daily_revenue_efficiency_min_3	PSI: 0.1238	⚠️ Moderado
3/36 daily_revenue_efficiency_min_3_is_sentinel	PSI: 0.0000	✅ Estável
4/36 daily_revenue_efficiency_ratio_ref_max_3_is_sentinel	PSI: 0.0000	✅ Estável
5/36 flag_plano_mensal	PSI: 0.0000	✅ Estável
6/36 flag_plano_mensal_lag_1	PSI: 0.0000	✅ Estável
7/36 flag_plano_mensal_max_3	PSI: 0.0000	✅ Estável
8/36 flag_valid_fee_max_3	PSI: 0.0000	✅ Estável
9/36 is_auto_renew	PSI: 0.0000	✅ Estável
10/36 log_total_plays	PSI: 0.0012	✅ Estável
11/36 log_total_plays_mean_3	PSI: 0.0043	✅ Estável
12/36 log_total_plays_mean_3_is_sentinel	PSI: 0.0000	✅ Estável
13/36 margem_liquida_mensal	PSI: 0.0951	✅ Estável
14/36 num_25	PSI: 0.0007	✅ Estável
15/36 num_50	PSI: 0.0003	✅ Estável
16/36 num_75	PSI: 0.0020	✅ Estável
17/36 num_985	PSI: 0.0039	✅ Estável
18/36 plays_per_unq_cap_min_6	PSI: 0.1777	⚠️ Moderado
19/36 plays_per_unq_cap_min_6_is_sentinel	PSI: 0.0000	✅ Estável
20/36 revenue_per_hour_listened_cap	PSI: 0.2197	⚠️ Moderado

21/36 total_plays_mean_3_is_sentinel	PSI: 0.0000	Estável
22/36 total_secs_mean_3	PSI: 0.0012	Estável
23/36 total_secs_mean_3_is_sentinel	PSI: 0.0000	Estável
24/36 total_secs_ratio_ref_max_6	PSI: 0.2781	Instável
25/36 total_secs_ratio_ref_max_6_is_sentinel	PSI: 0.0000	Estável
26/36 usage_intensity_per_tenure_cap	PSI: 0.9098	Instável
27/36 avg_secs_per_unq_cap_group_te	PSI: 0.0023	Estável
28/36 early_drop_rate_group_te	PSI: 0.0013	Estável
29/36 faixa_idade_te	PSI: 0.0156	Estável
30/36 gender_clean_te	PSI: 0.0149	Estável
31/36 plays_behavior_vs_completion_collapsed_te	PSI: 0.0023	Estável
32/36 plays_per_unq_behavior_te	PSI: 0.0010	Estável
33/36 registered_via_group_te	PSI: 0.0188	Estável
34/36 revenue_per_hour_tier_te	PSI: 0.8978	Instável
35/36 total_plays_group_te	PSI: 0.0012	Estável
36/36 usage_intensity_tier_te	PSI: 1.1127	Instável

📊 Resumo de Estabilidade:

- ✓ Estáveis (PSI < 0.1): 29
- ⚠️ Moderadas (0.1 ≤ PSI < 0.25): 3
- ✗ Instáveis (PSI ≥ 0.25): 4

⚠️ Features instáveis (candidatas à remoção):

- ✗ usage_intensity_tier_te: PSI = 1.1127
- ✗ usage_intensity_per_tenure_cap: PSI = 0.9098
- ✗ revenue_per_hour_tier_te: PSI = 0.8978
- ✗ total_secs_ratio_ref_max_6: PSI = 0.2781

Conclusão

Apesar das features apresentarem instabilidade, são métricas "core" do negócio. Elas capturam a dinâmica de engajamento e monetização. O drift (PSI alto) pode ser apenas o reflexo de uma mudança real no comportamento dos usuários ou no mix da base, e o modelo **precisa** aprender isso para ser útil.

* **Interação de Variáveis:** O LightGBM é excelente em encontrar interações. Talvez `usage_intensity` sozinha tenha drift, mas combinada com outra feature, ela se torne o sinal mais estável para o target.

Consolidação + Limpeza de Sentinelas

In [42]:

```
print("\n" + "=" * 80)
print("✓ CONSOLIDAÇÃO FINAL + LIMPEZA DE SENTINELAS")
print("=" * 80)

SENTINEL_SUFFIX = "_is_sentinel"

def parent_of_sentinel(sentinel_col: str) -> str:
    """Retorna o nome da variável pai de uma sentinel"""
    return sentinel_col.replace(SENTINEL_SUFFIX, "")

def remove_orphan_sentinels(features_list):
    """
    Remove sentinelas cujo pai não existe mais na lista de features.
    Retorna:
        - lista limpa
        - lista de sentinelas removidas
    """
    features_set = set(features_list)
    removed_orphans = []
    cleaned = []

    for f in features_list:
        if f.endswith(SENTINEL_SUFFIX):
            parent = parent_of_sentinel(f)
            if parent not in features_set:
                removed_orphans.append(f)
                print(f"✓ Removendo sentinel: {f} (pai '{parent}' foi removido)")
            continue
        cleaned.append(f)

    return cleaned, removed_orphans

# ✅ ALTERAÇÃO: NÃO aplicar filtro de PSI (manter todas as features após correlação)
# Apenas registramos o PSI para análise, mas não removemos automaticamente
features_pre_orphan_cleanup = features_after_corr # ← Mudança aqui

print(f"⭐ Features após correlação (PSI registrado, mas não filtrado): {len(features_pre_orphan_cleanup)}")

# 2) Remover sentinelas órfãs
features_final, orphan_sentinels_removed = remove_orphan_sentinels(features_pre_orphan_cleanup)

print(f"\n📊 Resumo da limpeza de sentinelas:")
print(f"  Sentinelas removidas: {len(orphan_sentinels_removed)}")
if orphan_sentinels_removed:
    for x in orphan_sentinels_removed:
        print(f"    - {x}")

print(f"\n✅ Features finais após limpeza: {len(features_final)}")

# ✅ OPCIONAL: Exibir quais features têm PSI alto (para referência/documentação)
if len(unstable_features) > 0:
    print(f"\n📋 Features com PSI alto (mantidas para avaliação no modelo):")
    for feat in unstable_features:
        if feat in features_final: # Só lista se ainda estiver na lista final
            psi_val = df_psi[df_psi['feature'] == feat]['psi'].values[0]
            print(f"    ⚠️ {feat}: PSI = {psi_val:.4f}")

```

```
=====
✓ CONSOLIDAÇÃO FINAL + LIMPEZA DE SENTINELAS
=====
⭐ Features após correlação (PSI registrado, mas não filtrado): 36
✓ Removendo sentinel: daily_revenue_efficiency_ratio_ref_max_3_is_sentinel (pai 'daily_revenue_efficiency_ratio_ref_max_3' foi removido)
✓ Removendo sentinel: total_plays_mean_3_is_sentinel (pai 'total_plays_mean_3' foi removido)

📊 Resumo da limpeza de sentinelas:
  Sentinelas removidas: 2
    - daily_revenue_efficiency_ratio_ref_max_3_is_sentinel
    - total_plays_mean_3_is_sentinel

✅ Features finais após limpeza: 34

📋 Features com PSI alto (mantidas para avaliação no modelo):
  ⚠️ usage_intensity_tier_te: PSI = 1.1127
  ⚠️ usage_intensity_per_tenure_cap: PSI = 0.9098
  ⚠️ revenue_per_hour_tier_te: PSI = 0.8978
  ⚠️ total_secs_ratio_ref_max_6: PSI = 0.2781
```

```
In [43]: print(features_final)
```

```
['daily_revenue_efficiency', 'daily_revenue_efficiency_min_3', 'daily_revenue_efficiency_min_3_is_sentinel',
'flag_plano_mensal', 'flag_plano_mensal_lag_1', 'flag_plano_mensal_max_3', 'flag_valid_fee_max_3', 'is_auto_renew',
'log_total_plays', 'log_total_plays_mean_3', 'log_total_plays_mean_3_is_sentinel', 'margem_liquida_mensal', 'num_25',
'num_50', 'num_75', 'num_985', 'plays_per_unq_cap_min_6', 'plays_per_unq_cap_min_6_is_sentinel',
'revenue_per_hour_listened_cap', 'total_secs_mean_3', 'total_secs_mean_3_is_sentinel', 'total_secs_ratio_ref_max_6',
'total_secs_ratio_ref_max_6_is_sentinel', 'usage_intensity_per_tenure_cap', 'avg_secs_per_unq_cap_group_te',
'early_drop_rate_group_te', 'faixa_idade_te', 'gender_clean_te', 'plays_behavior_vs_completion_collapsed_te',
'plays_per_unq_behavior_te', 'registered_via_group_te', 'revenue_per_hour_tier_te', 'total_plays_group_te',
'usage_intensity_tier_te']
```

Existem variáveis que são redundantes. Estas vou remover manualmente, buscando evitar o overfitting

Conclusão

```
In [44]: print("\n" + "=" * 80)
print("CONSOLIDAÇÃO E PREPARAÇÃO DO DATAFRAME FINAL")
print("=" * 80)

# Definir colunas de controle (IDs, Target, Partição)
control_cols = ['msno', 'safra', 'target_win', 'partition']

# Remoção manual adicional. Técnicas não capturaram, mas as variáveis em questão poderiam ocasionar overfitting por redundância
features_to_remove_manual = [
    # Categorias como números, ocasionando ordenação que não existe (ex: 1 < 2 < 3 < 4 < 5), e alta cardinalidade
    'payment_method_id',
    # Redundante com flag_plano_mensal (referencia) e flag_plano_mensal_max_3 (tendência em 3m).
    # A ideia de manter as duas acima se dá porque apesar de variáveis transformadas serem mais estáveis,
    # existe a possibilidade do modelo melhor lidar com o valor no mês referência.
    'flag_plano_mensal_lag_1',
    # Redundante com registered_via_group_te
    'is_auto_renew',
]
]

features_final = [c for c in features_final if c not in features_to_remove_manual]

# Montar lista final de colunas para o DataFrame
# Garantimos que as colunas de controle estejam presentes
cols_to_select = control_cols + features_final

# Criar o DataFrame Final (Master Selecionado)
# Este DF será usado para o LightGBM e Random Forest
df_final_model = df_master.select(*cols_to_select)

# Resumo Estatístico da Seleção
total_inicial = len(num_cols + te_cols)
total_final = len(features_final)

print(f"RESUMO DA SELEÇÃO DE FEATURES:")
print(f"- Features Iniciais: {total_inicial}")
print(f"- (-) Filtro Variância (CV): {len(low_variance_features)}")
print(f"- (-) Filtro Correlação: {len(features_to_remove_corr)}")
print(f"- (-) Limpeza de Sentinelas: {len(orphan_sentinels_removed)}")
print(f"- (-) Remoção Manual: {len(features_to_remove_manual)}")
print(f"- ( Registrado, não filtrado) PSI Alto: {len(unstable_features)}")
print(f"- ( Features Finais: {total_final} )")
print(f"- ( Redução Total: {100 * (1 - total_final / total_inicial):.1f}% )")

# 6. Verificar integridade
print("\nVerificando integridade do DataFrame Final...")
print(f"- Linhas totais: {df_final_model.count():,}")
print(f"- Colunas totais: {len(df_final_model.columns)}")
```

```
=====
CONSOLIDAÇÃO E PREPARAÇÃO DO DATAFRAME FINAL
=====
```

```
RESUMO DA SELEÇÃO DE FEATURES:
```

- Features Iniciais: 57
- (-) Filtro Variância (CV): 3
- (-) Filtro Correlação: 18
- (-) Limpeza de Sentinelas: 2
- (-) Remoção Manual: 3
- (Registrado, não filtrado) PSI Alto: 4
- (Features Finais: 32
- (Redução Total: 43.9%

```
Verificando integridade do DataFrame Final...
```

- Linhas totais: 9,678,866
- Colunas totais: 36

```
In [45]: print(features_final)
```

```
['daily_revenue_efficiency', 'daily_revenue_efficiency_min_3', 'daily_revenue_efficiency_min_3_is_sentinel',
'flag_plano_mensal', 'flag_plano_mensal_max_3', 'flag_valid_fee_max_3', 'log_total_plays', 'log_total_plays_mean_3',
'log_total_plays_mean_3_is_sentinel', 'margem_liquida_mensal', 'num_25', 'num_50', 'num_75', 'num_985',
'plays_per_unq_cap_min_6', 'plays_per_unq_cap_min_6_is_sentinel', 'revenue_per_hour_listened_cap', 'total_secs_mean_3',
'total_secs_mean_3_is_sentinel', 'total_secs_ratio_ref_max_6', 'total_secs_ratio_ref_max_6_is_sentinel',
'usage_intensity_per_tenure_cap', 'avg_secs_per_unq_cap_group_te', 'early_drop_rate_group_te', 'faixa_idade_te',
'gender_clean_te', 'plays_behavior_vs_completion_collapsed_te', 'plays_per_unq_behavior_te', 'registered_via_group_te',
'revenue_per_hour_tier_te', 'total_plays_group_te', 'usage_intensity_tier_te']
```

Manter as variáveis num_25, num_50, num_75, num_985 pode ocasionar overfitting ao modelo. Porem, se mostra interessante analisar os resultados de feature importance dessas variáveis ao treinar um LightGBM _baseline_.

```
In [46]: features_final = ['daily_revenue_efficiency', 'daily_revenue_efficiency_min_3', 'daily_revenue_efficiency_min_3_is_sentinel',
'flag_plano_mensal', 'flag_plano_mensal_max_3', 'flag_valid_fee_max_3', 'log_total_plays', 'log_total_plays_mean_3',
'log_total_plays_mean_3_is_sentinel', 'margem_liquida_mensal', 'num_25', 'num_50', 'num_75', 'num_985', 'plays_per_unq_cap_min_6',
'plays_per_unq_cap_min_6_is_sentinel', 'revenue_per_hour_listened_cap', 'total_secs_mean_3', 'total_secs_mean_3_is_sentinel',
'total_secs_ratio_ref_max_6', 'total_secs_ratio_ref_max_6_is_sentinel', 'usage_intensity_per_tenure_cap',
'avg_secs_per_unq_cap_group_te', 'early_drop_rate_group_te', 'faixa_idade_te', 'gender_clean_te',
'plays_behavior_vs_completion_collapsed_te', 'plays_per_unq_behavior_te', 'registered_via_group_te', 'revenue_per_hour_tier_te',
'total_plays_group_te', 'usage_intensity_tier_te']
```

```
In [49]: from collections import Counter

dups = [c for c, n in Counter(features_final).items() if n > 1]
print("Duplicadas em features_final:", dups)
```

```
Duplicadas em features_final: []
```

```
In [50]: df_cols = df_final_model.columns
dups_df = [c for c, n in Counter(df_cols).items() if n > 1]
print("Duplicadas no DataFrame:", dups_df)
```

```
Duplicadas no DataFrame: []
```

```
In [51]: # Gold path
gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"

# Salvar dataset final
print("\n💾 Salvando df_final_model...")

output_path = gold_path + "df_master_lightgbm"
df_final_model.write.mode("overwrite").parquet(output_path)

print(f"✅ Dataset final salvo em: {output_path}")
```

```
💾 Salvando df_final_model...
✅ Dataset final salvo em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_master_lightgbm
```

12.7. Decisão por modelo: Random Forest Regressor

Utilização de uma Única ABT

```
#### Objetivo
```

Considerando a finalidade de comparar o desempenho preditivo de dois algoritmos baseados em árvores — Random Forest Regressor e LightGBM Regressor — na previsão da margem líquida mensal (correspondente ao valor observado no mês subsequente ao mês de referência), a decisão que garante uma comparação metodologicamente correta entre os modelos depende da utilização de uma única ABT como fonte de dados para ambos.

A base analítica passou por um processo estruturado de preparação, contemplando:

- * Remoção de variáveis com variância quase zero, reduzindo atributos com baixo poder informacional;
- * Análise de correlação com a variável target, priorizando variáveis com maior potencial preditivo;
- * Tratamento de valores ausentes e sentinelas (ex.: -99999, -99998) em variáveis numéricas;
- * Codificação de variáveis categóricas via Target Encoding, transformando categorias em valores numéricos baseados no comportamento histórico da variável alvo;
- * Avaliação de estabilidade das variáveis via PSI (Population Stability Index) dentro da base de treino, visando identificar possíveis distribuições instáveis;
- * O resultado desse processo foi uma ABT consistente, numérica e adequada para algoritmos baseados em árvores de decisão.

Justificativa

A utilização da mesma base de dados e do mesmo pré-processamento para os dois algoritmos é uma prática recomendada em estudos comparativos de modelos de ML, pois garante:

3.1 Comparabilidade justa

Ao manter fixos as variáveis de entrada, a variável alvo, os tratamentos aplicados e a divisão entre treino, validação e teste, assegura-se que qualquer diferença de desempenho observada seja decorrente exclusivamente do algoritmo, e não de variações na base de dados.

3.2 Controle experimental

Esse procedimento caracteriza um ambiente controlado, no qual a ABT funciona como constante e o modelo se torna a única variável do experimento. Isso permite uma avaliação técnica clara sobre qual abordagem (bagging no Random Forest ou boosting no LightGBM) se adapta melhor ao problema proposto.

A estrutura final da ABT é especialmente compatível com modelos de árvore porque:

- * Todos os atributos encontram-se em formato numérico, requisito essencial para os algoritmos utilizados (**sendo necessário somente o tratamento de nulos para Random Forest**);
- * Modelos baseados em árvores não exigem normalização ou padronização das variáveis;
- * Esses modelos lidam bem com relações não lineares, interações entre variáveis e outliers "moderados".

Além disso, o uso de Target Encoding é apropriado para ambos os algoritmos, pois fornece uma representação numérica informativa das categorias, potencialmente aumentando o poder preditivo.

Diferenças esperadas entre os algoritmos

Embora utilizem a mesma base, os algoritmos possuem estratégias distintas de aprendizado:

Característica Random Forest LightGBM		
----- ----- -----		
Estratégia Bagging (múltiplas árvores independentes) Boosting (árvore sequencial que corrigem erros anteriores)		
Sensibilidade a ruído Menor Maior, porém com maior capacidade de ajuste		
Capacidade de modelar padrões complexos Alta Muito alta		
Eficiência computacional Moderada Elevada		

Conclusão

O uso da mesma ABT para treinar e avaliar os modelos Random Forest e LightGBM é tecnicamente adequado e metodologicamente correto para fins de comparação de desempenho. Essa abordagem garante que as diferenças observadas nos resultados refletem exclusivamente as características de cada algoritmo, proporcionando uma análise justa, controlada e confiável.

Carregando base

```
In [52]: gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"
df_master_lightgbm = spark.read.parquet(gold_path + "df_master_lightgbm")
```

```
In [53]: df_master_lightgbm.count()
```

```
Out[53]: 9678866
```

```
In [54]: df_master_lightgbm.groupBy("partition").count().show()
```

partition	count
train	6262831
oot	1850732
test	1565303

In [55]: df_master_lightgbm.printSchema()

```
root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- target_win: double (nullable = true)
 |-- partition: string (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- daily_revenue_efficiency_min_3_is_sentinel: byte (nullable = true)
 |-- flag_plano_mensal: byte (nullable = true)
 |-- flag_plano_mensal_max_3: byte (nullable = true)
 |-- flag_valid_fee_max_3: byte (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- log_total_plays_mean_3_is_sentinel: byte (nullable = true)
 |-- margem_liquida_mensal: double (nullable = true)
 |-- num_25: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- plays_per_unq_cap_min_6: double (nullable = true)
 |-- plays_per_unq_cap_min_6_is_sentinel: byte (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- total_secs_mean_3_is_sentinel: byte (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- total_secs_ratio_ref_max_6_is_sentinel: byte (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- avg_secs_per_unq_cap_group_te: double (nullable = true)
 |-- early_drop_rate_group_te: double (nullable = true)
 |-- faixa_idade_te: double (nullable = true)
 |-- gender_clean_te: double (nullable = true)
 |-- plays_behavior_vs_completion_collapsed_te: double (nullable = true)
 |-- plays_per_unq_behavior_te: double (nullable = true)
 |-- registered_via_group_te: double (nullable = true)
 |-- revenue_per_hour_tier_te: double (nullable = true)
 |-- total_plays_group_te: double (nullable = true)
 |-- usage_intensity_tier_te: double (nullable = true)
```

Identificando numéricas com nulos em df_master_lightgbm

```
In [57]:  
print("\n" + "=" * 80)  
print("VERIFICAÇÃO DE NULOS EM VARIÁVEIS NUMÉRICAS (df_master_lightgbm)")  
print("=" * 80)  
  
# 1) Identificar colunas numéricas no df_master_lightgbm  
numeric_types = (T.IntegerType, T.LongType, T.FloatType, T.DecimalType, T.ShortType)  
  
numeric_cols_all = [  
    f.name for f in df_master_lightgbm.schema.fields  
    if isinstance(f.dataType, numeric_types)  
]  
  
print(f"⭐ Total de colunas numéricas detectadas: {len(numeric_cols_all)}")  
  
# Se você quiser restringir apenas às features de modelagem (e tiver a lista):  
# numeric_cols_model = [c for c in features_final if c in numeric_cols_all]  
# Caso contrário, usamos todas as numéricas:  
numeric_cols_model = numeric_cols_all  
  
print(f"⭐ Colunas numéricas consideradas para verificação: {len(numeric_cols_model)}")  
  
# 2) Contagem de nulos por coluna  
exprs_nulls = [F.sum(F.col(c).isNull().cast("int")).alias(c) for c in numeric_cols_model]  
df_null_counts = df_master_lightgbm.select(exprs_nulls)  
  
null_counts_row = df_null_counts.collect()[0].asDict()  
  
total_rows = df_master_lightgbm.count()  
  
null_report = []  
for col, n_nulls in null_counts_row.items():  
    n_nulls = int(n_nulls)  
    if n_nulls > 0:  
        perc = n_nulls / total_rows  
        null_report.append({  
            "feature": col,  
            "n_nulls": n_nulls,  
            "perc_nulls": perc  
        })  
  
import pandas as pd  
  
if null_report:  
    df_null_report = pd.DataFrame(null_report).sort_values("perc_nulls", ascending=False)  
  
    print(f"\n⚠️ Variáveis numéricas com nulos: {len(df_null_report)}")  
    print(df_null_report.to_string(index=False, formatters={  
        "perc_nulls": lambda x: f"{x:.4%}"  
    }))  
else:  
    df_null_report = pd.DataFrame(columns=["feature", "n_nulls", "perc_nulls"])  
    print("\n✅ Nenhuma variável numérica com nulos encontrada.")
```

```
=====  
VERIFICAÇÃO DE NULOS EM VARIÁVEIS NUMÉRICAS (df_master_lightgbm)  
=====  
⭐ Total de colunas numéricas detectadas: 26  
⭐ Colunas numéricas consideradas para verificação: 26  
  
⚠️ Variáveis numéricas com nulos: 5  
      feature  n_nulls  perc_nulls  
daily_revenue_efficiency_min_3  2314344   23.9113%  
      plays_per_unq_cap_min_6  2216132   22.8966%  
      total_secs_ratio_ref_max_6  2200868   22.7389%  
      log_total_plays_mean_3  1914940   19.7848%  
      total_secs_mean_3  1914940   19.7848%
```

Tratando numéricas com nulos em df_master_lightgbm para construir df_master_random_forest

```
#### Contexto
```

Por que manter as Sentinelas?

As variáveis com nulos têm entre **20% a 24% de nulos**.

* Semântica: O fato de ser nulo nessas variáveis provavelmente significa que o usuário não teve atividade suficiente naquele período (ex: não deu play em nada nos últimos 3 ou 6 meses).

O papel da Sentinel: A flag `_is_sentinel` diz ao modelo: "Ei, ignore o valor numérico aqui, este dado é um 'buraco' no histórico".

* O papel da Imputação: Como a Random Forest do Sklearn não aceita o "buraco" (NaN), precisamos preencher com um número.

Qual a melhor estratégia de tratamento para esses 5 casos?

Tendo as flags sentinelas para essas 5 variáveis, a melhor abordagem não é a mediana, mas sim preenchimento com zero.

Por que Zero e não Mediana?

1. Consistência: Imputar a mediana implica dizer ao modelo que um usuário que não teve atividade (nulo) se comporta como um usuário "médio". Isso é mentira e confunde o modelo.

2. Destaque: Ao usar zero e manter as respectivas `_is_sentinel`, o modelo de árvore "ganha" duas chances de acertar:

* Pela Flag: `if is_sentinel == 1 -> Margem provável.`

* Pelo Valor: `if feature <= 0 -> Margem provável.`

Execucao

```
In [58]: print("\n" + "=" * 80)
print("IMPUTAÇÃO ESTRATÉGICA (ZERO + PROTEÇÃO DE SENTINELA)")
print("=" * 80)

# Lista das colunas críticas identificadas
cols_com_nulos = [
    'daily_revenue_efficiency_min_3',
    'plays_per_unq_cap_min_6',
    'total_secs_ratio_ref_max_6',
    'log_total_plays_mean_3',
    'total_secs_mean_3'
]

df_temp = df_master_lightgbm

for col in cols_com_nulos:
    print(f"Imputando {col} com 0.0 (Representando ausência de atividade)...")
    df_temp = df_temp.withColumn(col, F.when(F.col(col).isNull(), F.lit(0.0)).otherwise(F.col(col)))

# Verificação final se sobrou algum nulo em outras colunas (caso existam)
# Se houver outras, usamos a mediana como "rede de segurança"
remaining_nulls = [c for c in numeric_cols_model if c not in cols_com_nulos]

for col in remaining_nulls:
    null_count = df_master_lightgbm.filter(F.col(col).isNull()).count()
    if null_count > 0:
        mediana_val = df_temp.approxQuantile(col, [0.5], 0.01)[0]
        print(f"⚠️ Coluna secundária {col} com {null_count} nulos. Imputando com mediana: {mediana_val}")
        df_temp = df_temp.withColumn(col, F.when(F.col(col).isNull(), F.lit(mediana_val)).otherwise(F.col(col)))

df_master_random_forest = df_temp

print("\n✅ Imputação concluída. As flags '_is_sentinel' foram preservadas para manter a semântica.")
print("✅ O DataFrame está pronto para Random Forest (sem NaNs) e LightGBM.")
print("=" * 80)
```

```
=====
IMPUTAÇÃO ESTRATÉGICA (ZERO + PROTEÇÃO DE SENTINELA)
=====
Imputando daily_revenue_efficiency_min_3 com 0.0 (Representando ausência de atividade)...
Imputando plays_per_unq_cap_min_6 com 0.0 (Representando ausência de atividade)...
Imputando total_secs_ratio_ref_max_6 com 0.0 (Representando ausência de atividade)...
Imputando log_total_plays_mean_3 com 0.0 (Representando ausência de atividade)...
Imputando total_secs_mean_3 com 0.0 (Representando ausência de atividade)...

✅ Imputação concluída. As flags '_is_sentinel' foram preservadas para manter a semântica.
✅ O DataFrame está pronto para Random Forest (sem NaNs) e LightGBM.
=====
```

Validação

```
In [59]: print("\n" + "=" * 80)
print("VALIDAÇÃO DE NULOS EM VARIÁVEIS NUMÉRICAS (df_master_random_forest)")
print("=" * 80)

# 1) Identificar colunas numéricas no df_master_random_forest
numeric_types = (T.IntegerType, T.LongType, T.FloatType, T.DoubleType, T.DecimalType, T.ShortType)

numeric_cols_all = [
    f.name for f in df_master_random_forest.schema.fields
    if isinstance(f.dataType, numeric_types)
]

print(f"⭐ Total de colunas numéricas detectadas: {len(numeric_cols_all)}")

# Se você quiser restringir apenas às features de modelagem (e tiver a lista):
# numeric_cols_model = [c for c in features_final if c in numeric_cols_all]
# Caso contrário, usamos todas as numéricas:
numeric_cols_model = numeric_cols_all

print(f"⭐ Colunas numéricas consideradas para verificação: {len(numeric_cols_model)}")

# 2) Contagem de nulos por coluna
exprs_nulls = [F.sum(F.col(c).isNull().cast("int")).alias(c) for c in numeric_cols_model]
df_null_counts = df_master_random_forest.select(exprs_nulls)

null_counts_row = df_null_counts.collect()[0].asDict()

total_rows = df_master_random_forest.count()

null_report = []
for col, n_nulls in null_counts_row.items():
    n_nulls = int(n_nulls)
    if n_nulls > 0:
        perc = n_nulls / total_rows
        null_report.append({
            "feature": col,
            "n_nulls": n_nulls,
            "perc_nulls": perc
        })
    else:
        df_null_report = pd.DataFrame(null_report).sort_values("perc_nulls", ascending=False)

        print(f"\n⚠️ Variáveis numéricas com nulos: {len(df_null_report)}")
        print(df_null_report.to_string(index=False, formatters={
            "perc_nulls": lambda x: f"{x:.4%}"
        }))
else:
    df_null_report = pd.DataFrame(columns=["feature", "n_nulls", "perc_nulls"])
    print("\n✅ Nenhuma variável numérica com nulos encontrada.")

import pandas as pd
```

```
=====
VALIDAÇÃO DE NULOS EM VARIÁVEIS NUMÉRICAS (df_master_random_forest)
=====

⭐ Total de colunas numéricas detectadas: 26
⭐ Colunas numéricas consideradas para verificação: 26

✅ Nenhuma variável numérica com nulos encontrada.
```

Conclusao

```
In [60]: # Gold path
gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"

# Salvar dataset final
print("\n💾 Salvando df_master_random_forest...")

output_path = gold_path + "df_master_random_forest"
df_master_random_forest.write.mode("overwrite").parquet(output_path)

print(f"✅ Dataset final salvo em: {output_path}")
```

```
💾 Salvando df_master_random_forest...
✅ Dataset final salvo em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_master_random_forest
```

13. Modelagem e Validação: Algoritmos Supervisionados

```
In [4]: gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"
```

13.1. Modelo Baseline Linear - Regressão Linear Simples

13.1.1. Carregando ABT e separando treino, teste e valid

```
In [1]: df_master_elastic_net = spark.read.parquet(gold_path + "df_master_elastic_net")  
  
In [2]: # Criando os DataFrames de treino, teste e OOT  
df_train = df_master_elastic_net.filter(F.col("partition").isin("train"))  
df_test = df_master_elastic_net.filter(F.col("partition").isin("test"))  
df_oot = df_master_elastic_net.filter(F.col("partition").isin("oot"))  
  
print(f"Treino: {df_train.count()} | Teste: {df_test.count()} | OOT: {df_oot.count()}")  
  
Treino: 6262831 | Teste: 1565303 | OOT: 1850732
```

13.1.2. Modelo Baseline (Regressão Linear Simples)

```
In [3]: from pyspark.ml.regression import LinearRegression  
from pyspark.ml.evaluation import RegressionEvaluator  
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator  
  
In [4]: # Definindo o modelo baseline  
# regParam=0 significa sem regularização (Lasso/Ridge)  
lr_baseline = LinearRegression(featuresCol='features', labelCol='target_win', regParam=0)  
  
# Treinando  
model_baseline = lr_baseline.fit(df_train)  
  
# Fazendo previsões no teste  
predictions_baseline = model_baseline.transform(df_test)
```

13.1.2.1. Resultados

```
In [5]: # Avaliando com RMSE e MAE  
evaluator_rmse = RegressionEvaluator(labelCol="target_win", predictionCol="prediction", metricName="rmse")  
evaluator_mae = RegressionEvaluator(labelCol="target_win", predictionCol="prediction", metricName="mae")  
evaluator_r2 = RegressionEvaluator(labelCol="target_win", predictionCol="prediction", metricName="r2")  
  
rmse_base = evaluator_rmse.evaluate(predictions_baseline)  
mae_base = evaluator_mae.evaluate(predictions_baseline)  
r2_base = evaluator_r2.evaluate(predictions_baseline)  
  
print(f"--- RESULTADOS BASELINE ---")  
print(f"RMSE: {rmse_base:.4f}")  
print(f"MAE: {mae_base:.4f}")  
print(f"R²: {r2_base:.4f}")  
  
--- RESULTADOS BASELINE ---  
RMSE: 37.1594  
MAE: 20.0372  
R²: 0.6233
```

```
In [ ]: df_train.groupBy("partition").agg(F.mean(F.col("target_win"))).show()
df_test.groupBy("partition").agg(F.mean(F.col("target_win"))).show()
```

```
+-----+-----+
|partition| avg(target_win)|
+-----+-----+
|   train|43.67598854048888|
+-----+-----+  
+-----+-----+
|partition| avg(target_win)|
+-----+-----+
|   test|43.738913200408426|
+-----+-----+
```

13.1.2.2. Conclusão

Os resultados do modelo baseline revelam que a estrutura linear das features selecionadas possui sim certo poder explicativo sobre a margem líquida do mês subsequente. Isso mostra que a decisão de não atentar-se somente ao valor de mediana mas também da média na construção das variáveis + winsorização da target foram decisões válidas.

Abaixo, detalho o que cada métrica nos diz:

O Poder de Explicação ($R^2 = 0.6233$)

Um R^2 de **62,3%** indica que mais de seis décimos da variabilidade da margem líquida futura são explicados pelas variáveis atuais (como eficiência de receita, comportamento de uso e histórico de pagamentos). Para um modelo de regressão em dados comportamentais de milhões de clientes, este é um valor agradável, sugerindo que a etapa de *Feature Engineering* e a seleção por *Information Value* (IV) foram extremamente eficazes.

A Magnitude do Erro ($MAE = 20.03$)

O Erro Médio Absoluto (MAE) determina que, em média, o modelo erra a margem líquida do mês seguinte em **20 NTD**. Sendo assim, considerando uma média de 43.73, o erro do modelo médio é de aproximadamente 45.8% em relação à média. Em termos absolutos, o modelo está errando "quase metade" do valor médio da target --> alta volatilidade.

Sensibilidade a Outliers ($RSME = 37.15$)

O **RMSE de 37.15** é quase o valor da média da target e 85% maior que o MAE. Isso confirma que a base possui "caudas longas". Existem erros de predição muito grandes que estão puxando a métrica para cima. Provavelmente, o modelo está subestimando clientes com margens altíssimas ou não conseguindo prever prejuízos severos (margens muito negativas). Em uma distribuição normal, o RMSE e o MAE seriam mais próximos; essa distância revela o "grito" dos outliers.

Este cenário é o **caso de uso perfeito para a Elastic Net**:

- * A regressão linear pura (baseline) está tentando se ajustar a esses outliers para minimizar o erro quadrático, o que pode estar distorcendo os coeficientes.
- * A regularização (L1 e L2) servirá como um "freio de mão", impedindo que o modelo dê importância excessiva a variáveis que tentam explicar esses casos extremos, buscando um modelo que erre menos na "massa" de clientes (o centro da distribuição), mesmo que isso signifique ignorar os outliers.

13.2. Elastic Net

13.2.1. Treinando a Elastic Net com Cross-Validation

13.2.1.1. Configurações iniciais

```
In [10]: # 1. Definir o estimador
en = LinearRegression(featuresCol='features', labelCol='target_win')

# 2. Criar a grade de parâmetros (Grid Search) testando diferentes forças de multa e diferentes misturas
paramGrid = (ParamGridBuilder()
             .addGrid(en.regParam, [0.01, 0.1, 0.5, 1.0]) # Testa de baixa regularização a regularização forte
             .addGrid(en.elasticNetParam, [0.2, 0.5, 0.8]) # Mais Ridge a mais Lasso, incluindo o meio termo
             .build())
```

```
In [11]: # 3. Configurar o Cross-Validator
# numFolds=3 para não demorar uma eternidade com 9 milhões de linhas
cv = CrossValidator(estimator=en,
                    estimatorParamMaps=paramGrid,
                    evaluator=evaluator_rmse, # Escolher o modelo com menor RMSE na validacao cruzada, buscando penalizar erros
                    numFolds=3, # Similar a tecnica de OHE utilizada, para cada combinacao de hiperparam. FOLD 1: treina em 2+3,
                    validateIn 1...no final: media do RMSE dos 3 folds
                    parallelism=4) # Quantos modelos o motor vai treinar ao mesmo tempo

# 4. Treinar o modelo final
print("Iniciando treinamento da Elastic Net com Cross-Validation...")
cv_model = cv.fit(df_train)
print("Finalizado!")
```

```
Iniciando treinamento da Elastic Net com Cross-Validation...
Finalizado!
```

13.2.1.2. Analisando o resultado de todas as combinacoes

```
In [12]: # Criando um DataFrame com os resultados de todas as combinações
params = cv_model.getEstimatorParamMaps()
metrics = cv_model.avgMetrics

results = []
for p, m in zip(params, metrics):
    results.append({
        'regParam': p[en.regParam],
        'elasticNetParam': p[en.elasticNetParam],
        'RMSE_CV': m
    })

df_results = pd.DataFrame(results).sort_values(by='RMSE_CV')
print(df_results)
```

	regParam	elasticNetParam	RMSE_CV
1	0.01	0.5	37.222246
0	0.01	0.2	37.222254
2	0.01	0.8	37.222309
3	0.10	0.2	37.222975
4	0.10	0.5	37.225236
5	0.10	0.8	37.229027
6	0.50	0.2	37.238331
7	0.50	0.5	37.265339
9	1.00	0.2	37.270507
8	0.50	0.8	37.305334
10	1.00	0.5	37.346134
11	1.00	0.8	37.440051

Resultados extremamente parecidos com o baseline.

13.2.1.3. Enxergando o melhor modelo com base em df_test

```
In [13]: # 5. Melhor modelo e métricas
best_model = cv_model.bestModel
print(f"Melhor regParam: {best_model._java_obj.getRegParam()}")
print(f"Melhor elasticNetParam: {best_model._java_obj.getElasticNetParam()}")
```

```
Melhor regParam: 0.01
Melhor elasticNetParam: 0.5
```

```
In [14]: # Fazendo previsões no df_test com o melhor modelo
predictions_en = best_model.transform(df_test)

# Avaliando
evaluator_rmse = RegressionEvaluator(labelCol="target_win", predictionCol="prediction", metricName="rmse")
evaluator_mae = RegressionEvaluator(labelCol="target_win", predictionCol="prediction", metricName="mae")
evaluator_r2 = RegressionEvaluator(labelCol="target_win", predictionCol="prediction", metricName="r2")

print(f"--- RESULTADOS ELASTIC NET (MELHOR MODELO) ---")
print(f"RMSE: {evaluator_rmse.evaluate(predictions_en):.4f}")
print(f"MAE: {evaluator_mae.evaluate(predictions_en):.4f}")
print(f"R²: {evaluator_r2.evaluate(predictions_en):.4f}")
```

```
--- RESULTADOS ELASTIC NET (MELHOR MODELO) ---
RMSE: 37.1594
MAE: 20.0390
R²: 0.6233
```

Por que ficou idêntico?

O `regParam` vencedor foi **0.01** (o menor valor na grade). Uma multa de 0.01 em um dataset de 1.5 milhões de linhas é tão pequena que o modelo se comporta quase exatamente como uma Regressão Linear comum (o seu baseline).

O que isso permite pressupor sobre os dados utilizados?

- Sinal Forte e Estável:** As etapas anteriores de seleção de variáveis (IV, Correlação, Filtro de Variância) foram bem feitas que já foi entregue ao modelo apenas o "filé" dos dados. Não sobrou ruído ou redundância para a Elastic Net precisar "limpar";
- Ausência de Overfitting:** Se o modelo com regularização não conseguiu bater o modelo sem regularização, significa que o baseline não está sofrendo de overfitting. Ele está capturando padrões reais que se repetem no treino e no teste;
- Linearidade:** O fenômeno da margem líquida, dentro do que é explicável, é sim linear. Tentar "forçar" uma simplificação via L1 ou encolhimento via L2 não traz ganho porque o modelo já está no seu limite de aprendizado.

13.2.2. Análise de Coeficientes do Melhor Modelo

13.2.2.1. Extraindo os coeficientes

```
In [15]: # Tenta pegar o metadata cru
metadata = df_master_elastic_net.schema["features"].metadata

# Vamos inspecionar o que tem dentro
print("Chaves do metadata:", metadata.keys())

# Tenta acessar de várias formas possíveis
if "ml_attr" in metadata:
    ml_attr = metadata["ml_attr"]
    print("Chaves de ml_attr:", ml_attr.keys())

    # Tenta pegar attrs
    if "attrs" in ml_attr:
        attrs = ml_attr["attrs"]
        print("Tipos de attrs:", attrs.keys())

        # Junta numeric + binary + nominal
        from itertools import chain
        all_attrs = list(chain(*attrs.values()))
        feature_names = [a["name"] for a in sorted(all_attrs, key=lambda x: x["idx"])]

    elif "numAttrs" in ml_attr:
        # Às vezes vem só o número de atributos
        numAttrs = ml_attr["numAttrs"]
        feature_names = [f"f_{i}" for i in range(numAttrs)]
    else:
        feature_names = None
else:
    feature_names = None

print("Nomes extraídos:", feature_names)

Chaves do metadata: dict_keys(['ml_attr'])
Chaves de ml_attr: dict_keys(['numAttrs'])
Nomes extraídos: ['f_0', 'f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10', 'f_11', 'f_12', 'f_13', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19', 'f_20', 'f_21', 'f_22', 'f_23', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28', 'f_29', 'f_30', 'f_31', 'f_32', 'f_33', 'f_34', 'f_35', 'f_36', 'f_37', 'f_38', 'f_39', 'f_40', 'f_41', 'f_42', 'f_43', 'f_44', 'f_45', 'f_46', 'f_47', 'f_48', 'f_49', 'f_50', 'f_51', 'f_52', 'f_53', 'f_54', 'f_55', 'f_56', 'f_57', 'f_58', 'f_59', 'f_60', 'f_61', 'f_62']
```

```
In [16]: print(df_master_elastic_net.columns)

['msno', 'target_win', 'features', 'daily_revenue_efficiency_min_3', 'revenue_per_hour_listened_cap', 'actual_amount_paid', 'flag_plano_mensal_max_3', 'log_total_secs_mean_3', 'num_unq', 'usage_intensity_per_tenure_cap', 'payment_method_group_ohe', 'revenue_tier_ohe', 'revenue_per_hour_tier_ohe', 'plays_behavior_vs_volume_collapsed_ohe', 'plays_behavior_vs_completion_collapsed_ohe', 'usage_intensity_tier_ohe', 'registered_via_group_ohe', 'faixa_idade_ohe', 'partition', 'safra']
```

Numéricas (primeiro)

idx Feature	
----: -----	
0 daily_revenue_efficiency_min_3	
1 revenue_per_hour_listened_cap	
2 actual_amount_paid	
3 flag_plano_mensal_max_3	
4 log_total_secs_mean_3	
5 num_unq	
6 usage_intensity_per_tenure_cap	

A partir do idx 7 → **categorias OHE**
Cada OHE vira **várias dimensões**, somando até 63:

```
| idx inicial | OHE |
|-----:|----|
| 7 | payment_method_group_ohe |
| ... | revenue_tier_ohe |
| ... | revenue_per_hour_tier_ohe |
| ... | plays_behavior_vs_volume_collapsed_ohe |
| ... | plays_behavior_vs_completion_collapsed_ohe |
| ... | usage_intensity_tier_ohe |
| ... | registered_via_group_ohe |
| ... | faixa_idade_ohe |
```

13.2.2.2. Resultados + Conclusao

```
In [17]: coeffs = best_model.coefficients.toArray()

df_coeffs = pd.DataFrame({
    "idx": range(len(coeffs)),
    "coefficient": coeffs,
    "abs_coefficient": np.abs(coeffs)
}).sort_values("abs_coefficient", ascending=False)

print("Intercept:", best_model.intercept)
df_coeffs.head(20)
```

Intercept: 45.24295344822473

Out[17]:		idx	coefficient	abs_coefficient
	3	3	30.811709	30.811709
	5	5	-11.816001	11.816001
	12	12	-11.118940	11.118940
	11	11	9.112368	9.112368
	15	15	8.306271	8.306271
	18	18	-4.397499	4.397499
	4	4	-4.313835	4.313835
	7	7	4.134709	4.134709
	2	2	-3.676072	3.676072
	9	9	-3.608015	3.608015
	20	20	2.563233	2.563233
	25	25	2.440682	2.440682
	19	19	2.134992	2.134992
	47	47	-1.359741	1.359741
	16	16	-1.343261	1.343261
	23	23	-1.237598	1.237598
	24	24	1.195239	1.195239
	13	13	-1.062525	1.062525
	8	8	-1.062525	1.062525
	31	31	-1.051510	1.051510

idx	Feature	Interpretação
3	flag_plano_mensal_max_3	Maior impacto positivo → clientes com plano mensal recorrente têm forte aumento de margem
5	num_unq	Impacto negativo → comportamento errático / pouco engajamento prejudica margem
2	actual_amount_paid	Negativo (interessante!) → pagamento maior pode estar associado a promoções/descontos
4	log_total_secs_mean_3	Negativo → consumo alto ≠ margem alta (custo variável)
6	usage_intensity_per_tenure_cap	Negativo → heavy users podem pressionar custos
7+	OHE	Efeitos marginais de perfil / canal / comportamento

✓ Conclusão forte:

O modelo capturou **trade-off entre uso e custo**, algo extremamente realista para streaming / assinatura.

Intercepto: está correto?

O intercepto de 45.24 deve ser comparado com a média da target, 43.73. Por serem próximos, o intercepto do modelo é consistente com a média da margem líquida, indicando que o modelo está bem calibrado. As variáveis explicativas atuam como ajustes marginais positivos ou negativos em torno desse valor médio.

13.2.3. OLS assumptions

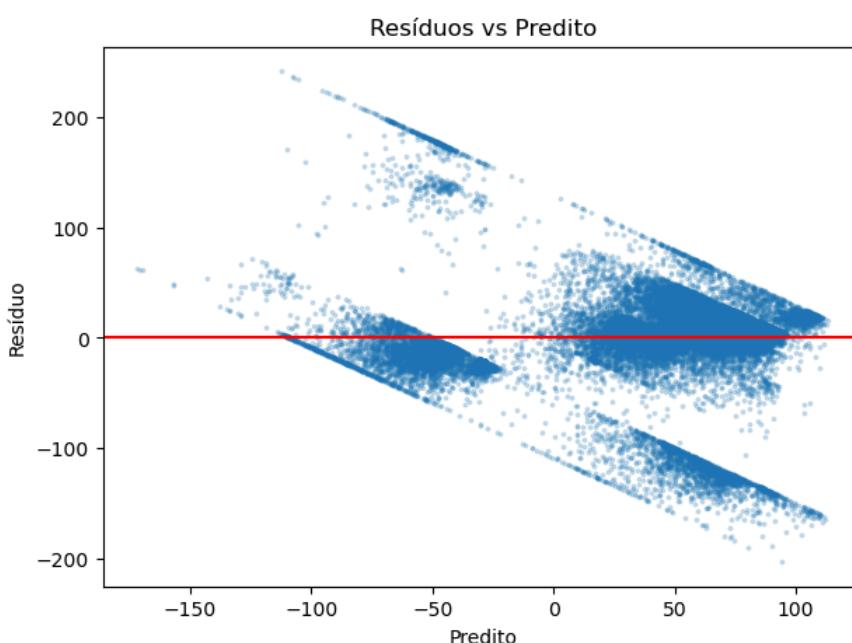
As suposições clássicas do OLS são avaliadas como diagnóstico de qualidade do modelo. Dado o foco preditivo, o grande volume de dados e o uso de regularização, eventuais violações **não comprometem** a validade prática do modelo.

13.2.3.1. Amostra + Resíduos

```
In [ ]: pred_test = best_model.transform(df_test).select("target_win", "prediction", "safra")
pred_test = pred_test.withColumn("residual", F.col("target_win") - F.col("prediction"))
pred_test_s = pred_test.sample(False, 0.03, seed=42)
pred_pd = pred_test_s.toPandas()
```

13.2.3.2. Linearidade — Resíduos vs Preditivo

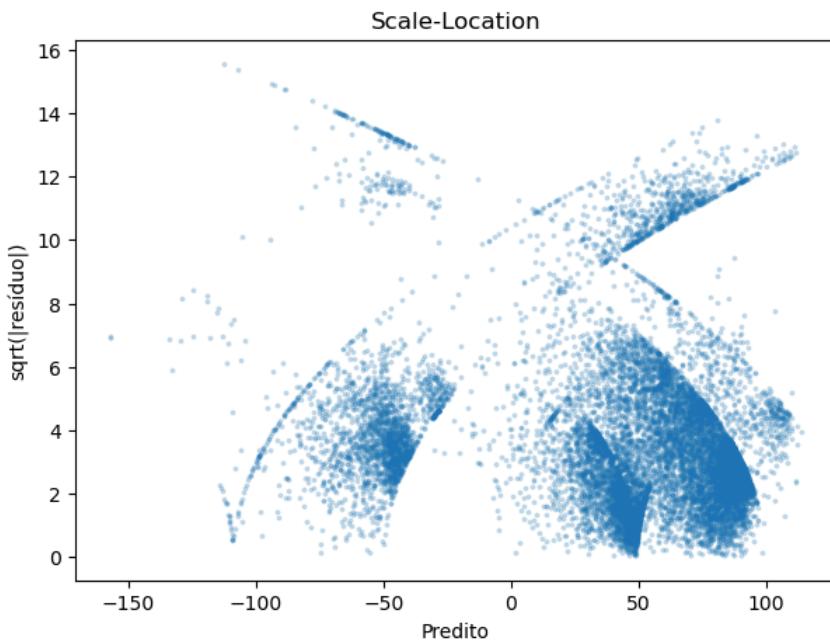
```
In [ ]: plt.figure(figsize=(7,5))
plt.scatter(pred_pd["prediction"], pred_pd["residual"], s=3, alpha=0.2)
plt.axhline(0, color="red")
plt.xlabel("Preditivo")
plt.ylabel("Resíduo")
plt.title("Resíduos vs Preditivo")
plt.show()
```



No gráfico de Resíduos vs Preditos (linearidade), há padrões estruturados claros — faixas inclinadas e agrupamentos — indicando que a relação entre preditores e resposta não é perfeitamente linear. Em um contexto de inferência OLS isso seria uma violação séria, pois os erros não estão distribuídos aleatoriamente em torno de zero ao longo de todo o domínio das previsões. Porém, sob a ótica preditiva, o impacto prático é limitado: o Elastic Net (assim como a regressão linear baseline) já está extraíndo praticamente todo o sinal linear disponível, e o fato de ambos terem apresentado RMSE idêntico sugere que eventuais não linearidades restantes têm baixo ganho marginal de modelagem ou estão diluídas no ruído. Ou seja, há evidência de não linearidade estrutural, mas ela não está se traduzindo em perda relevante de desempenho preditivo.

13.2.3.3. Homoscedasticidade — Scale-Location

```
In [ ]: plt.figure(figsize=(7,5))
plt.scatter(
    pred_pd["prediction"],
    np.sqrt(np.abs(pred_pd["residual"])),
    s=3,
    alpha=0.2
)
plt.xlabel("Preditos")
plt.ylabel("sqrt(|resíduo|)")
plt.title("Scale-Location")
plt.show()
```



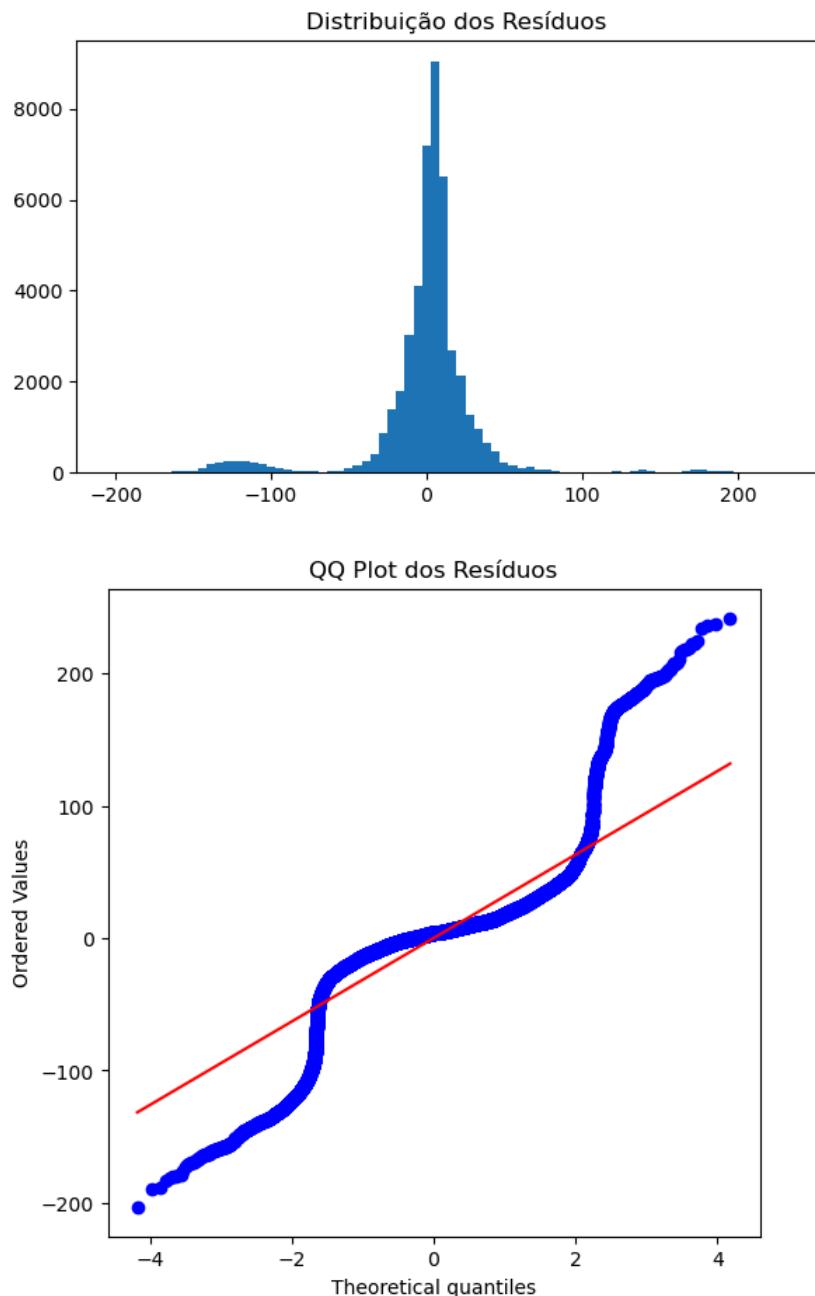
Observa-se aumento e variação irregular da dispersão dos resíduos ao longo dos valores preditos, caracterizando heterocedasticidade. Em OLS clássico isso afetaria a eficiência dos estimadores e a validade de testes t/F, mas não gera viés nas previsões médias. Como o foco aqui é previsão e o modelo foi estimado com regularização (Elastic Net) e grande volume amostral, essa violação tem efeito principalmente inferencial, não preditivo. O padrão indica que existem regiões do espaço de previsão com maior incerteza, mas o erro médio agregado (RMSE) já captura isso; não há evidência de instabilidade que comprometa o uso prático do modelo.

13.2.3.4. Normalidade dos resíduos — Histograma + QQ Plot

```
In [ ]: import scipy.stats as stats
```

```
plt.figure(figsize=(7,4))
plt.hist(pred_pd["residual"], bins=80)
plt.title("Distribuição dos Resíduos")
plt.show()

plt.figure(figsize=(6,6))
stats.probplot(pred_pd["residual"], dist="norm", plot=plt)
plt.title("QQ Plot dos Resíduos")
plt.show()
```



Quanto à distribuição dos resíduos, o histograma mostra forte concentração próxima de zero com caudas longas e assimetria, indicando que os erros não seguem normalidade. O QQ-plot reforça isso de forma clara: há desvios acentuados nas caudas (curvatura em "S"), evidenciando presença de outliers. Para OLS inferencial isso violaria a suposição de normalidade dos erros, mas em modelos voltados à previsão essa condição não é necessária para que as estimativas pontuais sejam válidas. Com grande amostra, a média dos erros tende a se estabilizar (Lei dos Grandes Números), e o fato de o desempenho do Elastic Net não superar o baseline indica que a não-normalidade está mais ligada à natureza do fenômeno (ruído assimétrico ou eventos extremos) do que a falha estrutural do modelo.

```
#### 13.2.3.5. Independência temporal — resíduos por safra
```

```
In [ ]: res_by_safra = (
    pred_test
    .groupBy("safra")
    .agg(
        F.mean("residual").alias("mean_residual"),
        F.mean(F.abs("residual")).alias("mae_like")
    )
    .orderBy("safra")
)

res_by_safra.toPandas()
```

	safra	mean_residual	mae_like
0	201601	-0.809764	34.322160
1	201602	-2.607234	22.926813
2	201603	1.426307	19.150281
3	201604	-2.301099	19.328098
4	201605	2.065811	17.299251
5	201606	-2.638728	19.489024
6	201607	3.474800	16.225053
7	201608	2.801835	15.582985
8	201609	-1.480842	16.673730

Por fim, avaliando a independência temporal, a tabela por safra mostra médias de resíduos oscilando ao redor de zero, sem tendência persistente de viés positivo ou negativo ao longo do tempo. Os valores de `mean_residual` alternam sinal e permanecem de baixa magnitude relativa, enquanto o `mae_like` apresenta leve variação mas sem explosões ou degradação progressiva. Isso sugere ausência de autocorrelação temporal forte ou drift sistemático, indicando que o modelo mantém estabilidade preditiva entre períodos. Em termos práticos, isso é um ponto positivo: mesmo com violações de pressupostos clássicos (linearidade perfeita, homocedasticidade e normalidade), não há evidência de deterioração temporal do erro, reforçando que o modelo é estatisticamente imperfeito sob a ótica clássica, mas operacionalmente consistente para fins preditivos.

13.2.4. Validação Out-Of-Time

13.2.4.1. Métricas melhor modelo em `df_oot`

```
In [18]: # Avaliando no Out-Of-Time (Meses de Outubro e Novembro)
predictions_oot = best_model.transform(df_oot)

rmse_oot = evaluator_rmse.evaluate(predictions_oot)
mae_oot = evaluator_mae.evaluate(predictions_oot)
r2_oot = evaluator_r2.evaluate(predictions_oot)

print(f"--- RESULTADOS NO OUT-OF-TIME (OOT) ---")
print(f"RMSE: {rmse_oot:.4f}")
print(f"MAE: {mae_oot:.4f}")
print(f"R²: {r2_oot:.4f}")

--- RESULTADOS NO OUT-OF-TIME (OOT) ---
RMSE: 32.2688
MAE: 16.6188
R²: 0.6329
```

13.2.4.2. Interpretando e estudando os resultados

```
In [19]: def describe_target(df, name):
    stats = (df
        .select(
            F.lit(name).alias("set"),
            F.mean("target_win").alias("mean"),
            F.stddev("target_win").alias("std"),
            F.expr("percentile_approx(target_win, 0.01)").alias("p01"),
            F.expr("percentile_approx(target_win, 0.50)").alias("p50"),
            F.expr("percentile_approx(target_win, 0.99)").alias("p99"),
        ))
    return stats

desc = describe_target(df_test, "test").unionByName(describe_target(df_oot, "oot"))
desc.show(truncate=False)
```

set	mean	std	p01	p50	p99
test	43.738913200408426	60.54488958218138	-108.8147222	60.482212700000005	129.0401291
oot	51.5203238170067	53.255595469834894	-97.1938973	60.8371468	127.9270382

O modelo apresenta desempenho **superior** no conjunto Out-of-Time quando comparado ao conjunto de teste, tanto em termos de erro absoluto quanto de poder explicativo. Isso é um fortíssimo indicativo de robustez e ausência de overfitting. Em projetos reais, é mais comum o OOT piorar; aqui ocorre o contrário. Qual o motivo?

1. Média maior no OOT

- test mean ≈ 43.7
- oot mean ≈ 51.5

OOT tem clientes mais lucrativos em média, o que facilita a predição.

2. Menor variância no OOT

- std test ≈ 60.5
- std oot ≈ 53.3

OOT é menos volátil, com menos dispersão.

3. Caudas menos extremas

- p01 menos negativo (mesmo que em pouca coisa)
- p99 levemente menor (mesmo que em pouquíssima coisa)

Menos outliers severos → **RMSE naturalmente cai**

✓ Conclusão estatística

> "O melhor desempenho no conjunto Out-of-Time é explicado por uma distribuição da variável alvo mais estável, com menor variância e menor incidência de valores extremos, o que favorece modelos lineares. Isso reforça que a melhoria no OOT não decorre de vazamento de informação, mas sim de uma mudança no regime estatístico da target."

13.2.5. Conclusão Geral

O modelo Elastic Net apresentou desempenho robusto e estável, com melhora significativa no conjunto out-of-time, explicada por uma distribuição mais estável da variável alvo. As análises diagnósticas indicam que, embora existam violações esperadas das suposições clássicas do OLS — especialmente heterocedasticidade e não normalidade dos resíduos —, o modelo captura adequadamente a estrutura linear média dos dados, apresenta estabilidade temporal e não demonstra sinais de overfitting. Esses resultados confirmam a adequação do modelo para uso preditivo em larga escala.

13.2.6. Salvando bases

```
In [20]: df_pred_train = (best_model
    .transform(df_train)
    .withColumn("residual", F.col("target_win") - F.col("prediction")))
    .withColumn("set", F.lit("train")))

df_pred_test = (best_model
    .transform(df_test)
    .withColumn("residual", F.col("target_win") - F.col("prediction")))
    .withColumn("set", F.lit("test")))

df_pred_oot = (best_model
    .transform(df_oot)
    .withColumn("residual", F.col("target_win") - F.col("prediction")))
    .withColumn("set", F.lit("oot")))

df_predictions_final = (df_pred_train.unionByName(df_pred_test).unionByName(df_pred_oot))
```

```
In [21]: # 1. cache
df_predictions_final = df_predictions_final.persist()
df_predictions_final.count()

# 2. salvar particionado
df_predictions_final.write \
    .mode("overwrite") \
    .partitionBy("safra") \
    .parquet("C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_predictions_elastic_net")
```

13.2.7. Salvando o modelo

```
In [22]: path_model = "C:/Users/Gustavo/Downloads/datamaster/models/"
# Salvando o modelo
best_model.write().overwrite().save(path_model + "elastic_net_model")

print(f"Modelo salvo com sucesso em: {path_model}")

Modelo salvo com sucesso em: C:/Users/Gustavo/Downloads/datamaster/models/
```

13.3. Modelo Baseline de Árvore - Decision Tree Regressor

13.3.1. Carregando bibliotecas e bases

```
In [9]: from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, root_mean_squared_error, mean_absolute_error, r2_score

In [4]: df_master_random_forest = spark.read.parquet(gold_path + "df_master_random_forest")
```

13.3.2. Transformando a ABT de Arvores de Spark DF para Pandas DF + separando train, test e oot

```
In [6]: # Converter para Pandas (se ainda não fez)
df_rf_pd = df_master_random_forest.toPandas()

In [7]: df_rf_pd.shape
```

Out[7]: (9678866, 36)

```
In [ ]: feature_cols = [
    "daily_revenue_efficiency",
    "daily_revenue_efficiency_min_3",
    "daily_revenue_efficiency_min_3_is_sentinel",
    "flag_plano_mensal",
    "flag_plano_mensal_max_3",
    "flag_valid_fee_max_3",
    "log_total_plays",
    "log_total_plays_mean_3",
    "log_total_plays_mean_3_is_sentinel",
    "num_25",
    "num_50",
    "num_75",
    "num_985",
    "plays_per_unq_cap_min_6",
    "plays_per_unq_cap_min_6_is_sentinel",
    "revenue_per_hour_listened_cap",
    "total_secs_mean_3",
    "total_secs_mean_3_is_sentinel",
    "total_secs_ratio_ref_max_6",
    "total_secs_ratio_ref_max_6_is_sentinel",
    "usage_intensity_per_tenure_cap",
    "avg_secs_per_unq_cap_group_te",
    "early_drop_rate_group_te",
    "faixa_idade_te",
    "gender_clean_te",
    "plays_behavior_vs_completion_collapsed_te",
    "plays_per_unq_behavior_te",
    "registered_via_group_te",
    "revenue_per_hour_tier_te",
    "total_plays_group_te",
    "usage_intensity_tier_te"
]

# Split
train_pd = df_rf_pd[df_rf_pd["partition"] == "train"].copy()
test_pd = df_rf_pd[df_rf_pd["partition"] == "test"].copy()
oot_pd = df_rf_pd[df_rf_pd["partition"] == "oot"].copy()

X_train = train_pd[feature_cols]
y_train = train_pd["target_win"]

X_test = test_pd[feature_cols]
y_test = test_pd["target_win"]

X_oot = oot_pd[feature_cols]
y_oot = oot_pd["target_win"]
```

```
In [10]: print("Distribuicao dos dataframes: ")
print(f"Train: {train_pd.shape}")
print(f"Test: {test_pd.shape}")
print(f"OOT: {oot_pd.shape}")
```

```
Distribuicao dos dataframes:
Train: (6262831, 36)
Test: (1565303, 36)
OOT: (1850732, 36)
```

13.3.3. Decision Tree simples + Resultados

```
In [ ]: baseline_tree = DecisionTreeRegressor(
    max_depth=10,
    min_samples_split=100,
    min_samples_leaf=50,
    random_state=42
)

baseline_tree.fit(X_train, y_train)
```

▼ `DecisionTreeRegressor` ⓘ ⓘ
 ► Parameters

```
In [ ]: for name, X, y in [ ("TEST", X_test, y_test), ("OOT", X_oot, y_oot)]:  
    preds = baseline_tree.predict(X)  
    rmse = root_mean_squared_error(y, preds)  
    mae = mean_absolute_error(y, preds)  
    r2 = r2_score(y, preds)
```

```
    print(f"--- Baseline Decision Tree - {name} ---")  
    print(f"RMSE: {rmse:.4f}")  
    print(f"MAE: {mae:.4f}")  
    print(f"R²: {r2:.4f}")  
    print()
```

```
--- Baseline Decision Tree - TEST ---  
RMSE: 37.6916  
MAE: 20.3338  
R²: 0.6124
```

```
--- Baseline Decision Tree - OOT ---  
RMSE: 31.3041  
MAE: 15.3203  
R²: 0.6545
```

Conclusões

Os resultados estão praticamente idênticos aos que foram encontrados tanto para o baseline linear quanto para a elastic net! O que isso pode significar?

Possibilidades

1. "Teto de Vidro" dos Dados (Data Signal):

Muitas vezes, o sinal preditivo nos dados é tão forte em algumas variáveis (como a `margem_liquida_mensal` do mês anterior ou o `target_encoding`) que qualquer modelo razoável consegue capturar esse sinal rapidamente. O que isso significa? Que atingiu-se o limite do que essas variáveis conseguem explicar. O erro restante pode ser ruído puro (erro irreductível) ou falta de variáveis que capturem comportamentos que não estão na ABT;

2. Árvore de Decisão "Baseline" conservadora demais:

Usar `max_depth=10` em um dataset de milhões de linhas, faz com que se torne quase um modelo linear simplificado. Não tem "espaço" para criar interações complexas que o LightGBM criaria, por exemplo;

3. Fenômeno da Linearidade Dominante

O Target Encoding transformou relações não-lineares em lineares. A "Armadilha" do Target Encoding: ao trocar uma categoria pela média do target, se está basicamente entregando para a Regressão Linear a informação que ela precisaria de uma árvore para descobrir. Isso faz com que o modelo linear "encoste" na performance de modelos complexos.

O LightGBM com **Hyperopt** vai tentar buscar o que esses modelos não conseguiram: **Interações de alta ordem**.

13.4. LightGBM

13.4.1. Carregando bibliotecas e bases

```
In [4]: import lightgbm as lgb  
from hyperopt import hp, fmin, tpe, Trials, STATUS_OK  
from sklearn.metrics import root_mean_squared_error, mean_absolute_error, r2_score
```

```
c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\atpe.py:19: UserWarning: pkg_resources is  
deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slated  
for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.  
    import pkg_resources
```

```
In [5]: df_master_lightgbm = spark.read.parquet(gold_path + "df_master_lightgbm")
```

```
In [6]: # Definir as 31 features
feature_cols = [
    "daily_revenue_efficiency",
    "daily_revenue_efficiency_min_3",
    "daily_revenue_efficiency_min_3_is_sentinel",
    "flag_plano_mensal",
    "flag_plano_mensal_max_3",
    "flag_valid_fee_max_3",
    "log_total_plays",
    "log_total_plays_mean_3",
    "log_total_plays_mean_3_is_sentinel",
    "num_25",
    "num_50",
    "num_75",
    "num_985",
    "plays_per_unq_cap_min_6",
    "plays_per_unq_cap_min_6_is_sentinel",
    "revenue_per_hour_listened_cap",
    "total_secs_mean_3",
    "total_secs_mean_3_is_sentinel",
    "total_secs_ratio_ref_max_6",
    "total_secs_ratio_ref_max_6_is_sentinel",
    "usage_intensity_per_tenure_cap",
    "avg_secs_per_unq_cap_group_te",
    "early_drop_rate_group_te",
    "faixa_idade_te",
    "gender_clean_te",
    "plays_behavior_vs_completion_collapsed_te",
    "plays_per_unq_behavior_te",
    "registered_via_group_te",
    "revenue_per_hour_tier_te",
    "total_plays_group_te",
    "usage_intensity_tier_te"
]

# Converter para Pandas (processo pesado, mas necessário para usar com LightGBM e Hyperopt)
print("Convertendo Spark → Pandas...")
df_lgbm_pd = df_master_lightgbm.select(["msno", "safra", "partition", "target_win"] + feature_cols).toPandas()
print("Feito!")


```

Convertendo Spark → Pandas...
Feito!

```
In [7]: # Split
train_pd = df_lgbm_pd[df_lgbm_pd["partition"] == "train"].copy()
test_pd = df_lgbm_pd[df_lgbm_pd["partition"] == "test"].copy()
oot_pd = df_lgbm_pd[df_lgbm_pd["partition"] == "oot"].copy()

X_train = train_pd[feature_cols]
y_train = train_pd["target_win"]

X_test = test_pd[feature_cols]
y_test = test_pd["target_win"]

X_oot = oot_pd[feature_cols]
y_oot = oot_pd["target_win"]

print(f"Train: {X_train.shape} | Test: {X_test.shape} | OOT: {X_oot.shape}")


```

Train: (6262831, 31) | Test: (1565303, 31) | OOT: (1850732, 31)

13.4.2. Configurações Hyperopt

```
In [8]: # Definir espaço de busca para Hyperopt
space = {
    'num_leaves': hp.quniform('num_leaves', 20, 100, 5), # Mais folhas = mais complexidade
    'learning_rate': hp.loguniform('learning_rate', np.log(0.005), np.log(0.2)),
    'n_estimators': hp.quniform('n_estimators', 100, 800, 50),
    'max_depth': hp.quniform('max_depth', 5, 15, 1), # Profundidade maior para interações
    'min_child_samples': hp.quniform('min_child_samples', 20, 200, 10),
    'subsample': hp.uniform('subsample', 0.6, 1.0),
    'colsample_bytree': hp.uniform('colsample_bytree', 0.6, 1.0),
    'reg_alpha': hp.loguniform('reg_alpha', np.log(0.001), np.log(10)),
    'reg_lambda': hp.loguniform('reg_lambda', np.log(0.001), np.log(10)),
    'min_split_gain': hp.uniform('min_split_gain', 0.0, 1.0)
}
```

In [9]:

```
# Função objetivo para Hyperopt (o que ele vai otimizar)
def objective(params):
    # Converter para int os parâmetros discretos
    params['num_leaves'] = int(params['num_leaves'])
    params['n_estimators'] = int(params['n_estimators'])
    params['max_depth'] = int(params['max_depth'])
    params['min_child_samples'] = int(params['min_child_samples'])

    # Criar modelo
    model = lgb.LGBMRegressor(
        **params,
        objective='regression',
        metric='rmse',
        boosting_type='gbdt',
        random_state=42,
        n_jobs=-1,
        verbose=-1
    )

    # Treinar com early stopping
    model.fit(
        X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric='rmse',
        callbacks=[lgb.early_stopping(stopping_rounds=30, verbose=False)]
    )

    # Avaliar no TEST (métrica de otimização)
    preds = model.predict(X_test)
    rmse = root_mean_squared_error(y_test, preds)

    return {'loss': rmse, 'status': STATUS_OK}
```

13.4.3. Encontrando melhores hiperparâmetros

In [10]:

```
print("\n" + "*60)
print("INICIANDO OTIMIZAÇÃO BAYESIANA (HYPEROPT)")
print("*60)
print("Isso pode demorar alguns minutos...")
print("Aguarde... 🌐\n")

trials = Trials()
best = fmin(
    fn=objective,
    space=space,
    algo=tpe.suggest, # Tree-structured Parzen Estimator (Bayesian Optimization)
    max_evals=50,
    trials=trials,
    rstate=np.random.default_rng(42),
    verbose=1
)

print("\n" + "*60)
print("OTIMIZAÇÃO CONCLUÍDA!")
print("*60)
print("\n💡 Melhores Hiperparâmetros:")
for key, value in best.items():
    print(f"  {key}: {value}")
```

```
=====
INICIANDO OTIMIZAÇÃO BAYESIANA (HYPEROPT)
=====
Isso pode demorar alguns minutos...
Aguarde... 🌐

100%[██████████] 50/50 [49:54<00:00, 59.88s/trial, best loss: 35.90716378326171]
```

```
=====
OTIMIZAÇÃO CONCLUÍDA!
=====
```

```
💡 Melhores Hiperparâmetros:
  colsample_bytree: 0.793176180116679
  learning_rate: 0.03931612462557681
  max_depth: 12.0
  min_child_samples: 90.0
  min_split_gain: 0.6435249382987236
  n_estimators: 350.0
  num_leaves: 60.0
  reg_alpha: 0.013905898414872837
  reg_lambda: 0.020059873982589828
  subsample: 0.6701347222884695
```

13.4.4. Treinando modelo com melhores hiperparâmetros

```
In [11]: # Reconstruir params
best_params = {
    'num_leaves': int(best['num_leaves']),
    'learning_rate': best['learning_rate'],
    'n_estimators': int(best['n_estimators']),
    'max_depth': int(best['max_depth']),
    'min_child_samples': int(best['min_child_samples']),
    'subsample': best['subsample'],
    'colsample_bytree': best['colsample_bytree'],
    'reg_alpha': best['reg_alpha'],
    'reg_lambda': best['reg_lambda'],
    'min_split_gain': best['min_split_gain']
}
```

```
In [12]: print("\n" + "*60)
print("TREINANDO MODELO FINAL COM MELHORES PARÂMETROS")
print("*60)

model_lgbm_final = lgb.LGBMRegressor(
    **best_params,
    objective='regression',
    metric='rmse',
    boosting_type='gbdt',
    random_state=42,
    n_jobs=-1,
    verbose=-1
)

model_lgbm_final.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)],
    eval_metric='rmse',
    callbacks=[lgb.early_stopping(stopping_rounds=30, verbose=False)]
)

print(f"✅ Melhor iteração (early stopping): {model_lgbm_final.best_iteration_}")
```

```
=====
TREINANDO MODELO FINAL COM MELHORES PARÂMETROS
=====
✅ Melhor iteração (early stopping): 350
```

13.4.5. Resultados (Test + OOT)

```
In [13]: print("\n" + "*60)
print("AVALIAÇÃO FINAL - LIGHTGBM (TUNED)")
print("*60)

for name, X, y in [("TEST", X_test, y_test), ("OOT", X_oot, y_oot)]:
    preds = model_lgbm_final.predict(X)
    rmse = root_mean_squared_error(y, preds)
    mae = mean_absolute_error(y, preds)
    r2 = r2_score(y, preds)

    print(f"\n--- {name} ---")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE: {mae:.4f}")
    print(f"R²: {r2:.4f}")
```

```
=====
AVALIAÇÃO FINAL - LIGHTGBM (TUNED)
=====

--- TEST ---
RMSE: 35.9082
MAE: 18.0202
R²: 0.6483

--- OOT ---
RMSE: 30.3142
MAE: 13.9768
R²: 0.6760
```

13.4.6. Comparação dos modelos executados até então

```
In [21]: print("\n" + "*60")
print("📊 COMPARAÇÃO DE TODOS OS MODELOS (OOT)")
print("*60)

preds_lgbm_oot = model_lgbm_final.predict(X_oot)
rmse_lgbm = root_mean_squared_error(y_oot, preds_lgbm_oot)
mae_lgbm = mean_absolute_error(y_oot, preds_lgbm_oot)
r2_lgbm = r2_score(y_oot, preds_lgbm_oot)

comparison = pd.DataFrame({
    'Modelo': ['Elastic Net', 'Decision Tree (Baseline)', 'LightGBM (Tuned)'],
    'RMSE OOT': [32.27, 31.30, rmse_lgbm],
    'MAE OOT': [16.62, 15.32, mae_lgbm],
    'R² OOT': [0.633, 0.654, r2_lgbm]
})

print(comparison.to_string(index=False))

# Calcular ganho percentual
ganho_rmse = ((32.27 - rmse_lgbm) / 32.27) * 100
ganho_mae = ((16.62 - mae_lgbm) / 16.62) * 100
ganho_r2 = ((r2_lgbm - 0.633) / 0.633) * 100

print(f"\n🎯 Ganho do LightGBM vs Elastic Net:")
print(f"  RMSE: {ganho_rmse:+.2f}%")
print(f"  MAE: {ganho_mae:+.2f}%")
print(f"  R²: {ganho_r2:+.2f}%")
```

```
=====
📊 COMPARAÇÃO DE TODOS OS MODELOS (OOT)
=====
      Modelo   RMSE OOT   MAE OOT   R² OOT
      Elastic Net  32.27000 16.620000 0.633000
      Decision Tree (Baseline) 31.30000 15.320000 0.654000
      LightGBM (Tuned) 30.31711 13.956646 0.675925

🎯 Ganho do LightGBM vs Elastic Net:
RMSE: +6.05%
MAE: +16.02%
R²: +6.78%
```

13.4.7. Conclusões sobre os resultados

13.4.7.1. Por que as métricas ficaram tão próximas no começo e o que isso indica

O fato do baseline linear, da Elastic Net e da árvore simples terem produzido métricas muito parecidas não indica necessariamente sinal de erro — mas sim um achado sobre o pipeline e sobre o tipo de sinal presente nos dados.

1. Winsorização da target reduz o “prêmio” dos modelos não-lineares.

Antes, a `target` tinha cauda longa (máximo ~1920), mas apliquei winsorization em `p1` e `p99`. Isso “cortou” extremos que inflamavam o RMSE e que modelos como LightGBM costumam capturar melhor via regras locais (split em regiões raras). Ao limitar a amplitude dos valores, reduz-se a influência de poucos pontos extremos na função de perda, tornando o problema mais estável e com “erro residual” mais parecido entre famílias de modelos. Em resumo, faz com que modelos lineares não sejam tão penalizados por eventos raros e muito distantes da média;

2. Feature Engineering/Selection que “linearizou” o problema.

O uso de **target encoding** (validado, sem leakage) é poderoso: transforma relações categóricas complexas em variáveis numéricas com forte poder explicativo. Isso costuma aproximar bastante um modelo linear do desempenho de modelos de árvore, porque parte da “não-linearidade”/interação já foi embutida na feature;

3. Por conter boa quantia de observações, modelos simples generalizam muito bem.

Com 6M+ linhas em treino, o baseline linear já estima coeficientes com variância baixíssima. A Elastic Net acabou escolhendo regularização fraca (ou praticamente nula) por existirem muitos dados, features boas e winsorização que reduz o impacto de outliers severos. Isso explica por que o “melhor Elastic Net” ficou muito perto do baseline linear no TEST.

Conclusão metodológica:

Métricas próximas aqui indicam **pipeline sólido**, não pipeline fraco. Quando diferentes modelos convergem, geralmente significa que:

- Não tem “vazamento óbvio” (que costuma gerar ganhos artificiais enormes em um modelo e não em outro);
- O split temporal OOT está coerente;
- O conjunto de features carrega sinal estável (não depende de um “truque” do algoritmo).

13.4.7.2. Análise dos resultados do LightGBM Tuned

É possível notar um ganho relevante, especialmente no OOT, que é a métrica que importa para estabilidade temporal:

- **RMSE caiu ~6%** vs Elastic Net → melhora real na qualidade das previsões, sobretudo em erros maiores.
- **MAE caiu ~16%** vs Elastic Net → melhora muito forte no erro “típico”, o que costuma ser o mais acionável para tomada de decisão.
- **R² subiu de 0.633 para 0.676** → ganho claro de explicação de variância fora do tempo.

Mesmo com o pipeline correto e já “forte”, o LightGBM (com tuning bayesiano via hyperopt) conseguiu extrair interações e não-linearidades residuais que os modelos lineares não capturaram — com ganho consistente no OOT, o que é o melhor sinal possível.

13.4.8. Feature Importance

13.4.8.1. Gain/Split

Conceitos

Métrica O que mede Interpretação prática
----- ----- -----
GAIN Quanto a feature reduz o erro do modelo Impacto total na qualidade das previsões
SPLIT Quantas vezes a feature foi usada para dividir nós Popularidade / frequência de uso
GAIN/SPLIT Eficiência média por uso O quão forte cada divisão dessa feature é

💡 Resumo simples

1. GAIN alto = variável importante no resultado final;
2. SPLIT alto = variável muito usada, mas não necessariamente decisiva;
3. GAIN/SPLIT alto = quando aparece, ela muda muito o jogo.

Valores + Gráficos

In [22]: booster = model_lgbm_final.booster_

```
# Importâncias nativas do booster
imp_gain = booster.feature_importance(importance_type="gain")
imp_split = booster.feature_importance(importance_type="split")

df_imp = pd.DataFrame({
    "feature": booster.feature_name(),
    "gain": imp_gain,
    "split": imp_split
})

df_imp["gain_per_split"] = df_imp["gain"] / df_imp["split"].replace(0, np.nan)
df_imp = df_imp.sort_values("gain", ascending=False)

print("Top 20 por GAIN:")
print(df_imp.head(20).to_string(index=False))

print("\nTop 20 por SPLIT:")
print(df_imp.sort_values("split", ascending=False).head(20).to_string(index=False))

print("\nTop 20 por GAIN/SPLIT (efeito por uso):")
print(df_imp.sort_values("gain_per_split", ascending=False).head(20).to_string(index=False))
```

Top 20 por GAIN:

	feature	gain	split	gain_per_split
	flag_plano_mensal_max_3	8.591049e+10	280	3.068232e+08
	flag_plano_mensal	4.363354e+10	425	1.026671e+08
daily_revenue_efficiency_min_3	2.809390e+10	2125	1.322066e+07	
daily_revenue_efficiency	1.733019e+10	2045	8.474420e+06	
log_total_plays	1.194809e+10	2071	5.769238e+06	
total_secs_mean_3	2.774079e+09	1252	2.215718e+06	
registered_via_group_te	2.396926e+09	967	2.478724e+06	
revenue_per_hour_listened_cap	2.117772e+09	791	2.677334e+06	
faixa_idade_te	8.184961e+08	1161	7.049924e+05	
usage_intensity_per_tenure_cap	6.873069e+08	1137	6.044916e+05	
daily_revenue_efficiency_min_3_is_sentinel	5.730088e+08	106	5.405743e+06	
flag_valid_fee_max_3	5.004012e+08	343	1.458896e+06	
total_plays_group_te	4.630414e+08	205	2.258738e+06	
log_total_plays_mean_3	4.270165e+08	1343	3.179572e+05	
plays_per_unq_cap_min_6	2.190789e+08	888	2.467104e+05	
total_secs_ratio_ref_max_6	1.749211e+08	905	1.932829e+05	
plays_behavior_vs_completion_collapsed_te	1.536894e+08	610	2.519498e+05	
num_50	1.381977e+08	493	2.803199e+05	
early_drop_rate_group_te	1.159488e+08	307	3.776833e+05	
usage_intensity_tier_te	1.123034e+08	275	4.083761e+05	

Top 20 por SPLIT:

	feature	gain	split	gain_per_split
	daily_revenue_efficiency_min_3	2.809390e+10	2125	1.322066e+07
	log_total_plays	1.194809e+10	2071	5.769238e+06
daily_revenue_efficiency	1.733019e+10	2045	8.474420e+06	
log_total_plays_mean_3	4.270165e+08	1343	3.179572e+05	
total_secs_mean_3	2.774079e+09	1252	2.215718e+06	
faixa_idade_te	8.184961e+08	1161	7.049924e+05	
usage_intensity_per_tenure_cap	6.873069e+08	1137	6.044916e+05	
registered_via_group_te	2.396926e+09	967	2.478724e+06	
total_secs_ratio_ref_max_6	1.749211e+08	905	1.932829e+05	
plays_per_unq_cap_min_6	2.190789e+08	888	2.467104e+05	
num_25	9.945608e+07	856	1.161870e+05	
revenue_per_hour_listened_cap	2.117772e+09	791	2.677334e+06	
plays_behavior_vs_completion_collapsed_te	1.536894e+08	610	2.519498e+05	
num_985	5.186502e+07	509	1.018959e+05	
num_50	1.381977e+08	493	2.803199e+05	
flag_plano_mensal	4.363354e+10	425	1.026671e+08	
gender_clean_te	9.378429e+07	382	2.455086e+05	
flag_valid_fee_max_3	5.004012e+08	343	1.458896e+06	
plays_per_unq_behavior_te	2.293455e+07	336	6.825759e+04	
early_drop_rate_group_te	1.159488e+08	307	3.776833e+05	

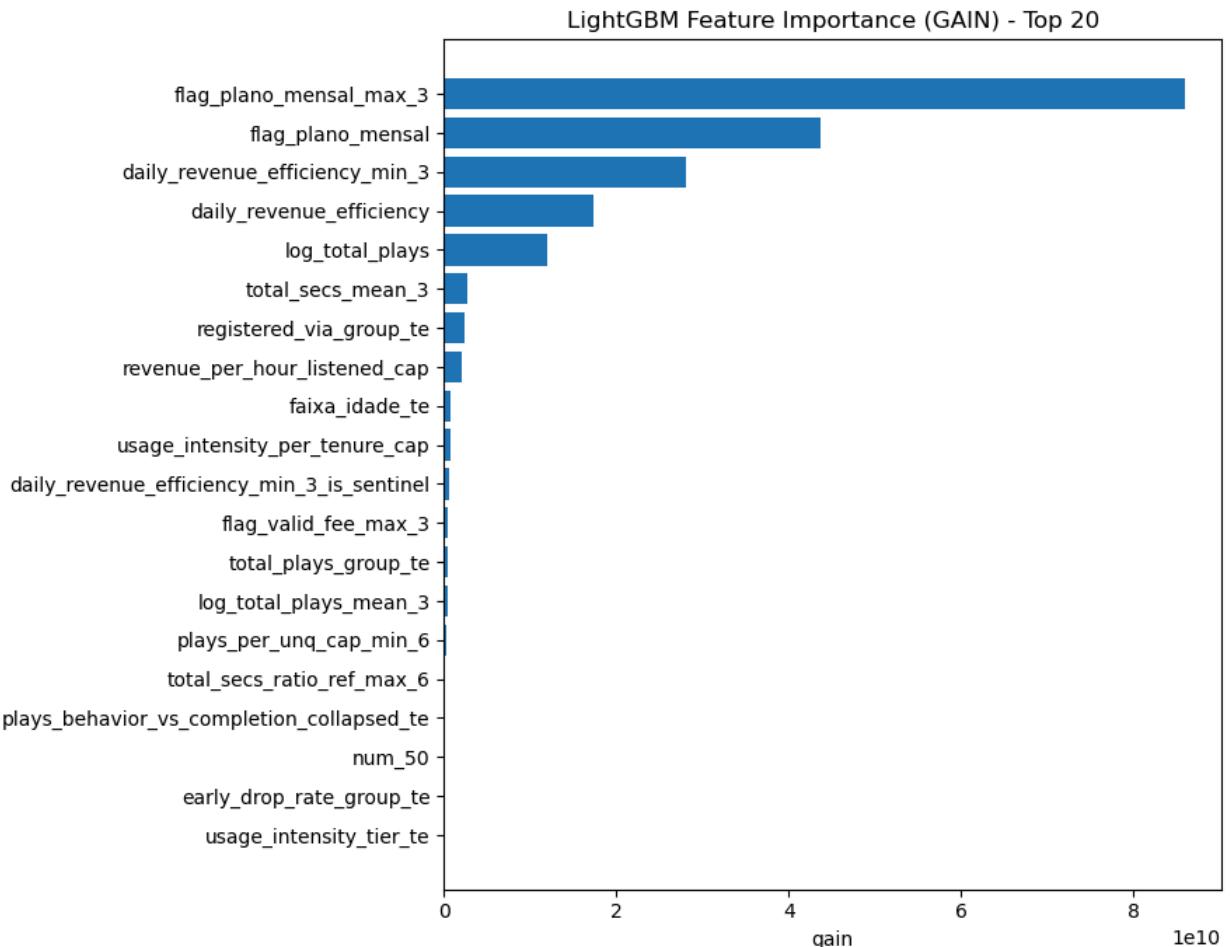
Top 20 por GAIN/SPLIT (efeito por uso):

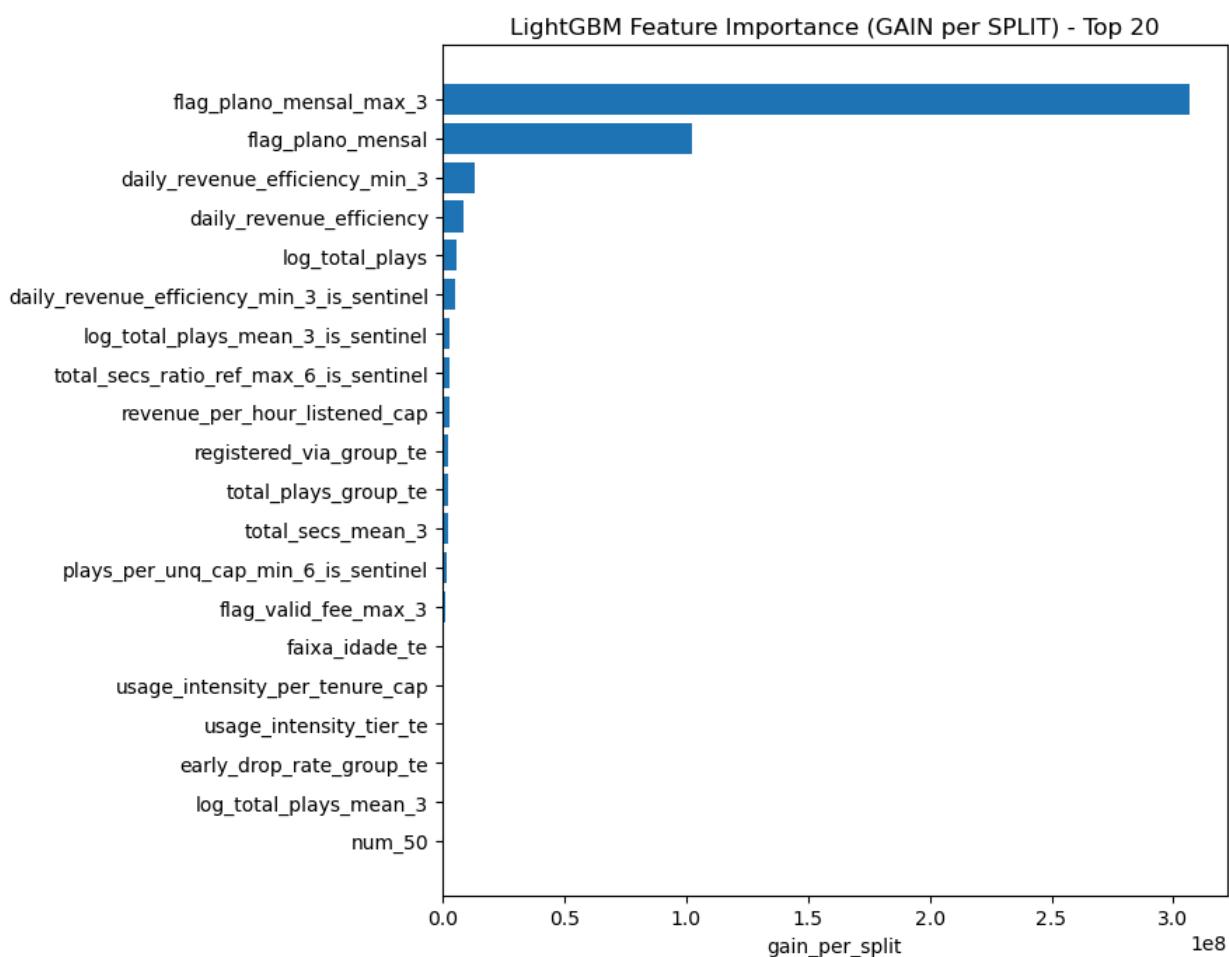
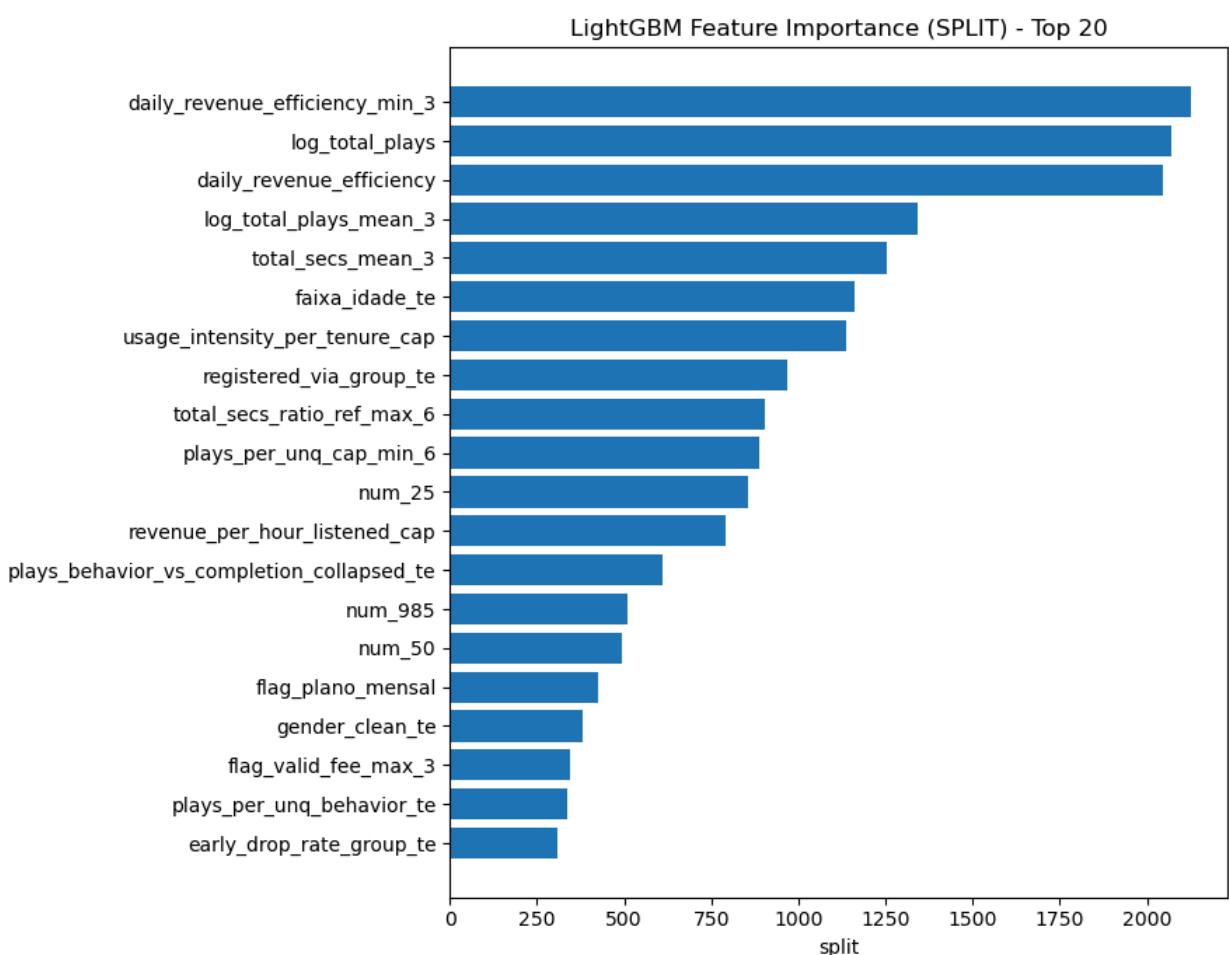
	feature	gain	split	gain_per_split
	flag_plano_mensal_max_3	8.591049e+10	280	3.068232e+08
	flag_plano_mensal	4.363354e+10	425	1.026671e+08
daily_revenue_efficiency_min_3	2.809390e+10	2125	1.322066e+07	
daily_revenue_efficiency	1.733019e+10	2045	8.474420e+06	
log_total_plays	1.194809e+10	2071	5.769238e+06	
daily_revenue_efficiency_min_3_is_sentinel	5.730088e+08	106	5.405743e+06	
log_total_plays_mean_3_is_sentinel	2.396792e+07	8	2.995990e+06	
total_secs_ratio_ref_max_6_is_sentinel	2.712760e+06	1	2.712760e+06	
revenue_per_hour_listened_cap	2.117772e+09	791	2.677334e+06	
registered_via_group_te	2.396926e+09	967	2.478724e+06	
total_plays_group_te	4.630414e+08	205	2.258738e+06	
total_secs_mean_3	2.774079e+09	1252	2.215718e+06	
plays_per_unq_cap_min_6_is_sentinel	5.512227e+07	28	1.968652e+06	
flag_valid_fee_max_3	5.004012e+08	343	1.458896e+06	
faixa_idade_te	8.184961e+08	1161	7.049924e+05	

```
usage_intensity_per_tenure_cap 6.873069e+08    1137    6.044916e+05
                                usage_intensity_tier_te 1.123034e+08    275    4.083761e+05
                                early_drop_rate_group_te 1.159488e+08    307    3.776833e+05
                                log_total_plays_mean_3 4.270165e+08    1343    3.179572e+05
                                num_50 1.381977e+08     493    2.803199e+05
```

```
In [23]: def plot_barh(df, col, title, topn=20):
    d = df.sort_values(col, ascending=False).head(topn).iloc[::-1]
    plt.figure(figsize=(9, 7))
    plt.barh(d["feature"], d[col])
    plt.title(title)
    plt.xlabel(col)
    plt.tight_layout()
    plt.show()

plot_barh(df_imp, "gain", "LightGBM Feature Importance (GAIN) - Top 20", 20)
plot_barh(df_imp.sort_values("split", ascending=False), "split", "LightGBM Feature Importance (SPLIT) - Top 20", 20)
plot_barh(df_imp.sort_values("gain_per_split", ascending=False), "gain_per_split", "LightGBM Feature Importance (GAIN per SPLIT) - Top 20", 20)
```





Conclusões

Features "doras" do modelo

💡 flag_plano_mensal_max_3 :

- Maior gain disparado;
- Poucos splits (280) → mas cada uso tem impacto gigante;
- Gain/Split mais alto do modelo

Essa variável é um divisor de águas. Quando o modelo encontra esse sinal, ele praticamente redefine a previsão.

💡 flag_plano_mensal

- Mesmo padrão, só um pouco menos extrema.

O modelo está dizendo claramente: o tipo/status de plano do usuário é um dos fatores mais determinantes do comportamento alvo.

Elas serem “parecidas” é um problema?

Sozinhas, não. Árvores (LightGBM incluso) lidam muito bem com variáveis correlacionadas. Diferente de regressão linear, aqui não rola aquele caos de multicolinearidade quebrando coeficiente. O que acontece na prática é:

- O modelo escolhe uma delas primeiro pra dividir (podemos pressupor que `flag_plano_mensal_max_3`, pelos valores de gain/split); e
- A outra pode ou não entrar depois, refinando subgrupos.

`flag_plano_mensal` diz respeito à situação no mês atual, enquanto `flag_plano_mensal_max_3` a melhor situação nos últimos 3 meses. Apesar de similares, respondem a perguntas diferentes:

- A pessoa está com plano mensal ativo agora?
- A pessoa já teve plano mensal recentemente?

E isso indica segmentação comportamental rica, não pura redundância.

Quando se tornaria um problema?

Se uma for cópia quase perfeita da outra. Por exemplo, 98% das linhas onde uma é 1 a outra também é 1. Para checar:

In [30]: `pd.crosstab(X_oot.flag_plano_mensal, X_oot.flag_plano_mensal_max_3, normalize='all')`

Out[30]:	flag_plano_mensal_max_3	0	1
	flag_plano_mensal		
0	0.094034	0.019982	
1	0.000000	0.885983	

Reescrevendo para ficar melhor:

mensal max_3 significado proporção
----- ----- ----- -----
0 0 Nunca foi mensal (0,0) 9.40%
0 1 Ex-mensal recente 1.99%
1 0 Impossível 0.00%
1 1 Mensal atual 88.60%

Existe diferença real entre elas?

Só existe diferença real aqui: mensal = 0, max_3 (2%) = 1 → Usuário já foi mensal recentemente, mas não é mais. E é exatamente esse grupo que faz o modelo manter as duas variáveis.

Pensando que o foco é a performance do modelo, a melhor escolha é manter as duas: o modelo está usando a combinação para identificar **ex-mensais**.

Features operárias do modelo (muito usadas)

daily_revenue_efficiency_min_3 , daily_revenue_efficiency , log_total_plays

* São MUITO usadas (2000+ splits);

* Têm GAIN alto, mas GAIN/SPLIT bem menor que as listadas acima.

Essas variáveis são o “ajuste fino” do modelo. Elas ajudam o modelo a refinar a previsão em muitos pontos diferentes, mas nenhuma divisão sozinha é explosiva.

Padrão identificado:

1. Variáveis de comportamento contínuo = muitas decisões pequenas
2. Variáveis de plano/status = poucas decisões muito fortes

Variáveis “decisivas” por gain/split

Além dos planos, aparecem bem:

daily_revenue_efficiency_min_3_is_sentinel , log_total_plays_mean_3_is_sentinel , outras _is_sentinel . Com isso, podemos concluir que o modelo está aprendendo que a ausência ou valor extremo (sentinel) dessas métricas é altamente informativo. Ou seja:

- Usuário sem histórico suficiente
- Usuário com comportamento anômalo
- Dados faltantes que não são aleatórios

Se mostram como sinais que operam quase como flags de risco ou exceção.

Possível estrutura aprendida pelo modelo

1. Tipo do usuário: plano, status, flags;
2. Como ele se comporta: uso, plays, tempo, eficiência.

Features que parecem fortes mas são só “frequentes”

log_total_plays_mean_3 , faixa_idade_te : muitos splits, baixo gain/split. Elas ajudam, mas são fracas individualmente. Servem mais como ajustes marginais.

13.4.8.2. SHAP

Separando amostra para features

In [31]:

```
import shap

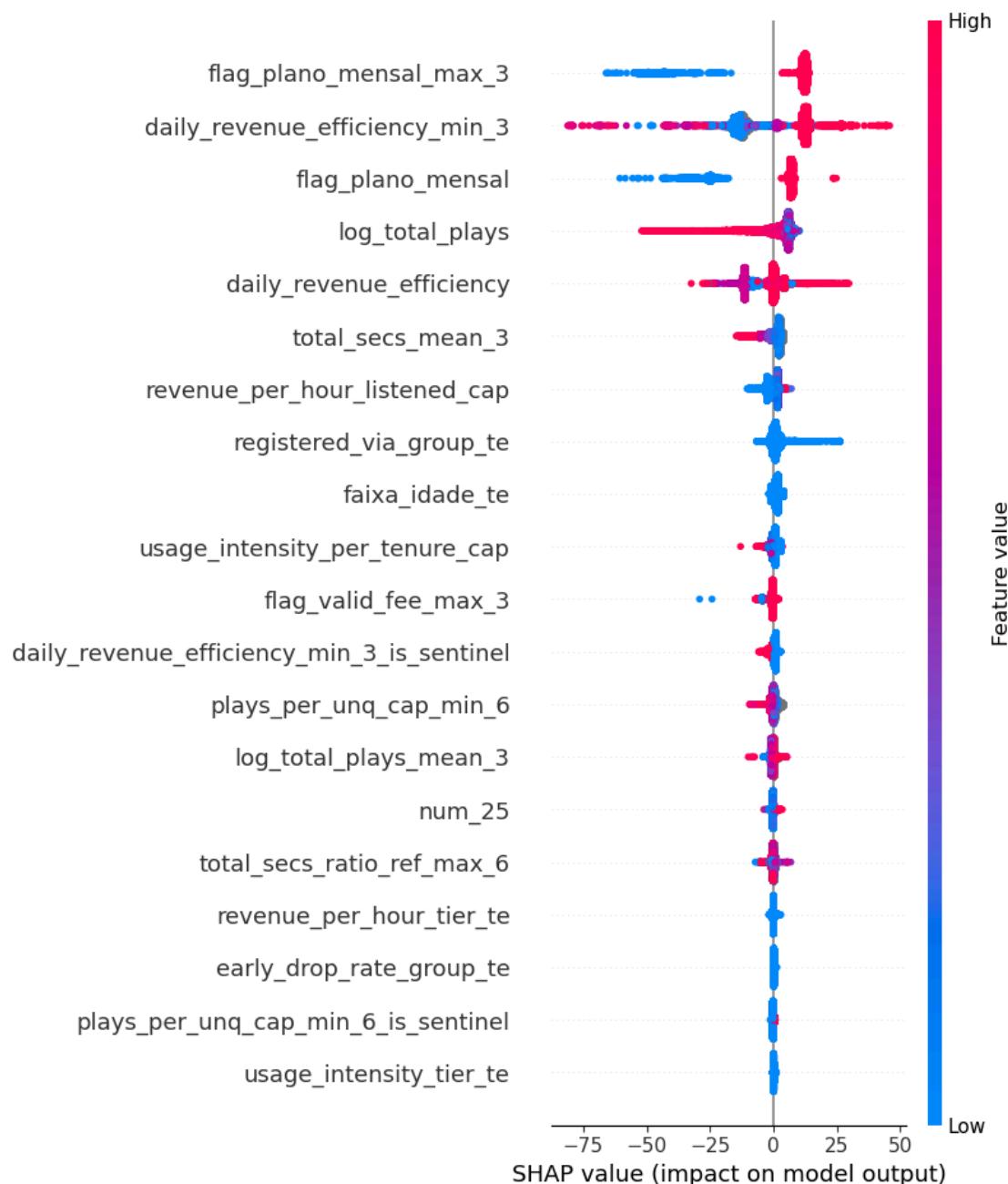
# Amostra para SHAP (ajuste se sua máquina aguentar)
N_SHAP = 20000
X_shap = X_test.sample(min(N_SHAP, len(X_test)), random_state=42)

# Para regressão com LightGBM, TreeExplainer é o correto
explainer = shap.TreeExplainer(model_lgbm_final)

# SHAP values
shap_values = explainer.shap_values(X_shap)
```

Summary Plot - Beeswarm

```
In [32]: shap.summary_plot(shap_values, X_shap, max_display=20)
```



Interpretação do SHAP – Modelo LightGBM

Agora estamos analisando como o modelo toma decisões individuais, não apenas a importância global das variáveis.

Vou dividir em três partes:

1. Como ler o **SHAP summary plot**
2. O que ele está dizendo sobre o modelo

1. Como ler o SHAP Summary Plot

- Cada linha representa uma feature;
- Cada ponto representa uma observação (usuário/linha do dataset).

Eixo X (horizontal)

O valor no eixo X é o **SHAP value**, que mostra o impacto daquela variável na predição.

- Valores **positivos** → empurram a predição **para cima**
- Valores **negativos** → empurram a predição **para baixo**
- Quanto mais distante do zero, maior o impacto

Cor dos pontos

A cor indica o **valor real da feature** naquela observação:

- **Azul** → valor baixo
- **Vermelho/Rosa** → valor alto

Isso permite ver o padrão quando a variável é alta, ela aumenta ou diminui a predição?

2. O que o gráfico revela sobre o modelo

`flag_plano_mensal_max_3`

- Pontos vermelhos (valor = 1) estão à direita → aumentam a predição
- Pontos azuis (valor = 0) estão à esquerda → diminuem a predição

Interpretação:

Ter sido mensal nos últimos 3 meses aumenta fortemente a previsão do modelo.

`flag_plano_mensal`

Mesmo padrão, mas com impacto um pouco menor.

O modelo diferencia:

- Nunca foi mensal
- Já foi mensal
- É mensal agora

`daily_revenue_efficiency_min_3`

Padrão bem claro e monotônico:

- Eficiência alta → aumenta predição
- Eficiência baixa → reduz predição

Variável comportamental forte e consistente.

`log_total_plays`

- Muito uso → impacto positivo
- Pouco uso → impacto negativo

O modelo está capturando **engajamento real do usuário**.

`total_secs_mean_3`

- Mais tempo ouvindo → aumenta a previsão
- Menos tempo → diminui

Reflete intensidade de consumo.

Variáveis sentinel

Exemplo: `daily_revenue_efficiency_min_3_is_sentinel`. Quando essa flag está ativa (valor alto), ela costuma empurrar a predição de forma consistente. A ausência de histórico ou comportamento extremo é informativa para o modelo.

Predição individual

In [27]:

```
i = 0
x0 = X_shap.iloc[i:i+1]

pred0 = model_lgbm_final.predict(x0)[0]
base_value = explainer.expected_value

print("Pred:", pred0)
print("Base value (E[f(x)]):", base_value)
print("Base + sum(SHAP):", base_value + shap_values[1].sum())

Pred: 61.94651290453658
Base value (E[f(x)]): 43.675988800888376
Base + sum(SHAP): 61.94651290453632
```

Entendendo a predição individual

Valores fornecidos:

- * Pred: 61.9465
- * Base value: 43.6759
- * Base + sum(SHAP): 61.9465

SHAP segue a equação:

$$\text{Predição Final} = E[f(X)] + \sum_{j=1}^M \phi_j$$

Onde:

- * **Valor Base ($E[f(X)]$):** É a média das predições do modelo sobre todo o conjunto de treino. Representa o que o modelo preveria se não conhecesse nenhuma informação sobre o cliente.
- * **ϕ_j (SHAP Value):** É o impacto (positivo ou negativo) da variável j para deslocar a predição do valor médio até o valor final observado.
- * **M:** Número total de variáveis (features).

Se o valor médio da margem líquida no treino é **43.67** (Valor Base) e o modelo prevê **61.94** para um cliente específico, a soma de todos os SHAP values desse cliente será exatamente **18.27**. O gráfico de SHAP detalha quais variáveis "empurraram" a predição para cima (ex: alta eficiência de receita) e quais "puxaram" para baixo (ex: baixa intensidade de uso).

Exemplo ilustrativo:

Feature SHAP Efeito		
----- ----- -----		
Plano recente +12 Aumenta		
Plano atual +6 Aumenta		
Alta eficiência +5 Aumenta		
Baixo tempo de uso -3 Diminui		
Idade -2 Diminui		

Soma dos SHAP ≈ +18.27

Interpretação dessa predição

O usuário está bem acima da média do modelo:

- Média geral: 43.7
- Usuário: 61.9

As características dele empurraram a previsão para cima, provavelmente por:

- Ser ou ter sido usuário mensal
- Ter boa eficiência de receita
- Ter bom nível de uso

Resumo

Elemento Significado
----- -----
Base value Previsão média do modelo
SHAP value Impacto individual de cada variável
SHAP > 0 Aumenta a predição
SHAP < 0 Diminui a predição
Summary plot Direção e força do efeito de cada variável

O modelo mostra um comportamento coerente:

- Variáveis de plano geram grandes mudanças
- Variáveis de uso fazem ajustes graduais
- SHAP confirma os padrões vistos no GAIN

13.4.9. Salvar base

```
In [14]: gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"
```

```
In [15]: print("=*60")
print("GERANDO PREDIÇÕES (TRAIN/TEST/OOT)")
print("=*60)

def add_predictions(pdf, set_name):
    """Adiciona predições e resíduos ao dataframe"""
    pdf = pdf.copy()
    pdf["prediction_lgbm"] = model_lgbm_final.predict(pdf[feature_cols])
    pdf["residual_lgbm"] = pdf["target_win"] - pdf["prediction_lgbm"]
    pdf["set"] = set_name
    return pdf

# Gerar predições
train_pred_pd = add_predictions(train_pd, "train")
test_pred_pd = add_predictions(test_pd, "test")
oot_pred_pd = add_predictions(oot_pd, "oot")

print(f"✓ Train: {train_pred_pd.shape}")
print(f"✓ Test: {test_pred_pd.shape}")
print(f"✓ OOT: {oot_pred_pd.shape}")

# Selecionar colunas relevantes (reduz tamanho do arquivo)
keep_cols = [
    "msno",
    "safra",
    "partition",
    "set",
    "target_win",
    "prediction_lgbm",
    "residual_lgbm"
] + feature_cols

train_pred_pd = train_pred_pd[keep_cols]
test_pred_pd = test_pred_pd[keep_cols]
oot_pred_pd = oot_pred_pd[keep_cols]

# Consolidar tudo
all_pred_pd = pd.concat([train_pred_pd, test_pred_pd, oot_pred_pd], ignore_index=True)
print(f"\n✓ Base consolidada: {all_pred_pd.shape}")
print(f"    Colunas: {list(all_pred_pd.columns)})
```

```
=====
GERANDO PREDIÇÕES (TRAIN/TEST/OOT)
=====

✓ Train: (6262831, 38)
✓ Test: (1565303, 38)
✓ OOT: (1850732, 38)

✓ Base consolidada: (9678866, 38)
    Colunas: ['msno', 'safra', 'partition', 'set', 'target_win', 'prediction_lgbm', 'residual_lgbm',
'daily_revenue_efficiency', 'daily_revenue_efficiency_min_3', 'daily_revenue_efficiency_min_3_is_sentinel',
'flag_plano_mensal', 'flag_plano_mensal_max_3', 'flag_valid_fee_max_3', 'log_total_plays', 'log_total_plays_mean_3',
'log_total_plays_mean_3_is_sentinel', 'num_25', 'num_50', 'num_75', 'num_985', 'plays_per_unq_cap_min_6',
'plays_per_unq_cap_min_6_is_sentinel', 'revenue_per_hour_listened_cap', 'total_secs_mean_3',
'total_secs_mean_3_is_sentinel', 'total_secs_ratio_ref_max_6', 'total_secs_ratio_ref_max_6_is_sentinel',
'usage_intensity_per_tenure_cap', 'avg_secs_per_unq_cap_group_te', 'early_drop_rate_group_te', 'faixa_idade_te',
'gender_clean_te', 'plays_behavior_vs_completion_collapsed_te', 'plays_per_unq_behavior_te', 'registered_via_group_te',
'revenue_per_hour_tier_te', 'total_plays_group_te', 'usage_intensity_tier_te']
```

```
In [ ]: print("=*60")
print("CONVERTENDO E SALVANDO DIRETO")
print("=*60)

# Caminho de saída
out_path = gold_path + "df_predictions_lightgbm"

# Converter tudo de uma vez
df_predictions_lightgbm = spark.createDataFrame(all_pred_pd)

# Salvar particionado
(
    df_predictions_lightgbm.write
    .mode("overwrite")
    .partitionBy("safra")
    .parquet(out_path)
)

print(f"✓ Sucesso! Predições salvas em: {out_path}")
```

```
=====
CONVERTENDO E SALVANDO DIRETO
=====

✓ Sucesso! Predições salvas em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_predictions_lightgbm
```

Obs.: depois me liguei que não precisava ter mandado para Spark DF para salvar como .parquet, o Pandas também faz isso e seria menos pesado.

13.4.10. Salvar modelo + metadados

```
In [18]: path_model = "C:/Users/Gustavo/Downloads/datamaster/models/"
```

```
In [20]: print("\n" + "*60)
print("SALVANDO MODELO E METADADOS")
print("*60)

import os
import json
import joblib

# Criar diretório
os.makedirs(path_model, exist_ok=True)

# 4.1 Salvar modelo completo (joblib) - para reuso em Python
model_path_pkl = os.path.join(path_model, "model.pkl")
joblib.dump(model_lgbm_final, model_path_pkl)
print(f"✓ Modelo (joblib) salvo: {model_path_pkl}")

# 4.2 Salvar booster nativo (formato LightGBM) - portável
booster_path = os.path.join(path_model, "booster.txt")
model_lgbm_final.booster_.save_model(booster_path)
print(f"✓ Booster (LightGBM) salvo: {booster_path}")

# 4.3 Salvar metadados (features, params, best_iteration)
metadata = {
    "model_type": "LightGBMRegressor",
    "optimization": "Hyperopt (Bayesian)",
    "n_trials": 50,
    "feature_cols": feature_cols,
    "n_features": len(feature_cols),
    "best_iteration": int(getattr(model_lgbm_final, "best_iteration_", -1)),
    "params": model_lgbm_final.get_params(),
    "train_shape": list(X_train.shape),
    "test_shape": list(X_test.shape),
    "oot_shape": list(X_oot.shape)
}

metadata_path = os.path.join(path_model, "metadata.json")
with open(metadata_path, "w", encoding="utf-8") as f:
    json.dump(metadata, f, ensure_ascii=False, indent=2, default=str)
print(f"✓ Metadados salvos: {metadata_path}")

print("\n" + "*60)
print("✓ TUDO SALVO COM SUCESSO!")
print("*60)
```

```
=====
SALVANDO MODELO E METADADOS
=====
✓ Modelo (joblib) salvo: C:/Users/Gustavo/Downloads/datamaster/models/model.pkl
✓ Booster (LightGBM) salvo: C:/Users/Gustavo/Downloads/datamaster/models/booster.txt
✓ Metadados salvos: C:/Users/Gustavo/Downloads/datamaster/models/metadata.json

=====
✓ TUDO SALVO COM SUCESSO!
=====
```

13.5. Random Forest Regressor

13.5.1. Carregando bibliotecas e bases

```
In [7]: from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import root_mean_squared_error, mean_absolute_error, r2_score
from hyperopt import hp, fmin, tpe, Trials, STATUS_OK
```

```
c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\atpe.py:19: UserWarning: pkg_resources is
deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slated
for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.
    import pkg_resources
```

```
In [4]: df_master_random_forest = spark.read.parquet(gold_path + "df_master_random_forest")
```

```
In [5]: # Definir as 31 features
feature_cols = [
    "daily_revenue_efficiency",
    "daily_revenue_efficiency_min_3",
    "daily_revenue_efficiency_min_3_is_sentinel",
    "flag_plano_mensal",
    "flag_plano_mensal_max_3",
    "flag_valid_fee_max_3",
    "log_total_plays",
    "log_total_plays_mean_3",
    "log_total_plays_mean_3_is_sentinel",
    "num_25",
    "num_50",
    "num_75",
    "num_985",
    "plays_per_unq_cap_min_6",
    "plays_per_unq_cap_min_6_is_sentinel",
    "revenue_per_hour_listened_cap",
    "total_secs_mean_3",
    "total_secs_mean_3_is_sentinel",
    "total_secs_ratio_ref_max_6",
    "total_secs_ratio_ref_max_6_is_sentinel",
    "usage_intensity_per_tenure_cap",
    "avg_secs_per_unq_cap_group_te",
    "early_drop_rate_group_te",
    "faixa_idade_te",
    "gender_clean_te",
    "plays_behavior_vs_completion_collapsed_te",
    "plays_per_unq_behavior_te",
    "registered_via_group_te",
    "revenue_per_hour_tier_te",
    "total_plays_group_te",
    "usage_intensity_tier_te"
]

# Converter para Pandas (processo pesado, mas necessário para usar com LightGBM e Hyperopt)
print("Convertendo Spark → Pandas...")
df_rf_pd = df_master_random_forest.select(["msno", "safra", "partition", "target_win"] + feature_cols).toPandas()
print("Feito!")
```

```
Convertendo Spark → Pandas...
Feito!
```

```
In [6]: # Split
train_pd = df_rf_pd[df_rf_pd["partition"] == "train"].copy()
test_pd = df_rf_pd[df_rf_pd["partition"] == "test"].copy()
oot_pd = df_rf_pd[df_rf_pd["partition"] == "oot"].copy()

X_train = train_pd[feature_cols]
y_train = train_pd["target_win"]

X_test = test_pd[feature_cols]
y_test = test_pd["target_win"]

X_oot = oot_pd[feature_cols]
y_oot = oot_pd["target_win"]

print(f"Train: {X_train.shape} | Test: {X_test.shape} | OOT: {X_oot.shape}")
```

```
Train: (6262831, 31) | Test: (1565303, 31) | OOT: (1850732, 31)
```

13.5.2. Configurações Hyperopt

```
#### 13.5.2.1. Primeira configuração (pesada para os 6 milhões de registros de treino)
```

```
In [ ]: # Definir espaço de busca para Hyperopt
space_rf = {
    'n_estimators': hp.quniform('n_estimators', 100, 500, 50),
    'max_depth': hp.quniform('max_depth', 5, 20, 1),
    'min_samples_split': hp.quniform('min_samples_split', 2, 20, 1),
    'min_samples_leaf': hp.quniform('min_samples_leaf', 1, 5, 1),
    'max_features': hp.choice('max_features', ['sqrt', 'log2', 0.5, 0.7]),
    'bootstrap': hp.choice('bootstrap', [True, False]),
    'max_samples': hp.uniform('max_samples', 0.6, 1.0) # só funciona se bootstrap=True
}
```

```
In [12]: def objective_rf(params):
    # Converter para int os parâmetros discretos
    params['n_estimators'] = int(params['n_estimators'])
    params['max_depth'] = int(params['max_depth'])
    params['min_samples_split'] = int(params['min_samples_split'])
    params['min_samples_leaf'] = int(params['min_samples_leaf'])

    # Se bootstrap=False, max_samples não é usado (remover para evitar warning)
    if not params['bootstrap']:
        params.pop('max_samples', None)

    # Criar modelo
    model = RandomForestRegressor(
        **params,
        random_state=42,
        n_jobs=-1,
        verbose=0
    )

    # Treinar
    model.fit(X_train, y_train)

    # Avaliar no TEST
    preds = model.predict(X_test)
    rmse = root_mean_squared_error(y_test, preds)

    return {'loss': rmse, 'status': STATUS_OK}
```

13.5.2.2. Segunda configuração (melhor desempenho)

```
In [22]: # Espaço de busca mais leve
space_rf = {
    'n_estimators': hp.quniform('n_estimators', 50, 200, 50),
    'max_depth': hp.quniform('max_depth', 5, 15, 1),
    'min_samples_leaf': hp.quniform('min_samples_leaf', 5, 50, 5),
    'max_features': hp.choice('max_features', ['sqrt', 0.5]),
    'bootstrap': hp.choice('bootstrap', [True])
}
```

```
In [25]: def objective_rf(params):
    # Converter para int
    params['n_estimators'] = int(params['n_estimators'])
    params['max_depth'] = int(params['max_depth'])
    params['min_samples_leaf'] = int(params['min_samples_leaf'])

    # Criar modelo
    model = RandomForestRegressor(
        **params,
        random_state=42,
        n_jobs=-1,
        verbose=0
    )

    # Treinar na AMOSTRA
    model.fit(X_train_sample, y_train_sample)

    # Avaliar no TEST completo
    preds = model.predict(X_test)
    rmse = root_mean_squared_error(y_test, preds)

    return {'loss': rmse, 'status': STATUS_OK}
```

13.5.3. Encontrando melhores hiperparâmetros

13.5.3.1. Primeira execução

```
In [13]: print("\n" + "*60)
print("INICIANDO OTIMIZAÇÃO BAYESIANA VIA HYPEROPT")
print("*60)
print("Isso pode levar um tempo. Aguarde...")

trials_rf = Trials()
best_rf = fmin(
    fn=objective_rf,
    space=space_rf,
    algo=tpe.suggest,
    max_evals=50,
    trials=trials_rf,
    rstate=np.random.default_rng(42),
    verbose=1
)

print("\n" + "*60)
print("OTIMIZAÇÃO CONCLUÍDA!")
print("*60)
print("\n💡 Melhores Hiperparâmetros:")
for key, value in best_rf.items():
    print(f"  {key}: {value}")


=====
INICIANDO OTIMIZAÇÃO BAYESIANA VIA HYPEROPT
=====
Isso pode levar um tempo. Aguarde...
2%|          | 1/50 [13:40<11:09:44, 820.10s/trial, best loss: 37.59801678612831]
```

```
-----
KeyboardInterrupt                                     Traceback (most recent call last)
Cell In[13], line 7
      4 print("Isso pode levar um tempo. Aguarde...")
      5 trials_rf = Trials()
----> 6 best_rf = fmin(
     7     fn=objective_rf,
     8     space=space_rf,
     9     algo=tpe.suggest,
    10    max_evals=50,
    11    trials=trials_rf,
    12    rstate=np.random.default_rng(42),
    13    verbose=1
    14
    15 )
    16 print("\n" + "*60)
    17 print("OTIMIZAÇÃO CONCLUÍDA!")
```

```
File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\fmin.py:540, in fmin(fn, space, algo, max_evals, timeout, loss_threshold, trials, rstate, allow_trials_fmin, pass_expr_memo_ctrl, catch_eval_exceptions, verbose, return_argmin, points_to_evaluate, max_queue_len, show_progressbar, early_stop_fn, trials_save_file)
    537     fn = __objective_fmin_wrapper(fn)
    538 if allow_trials_fmin and hasattr(trials, "fmin"):
--> 539     return trials.fmin(
    540         fn,
    541         space,
    542         algo=algo,
    543         max_evals=max_evals,
    544         timeout=timeout,
    545         loss_threshold=loss_threshold,
    546         max_queue_len=max_queue_len,
    547         rstate=rstate,
    548         pass_expr_memo_ctrl=pass_expr_memo_ctrl,
    549         verbose=verbose,
    550         catch_eval_exceptions=catch_eval_exceptions,
    551         return_argmin=return_argmin,
    552         show_progressbar=show_progressbar,
    553         early_stop_fn=early_stop_fn,
    554         trials_save_file=trials_save_file,
    555     )
    556
    557 if trials is None:
    558     if os.path.exists(trials_save_file):
```

```
File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\base.py:671, in Trials.fmin(self, fn, space, algo, max_evals, timeout, loss_threshold, max_queue_len, rstate, verbose, pass_expr_memo_ctrl, catch_eval_exceptions, return_argmin, show_progressbar, early_stop_fn, trials_save_file)
    666 # -- Stop-gap implementation!
    667 #     fmin should have been a Trials method in the first place
    668 #     but for now it's still sitting in another file.
    669 from .fmin import fmin
--> 670 return fmin(
    671     fn,
    672     space,
    673     algo=algo,
```

```

675     max_evals=max_evals,
676     timeout=timeout,
677     loss_threshold=loss_threshold,
678     trials=self,
679     rstate=rstate,
680     verbose=verbose,
681     max_queue_len=max_queue_len,
682     allow_trials_fmin=False, # -- prevent recursion
683     pass_expr_memo_ctrl=pass_expr_memo_ctrl,
684     catch_eval_exceptions=catch_eval_exceptions,
685     return_argmin=return_argmin,
686     show_progressbar=show_progressbar,
687     early_stop_fn=early_stop_fn,
688     trials_save_file=trials_save_file,
689 )

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\fmin.py:586, in fmin(fn, space, algo, max_evals, timeout, loss_threshold, trials, rstate, allow_trials_fmin, pass_expr_memo_ctrl, catch_eval_exceptions, verbose, return_argmin, points_to_evaluate, max_queue_len, show_progressbar, early_stop_fn, trials_save_file)
583 rval.catch_eval_exceptions = catch_eval_exceptions
585 # next line is where the fmin is actually executed
--> 586 rval.exhaust()
588 if return_argmin:
589     if len(trials.trials) == 0:

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\fmin.py:364, in FMinIter.exhaust(self)
362 def exhaust(self):
363     n_done = len(self.trials)
--> 364     self.run(self.max_evals - n_done, block_until_done=self.asynchronous)
365     self.trials.refresh()
366     return self

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\fmin.py:300, in FMinIter.run(self, N, block_until_done)
297     time.sleep(self.poll_interval_secs)
298 else:
299     # -- loop over trials and do the jobs directly
--> 300     self.serial_evaluate()
302 self.trials.refresh()
303 if self.trials_save_file != "":

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\fmin.py:178, in FMinIter.serial_evaluate(self, N)
176 ctrl = base.Ctrl(self.trials, current_trial=trial)
177 try:
--> 178     result = self.domain.evaluate(spec, ctrl)
179 except Exception as e:
180     logger.error("job exception: %s" % str(e))

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\hyperopt\base.py:892, in Domain.evaluate(self, config, ctrl, attach_attachments)
883 else:
884     # -- the "work" of evaluating `config` can be written
885     #   either into the pyll part (self.expr)
886     #   or the normal Python part (self.fn)
887     pyll_rval = pyll.rec_eval(
888         self.expr,
889         memo=memo,
890         print_node_on_error=self.rec_eval_print_node_on_error,
891     )
--> 892     rval = self.fn(pyll_rval)
894 if isinstance(rval, (float, int, np.number)):
895     dict_rval = {"loss": float(rval), "status": STATUS_OK}

Cell In[12], line 21, in objective_rf(params)
13 model = RandomForestRegressor(
14     **params,
15     random_state=42,
16     n_jobs=-1,
17     verbose=0
18 )
20 # Treinar
--> 21 model.fit(X_train, y_train)
23 # Avaliar no TEST
24 preds = model.predict(X_test)

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\sklearn\base.py:1365, in _fit_context.
<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
1358     estimator._validate_params()
1360 with config_context(
1361     skip_parameter_validation=
1362         prefer_skip_nested_validation or global_skip_validation

```

```

1363     )
1364 ):
-> 1365     return fit_method(estimator, *args, **kwargs)

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\sklearn\ensemble\_forest.py:486, in
BaseForest.fit(self, X, y, sample_weight)
    475 trees = [
    476     self._make_estimator(append=False, random_state=random_state)
    477     for i in range(n_more_estimators)
    478 ]
    480 # Parallel loop: we prefer the threading backend as the Cython code
    481 # for fitting the trees is internally releasing the Python GIL
    482 # making threading more efficient than multiprocessing in
    483 # that case. However, for joblib 0.12+ we respect any
    484 # parallel_backend contexts set at a higher level,
    485 # since correctness does not rely on using threads.
--> 486 trees = Parallel(
    487     n_jobs=self.n_jobs,
    488     verbose=self.verbose,
    489     prefer="threads",
    490 )(

491     delayed(_parallel_build_trees)(
    492         t,
    493         self.bootstrap,
    494         X,
    495         y,
    496         sample_weight,
    497         i,
    498         len(trees),
    499         verbose=self.verbose,
    500         class_weight=self.class_weight,
    501         n_samples_bootstrap=n_samples_bootstrap,
    502         missing_values_in_feature_mask=missing_values_in_feature_mask,
    503     )
    504     for i, t in enumerate(trees)
    505 )
    507 # Collect newly grown trees
    508 self.estimators_.extend(trees)

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\sklearn\utils\parallel.py:82, in
Parallel.__call__(self, iterable)
    73 warning_filters = warnings.filters
    74 iterable_with_config_and_warning_filters = (
    75     (
    76         _with_config_and_warning_filters(delayed_func, config, warning_filters),
(...):
    80     for delayed_func, args, kwargs in iterable
    81 )
---> 82 return super().__call__(iterable_with_config_and_warning_filters)

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\joblib\parallel.py:2072, in Parallel.__call__(self, iterable)
    2066 # The first item from the output is blank, but it makes the interpreter
    2067 # progress until it enters the Try/Except block of the generator and
    2068 # reaches the first `yield` statement. This starts the asynchronous
    2069 # dispatch of the tasks to the workers.
    2070 next(output)
-> 2072 return output if self.return_generator else list(output)

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\joblib\parallel.py:1682, in
Parallel._get_outputs(self, iterator, pre_dispatch)
    1679     yield
    1681     with self._backend.retrieval_context():
-> 1682         yield from self._retrieve()
    1684 except GeneratorExit:
    1685     # The generator has been garbage collected before being fully
    1686     # consumed. This aborts the remaining tasks if possible and warn
    1687     # the user if necessary.
    1688     self._exception = True

File c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\joblib\parallel.py:1800, in
Parallel._retrieve(self)
    1789 if self.return_ordered:
    1790     # Case ordered: wait for completion (or error) of the next job
    1791     # that have been dispatched and not retrieved yet. If no job
(...):
    1795     # control only have to be done on the amount of time the next
    1796     # dispatched job is pending.
    1797     if (nb_jobs == 0) or (
    1798         self._jobs[0].get_status(timeout=self.timeout) == TASK_PENDING
    1799     ):
-> 1800         time.sleep(0.01)
    1801         continue
    1803 elif nb_jobs == 0:

```

```

1804      # Case unordered: jobs are added to the list of jobs to
1805      # retrieve `self._jobs` only once completed or in error, which
(... ) 1811      # timeouts before any other dispatched job has completed and
1812      # been added to `self._jobs` to be retrieved.

```

KeyboardInterrupt:

Foram cinco minutos para rodar apenas um conjunto de parâmetros! RF é mais pesada que LightGBM, e tende a não acertar tão bem quanto ele. Vou amostrar os dados.

13.5.3.2. Construindo amostragem dos dados

Por que pegar amostra de dados?

- * Complexidade de Memória: O Random Forest tenta carregar e processar os dados de forma diferente. Com 6 milhões de linhas e 31 colunas, o custo de memória e CPU para criar uma única árvore profunda é gigantesco;
- * Falta de Binning: O LightGBM transforma variáveis numéricas em "bins", o que acelera o treino dezenas de vezes. O Random Forest padrão testa todos os pontos de corte possíveis, o que é impraticável nessa volumetria;
- * N_jobs=-1: Se o primeiro trial pegar um parâmetro de `n_estimators=500` e `max_depth=30`, o PC pode "engasgar" para processar tudo em paralelo.

Para este caso, não me parece fazer sentido ficar horas esperando um modelo que provavelmente será pior que o LightGBM. Como o RF atinge a convergência estatística muito antes de 6 milhões de linhas, treinar com uma amostra de 1 milhão de linhas (ou até 500k) dará resultados quase idênticos e será muito mais rápido.

```

In [27]: # Amostra de 1kk linhas - estatisticamente suficiente para achar bons parâmetros
X_train_sample = X_train.sample(n=min(1000000, len(X_train)), random_state=42)
y_train_sample = y_train.loc[X_train_sample.index]

print(f"Amostra de treino para Hyperopt: {X_train_sample.shape}")
print(f"Test completo: {X_test.shape}")
print(f"OOT completo: {X_oot.shape}")

Amostra de treino para Hyperopt: (1000000, 31)
Test completo: (1565303, 31)
OOT completo: (1850732, 31)

```

13.5.3.3. Segunda execução

```

In [28]: print("\n" + "*60)
print("INICIANDO OTIMIZAÇÃO BAYESIANA VIA HYPEROPT")
print("*60)
print("Isso pode alguns minutos. Aguarde...")

trials_rf = Trials()
best_rf = fmin(
    fn=objective_rf,
    space=space_rf,
    algo=tpe.suggest,
    max_evals=50,
    trials=trials_rf,
    rstate=np.random.default_rng(42),
    verbose=1
)

print("\n" + "*60)
print("OTIMIZAÇÃO CONCLUÍDA!")
print("*60)
print("\n💡 Melhores Hiperparâmetros (raw):")
print(best_rf)

```

```

=====
INICIANDO OTIMIZAÇÃO BAYESIANA VIA HYPEROPT
=====
Isso pode alguns minutos. Aguarde...
100%[██████████] 50/50 [51:56<00:00, 62.33s/trial, best loss: 36.06878821535873]

=====
OTIMIZAÇÃO CONCLUÍDA!
=====

💡 Melhores Hiperparâmetros (raw):
{'bootstrap': np.int64(0), 'max_depth': np.float64(13.0), 'max_features': np.int64(1), 'min_samples_leaf': np.float64(25.0),
'n_estimators': np.float64(150.0)}

```

13.5.4. Treinando modelo com melhores hiperparâmetros

```
In [29]:  
print("\n" + "*60)  
print("TREINANDO MODELO FINAL COM MELHORES PARÂMETROS")  
print("*60)  
  
# Reconstruir params (cuidado com hp.choice)  
best_params_rf = {  
    'n_estimators': int(best_rf['n_estimators']),  
    'max_depth': int(best_rf['max_depth']),  
    'min_samples_leaf': int(best_rf['min_samples_leaf']),  
    'max_features': ['sqrt', 0.5][best_rf['max_features']],  
    'bootstrap': [True][best_rf['bootstrap']], # Sempre True no nosso space  
    'random_state': 42,  
    'n_jobs': -1,  
    'verbose': 0  
}  
  
print("Parâmetros finais:")  
for k, v in best_params_rf.items():  
    print(f" {k}: {v}")  
  
# Treinar no dataset COMPLETO  
model_rf_final = RandomForestRegressor(**best_params_rf)  
model_rf_final.fit(X_train, y_train)  
  
print(f" ✅ Modelo treinado com {model_rf_final.n_estimators} árvores")
```

```
=====  
TREINANDO MODELO FINAL COM MELHORES PARÂMETROS  
=====  
Parâmetros finais:  
    n_estimators: 150  
    max_depth: 13  
    min_samples_leaf: 25  
    max_features: 0.5  
    bootstrap: True  
    random_state: 42  
    n_jobs: -1  
    verbose: 0  
✅ Modelo treinado com 150 árvores
```

13.5.5. Resultados (Test + OOT)

```
In [31]:  
print("\n" + "*60)  
print("AVALIAÇÃO FINAL - RANDOM FOREST (TUNED)")  
print("*60)  
  
for name, X, y in [("TEST", X_test, y_test), ("OOT", X_oot, y_oot)]:  
    preds = model_rf_final.predict(X)  
    rmse = root_mean_squared_error(y, preds)  
    mae = mean_absolute_error(y, preds)  
    r2 = r2_score(y, preds)  
  
    print(f"\n--- {name} ---")  
    print(f"RMSE: {rmse:.4f}")  
    print(f"MAE: {mae:.4f}")  
    print(f"R²: {r2:.4f}")
```

```
=====  
AVALIAÇÃO FINAL - RANDOM FOREST (TUNED)  
=====  
  
--- TEST ---  
RMSE: 36.1329  
MAE: 19.9147  
R²: 0.6438  
  
--- OOT ---  
RMSE: 30.6644  
MAE: 15.7750  
R²: 0.6685
```

13.5.6. Comparação dos modelos executados até então

```
In [32]:  
print("\n" + "*60)  
print("📊 COMPARAÇÃO DE TODOS OS MODELOS (OOT)")  
print("*60)  
  
preds_rf_oot = model_rf_final.predict(X_oot)  
rmse_rf = root_mean_squared_error(y_oot, preds_rf_oot)  
mae_rf = mean_absolute_error(y_oot, preds_rf_oot)  
r2_rf = r2_score(y_oot, preds_rf_oot)  
  
comparison = pd.DataFrame({  
    'Modelo': ['Elastic Net', 'Decision Tree (Baseline)', 'LightGBM (Tuned)', 'Random Forest (Tuned)'],  
    'RMSE OOT': [32.27, 31.30, 30.32, rmse_rf],  
    'MAE OOT': [16.62, 15.32, 13.96, mae_rf],  
    'R² OOT': [0.633, 0.654, 0.676, r2_rf]  
})  
  
print(comparison.to_string(index=False))  
  
# Ganhos  
ganho_rmse_en = ((32.27 - rmse_rf) / 32.27) * 100  
ganho_mae_en = ((16.62 - mae_rf) / 16.62) * 100  
diff_rmse_lgbm = ((rmse_rf - 30.32) / 30.32) * 100  
diff_mae_lgbm = ((mae_rf - 13.96) / 13.96) * 100  
  
print(f"\n🎯 Ganho do Random Forest vs Elastic Net:")  
print(f"    RMSE: {ganho_rmse_en:+.2f}%)")  
print(f"    MAE: {ganho_mae_en:+.2f}%)")  
  
print(f"\n🔍 Random Forest vs LightGBM:")  
print(f"    RMSE: {diff_rmse_lgbm:+.2f}%)")  
print(f"    MAE: {diff_mae_lgbm:+.2f}%)")
```

```
=====  
📊 COMPARAÇÃO DE TODOS OS MODELOS (OOT)  
=====  
          Modelo RMSE OOT   MAE OOT   R² OOT  
Elastic Net 32.270000 16.620000 0.633000  
Decision Tree (Baseline) 31.300000 15.320000 0.654000  
LightGBM (Tuned) 30.320000 13.960000 0.676000  
Random Forest (Tuned) 30.664446 15.775009 0.668457  
  
🎯 Ganho do Random Forest vs Elastic Net:  
RMSE: +4.98%  
MAE: +5.08%  
  
🔍 Random Forest vs LightGBM:  
RMSE: +1.14%  
MAE: +13.00%
```

13.5.7. Conclusões sobre os resultados

O Random Forest apresentou uma performance de generalização (OOT) robusta, com um R^2 de 0.668, superando o Elastic Net e a Decision Tree.

Ao comparar com o LightGBM, observamos que o Random Forest obteve um RMSE muito competitivo (apenas 1.14% superior), indicando excelente controle sobre grandes erros. No entanto, o MAE foi 13% superior ao LightGBM.

Essa divergência sugere que, enquanto o LightGBM é mais preciso na estimativa pontual da maioria dos clientes (menor MAE), o Random Forest oferece uma estimativa mais estável e conservadora, sendo menos propenso a overfitting em casos extremos, embora sacrifique a precisão média no processo.

13.5.8. Feature Importance

```
#### 13.5.8.1. Mean Decrease Impurity (MDI)
```

In [33]:

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

print("*"*60)
print("FEATURE IMPORTANCE - RANDOM FOREST (MDI/GAIN)")
print("*"*60)

# Extrair importâncias do modelo treinado
importances_rf = pd.DataFrame({
    'feature': feature_cols,
    'importance': model_rf_final.feature_importances_
}).sort_values('importance', ascending=False).reset_index(drop=True)

# Adicionar importância relativa (%)
importances_rf['importance_pct'] = (importances_rf['importance'] / importances_rf['importance'].sum()) * 100

# Adicionar importância acumulada
importances_rf['importance_cumsum'] = importances_rf['importance_pct'].cumsum()

print("\n📊 Top 15 Features (Random Forest):")
print(importances_rf.head(15).to_string(index=False))

# Identificar quantas features explicam 80% da importância
n_features_80 = (importances_rf['importance_cumsum'] <= 80).sum()
print(f"\n🔴 {n_features_80} features explicam 80% da importância total")
```

```
=====
FEATURE IMPORTANCE - RANDOM FOREST (MDI/GAIN)
=====
```

📊 Top 15 Features (Random Forest):

	feature	importance	importance_pct	importance_cumsum
1	flag_plano_mensal_max_3	0.383604	38.360427	38.360427
2	flag_plano_mensal	0.188339	18.833896	57.194322
3	daily_revenue_efficiency	0.131438	13.143784	70.338106
4	daily_revenue_efficiency_min_3	0.121355	12.135483	82.473589
5	flag_valid_fee_max_3	0.044264	4.426402	86.899991
6	log_total_plays	0.038924	3.892389	90.792381
7	revenue_per_hour_listened_cap	0.015729	1.572866	92.365246
8	total_secs_mean_3	0.014631	1.463149	93.828395
9	registered_via_group_te	0.011799	1.179863	95.008258
10	total_plays_group_te	0.009758	0.975780	95.984038
11	daily_revenue_efficiency_min_3_is_sentinel	0.009053	0.905270	96.889308
12	revenue_per_hour_tier_te	0.005128	0.512849	97.402157
13	log_total_plays_mean_3	0.004727	0.472711	97.874868
14	faixa_idade_te	0.004171	0.417101	98.291969
15	usage_intensity_per_tenure_cap	0.004099	0.409883	98.701852

🔴 3 features explicam 80% da importância total

Correcao: sao na verdade 4 features, pra passar de 70% para 82%*

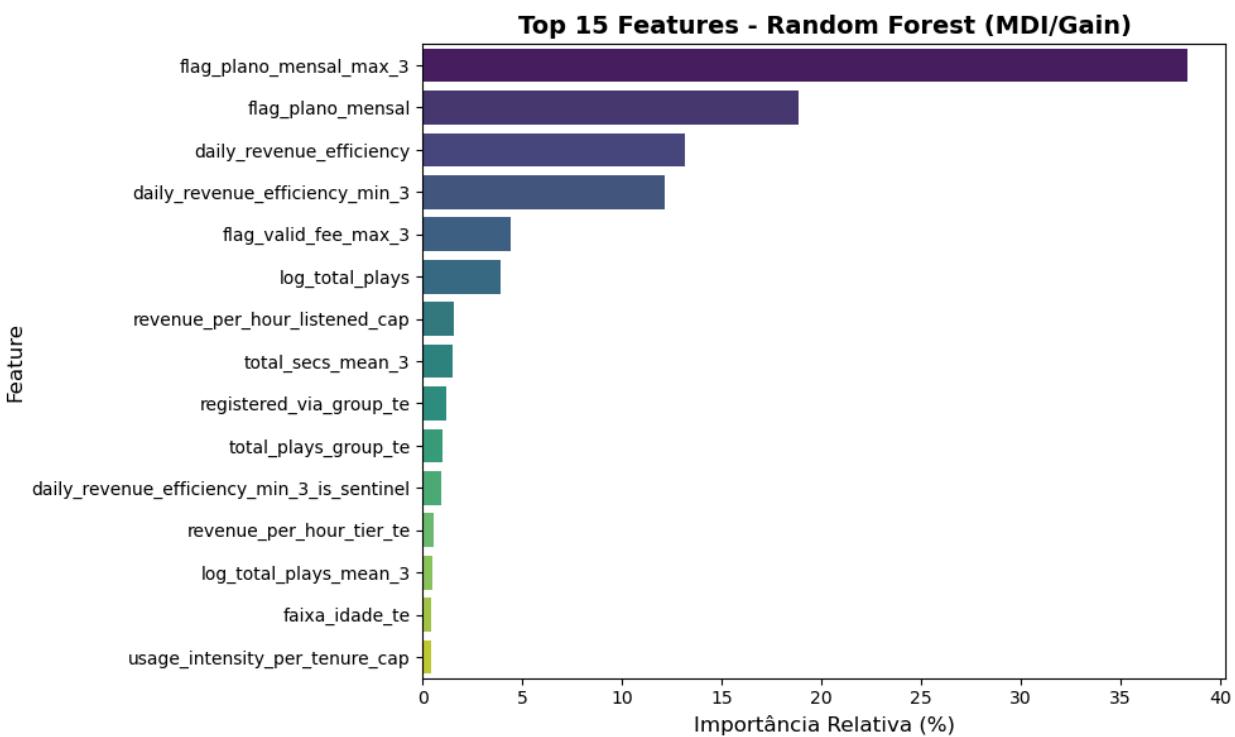
In [34]:

```
# =====
# VISUALIZAÇÃO 1: Top 15 Features (Barplot)
# =====
plt.figure(figsize=(10, 6))
top_15 = importances_rf.head(15)
sns.barplot(data=top_15, y='feature', x='importance_pct', palette='viridis')
plt.title('Top 15 Features - Random Forest (MDI/Gain)', fontsize=14, fontweight='bold')
plt.xlabel('Importância Relativa (%)', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.tight_layout()
plt.show()
```

C:\Users\Gustavo\AppData\Local\Temp\ipykernel_26864\1224318005.py:6: FutureWarning:

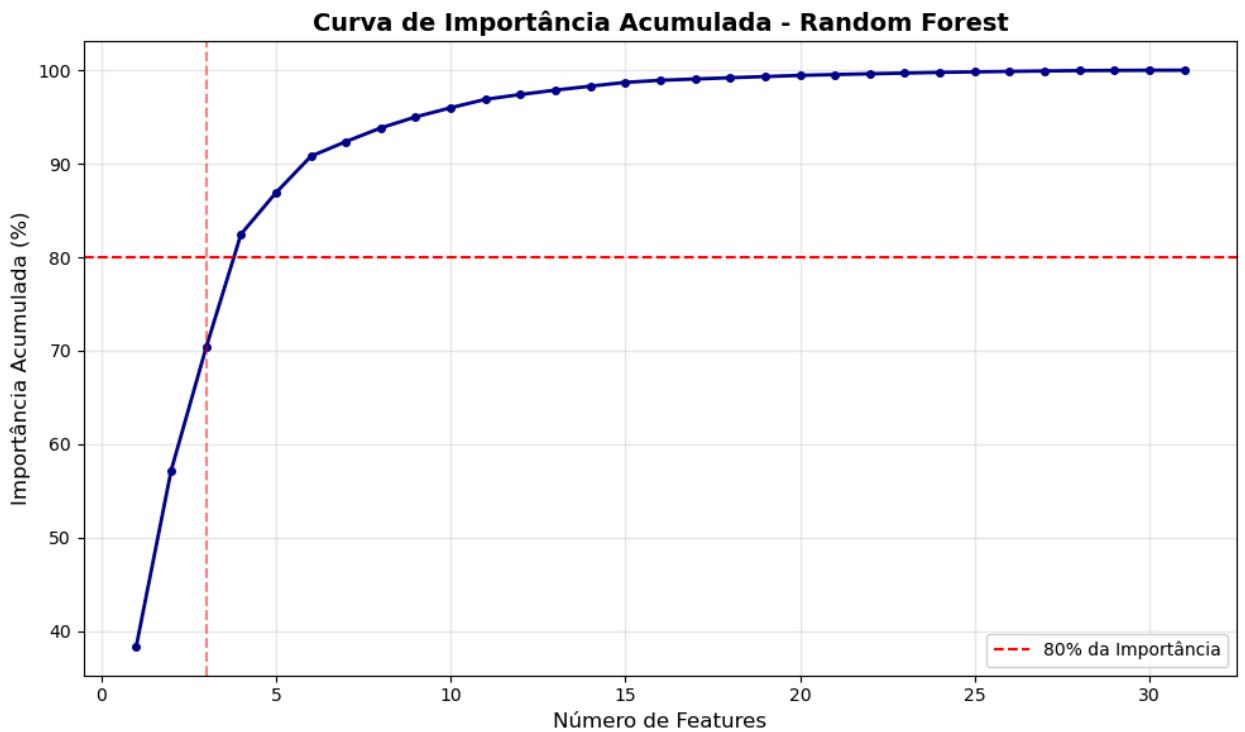
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=top_15, y='feature', x='importance_pct', palette='viridis')
```



In [35]:

```
# =====
# VISUALIZAÇÃO 2: Curva de Importância Acumulada
# =====
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(importances_rf) + 1), importances_rf['importance_cumsum'],
         marker='o', linewidth=2, markersize=4, color='darkblue')
plt.axhline(y=80, color='red', linestyle='--', linewidth=1.5, label='80% da Importância')
plt.axvline(x=n_features_80, color='red', linestyle='--', linewidth=1.5, alpha=0.5)
plt.title('Curva de Importância Acumulada - Random Forest', fontsize=14, fontweight='bold')
plt.xlabel('Número de Features', fontsize=12)
plt.ylabel('Importância Acumulada (%)', fontsize=12)
plt.grid(alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```



A importância nativa, baseada em MDI (Mean Decrease Impurity) ou Gain, mede o quanto cada variável contribuiu para reduzir a incerteza (impureza) nos nós das árvores durante o treinamento.

- Concentração de Importância: Apenas 3 features explicam 80% do comportamento do modelo. Isso indica um modelo muito eficiente, que foca em sinais claros de faturamento e tipo de contrato;
- Top Features: flag_plano_mensal_max_3 (38.36%) e flag_plano_mensal (18.83%) dominam, mostrando que a estratégia de assinatura é o maior preditor de rentabilidade;
- Eficiência de Receita: daily_revenue_efficiency aparece com destaque (13.14%), validando que o quanto eficiente o cliente é em gerar receita por dia é um diferencial crítico.

13.5.8.2. SHAP Values

```
In [36]:
import shap
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

print("*"*60)
print("SHAP VALUES - RANDOM FOREST")
print("*"*60)
print("⚠️ Isso pode levar alguns minutos. Aguarde...")

# =====
# 1. CRIAR AMOSTRA PARA O SHAP (Acelera o cálculo)
# =====
# SHAP no RF pode ser lento. Usamos 5.000 linhas do OOT
np.random.seed(42)
sample_indices = np.random.choice(X_oot.index, size=min(5000, len(X_oot)), replace=False)
X_shap = X_oot.loc[sample_indices]

print(f"Amostra para SHAP: {X_shap.shape}")

# =====
# 2. CALCULAR SHAP VALUES
# =====
explainer = shap.TreeExplainer(model_rf_final)
shap_values = explainer.shap_values(X_shap)

print(f"✅ SHAP values calculados: {shap_values.shape}")
```

```
c:\Users\Gustavo\anaconda3\envs\new_datamaster\Lib\site-packages\tqdm\auto.py:21: TqdmWarning: IPProgress not found.
Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
    from .autonotebook import tqdm

=====
SHAP VALUES - RANDOM FOREST
=====
⚠️ Isso pode levar alguns minutos. Aguarde...
Amostra para SHAP: (5000, 31)
✅ SHAP values calculados: (5000, 31)
```

```
In [46]:
i = 1
x0 = X_shap.iloc[i:i+1]

pred0 = model_rf_final.predict(x0)[0]
base_value = explainer.expected_value

print("Pred:", pred0)
print("Base value (E[f(x)]):", base_value)
print("Base + sum(SHAP):", base_value + shap_values[i].sum())

Pred: 84.92638753968073
Base value (E[f(x)]): [43.67735657]
Base + sum(SHAP): [84.92638754]
```

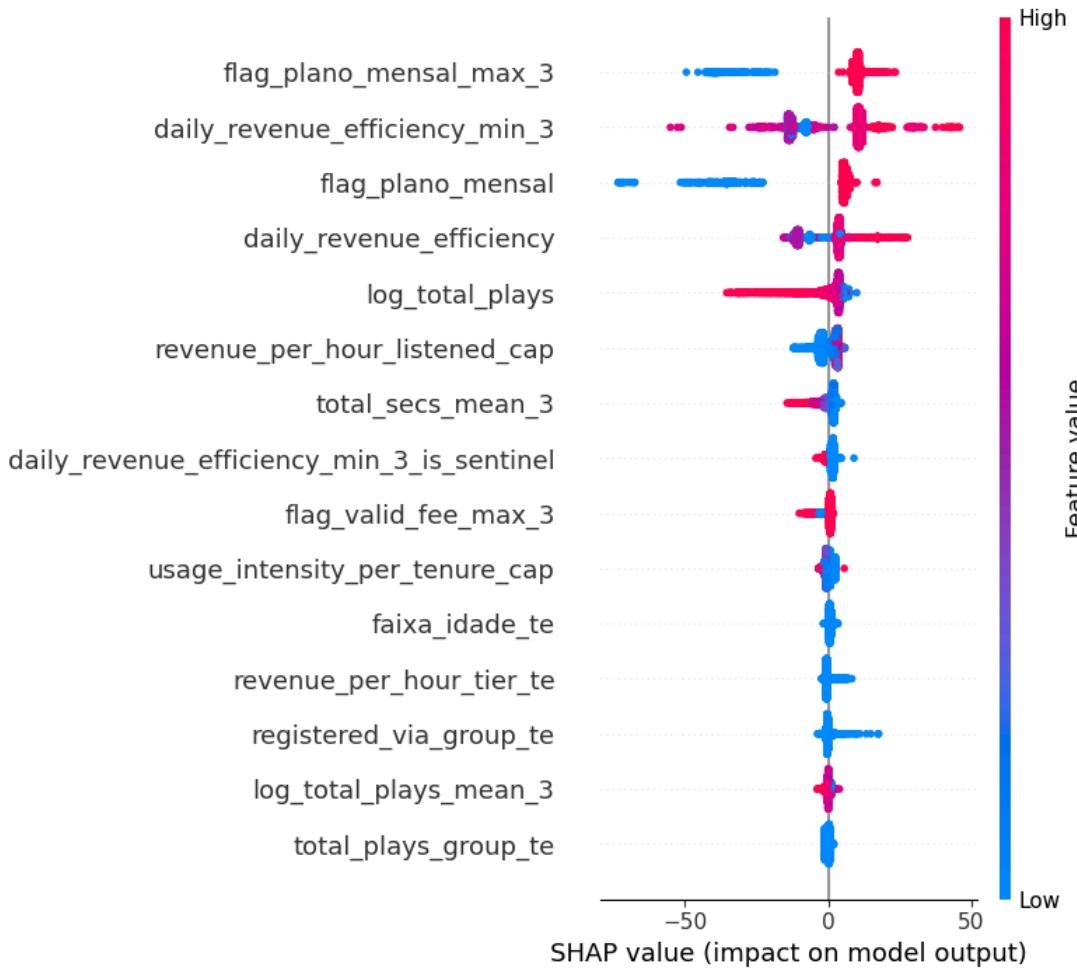
In [47]:

```
# =====
# 3. VISUALIZAÇÃO 1: Summary Plot (Beeswarm)
# =====
print("\n📊 Gerando Summary Plot (Beeswarm)...")
plt.figure(figsize=(10, 8))
shap.summary_plot(shap_values, X_shap, feature_names=feature_cols, show=False, max_display=15)
plt.title('SHAP Summary Plot - Random Forest (Top 15)', fontsize=14, fontweight='bold', pad=20)
plt.tight_layout()
plt.show()
```

📊 Gerando Summary Plot (Beeswarm)...

C:\Users\Gustavo\AppData\Local\Temp\ipykernel_26864\3263841930.py:6: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future version this function will no longer use the global RNG. Pass `rng` explicitly to opt-in to the new behaviour and silence this warning.
 shap.summary_plot(shap_values, X_shap, feature_names=feature_cols, show=False, max_display=15)

SHAP Summary Plot - Random Forest (Top 15)



In [49]:

```
# =====
# 5. TABELA: SHAP Importance (Mean Absolute SHAP)
# =====

shap_importance_rf = pd.DataFrame({
    'feature': feature_cols,
    'shap_importance': np.abs(shap_values).mean(axis=0)
}).sort_values('shap_importance', ascending=False).reset_index(drop=True)

# Adicionar importância relativa (%)
shap_importance_rf['shap_importance_pct'] = (
    shap_importance_rf['shap_importance'] / shap_importance_rf['shap_importance'].sum()
) * 100

# Adicionar importância acumulada
shap_importance_rf['shap_cumsum'] = shap_importance_rf['shap_importance_pct'].cumsum()

print("\n📊 Top 15 Features (SHAP):")
print(shap_importance_rf.head(15).to_string(index=False))

# Identificar quantas features explicam 80% da importância SHAP
n_features_80_shap = (shap_importance_rf['shap_cumsum'] <= 80).sum()
print(f"\n🎯 {n_features_80_shap} features explicam 80% da importância SHAP")
```

📊 Top 15 Features (SHAP):

	feature	shap_importance	shap_importance_pct	shap_cumsum
	flag_plano_mensal_max_3	12.492821	21.645152	21.645152
	daily_revenue_efficiency_min_3	11.952961	20.709787	42.354939
	flag_plano_mensal	9.309943	16.130474	58.485413
	daily_revenue_efficiency	7.164424	12.413133	70.898546
	log_total_plays	4.376658	7.583029	78.481574
	revenue_per_hour_listened_cap	2.542718	4.405532	82.887107
	total_secs_mean_3	2.118418	3.670386	86.557492
	daily_revenue_efficiency_min_3_is_sentinel	1.480944	2.565894	89.123387
	flag_valid_fee_max_3	0.943074	1.633977	90.757364
	usage_intensity_per_tenure_cap	0.909121	1.575150	92.332514
	faixa_idade_te	0.843946	1.462227	93.794741
	revenue_per_hour_tier_te	0.805660	1.395893	95.190634
	registered_via_group_te	0.520207	0.901315	96.091949
	log_total_plays_mean_3	0.473028	0.819571	96.911520
	total_plays_group_te	0.426099	0.738263	97.649783

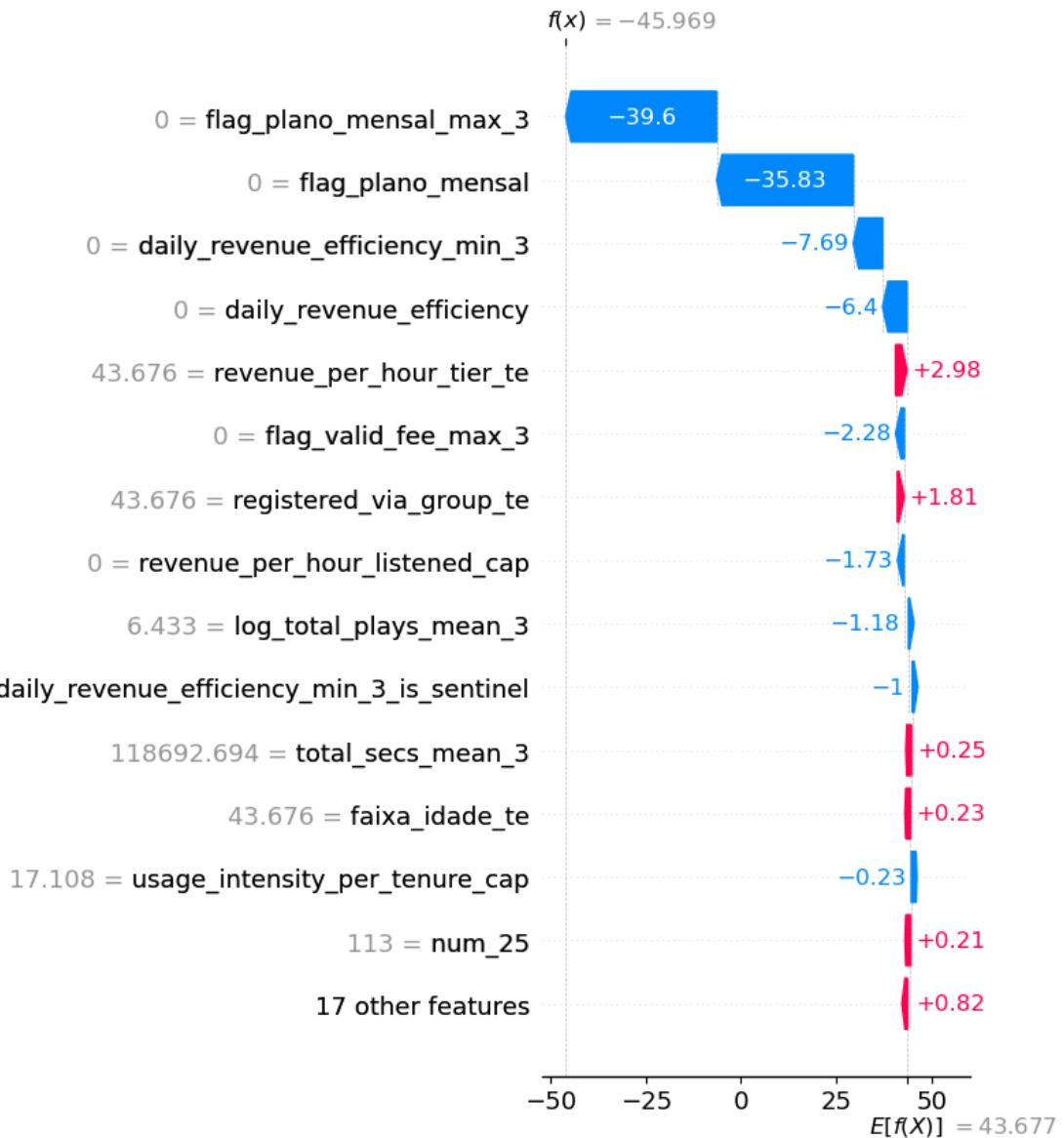
🎯 5 features explicam 80% da importância SHAP

Diferente do MDI, o SHAP é baseado na "Teoria dos Jogos" e mostra a contribuição marginal de cada variável para a predição final em relação a um valor base (média). Percebe-se que, para o SHAP, são necessárias 5 features (em vez de 3) para explicar ~80% da importância. O SHAP "distribui" melhor a importância, valorizando mais o histórico de 3 meses (`daily_revenue_efficiency_min_3`).

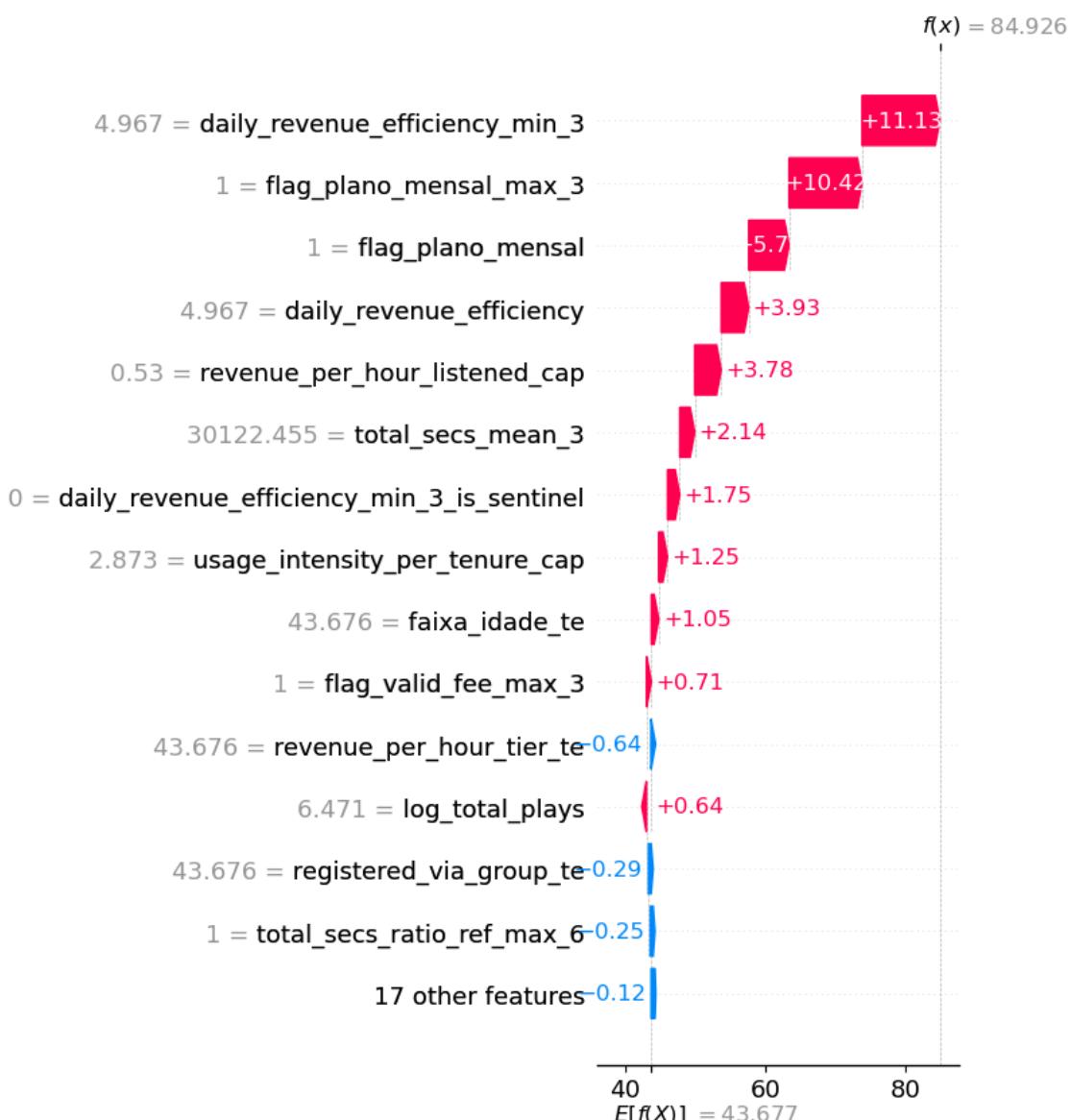
```
In [ ]: print("\nGerando Waterfall Plot (Exemplo Individual)...")
# Pegar as primeiras 3 observações da amostra
for i in range (0, 3):
    sample_idx = i
    plt.figure(figsize=(10, 8))
    shap.waterfall_plot(
        shap.Explanation(
            values=shap_values[sample_idx],
            base_values=explainer.expected_value,
            data=X_shap.iloc[sample_idx].values,
            feature_names=feature_cols
        ),
        max_display=15,
        show=False
    )
    plt.title(f'Waterfall Plot - Observação {sample_indices[sample_idx]}',
              fontsize=14, fontweight='bold', pad=20)
    plt.tight_layout()
    plt.show()
```

Gerando Waterfall Plot (Exemplo Individual)...

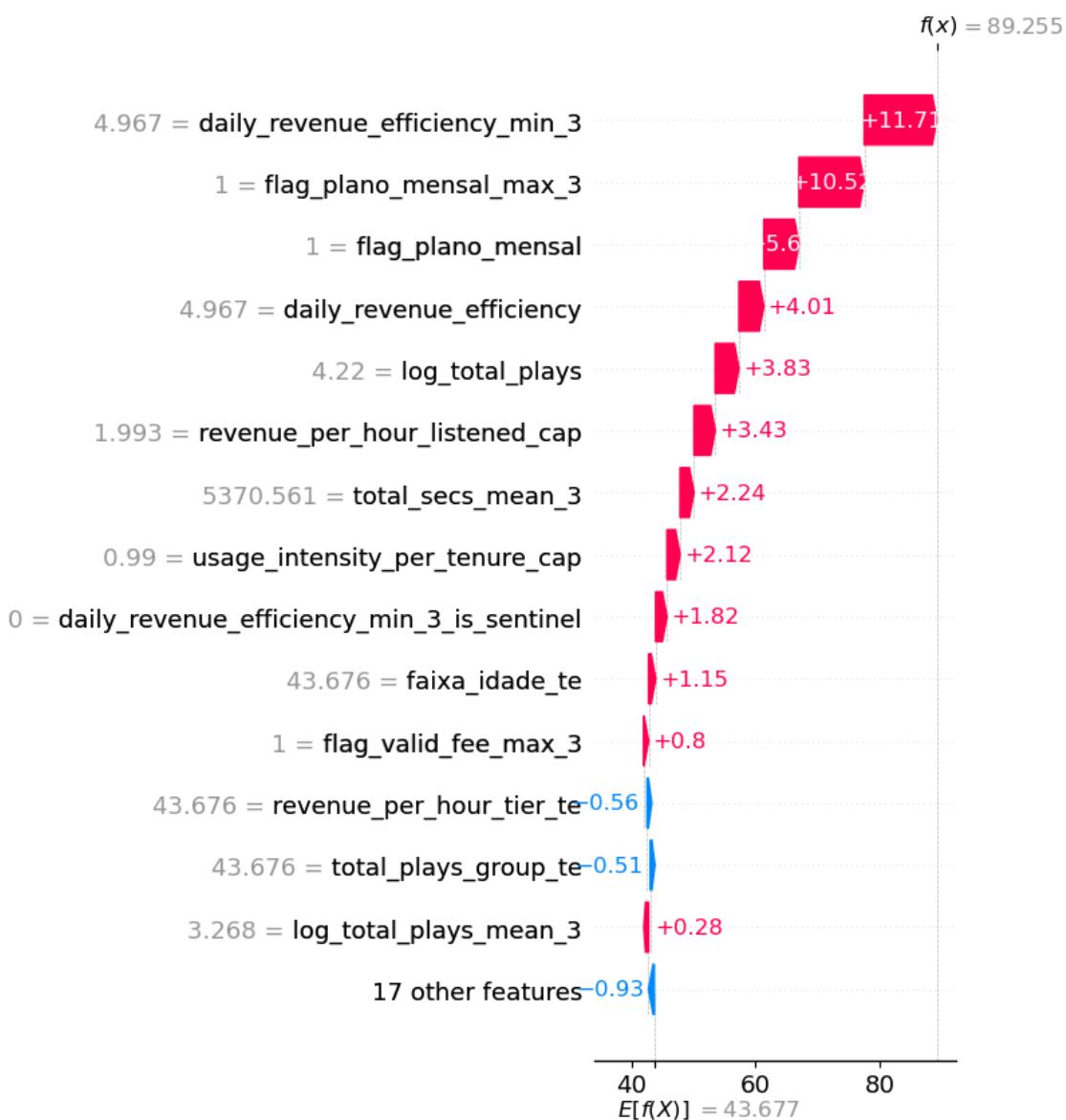
Waterfall Plot - Observação 1124269



Waterfall Plot - Observação 3071760



Waterfall Plot - Observação 6583412



- Primeiro cliente: exemplo de baixo valor. Percebe-se dominância das setas azuis, mostrando que a ausência do plano mensal (`flag_plano_mensal_max_3 == 0`) "derrubou" a predição em quase 40 pontos;
- Segundo e terceiro clientes: exemplo de sucesso. As setas vermelhas "empurram" o valor para a direita, principalmente pelo status do plano mensal.

In [55]:

```
# =====
# 8. COMPARAÇÃO: MDI vs SHAP (Top 10)
# =====
print("\n📊 Comparação: MDI (Nativo) vs SHAP (Top 10)...")

comparison_importance = pd.merge(
    importances_rf[['feature', 'importance_pct']].head(10).rename(columns={'importance_pct': 'MDI_pct'}),
    shap_importance_rf[['feature', 'shap_importance_pct']].head(10),
    on='feature',
    how='outer'
).fillna(0)

print("\n" + "*60)
print("COMPARAÇÃO: MDI vs SHAP (Top 10)")
print("*60)
print(comparison_importance.to_string(index=False))

# Visualização comparativa
fig, ax = plt.subplots(figsize=(12, 6))
x = np.arange(len(comparison_importance))
width = 0.35

ax.bars(x - width/2, comparison_importance['MDI_pct'], width, label='MDI (Nativo)', color='steelblue')
ax.bars(x + width/2, comparison_importance['shap_importance_pct'], width, label='SHAP', color='coral')

ax.set_yticks(x)
ax.set_yticklabels(comparison_importance['feature'])
ax.invert_yaxis()
ax.set_xlabel('Importância (%)', fontsize=12)
ax.set_title('Comparação: MDI vs SHAP - Random Forest (Top 10)', fontsize=14, fontweight='bold')
ax.legend()
ax.grid(axis='x', alpha=0.3)

plt.tight_layout()
plt.show()

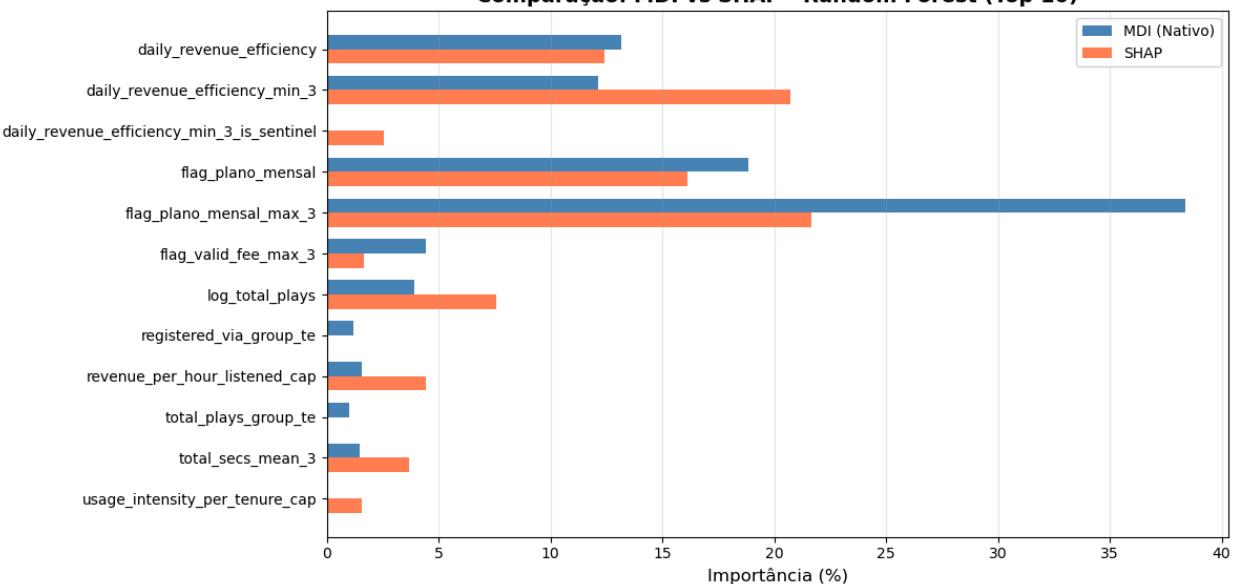
print("\n" + "*60)
print("✅ SHAP VALUES (RANDOM FOREST) CONCLUÍDO!")
print("*60)
```

📊 Comparação: MDI (Nativo) vs SHAP (Top 10)...

```
=====
COMPARAÇÃO: MDI vs SHAP (Top 10)
=====
```

feature	MDI_pct	shap_importance_pct
daily_revenue_efficiency	13.143784	12.413133
daily_revenue_efficiency_min_3	12.135483	20.709787
daily_revenue_efficiency_min_3_is_sentinel	0.000000	2.565894
flag_plano_mensal	18.833896	16.130474
flag_plano_mensal_max_3	38.360427	21.645152
flag_valid_fee_max_3	4.426402	1.633977
log_total_plays	3.892389	7.583029
registered_via_group_te	1.179863	0.000000
revenue_per_hour_listened_cap	1.572866	4.405532
total_plays_group_te	0.975780	0.000000
total_secs_mean_3	1.463149	3.670386
usage_intensity_per_tenure_cap	0.000000	1.575150

Comparação: MDI vs SHAP - Random Forest (Top 10)



Este comparativo é a chave dessa análise. Atraves dele, entende duas diferenças principais: onde o algoritmo foca (MDI) vs o que realmente impacta o resultado final (SHAP).

- Divergências Positivas: features como `daily_revenue_efficiency_min_3` ganham muito mais relevância no SHAP (20.7%) do que no MDI (12.1%). Isso sugere que, embora o modelo use o plano mensal para criar as divisões principais nas árvores, o histórico de eficiência é o que dá o "ajuste fino" no valor final da margem;
- Flags de "Sentinela": `daily_revenue_efficiency_min_3_is_sentinel` e `usage_intensity_per_tenure_cap` aparece no SHAP mas é quase nula no MDI nativo. O SHAP consegue captar sinais sutis que a importância nativa ignora.
- Features categoricas (com TE): `registered_via_group_te` e `total_plays_group_te` com valores significativos somente no MDI.

13.5.9. Salvar base

```
In [37]: gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"
```

```
In [38]: print("=*60")
print("GERANDO PREDIÇÕES (TRAIN/TEST/OOT)")
print("=*60")

def add_predictions(pdf, set_name):
    """Adiciona predições e resíduos ao dataframe"""
    pdf = pdf.copy()
    pdf["prediction_rfr"] = model_rf_final.predict(pdf[feature_cols])
    pdf["residual_rfr"] = pdf["target_win"] - pdf["prediction_rfr"]
    pdf["set"] = set_name
    return pdf

# Gerar predições
train_pred_pd = add_predictions(train_pd, "train")
test_pred_pd = add_predictions(test_pd, "test")
oot_pred_pd = add_predictions(oot_pd, "oot")

print(f"✓ Train: {train_pred_pd.shape}")
print(f"✓ Test: {test_pred_pd.shape}")
print(f"✓ OOT: {oot_pred_pd.shape}")

# Selecionar colunas relevantes (reduz tamanho do arquivo)
keep_cols = [
    "msno",
    "safra",
    "partition",
    "set",
    "target_win",
    "prediction_rfr",
    "residual_rfr"
] + feature_cols

train_pred_pd = train_pred_pd[keep_cols]
test_pred_pd = test_pred_pd[keep_cols]
oot_pred_pd = oot_pred_pd[keep_cols]

# Consolidar tudo
all_pred_pd = pd.concat([train_pred_pd, test_pred_pd, oot_pred_pd], ignore_index=True)
print(f"\n✓ Base consolidada: {all_pred_pd.shape}")
print(f"    Colunas: {list(all_pred_pd.columns)}")
```

=====

GERANDO PREDIÇÕES (TRAIN/TEST/OOT)

=====

✓ Train: (6262831, 38)
✓ Test: (1565303, 38)
✓ OOT: (1850732, 38)

✓ Base consolidada: (9678866, 38)
Colunas: ['msno', 'safra', 'partition', 'set', 'target_win', 'prediction_rfr', 'residual_rfr',
'daily_revenue_efficiency', 'daily_revenue_efficiency_min_3', 'daily_revenue_efficiency_min_3_is_sentinel',
'flag_plano_mensal', 'flag_plano_mensal_max_3', 'flag_valid_fee_max_3', 'log_total_plays', 'log_total_plays_mean_3',
'log_total_plays_mean_3_is_sentinel', 'num_25', 'num_50', 'num_75', 'num_985', 'plays_per_unq_cap_min_6',
'plays_per_unq_cap_min_6_is_sentinel', 'revenue_per_hour_listened_cap', 'total_secs_mean_3',
'total_secs_mean_3_is_sentinel', 'total_secs_ratio_ref_max_6', 'total_secs_ratio_ref_max_6_is_sentinel',
'usage_intensity_per_tenure_cap', 'avg_secs_per_unq_cap_group_te', 'early_drop_rate_group_te', 'faixa_idade_te',
'gender_clean_te', 'plays_behavior_vs_completion_collapsed_te', 'plays_per_unq_behavior_te', 'registered_via_group_te',
'revenue_per_hour_tier_te', 'total_plays_group_te', 'usage_intensity_tier_te']

```
In [39]: print("\n" + "*60)
print("SALVANDO DIRETO EM PARQUET (PANDAS)")
print("*60)

# Caminho de saída
out_path = gold_path + "df_predictions_random_forest"

# Salvar particionado por safra usando Pandas + PyArrow
all_pred_pd.to_parquet(
    out_path,
    engine='pyarrow',
    partition_cols=['safra'],
    compression='snappy',
    index=False
)

print(f"✅ Sucesso! Predições salvas em: {out_path}")
print(f"  Formato: Parquet particionado por safra")
print(f"  Engine: PyArrow")
print(f"  Compressão: Snappy")
```

```
=====
SALVANDO DIRETO EM PARQUET (PANDAS)
=====
✅ Sucesso! Predições salvas em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_predictions_random_forest
  Formato: Parquet particionado por safra
  Engine: PyArrow
  Compressão: Snappy
```

13.5.10. Salvar modelo + metadados

```
In [56]: path_model = "C:/Users/Gustavo/Downloads/datamaster/models/random_forest_regressor_model"
```

```
In [57]: print("\n" + "*60)
print("SALVANDO MODELO E METADADOS")
print("*60)
```

```
import os
import json
import joblib

# Criar diretório
os.makedirs(path_model, exist_ok=True)

# 4.1 Salvar modelo completo (joblib) - para reuso em Python
model_path_pkl = os.path.join(path_model, "model.pkl")
joblib.dump(model_rf_final, model_path_pkl)
print(f"✓ Modelo (joblib) salvo: {model_path_pkl}")

# 4.2 Salvar metadados (features, params, n_estimators)
metadata = {
    "model_type": "RandomForestRegressor",
    "optimization": "Hyperopt (Bayesian)",
    "n_trials": 50,
    "feature_cols": feature_cols,
    "n_features": len(feature_cols),
    "n_estimators": int(model_rf_final.n_estimators),
    "n_trees_trained": int(model_rf_final.n_estimators), # Total de árvores treinadas
    "params": model_rf_final.get_params(),
    "train_shape": list(X_train.shape),
    "test_shape": list(X_test.shape),
    "oot_shape": list(X_oot.shape)
}

metadata_path = os.path.join(path_model, "metadata.json")
with open(metadata_path, "w", encoding="utf-8") as f:
    json.dump(metadata, f, ensure_ascii=False, indent=2, default=str)
print(f"✓ Metadados salvos: {metadata_path}")

print("\n" + "*60)
print("✓ TUDO SALVO COM SUCESSO!")
print("*60)
```

```
=====
SALVANDO MODELO E METADADOS
=====
✓ Modelo (joblib) salvo: C:/Users/Gustavo/Downloads/datamaster/models/random_forest_regressor_model\model.pkl
✓ Metadados salvos: C:/Users/Gustavo/Downloads/datamaster/models/random_forest_regressor_model\metadata.json

=====
✓ TUDO SALVO COM SUCESSO!
=====
```

13.6. Escolha do modelo finalista

13.6.1. Carregando bases de predições EN, LGBM, RFR

```
In [5]: df_predictions_elastic_net = spark.read.parquet(gold_path + "df_predictions_elastic_net")
df_predictions_lightgbm = spark.read.parquet(gold_path + "df_predictions_lightgbm")
df_predictions_random_forest = spark.read.parquet(gold_path + "df_predictions_random_forest")
```

```
In [10]: df_predictions_elastic_net.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- target_win: double (nullable = true)
 |-- features: vector (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- actual_amount_paid: double (nullable = true)
 |-- flag_plano_mensal_max_3: integer (nullable = true)
 |-- log_total_secs_mean_3: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- payment_method_group_ohe: vector (nullable = true)
 |-- revenue_tier_ohe: vector (nullable = true)
 |-- revenue_per_hour_tier_ohe: vector (nullable = true)
 |-- plays_behavior_vs_volume_collapsed_ohe: vector (nullable = true)
 |-- plays_behavior_vs_completion_collapsed_ohe: vector (nullable = true)
 |-- usage_intensity_tier_ohe: vector (nullable = true)
 |-- registered_via_group_ohe: vector (nullable = true)
 |-- faixa_idade_ohe: vector (nullable = true)
 |-- partition: string (nullable = true)
 |-- prediction: double (nullable = true)
 |-- residual: double (nullable = true)
 |-- set: string (nullable = true)
 |-- safra: integer (nullable = true)
```

```
In [11]: df_predictions_lightgbm.printSchema()
```

```
root
 |-- msno: string (nullable = true)
 |-- partition: string (nullable = true)
 |-- set: string (nullable = true)
 |-- target_win: double (nullable = true)
 |-- prediction_lgbm: double (nullable = true)
 |-- residual_lgbm: double (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- daily_revenue_efficiency_min_3_is_sentinel: long (nullable = true)
 |-- flag_plano_mensal: long (nullable = true)
 |-- flag_plano_mensal_max_3: long (nullable = true)
 |-- flag_valid_fee_max_3: long (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- log_total_plays_mean_3_is_sentinel: long (nullable = true)
 |-- num_25: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- plays_per_unq_cap_min_6: double (nullable = true)
 |-- plays_per_unq_cap_min_6_is_sentinel: long (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- total_secs_mean_3_is_sentinel: long (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- total_secs_ratio_ref_max_6_is_sentinel: long (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- avg_secs_per_unq_cap_group_te: double (nullable = true)
 |-- early_drop_rate_group_te: double (nullable = true)
 |-- faixa_idade_te: double (nullable = true)
 |-- gender_clean_te: double (nullable = true)
 |-- plays_behavior_vs_completion_collapsed_te: double (nullable = true)
 |-- plays_per_unq_behavior_te: double (nullable = true)
 |-- registered_via_group_te: double (nullable = true)
 |-- revenue_per_hour_tier_te: double (nullable = true)
 |-- total_plays_group_te: double (nullable = true)
 |-- usage_intensity_tier_te: double (nullable = true)
 |-- safra: integer (nullable = true)
```

In [12]: df_predictions_random_forest.printSchema()

```
root
 |-- msno: string (nullable = true)
 |-- partition: string (nullable = true)
 |-- set: string (nullable = true)
 |-- target_win: double (nullable = true)
 |-- prediction_rfr: double (nullable = true)
 |-- residual_rfr: double (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- daily_revenue_efficiency_min_3: double (nullable = true)
 |-- daily_revenue_efficiency_min_3_is_sentinel: byte (nullable = true)
 |-- flag_plano_mensal: byte (nullable = true)
 |-- flag_plano_mensal_max_3: byte (nullable = true)
 |-- flag_valid_fee_max_3: byte (nullable = true)
 |-- log_total_plays: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- log_total_plays_mean_3_is_sentinel: byte (nullable = true)
 |-- num_25: double (nullable = true)
 |-- num_50: double (nullable = true)
 |-- num_75: double (nullable = true)
 |-- num_985: double (nullable = true)
 |-- plays_per_unq_cap_min_6: double (nullable = true)
 |-- plays_per_unq_cap_min_6_is_sentinel: byte (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- total_secs_mean_3_is_sentinel: byte (nullable = true)
 |-- total_secs_ratio_ref_max_6: double (nullable = true)
 |-- total_secs_ratio_ref_max_6_is_sentinel: byte (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- avg_secs_per_unq_cap_group_te: double (nullable = true)
 |-- early_drop_rate_group_te: double (nullable = true)
 |-- faixa_idade_te: double (nullable = true)
 |-- gender_clean_te: double (nullable = true)
 |-- plays_behavior_vs_completion_collapsed_te: double (nullable = true)
 |-- plays_per_unq_behavior_te: double (nullable = true)
 |-- registered_via_group_te: double (nullable = true)
 |-- revenue_per_hour_tier_te: double (nullable = true)
 |-- total_plays_group_te: double (nullable = true)
 |-- usage_intensity_tier_te: double (nullable = true)
 |-- safra: integer (nullable = true)
```

13.6.2. Juntando bases predições

```
In [19]: # Elastic Net
en = (df_predictions_elastic_net
      .filter(F.col("partition").isin(["test","oot"]))
      .select("msno","safra","partition","target_win",
              F.col("prediction").alias("pred_en")))

# LightGBM
lgbm = (df_predictions_lightgbm
         .filter(F.col("partition").isin(["test","oot"]))
         .select("msno","safra","partition","target_win",
                 F.col("prediction_lgbm").alias("pred_lgbm")))

# Random Forest
rf = (df_predictions_random_forest
      .filter(F.col("partition").isin(["test","oot"]))
      .select("msno","safra","partition","target_win",
              F.col("prediction_rfr").alias("pred_rf")))

# Interseção exata
df_cmp = (en
           .join(lgbm, on=["msno","safra","partition","target_win"], how="inner")
           .join(rf,   on=["msno","safra","partition","target_win"], how="inner")
           .cache()
          )

print("\n" + "="*90)
print("✅ DATASET COMPARÁVEL (interseção dos 3 modelos)")
print("="*90)
print("n total comparável =", df_cmp.count())
df_cmp.groupBy("partition").count().show()
```

```
=====
✅ DATASET COMPARÁVEL (interseção dos 3 modelos)
=====
n total comparável = 3416035
+-----+-----+
|partition|  count|
+-----+-----+
|      oot|1850732|
|     test|1565303|
+-----+-----+
```

13.6.3. Métricas (RMSE/MAE/R²) por modelo e partição

```
In [20]: def regression_metrics(df, y_col, yhat_col, label):
    # calcula SSE e SST por partição (para R2)
    base = (df
        .select("partition", F.col(y_col).alias("y"), F.col(yhat_col).alias("yhat"))
        .withColumn("err", F.col("y") - F.col("yhat"))
        .withColumn("ae", F.abs("err"))
        .withColumn("se", F.col("err")*F.col("err")))
    )

    # stats básicos
    stats = (base.groupBy("partition")
        .agg(
            F.count("*").alias("n"),
            F.sqrt(F.mean("se")).alias("rmse"),
            F.mean("ae").alias("mae"),
            F.mean("y").alias("y_mean"),
            F.mean("yhat").alias("yhat_mean"),
            F.sum("se").alias("sse")
        )
    )

    # sst
    ybar = base.groupBy("partition").agg(F.mean("y").alias("ybar"))
    sst = (base.join(ybar, on="partition")
        .withColumn("sst_term", (F.col("y") - F.col("ybar"))*(F.col("y") - F.col("ybar")))
        .groupBy("partition")
        .agg(F.sum("sst_term").alias("sst"))
    )

    out = (stats.join(sst, on="partition")
        .withColumn("r2", F.lit(1.0) - (F.col("sse")/F.col("sst")))
        .withColumn("Modelo", F.lit(label))
        .select("Modelo", "partition", "n", "rmse", "mae", "r2", "y_mean", "yhat_mean")
    )
    return out

m_en   = regression_metrics(df_cmp, "target_win", "pred_en",   "Elastic Net")
m_lgbm = regression_metrics(df_cmp, "target_win", "pred_lgbm", "LightGBM (Tuned)")
m_rf   = regression_metrics(df_cmp, "target_win", "pred_rf",   "Random Forest (Tuned)")

df_metrics = (m_en.unionByName(m_lgbm).unionByName(m_rf)
    .orderBy("partition", "Modelo"))

print("\n" + "="*90)
print("📊 MÉTRICAS POR PARTIÇÃO (test e oot) – dataset comparável")
print("=*90")
df_metrics.show(truncate=False)
```

```
=====
📊 MÉTRICAS POR PARTIÇÃO (test e oot) – dataset comparável
=====
+-----+-----+-----+-----+-----+
|Modelo      |partition|n      |rmse       |mae        |r2          |y_mean
|yhat_mean   |          |
+-----+-----+-----+-----+-----+
+-----+
|Elastic Net |oot      |1850732|32.26876092364628
|16.618794242397154|0.6328577852910099|51.52032381700692|51.84356100163868 |
|LightGBM (Tuned)|oot
|1850732|30.314178560769314|13.976792947903277|0.6759878071265872|51.52032381700692|53.221726439582774|
|Random Forest (Tuned)|oot
|1850732|30.664446308242887|15.775008724073242|0.6684568960357952|51.52032381700692|52.23245925917683 |
|Elastic Net     |test     |1565303|37.15942743267742|20.03902701539178|0.6233106674198143|43.7389132004085
|43.708222333246965|
|LightGBM (Tuned)|test     |1565303|35.90815556164754|18.020233209700674|0.6482521185407087|43.7389132004085
|45.0829227356214 |
|Random Forest (Tuned)|test     |1565303|36.13287227683309|19.914681533599442|0.6438357981479723|43.7389132004085
|44.54554415365792 |
+-----+-----+-----+-----+-----+
=====
```

```
In [22]: # Tabela OOT estilo "bonita"
oot_table = (df_metrics
    .filter(F.col("partition")=="oot")
    .select(
        "Modelo",
        F.round("rmse", 6).alias("RMSE OOT"),
        F.round("mae", 6).alias("MAE OOT"),
        F.round("r2", 6).alias("R² OOT"),
        "n"
    )
    .orderBy(F.col("RMSE OOT").asc())
)

print("\n" + "*60)
print("📊 COMPARAÇÃO DE TODOS OS MODELOS (OOT) – comparável")
print("*60)
oot_table.show(truncate=False)
```

=====				
📊 COMPARAÇÃO DE TODOS OS MODELOS (OOT) – comparável				
Modelo	RMSE OOT	MAE OOT	R² OOT	n
LightGBM (Tuned)	30.314179	13.976793	0.675988	1850732
Random Forest (Tuned)	30.664446	15.775009	0.668457	1850732
Elastic Net	32.268761	16.618794	0.632858	1850732

Desempenho Global (OOT)

- Menor RMSE: LightGBM, Random Forest, Elastic Net;
- Menor MAE: LightGBM, Random Forest, Elastic Net;
- Melhor R^2 : LightGBM, Random Forest, Elastic Net;

Vencedor: LightGBM.

13.6.4. Estabilidade temporal: Delta Test --> OOT

```
In [ ]: test_tbl = df_metrics.filter("partition='test'").select(
    "Modelo",
    F.col("rmse").alias("rmse_test"),
    F.col("mae").alias("mae_test"),
    F.col("r2").alias("r2_test")
)

oot_tbl = df_metrics.filter("partition='oot'").select(
    "Modelo",
    F.col("rmse").alias("rmse_oot"),
    F.col("mae").alias("mae_oot"),
    F.col("r2").alias("r2_oot")
)

stability = (test_tbl.join(oot_tbl, on="Modelo")
    .withColumn("rmse_ratio_oot_test", F.col("rmse_oot")/F.col("rmse_test"))
    .withColumn("mae_ratio_oot_test", F.col("mae_oot")/F.col("mae_test"))
    .withColumn("r2_delta_oot_test", F.col("r2_oot") - F.col("r2_test"))
    .select(
        "Modelo",
        F.round("rmse_test",6).alias("RMSE test"),
        F.round("rmse_oot",6).alias("RMSE oot"),
        F.round("rmse_ratio_oot_test",4).alias("RMSE (oot/test)"),
        F.round("mae_test",6).alias("MAE test"),
        F.round("mae_oot",6).alias("MAE oot"),
        F.round("mae_ratio_oot_test",4).alias("MAE (oot/test)"),
        F.round("r2_test",6).alias("R2 test"),
        F.round("r2_oot",6).alias("R2 oot"),
        F.round("r2_delta_oot_test",6).alias("ΔR2 (oot-test)")
    )
    .orderBy(F.col("RMSE (oot/test)").asc())
)

print("\n" + "="*90)
print("⌚ ESTABILIDADE (test -> oot): ratios e delta de R2")
print("=*90")
stability.show(truncate=False)
```

```
=====
⌚ ESTABILIDADE (test -> oot): ratios e delta de R2
=====
+-----+-----+-----+-----+-----+-----+-----+
--+
|Modelo      |RMSE test|RMSE oot |RMSE (oot/test)|MAE test |MAE oot  |MAE (oot/test)|R2 test |R2 oot   |ΔR2 (oot-
test)|
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
|LightGBM (Tuned) |35.908156|30.314179|0.8442       |18.020233|13.976793|0.7756       |0.648252|0.675988|0.027736
|
|Ran...d...m...o...r...f...o...r...t... (Tuned)|36.132872|30.664446|0.8487       |19.914682|15.775009|0.7921       |0.643836|0.668457|0.024621
|
|Elastic Net     |37.159427|32.268761|0.8684       |20.039027|16.618794|0.8293       |0.623311|0.632858|0.009547
|
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
```

- Melhor Ratio RMSE (menor queda): LightGBM, Random Forest, Elastic Net;
- Melhor Ratio MAE: LightGBM, Random Forest, Elastic Net;
- Maior ganho de \$R^2\$ (\$\Delta R^2\$): LightGBM, Random Forest, Elastic Net;

Vencedor: LightGBM.

13.6.5. Soma: target vs. prediction

```
In [24]: # =====#
# 3) FECHAMENTO AGREGADO: soma(y) vs soma(ŷ) (test e oot)
# =====#

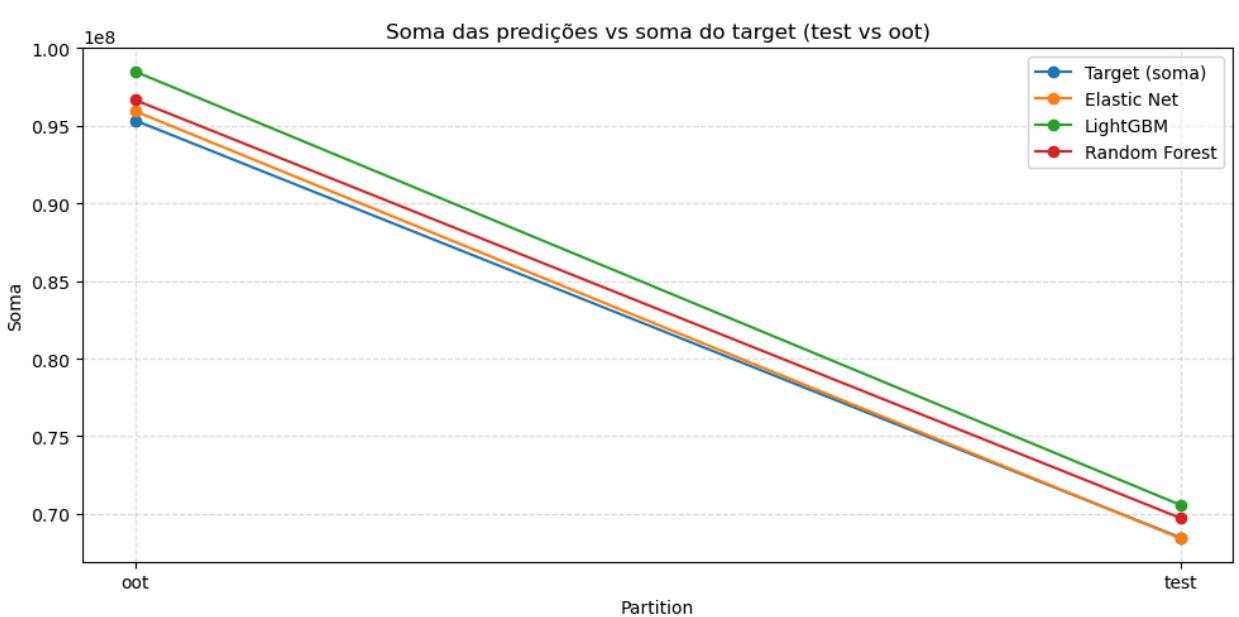
agg = (df_cmp.groupBy("partition")
    .agg(
        F.count("*").alias("n"),
        F.sum("target_win").alias("sum_y"),
        F.sum("pred_en").alias("sum_pred_en"),
        F.sum("pred_lgbm").alias("sum_pred_lgbm"),
        F.sum("pred_rf").alias("sum_pred_rf")
    )
    .withColumn("rel_err_en", (F.col("sum_pred_en") - F.col("sum_y")) / F.col("sum_y"))
    .withColumn("rel_err_lgbm", (F.col("sum_pred_lgbm") - F.col("sum_y")) / F.col("sum_y"))
    .withColumn("rel_err_rf", (F.col("sum_pred_rf") - F.col("sum_y")) / F.col("sum_y"))
    .orderBy("partition")
)

print("\n" + "*90")
print(" FECHAMENTO AGREGADO: soma predições vs soma target (e erro relativo)")
print("*90")
agg.select(
    "partition", "n",
    F.round("sum_y", 2).alias("sum_y"),
    F.round("sum_pred_en", 2).alias("sum_pred_en"),
    F.round("sum_pred_lgbm", 2).alias("sum_pred_lgbm"),
    F.round("sum_pred_rf", 2).alias("sum_pred_rf"),
    F.round("rel_err_en", 4).alias("rel_err_en"),
    F.round("rel_err_lgbm", 4).alias("rel_err_lgbm"),
    F.round("rel_err_rf", 4).alias("rel_err_rf"),
).show(truncate=False)

# Plot das somas (2 linhas, leve)
pdf_agg = agg.toPandas()
plt.figure(figsize=(10, 5))
plt.plot(pdf_agg["partition"], pdf_agg["sum_y"], marker="o", label="Target (soma)")
plt.plot(pdf_agg["partition"], pdf_agg["sum_pred_en"], marker="o", label="Elastic Net")
plt.plot(pdf_agg["partition"], pdf_agg["sum_pred_lgbm"], marker="o", label="LightGBM")
plt.plot(pdf_agg["partition"], pdf_agg["sum_pred_rf"], marker="o", label="Random Forest")
plt.title("Soma das predições vs soma do target (test vs oot)")
plt.xlabel("Partition"); plt.ylabel("Soma")
plt.grid(True, linestyle="--", alpha=0.4)
plt.legend()
plt.tight_layout()
plt.show()
```

```
=====
 FECHAMENTO AGREGADO: soma predições vs soma target (e erro relativo)
=====

+-----+-----+-----+-----+-----+-----+-----+
|partition|n |sum_y      |sum_pred_en |sum_pred_lgbm|sum_pred_rf | |rel_err_en|rel_err_lgbm|rel_err_rf|
+-----+-----+-----+-----+-----+-----+-----+
|oot     |1850732|9.535031194E7|9.594853734E7|9.849915222E7|9.666828379E7|0.0063   |0.033    |0.0138   |
|test    |1565303|6.846465205E7|6.841661154E7|7.056843421E7|6.97272739E7|-7.0E-4  |0.0307   |0.0184   |
+-----+-----+-----+-----+-----+-----+-----+
```



- Menor Erro Relativo (OOT): Elastic Net, Random Forest, LightGBM;
- Menor Erro Relativo (Test): Elastic Net, Random Forest, LightGBM;

Vencedor: Elastic Net.

13.6.6. Calibracao por decil

```
In [35]: df_oot = df_cmp.filter(F.col("partition") == "oot")

def decile_calibration(df, pred_col, label):
    # Cria a janela ordenando pela coluna de predição passada como parâmetro
    w = Window.orderBy(F.col(pred_col)) # <-- aqui usa pred_col

    tmp = (df.withColumn("rn", F.row_number().over(w)))

    n = tmp.count()
    tmp = tmp.withColumn("decile", F.floor((F.col("rn")-1) / (F.lit(n)/F.lit(10))) + 1)
    tmp = tmp.withColumn("decile", F.when(F.col("decile")>10, 10).otherwise(F.col("decile")))

    out = (tmp
        .withColumn("err", F.col("target_win") - F.col(pred_col)) # <-- aqui também
        .withColumn("ae", F.abs("err"))
        .groupBy("decile")
        .agg(
            F.count("*").alias("n"),
            F.sum("target_win").alias("sum_y"),
            F.sum(pred_col).alias("sum_pred"), # <-- e aqui
            F.mean("ae").alias("mae"),
            F.mean("target_win").alias("mean_y"),
            F.mean(pred_col).alias("mean_pred") # <-- e aqui
        )
        .withColumn("Modelo", F.lit(label))
        .orderBy("decile")
    )
    return out

cal_en = decile_calibration(df_oot, "pred_en", "Elastic Net")
cal_lgbm = decile_calibration(df_oot, "pred_lgbm", "LightGBM (Tuned)")
cal_rf = decile_calibration(df_oot, "pred_rf", "Random Forest (Tuned)")

df_cal = cal_en.unionByName(cal_lgbm).unionByName(cal_rf).cache()

list_cal = [cal_en, cal_lgbm, cal_rf]

print("\n" + "="*90)
print("CALIBRAÇÃO POR DECIL (OOT): soma(y) vs soma(ŷ) e MAE por decil")
print("=*90")
for df in list_cal:
    df.show(truncate=False)
```

```
=====
CALIBRAÇÃO POR DECIL (OOT): soma(y) vs soma(ŷ) e MAE por decil
=====
+-----+-----+-----+-----+-----+
|decile|n      |sum_y          |sum_pred       |mae           |mean_y        |mean_pred     |Modelo
+-----+-----+-----+-----+-----+
|1     |185074| -9761604.579586446 | -9188399.322666442 | 27.838144807401545 | -52.74433242695596 | -49.64716449996457 |Elastic
Net|
|2     |185073| 4094673.1806230242 | 4725561.764463908 | 18.820043601490433 | 22.124638281235104 | 25.53350172344917 |Elastic
Net|
|3     |185073| 7036904.979092318 | 7545851.085786953 | 12.99490413154297 | 38.022320809044636 | 40.77229571999672 |Elastic
Net|
|4     |185073| 8246268.4950592695 | 8449272.012420775 | 8.6782720250006 | 44.5568424084511 | 45.6537258942189 |Elastic
Net|
|5     |185073| 8754886.427995892 | 9423164.495432092 | 16.035367774610865 | 47.305044106897775 | 50.915933147634135 |Elastic
Net|
|6     |185074| 1.1634549488910038E7 | 1.175751239180553E7 | 26.709740375880887 | 62.86431097242205 | 63.52870955296546 |Elastic
Net|
|7     |185073| 1.4287137730887232E7 | 1.3997100661264764E7 | 13.502925574953757 | 77.1973098771146 | 75.63016032195277 |Elastic
Net|
|8     |185073| 1.596758383758687E7 | 1.5371814842604175E7 | 11.53591398765082 | 86.2772194625195 | 83.05811675719406 |Elastic
Net|
|9     |185073| 1.6956165560250074E7 | 1.619111532614104E7 | 12.100489046313267 | 91.61879669238665 | 87.4850211869967 |Elastic
Net|
|10    |185073| 1.813374681767806E7 | 1.7675544082429152E7 | 17.97202595375111 | 97.98159006272152 | 95.50579545600466 |Elastic
Net|
+-----+-----+-----+-----+-----+
-----+
|decile|n      |sum_y          |sum_pred       |mae           |mean_y        |mean_pred     |Modelo
+-----+-----+-----+-----+-----+
-----+
```

- Melhor MAE nos decis centrais (3 e 4): LightGBM, Random Forest, Elastic Net;
 - Melhor aderência em valores negativos (Decil 1): LightGBM, Elastic Net, Random Forest;
 - Melhor consistência linear (Mean Y vs Mean \hat{Y}): LightGBM, Random Forest, Elastic Net;

Vencedor: LightGBM.

13.6.7. Top 5% erros

```
In [ ]: def tail_metrics(df, pred_col, label, q=0.95):
    tmp = (df
        .select("target_win", F.col(pred_col).alias("pred"))
        .withColumn("residual", F.col("target_win") - F.col("pred"))
        .withColumn("abs_residual", F.abs("residual")))
    )
    thr = tmp.approxQuantile("abs_residual", [q], 0.001)[0]
    worst = tmp.filter(F.col("abs_residual") >= F.lit(thr))

    out = (worst
        .withColumn("ae", F.abs("residual"))
        .withColumn("se", F.col("residual")*F.col("residual")))
        .groupBy()
        .agg(
            F.count("*").alias("n_worst"),
            F.sqrt(F.mean("se")).alias("rmse_worst"),
            F.mean("ae").alias("mae_worst"),
            F.mean("residual").alias("bias_worst"),
            F.lit(thr).alias("thr_abs_residual")
        )
        .withColumn("Modelo", F.lit(label))
        .select("Modelo","n_worst","thr_abs_residual","rmse_worst","mae_worst","bias_worst")
    )
    return out

tail_en   = tail_metrics(df_oot, "pred_en",   "Elastic Net", q=0.95)
tail_lgbm = tail_metrics(df_oot, "pred_lgbm", "LightGBM (Tuned)", q=0.95)
tail_rf   = tail_metrics(df_oot, "pred_rf",   "Random Forest (Tuned)", q=0.95)

df_tail = tail_en.unionByName(tail_lgbm).unionByName(tail_rf)

print("\n" + "="*90)
print("🔥 CAUDA DE ERRO (OOT): métricas no Top 5% abs_residual")
print("=*90")
df_tail.select(
    "Modelo",
    "n_worst",
    F.round("thr_abs_residual",4).alias("thr_abs_residual"),
    F.round("rmse_worst",6).alias("RMSE worst"),
    F.round("mae_worst",6).alias("MAE worst"),
    F.round("bias_worst",6).alias("Bias worst (mean residual)")
).orderBy(F.col("RMSE worst").asc()).show(truncate=False)
```

```
=====
🔥 CAUDA DE ERRO (OOT): métricas no Top 5% abs_residual
=====
+-----+-----+-----+-----+-----+
|Modelo      |n_worst|thr_abs_residual|RMSE worst|MAE worst |Bias worst (mean residual)|
+-----+-----+-----+-----+-----+
|Random Forest (Tuned)|94317 |53.2389     |122.121025|117.423842|-64.702073   |
|LightGBM (Tuned)  |94350 |51.6017     |124.044288|119.031077|-64.016355   |
|Elastic Net     |94355 |71.937      |126.490609|123.741347|-69.722445   |
+-----+-----+-----+-----+-----+
```

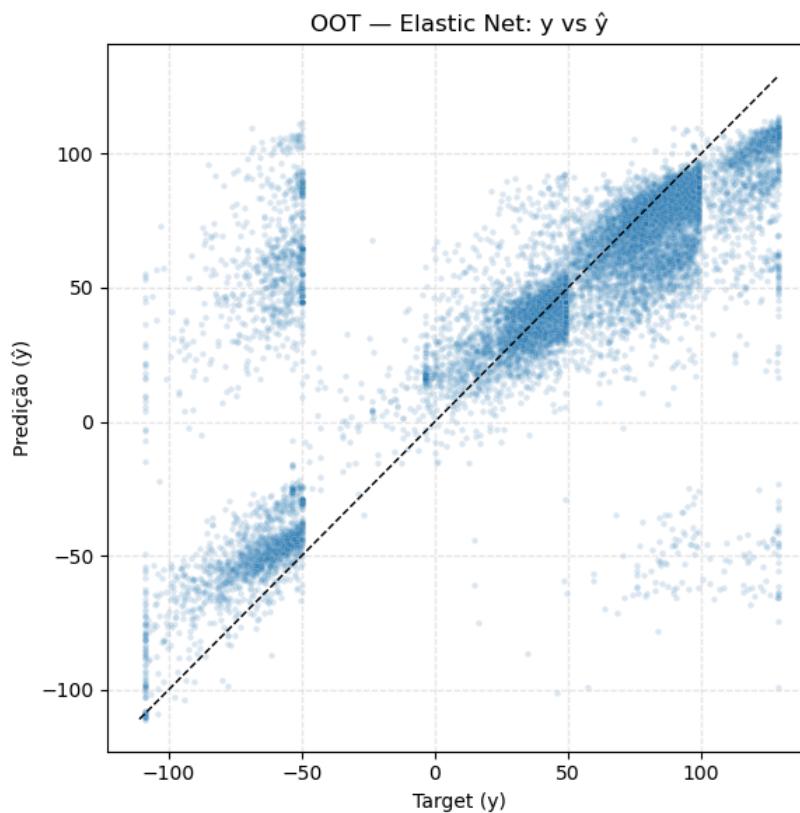
- Menor Threshold de Resíduo: LightGBM, Random Forest, Elastic Net;
- Melhor "Pior RMSE": Random Forest, LightGBM, Elastic Net;
- Melhor "Pior MAE": Random Forest, LightGBM, Elastic Net;
- Menor "Pior Viés": LightGBM, Random Forest, Elastic Net;

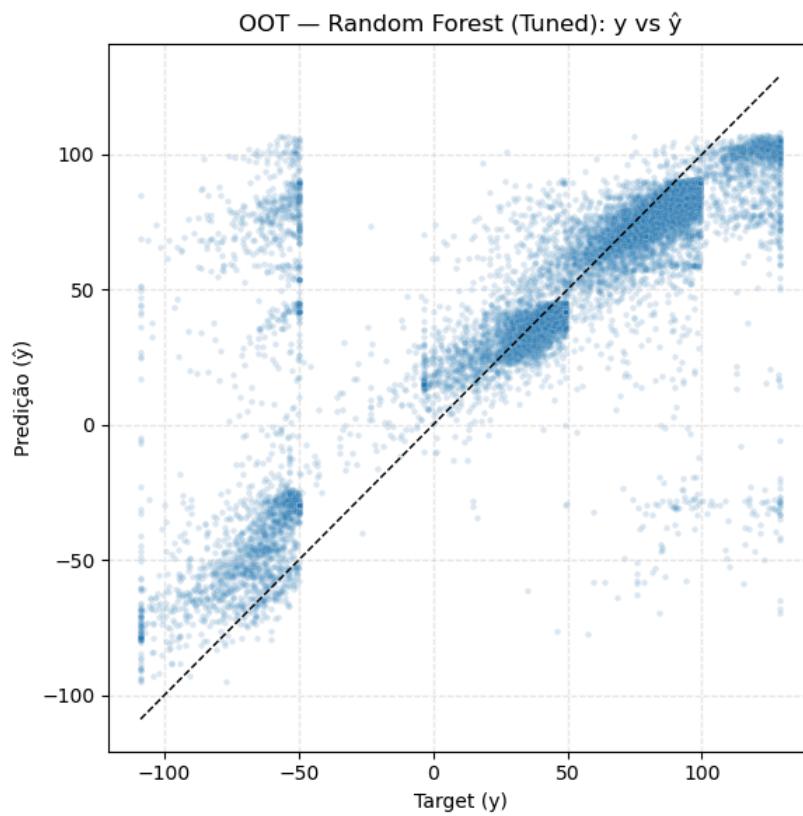
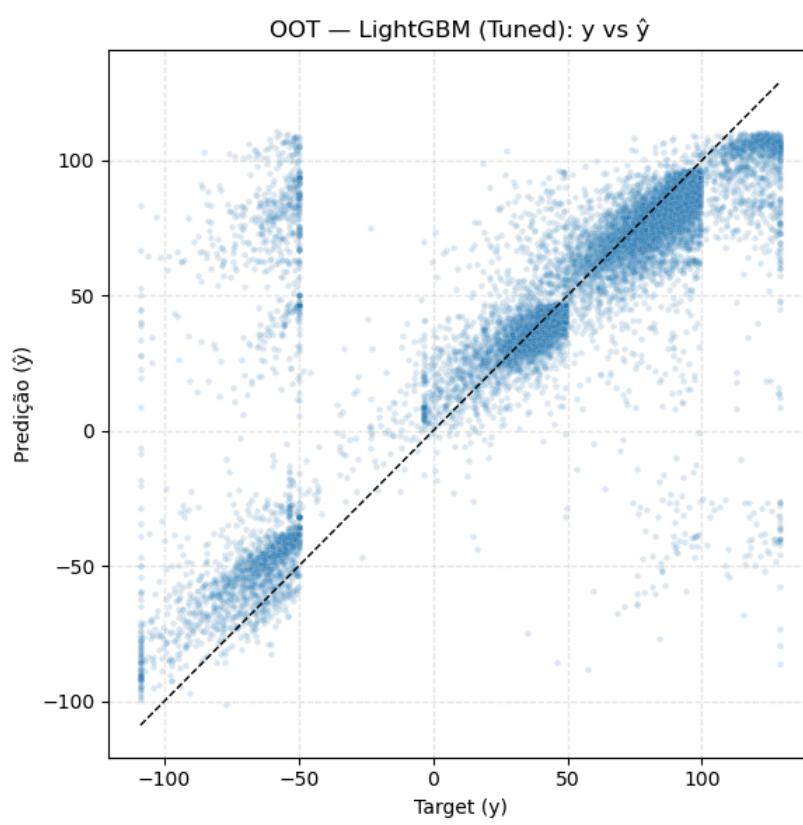
Vencedor: Empate entre Random Forest e LightGBM.

13.6.8. Scatter Plot: target vs. prediction

```
In [31]: # =====#
# 6) SCATTER: y vs yhat (OOT) - amostra igual pros 3 modelos
# =====#
# Amostragem única para os 3 (mesmas linhas)
pdf_sc = (df_oot
    .select("target_win","pred_en","pred_lgbm","pred_rf")
    .sample(False, 0.01, 42) # ajuste: 0.005 se ficar pesado
    .toPandas()
)
def scatter_y_vs_yhat(pdf, pred_col, title):
    plt.figure(figsize=(6,6))
    sns.scatterplot(data=pdf, x="target_win", y=pred_col, alpha=0.15, s=10)
    # linha y=x
    mn = min(pdf["target_win"].min(), pdf[pred_col].min())
    mx = max(pdf["target_win"].max(), pdf[pred_col].max())
    plt.plot([mn,mx],[mn,mx], linestyle="--", color="black", linewidth=1)
    plt.title(title)
    plt.xlabel("Target (y)")
    plt.ylabel("Predição ( $\hat{y}$ )")
    plt.grid(True, linestyle="--", alpha=0.3)
    plt.tight_layout()
    plt.show()

scatter_y_vs_yhat(pdf_sc, "pred_en", "OOT - Elastic Net: y vs  $\hat{y}$ ")
scatter_y_vs_yhat(pdf_sc, "pred_lgbm", "OOT - LightGBM (Tuned): y vs  $\hat{y}$ ")
scatter_y_vs_yhat(pdf_sc, "pred_rf", "OOT - Random Forest (Tuned): y vs  $\hat{y}$ )
```





Visualizações extremamente parecidas. Os modelos são bem similares entre si.

13.6.9. Escolha Final: LightGBM (Tuned)

Embora o Elastic Net apresente o melhor fechamento financeiro agregado (soma total), o LightGBM foi o vencedor absoluto na maioria das métricas avaliadas. Demonstrou ser o modelo mais estável temporalmente, com a melhor capacidade de distinguir perfis de clientes por decil e a maior precisão individual (MAE).

14. Agrupamento e Análise - Algoritmos Não Supervisionados

```
In [78]: gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"
```

14.1. Propostas de agrupamento

14.1.1. Proposta 1: Análise de Volatilidade e Incerteza

Premissa: Agrupar clientes não pelo que demonstram hoje, mas olhando como o comportamento destes oscila ao longo do tempo.

- * Valor de Negócio: Um cliente que gasta 50 NTD todo mês é diferente de um que gasta 100 NTD em um mês e 0 NTD no outro. O segundo perfil é o "Incerto", contrário do "Consistente". Se incerteza é risco, é importante identificar esses casos;
- * Granularidade do Dataframe: id do cliente (`msno`) agregado. Calcular o desvio padrão ou coeficiente de variação das features de uso, margem, e transações ao longo de 3/6/12 meses;
- * Ideia de análise: Criar clusters de "Consistentes", "Incertos" e "Declíneo".

14.1.2. Proposta 2: Segmentação de Valor e Engajamento

Premissa: Criar um "mapa de calor" dos clientes, baseando-me em quanto eles usam (engajamento) vs. quanto eles rendem (margem líquida no mês referência).

- * Valor de negócio previsto: Identificação dos clientes "VIPs" (alta margem, alto uso), "Eficientes" (alta margem, baixo uso - baixo custo operacional), "Drenos" (baixo lucro, mas usam muito a plataforma) e "Ausentes" (baixo lucro, baixo uso da plataforma);
- * Granularidade do Dataframe: id do cliente (`msno`) + `safra` ;
- * Manter target nula: se o cara tem target nula em M+1, ele é um **churn**. Agrupar esses caras permite ver se existe um "perfil pré-churn" (curiosamente, a premissa do outro modelo preditivo);
- * Ideia de análise: Comparar a proporção de churn (target nula) em cada cluster. Se o Cluster X tem 80% de target nula, descobrimos um "grupo de risco".

14.1.3. Proposta 3: Diagnóstico de Erros (Auditoria) do Modelo Finalista - LightGBM

Premissa: Pegar os n% de clientes onde o LightGBM mais errou (maiores resíduos) e clusterizar esses casos.

- * Valor de Negócio: Identificar e explicar para para `_stakeholders_` não só a força mas também a fraqueza do modelo: "O modelo executa boas previsões para o público X, exceto para o Grupo Y (ex: clientes muito jovens que acabaram de assinar), onde se mostra necessário uma abordagem diferente";
- * Granularidade do Dataframe: id do cliente (`msno`) + `safra` + `residual_lgbm` ;
- * Ideia de análise: Descobrir se o erro é aleatório ou se existe um "perfil de cliente" que a abordagem supervisionada escolhida (rentabilidade) é incapaz de captar.

14.2. Estudo de melhor algoritmo para a solução não supervisionada (agrupamento)

Depois de pesquisar e estudar sobre, separei e listei os algoritmos de agrupamento em diferentes famílias:

14.2.1. Família de Particionamento (Partition-Based)

```
#### 1.1 K-Means
```

O que faz: Divide os dados em k clusters minimizando a distância intra-cluster (soma dos quadrados das distâncias ao centróide).

Vantagens:

- Extremamente rápido e escalável
- Simples de interpretar (centróides são "perfis médios")
- Funciona bem em dados numéricos padronizados
- Implementação madura (sklearn, pyspark.ml)

Desvantagens:

- Assume clusters esféricos (distância Euclidiana)
- Sensível a outliers (um outlier puxa o centróide)
- Precisa definir k a priori
- Não lida bem com clusters de tamanhos/densidades muito diferentes

Melhor para:

- **A) Descoberta dos Perfis** (principal)
- **D) Jornada do Cliente** (rápido para rodar em múltiplas safras)

Quando NÃO usar: Se você tem muitos outliers ou clusters com formatos não-esféricos.

1.2 MiniBatch K-Means

O que faz: Versão do K-Means que usa mini-batches (amostras aleatórias) para atualizar os centróides, ao invés de processar todos os dados de uma vez.

Vantagens:

- **Muito mais rápido** que K-Means tradicional (10-100x)
- Escala para milhões de linhas
- Mesma interpretabilidade do K-Means

Desvantagens:

- Ligeiramente menos preciso que K-Means (mas a diferença é mínima)
- Mesmas limitações do K-Means (outliers, forma esférica)

Melhor para:

- **A) Descoberta dos Perfis** (quando você tem milhões de linhas)
- **D) Jornada do Cliente**

Quando usar: Sempre que você tiver mais de 500k linhas e quiser velocidade.

1.3 K-Medoids (PAM - Partitioning Around Medoids)

O que faz: Similar ao K-Means, mas usa **medoids** (pontos reais do dataset) ao invés de centróides (médias). Minimiza a soma das dissimilaridades.

Vantagens:

- **Robusto a outliers** (medoid é um ponto real, não uma média)
- Funciona com qualquer métrica de distância (não só Euclidiana)
- Mais interpretável que K-Means (o medoid é um cliente real)

Desvantagens:

- **Muito mais lento** que K-Means ($O(n^2)$ vs $O(n)$)
- Não escala bem para grandes datasets
- Implementações menos maduras

Melhor para:

- **A) Descoberta dos Perfis** (quando você tem outliers e quer robustez)

Quando usar: Datasets pequenos (<100k linhas) com outliers.

1.4 K-Prototypes

O que faz: Extensão do K-Means para dados **mistas** (numéricos + categóricos). Usa distância Euclidiana para numéricos e "matching dissimilarity" para categóricos.

Vantagens:

- Lida nativamente com categóricas (sem precisar de OHE)
- Mais eficiente que fazer OHE + K-Means

Desvantagens:

- Implementação menos madura (kmodes library)
- Tuning do parâmetro `gamma` (peso das categóricas) é chato
- Não tão rápido quanto MiniBatch K-Means

Melhor para:

- A) **Descoberta dos Perfis** (se você tiver muitas categóricas importantes)

Quando usar: Se você não quiser fazer OHE e tiver categóricas relevantes.

14.2.2. Família de Hierárquicos (Hierarchical)

2.1 Agglomerative Clustering (Hierárquico Aglomerativo)

O que faz: Começa com cada ponto sendo um cluster e vai **fundindo** os clusters mais próximos até formar uma árvore (dendrograma).

Vantagens:

- Não precisa definir `k` a priori (você corta o dendrograma onde quiser)
- Gera uma **hierarquia** de clusters (útil para storytelling)
- Funciona com várias métricas de linkage (ward, complete, average)

Desvantagens:

- **Muito lento** ($O(n^2 \log n)$ ou $O(n^3)$)
- Não escala para milhões de linhas
- Sensível a ruído e outliers

Melhor para:

- A) **Descoberta dos Perfis** (quando você quer explorar diferentes níveis de granularidade)
- **Visualização** (dendrograma é lindo para apresentações)

Quando usar: Datasets pequenos (<50k linhas) ou para validação/exploração inicial.

2.2 BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)

O que faz: Hierárquico otimizado para **grandes datasets**. Constrói uma árvore CF (Clustering Feature) que resume os dados.

Vantagens:

- **Muito mais rápido** que Agglomerative ($O(n)$)
- Escala para milhões de linhas
- Lida bem com ruído

Desvantagens:

- Assume clusters esféricos (como K-Means)
- Menos interpretável que Agglomerative
- Sensível ao parâmetro `threshold`

Melhor para:

- A) **Descoberta dos Perfis** (alternativa rápida ao Agglomerative)

Quando usar: Quando você quer hierárquico mas tem muitos dados.

14.2.3. Família de Densidade (Density-Based)

3.1 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

O que faz: Agrupa pontos que estão **densamente conectados** e marca pontos isolados como **ruido/outliers**.

Vantagens:

- **Detecta clusters de forma arbitrária** (não assume esferas)
- **Identifica outliers explicitamente** (label = -1)
- Não precisa definir k a priori

Desvantagens:

- Sensível aos parâmetros `eps` (raio) e `min_samples`
- Não funciona bem em **alta dimensionalidade** (curse of dimensionality)
- Clusters de densidades diferentes são problemáticos
- Não escala bem ($O(n^2)$ sem otimizações)

Melhor para:

- **C) Auditoria do Modelo** (detectar "bolsões" de erro sistemático)
- **Detecção de anomalias**

Quando usar: Quando você quer encontrar outliers ou clusters de forma irregular.

3.2 HDBSCAN (Hierarchical DBSCAN)

O que faz: Versão **hierárquica** do DBSCAN que constrói uma árvore de densidades e extrai clusters estáveis.

Vantagens:

- **Mais robusto** que DBSCAN (menos sensível a parâmetros)
- Lida com clusters de **densidades variadas**
- Identifica outliers
- Não precisa definir k ou `eps`

Desvantagens:

- Ainda sofre em alta dimensionalidade
- Mais lento que DBSCAN
- Implementação menos madura (hdbscan library)

Melhor para:

- **C) Auditoria do Modelo** (encontrar "níchos" de erro)
- **B) Filtro de Estabilidade** (detectar clientes "anômalos")

Quando usar: Quando você quer DBSCAN mas com menos tuning.

14.2.4.Família de Modelos de Mistura (Model-Based)

4.1 Gaussian Mixture Models (GMM)

O que faz: Assume que os dados vêm de uma **mistura de distribuições Gaussianas**. Cada cluster é uma Gaussiana (média + covariância).

Vantagens:

- **Clustering "soft"** (probabilidade de pertencimento a cada cluster)
- Captura clusters **elípticos** (mais flexível que K-Means)
- Permite calcular **incerteza** (entropia das probabilidades)
- Tem critérios estatísticos para escolher k (BIC, AIC)

Desvantagens:

- Mais lento que K-Means
- Sensível a inicialização
- Assume distribuições Gaussianas (pode não ser realista)
- Sofre em alta dimensionalidade

Melhor para:

- **B) Filtro de Estabilidade** (usar entropia como medida de incerteza)
- **A) Descoberta dos Perfilis** (quando você quer probabilidades)

Quando usar: Quando você quer medir "quão confiante" é a atribuição de cluster.

O que faz: Versão **Bayesiana** do GMM que automaticamente determina o número de componentes (clusters).

Vantagens:

- **Não precisa definir k** (ele "desliga" componentes desnecessários)
- Mesmas vantagens do GMM

Desvantagens:

- Ainda mais lento que GMM
- Mais complexo de interpretar

Melhor para:

- **Exploração** (quando você não sabe quantos clusters esperar)

Quando usar: Quando você quer GMM mas não quer escolher k .

14.2.5. Redução de Dimensionalidade e Clustering Unidos

5.1 PCA + K-Means

O que faz: Reduz a dimensionalidade com **PCA** (Principal Component Analysis) e depois aplica K-Means nos componentes principais.

Vantagens:

- **Reduz ruído** e correlação entre features
- **Acelera** o clustering (menos dimensões)
- Melhora a estabilidade do K-Means
- Permite visualização 2D/3D

Desvantagens:

- PCA é **linear** (pode perder estruturas não-lineares)
- Componentes principais são difíceis de interpretar

Melhor para:

- **A) Descoberta dos Perfis** (quando você tem muitas features correlacionadas)
- **D) Jornada do Cliente** (visualização da evolução)

Quando usar: Sempre que você tiver >20 features ou alta correlação.

14.2.6. Qual algoritmo para cada frente?

| Frente | Algoritmos Recomendados | Por quê? |

|-----|-----|-----|

| **A) Descoberta dos Perfis** | **PCA + K-Means** (principal) | Rápido, escalável, interpretável. PCA ajuda com correlação. |

| **B) Filtro de Estabilidade** | **GMM** (principal)

PCA + K-Means: baseline | GMM dá probabilidades (incerteza). K-Means permite identificar segmentação. |

| **C) Auditoria do Modelo** | **PCA + K-Means** (principal) | Encontrando o melhor K, o algoritmo permite uma para análise rápida e certeira. |

Algoritmo Principal: PCA + K-Means

- Atende todas menos volatilidade/incerteza
- Rápido, escalável, interpretável
- PCA remove correlação e ruído

Algoritmo Complementar: GMM

- Atende estudo de volatilidade/incerteza
- Dá probabilidades de pertencimento
- Permite calcular entropia como medida de incerteza

14.3. Feature _Engineering/_Selection_ - Construção das ABTs para algoritmos não supervisionados

```
In [3]: silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"  
gold_path = "C:/Users/Gustavo/Downloads/datamaster/dados/gold/"
```

14.3.1. Engineering

14.3.1.1. Features escolhidas

Objetivo do clustering:

Descobrir perfis comportamentais.

Então usamos variáveis que representem:

1 Engajamento

- total_secs
- num_unq
- total_plays
- log_total_secs
- completed_songs_rate
- early_drop_rate
- plays_per_unq_cap
- catalog_exploration_ratio_cap

2 Monetização

- margem_liquida_mensal
- actual_amount_paid
- daily_revenue_efficiency
- revenue_per_hour_listened_cap
- usage_intensity_per_tenure_cap

3 Tendência (comportamento recente)

- total_secs_ratio_ref_mean_3
- log_total_plays_mean_3
- total_secs_mean_3

4 Contratuais

- is_auto_renew
- flag_plano_mensal
- flag_has_transactions

5 Demografia / Perfil

- faixa_idade
- gender_clean
- tenure_faixa
- payment_method_group
- revenue_tier

14.3.1.2. Granularidade: msno + safra

Base da ABT

```
In [6]: print("\n" + "*80")
print("CONSTRUÇÃO DA ABT CLIENTE-MÊS")
print("*80)

silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"

# 1 Carregar base fonte
df_base = spark.read.parquet(silver_path + "df_feature_store_intermediate")

print("✓ Base carregada")
print("Linhas:", df_base.count())
print("Colunas:", len(df_base.columns))

# 2 Definir listas
features_num = [
    'total_secs',
    'num_unq',
    'total_plays',
    'log_total_secs',
    'completed_songs_rate',
    'early_drop_rate',
    'plays_per_unq_cap',
    'catalog_exploration_ratio_cap',
    'margem_liquida_mensal',
    'actual_amount_paid',
    'daily_revenue_efficiency',
    'revenue_per_hour_listened_cap',
    'usage_intensity_per_tenure_cap',
    'total_secs_ratio_ref_mean_3',
    'log_total_plays_mean_3',
    'total_secs_mean_3',
    'is_auto_renew',
    'flag_plano_mensal',
    'flag_has_transactions'
]

features_cat = [
    'faixa_idade',
    'gender_clean',
    'tenure_faixa',
    'payment_method_group',
    'revenue_tier'
]

control_cols = ['msno', 'safra']

# 3 Selecionar apenas colunas existentes
available = set(df_base.columns)

features_num = [c for c in features_num if c in available]
features_cat = [c for c in features_cat if c in available]

cols_final = control_cols + features_num + features_cat + ['target', 'target_win']

df_abt = df_base.select(*cols_final)

print("✓ ABT inicial criada")
print("Colunas finais:", len(df_abt.columns))
```

```
=====
CONSTRUÇÃO DA ABT CLIENTE-MÊS
=====

✓ Base carregada
Linhas: 11242865
Colunas: 321
✓ ABT inicial criada
Colunas finais: 28
```

Marcação de "censura". Última safra = censura (churn artificial).

```
In [7]: last_safra = df_abt.select(F.max("safra")).collect()[0][0]
df_abt = df_abt.withColumn("is_censored", F.when(F.col("safra") == F.lit(last_safra), 1).otherwise(0))
print("✓ Última safra:", last_safra)
df_abt.groupBy("is_censored").count().show()
```

```
✓ Última safra: 201612
+-----+-----+
|is_censored|  count|
+-----+-----+
|          0|10264066|
|          1|  978799|
+-----+-----+
```

Contexto. O churn foi operacionalizado como `target_win = NULL` (ausência de margem líquida M+1). Como o dataset possui uma **última safra observada** (ex.: 201611) e não contém o mês seguinte (M+1), ocorre **censura à direita (right censoring)**: todos os registros do último mês passam a ter churn "artificial", pois não é possível observar a renovação no período subsequente.

Diagnóstico. Ao calcular churn e taxas associadas usando todas as safras, a métrica fica viésada, pois incorpora o mês censurado (último mês) como churn em massa.

Correção implementada.

- Criou-se a flag `is_censored = 1` para a **última safra** (detectada dinamicamente via `max(safra)`).
- Métricas de comportamento (ex.: médias, desvios, CV, min, max) foram agregadas usando **todas as safras** disponíveis (janela *ALL*), pois descrevem perfil e volatilidade observada.
- Métricas dependentes de churn/target (`n_meses_churn` , `churn_rate` , `flag_churn_algum_mes`) passaram a ser calculadas em uma janela *NO_CENSOR*, isto é, **excluindo a safra censurada** (`is_censored = 0`).
- A ABT final do cliente agregado foi construída via `join` de (*ALL* + *NO_CENSOR*), garantindo que a variável `churn_rate_no_censor` represente churn observado de forma consistente e comparável ao longo do tempo.

Rastreabilidade.

- A safra censurada é registrada no log como `last_safra` .
- A presença de censura por cliente é auditável via `n_meses_censurados` e `n_meses_observados_no_censor` .

Impacto esperado.

- Remove viés estrutural nas métricas de churn no fim da janela histórica.
- Preserva métricas comportamentais e de volatilidade para segmentação, auditoria e filtros de estabilidade.

Tratamento inicial de sentinelas

```
In [8]: # 3. TRATAMENTO DE SENTINELAS (IGUAL AO LGBM)
print("\n" + "=" * 90)
print("🔧 TRATAMENTO DE SENTINELAS")
print("=" * 90)

for col in features_num:
    df_abt = df_abt.withColumn(col, F.when(F.col(col) < -10000, None).otherwise(F.col(col)))

print("✓ Tratamento de sentinelas aplicado nas features transformadas")
```

```
=====
🔧 TRATAMENTO DE SENTINELAS
=====
✓ Tratamento de sentinelas aplicado nas features transformadas
```

Checkpoint para analisar a divisão de registros e features

```
In [9]: df_abt.selectExpr("count(1) as n").show()
df_abt.printSchema()
df_abt.select("safra").groupBy("safra").count().orderBy("safra").show()
```

```
+-----+
|      n|
+-----+
|11242865|
+-----+  
  
root
|-- msno: string (nullable = true)
|-- safra: integer (nullable = true)
|-- total_secs: double (nullable = true)
|-- num_unq: double (nullable = true)
|-- total_plays: double (nullable = true)
|-- log_total_secs: double (nullable = true)
|-- completed_songs_rate: double (nullable = true)
|-- early_drop_rate: double (nullable = true)
|-- plays_per_unq_cap: double (nullable = true)
|-- catalog_exploration_ratio_cap: double (nullable = true)
|-- margem_liquida_mensal: double (nullable = true)
|-- actual_amount_paid: float (nullable = true)
|-- daily_revenue_efficiency: double (nullable = true)
|-- revenue_per_hour_listened_cap: double (nullable = true)
|-- usage_intensity_per_tenure_cap: double (nullable = true)
|-- total_secs_ratio_ref_mean_3: double (nullable = true)
|-- log_total_plays_mean_3: double (nullable = true)
|-- total_secs_mean_3: double (nullable = true)
|-- is_auto_renew: integer (nullable = true)
|-- flag_plano_mensal: integer (nullable = true)
|-- flag_has_transactions: integer (nullable = true)
|-- faixa_idade: string (nullable = true)
|-- gender_clean: string (nullable = true)
|-- tenure_faixa: string (nullable = true)
|-- payment_method_group: string (nullable = true)
|-- revenue_tier: string (nullable = true)
|-- target: double (nullable = true)
|-- target_win: double (nullable = true)
|-- is_censored: integer (nullable = false)  
  
+-----+-----+
| safra| count|
+-----+-----+
|201601| 888663|
|201602| 918116|
|201603| 871651|
|201604| 831565|
|201605| 849653|
|201606| 844884|
|201607| 967936|
|201608| 980643|
|201609| 991285|
|201610|1048557|
|201611|1071113|
|201612| 978799|
+-----+-----+
```

```
##### Target Encoding
```

```
In [14]: print("\n" + "*80")
print("TARGET ENCODING (K-FOLD + SMOOTHING)")
print("*80")

# Criar target binário (0 = churn, 1 = ativo)
df_clustering_raw = df_abt.withColumn("target_te", F.when(F.col("target_win").isNull(), 0.0).otherwise(1.0))

# Adicionar fold_id
w = Window.orderBy(F.rand(seed=42))
n_folds = 5

df_clustering_raw = (df_clustering_raw
    .withColumn("row_id_tmp", F.row_number().over(w))
    .withColumn("fold_id", (F.col("row_id_tmp") % n_folds).cast("int"))
    .drop("row_id_tmp"))

print(f"✓ Folds criados: {n_folds}")

# Função de TE
def run_te_clustering(df, cat_col, alpha=10):
    """Target Encoding com K-Fold para clustering"""

    global_mean = df.select(F.mean("target_te")).collect()[0][0]

    stats_fold = (
        df
        .groupBy("fold_id", cat_col)
        .agg(
            F.count("*").alias("cnt"),
            F.sum("target_te").alias("sum_target")
        )
    )

    stats_te = (
        stats_fold
        .groupBy(cat_col)
        .agg(
            F.sum("cnt").alias("total_cnt"),
            F.sum("sum_target").alias("total_sum")
        )
        .join(stats_fold, cat_col, "left")
        .withColumn("cnt_others", F.col("total_cnt") - F.col("cnt"))
        .withColumn("sum_others", F.col("total_sum") - F.col("sum_target"))
        .withColumn(
            "mean_others",
            F.when(F.col("cnt_others") > 0,
                   F.col("sum_others") / F.col("cnt_others"))
            .otherwise(global_mean)
        )
        .withColumn(
            f"{cat_col}_te",
            (F.col("cnt_others") * F.col("mean_others") +
             alpha * global_mean) /
            (F.col("cnt_others") + alpha)
        )
        .select("fold_id", cat_col, f"{cat_col}_te")
    )

    result_df = df.join(stats_te, on=["fold_id", cat_col], how="left")

    return result_df

# Processar TE com checkpoints
df_te_master = df_clustering_raw

for idx, cat_col in enumerate(features_cat, start=1):
    print(f"🔧 [{idx}/{len(features_cat)}] Processando TE: {cat_col}")
    df_te_master = run_te_clustering(df_te_master, cat_col, alpha=10)

    # Checkpoint a cada 3 features (mesmo que sendo somente 5 categoricas, funcionou em cima, prefiro manter como deu certo)
    if idx % 3 == 0:
        temp_path = silver_path + f"clustering_te_checkpoint_{idx}_v3"
        df_te_master.write.mode("overwrite").parquet(temp_path)
        df_te_master = spark.read.parquet(temp_path)
        print(f"💾 Checkpoint {idx} salvo e recarregado")

print("✓ Target Encoding concluído")
```

```
=====
TARGET ENCODING (K-FOLD + SMOOTHING)
=====

✓ Folds criados: 5
🔧 [1/5] Processando TE: faixa_idade
🔧 [2/5] Processando TE: gender_clean
🔧 [3/5] Processando TE: tenure_faixa
💾 Checkpoint 3 salvo e recarregado
```

```
    ↗ [4/5] Processando TE: payment_method_group  
    ↗ [5/5] Processando TE: revenue_tier  
✓ Target Encoding concluído
```

Consolidar e salvar ABT

```
In [15]: print("\n" + "*80")
print("CONSOLIDAÇÃO FINAL")
print("*80)

# Colunas TE
te_cols = [f"{c}_te" for c in features_cat]

# Colunas finais para clustering
features_final = features_num + te_cols

# Selecionar colunas finais
cols_abt_final = control_cols + features_final + ['target_win', 'is_censored']

df_abt_final = df_te_master.select(*cols_abt_final)

# Estatísticas
n_total = df_abt_final.count()
n_censored = df_abt_final.filter(F.col("is_censored") == 1).count()
n_churn = df_abt_final.filter((F.col("target_win").isNull()) & (F.col("is_censored") == 0)).count()
n_active = df_abt_final.filter((F.col("target_win").isNotNull()) & (F.col("is_censored") == 0)).count()

print(f"\n📊 Estatísticas da ABT:")
print(f"  Total de registros: {n_total:,}")
print(f"  Censurados (safra {last_safra}): {n_censored:,} ({100*n_censored/n_total:.2f}%)")
print(f"  Churn real: {n_churn:,} ({100*n_churn/(n_total-n_censored):.2f}% das safras válidas)")
print(f"  Ativos: {n_active:,} ({100*n_active/(n_total-n_censored):.2f}% das safras válidas)")
print(f"  Features finais: {len(features_final)}")
```

```
=====
CONSOLIDAÇÃO FINAL
=====
```

```
📊 Estatísticas da ABT:
Total de registros: 11,242,865
Censurados (safra 201612): 978,799 (8.71%)
Churn real: 585,200 (5.70% das safras válidas)
Ativos: 9,678,866 (94.30% das safras válidas)
Features finais: 24
```

```
In [16]: # Salvar ABT final
df_abt_final.write.mode("overwrite").parquet(silver_path + "df_abt_clustering_cliente_mes")

print(f"\n💾 ABT salva em: {silver_path + 'df_abt_clustering_cliente_mes'}")

# Teste de sanidade
df_test = spark.read.parquet(silver_path + "df_abt_clustering_cliente_mes")
df_test.select("msno", "safra", "is_censored").limit(5).show()

print("\n" + "*80)
print("✅ ABT CLUSTERING (cliente/mes) CRIADA COM SUCESSO")
print("*80)
```

```
💾 ABT salva em: C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_abt_clustering_cliente_mes
+-----+-----+
| msno | safra | is_censored |
+-----+-----+
| HxCZ6/TGgLIRfGHNR... | 201605 | 0 |
| IXY1LHeahfL9U2iZc... | 201612 | 1 |
| 6rWF1UM1AygJU8Bt1... | 201602 | 0 |
| X7eHiAxRR8fGtmMqJ... | 201608 | 0 |
| N/ZpdG9Tw81R5YHXz... | 201611 | 0 |
+-----+-----+
```

```
=====
✅ ABT CLUSTERING (cliente/mes) CRIADA COM SUCESSO
=====
```

14.3.1.3. Granularidade: msno (agregado)

Base da ABT

```
In [19]: print("\n" + "=" * 90)
print("🔧 CONSTRUÇÃO DA ABT – CLIENTE AGREGADO (msno)")
print("=" * 90)

df_cliente_mes = spark.read.parquet(silver_path + "df_abt_clustering_cliente_mes")

print(f"✓ Linhas totais: {df_cliente_mes.count():,}")

last_safra = df_cliente_mes.select(F.max("safra")).collect()[0][0]

df_cliente_mes = df_cliente_mes.withColumn("is_censored", F.when(F.col("safra") == last_safra, 1).otherwise(0).cast("byte"))

print(f"🔴 Última safra (censurada): {last_safra}")

df_cliente_mes_no_censor = df_cliente_mes.filter(F.col("is_censored") == 0)

print(f"✓ Linhas válidas (sem censura): {df_cliente_mes_no_censor.count():,}")
```

```
=====
🔧 CONSTRUÇÃO DA ABT – CLIENTE AGREGADO (msno)
=====
✓ Linhas totais: 11,242,865
🔴 Última safra (censurada): 201612
✓ Linhas válidas (sem censura): 10,264,066
```

Features à agregar

```
In [21]: exclude_cols = [
    "msno", "safra", "target_win", "is_censored",
    "fold_id", "partition", "features", "features_raw"
]

candidate_cols = [c for c in df_cliente_mes.columns if c not in exclude_cols]

te_cols = [c for c in candidate_cols if c.endswith("_te")]
flag_cols = [c for c in candidate_cols if c.startswith("flag_") or c.startswith("is_")]
num_cols = [c for c in candidate_cols if c not in te_cols and c not in flag_cols]

print(f"✓ Numéricas: {len(num_cols)}")
print(f"✓ Target Encoded: {len(te_cols)}")
print(f"✓ Flags: {len(flag_cols)}")
```

```
✓ Numéricas: 16
✓ Target Encoded: 5
✓ Flags: 3
```

Agregados, com censura

```
In [ ]: agg_exprs = []

# Numéricas
for c in num_cols:
    agg_exprs.extend([
        F.mean(c).alias(f"{c}_mean"),
        F.coalesce(F.stddev(c), F.lit(0.0)).alias(f"{c}_std"),
        F.min(c).alias(f"{c}_min"),
        F.max(c).alias(f"{c}_max")
    ])

# TE
for c in te_cols:
    agg_exprs.append(F.mean(c).alias(f"{c}_mean"))

# Flags
for c in flag_cols:
    agg_exprs.extend([
        F.mean(c).alias(f"{c}_rate"),
        F.sum(F.col(c).cast("double")).alias(f"{c}_sum")
    ])

# Exposição temporal
agg_exprs.extend([
    F.count("*").alias("n_meses_observados_all"),
    F.sum("is_censored").alias("n_meses_censurados")
])

df_all = df_cliente_mes.groupBy("msno").agg(*agg_exprs)

print(f"✓ Clientes agregados (ALL): {df_all.count():,}")
```

```
✓ Clientes agregados (ALL): 1,563,999
```

```
##### Coeficiente de variação
```

```
In [23]: for c in num_cols:  
    df_all = df_all.withColumn(f"{c}_cv",  
        F.when(F.abs(F.col(f"{c}_mean")) > 1e-6, F.col(f"{c}_std") / F.abs(F.col(f"{c}_mean")))  
        .otherwise(F.lit(0.0)))
```

```
##### Agregados, sem censura
```

```
In [24]: df_churn = df_cliente_mes_no_censor.groupBy("msno").agg(  
    F.count("*").alias("n_meses_observados_no_censor"),  
    F.sum(F.when(F.col("target_win").isNull(), 1).otherwise(0)).alias("n_meses_churn_no_censor"),  
    F.max(F.when(F.col("target_win").isNull(), 1).otherwise(0)).alias("flag_churn_algum_mes_no_censor"))  
)  
  
df_churn = df_churn.withColumn("churn_rate_no_censor",  
    F.when(F.col("n_meses_observados_no_censor") > 0, F.col("n_meses_churn_no_censor") / F.col("n_meses_observados_no_censor"))  
    .otherwise(F.lit(0.0)))  
  
print(f"✓ Clientes com churn calculado: {df_churn.count():,}")
```

```
✓ Clientes com churn calculado: 1,530,217
```

```
##### Salvar base
```

```
In [26]: df_cliente_agregado = (df_all  
    .join(df_churn, on="msno", how="left")  
    .fillna({  
        "n_meses_observados_no_censor": 0,  
        "n_meses_churn_no_censor": 0,  
        "flag_churn_algum_mes_no_censor": 0,  
        "churn_rate_no_censor": 0.0}))  
  
print("\n" + "=" * 90)  
print("✓ ABT CLIENTE AGREGADO FINALIZADA")  
print("=" * 90)  
  
df_cliente_agregado.write.mode("overwrite").parquet(silver_path + "df_abt_clustering_cliente_agregado")  
  
print(f"\n💾 ABT Cliente Agregado salva em: {silver_path + 'df_abt_clustering_cliente_agregado'}")  
print(f"✓ Clientes únicos: {df_cliente_agregado.count():,}")  
print(f"✓ Total de colunas: {len(df_cliente_agregado.columns)}")
```

```
=====  
✓ ABT CLIENTE AGREGADO FINALIZADA  
=====  
💾 ABT Cliente Agregado salva em: C:/Users/Gustavo/Downloads/datamaster/dados/silver/df_abt_clustering_cliente_agregado  
✓ Clientes únicos: 1,563,999  
✓ Total de colunas: 98
```

14.3.2. Selection

```
#### 14.3.2.1. PCA (Redução de Dimensionalidade)
```

```
##### Conceito
```

O que é o PCA (Principal Component Analysis)?

Estatisticamente, o PCA é uma técnica de redução de dimensionalidade linear que transforma um conjunto de variáveis correlacionadas em um novo conjunto de variáveis não correlacionadas, chamadas de **Componentes Principais**.

1. A Intuição: Maximização da Variância

Imagine uma nuvem de pontos em 3D. O PCA busca uma nova "reta" (eixo) que atravesse essa nuvem na direção onde os pontos estão mais espalhados.

- O Primeiro Componente Principal (PC1) é a direção que captura a maior variância possível dos dados originais.
- O Segundo Componente Principal (PC2) é ortogonal (faz 90°) ao primeiro e captura a maior parte da variância restante.
- E assim por diante.

2. Por que usar neste caso?

Como foram geradas muitas métricas agregadas (`mean`, `std`, `min`, `max`, `cv`) sobre as mesmas variáveis originais, a multicolinearidade será altíssima.

- O PCA "limpa" essa redundância.
- Ele condensa, por exemplo, 10 variáveis de "intensidade de uso" em apenas 1 ou 2 componentes que explicam 95% do comportamento.

3. O "Preço" do PCA: Interpretabilidade

A maior desvantagem é que você perde o nome da feature. Em vez de saber que o cliente tem "alta média de plays", você saberá que ele tem um valor alto no "Componente 1". Para clustering, isso geralmente é aceitável, pois o objetivo é agrupar perfis similares, não necessariamente explicar cada variável isoladamente no início.

Execução - msno (agregado)

```
In [86]: df_cliente_agregado = spark.read.parquet(silver_path + "df_abt_clustering_cliente_agregado")
```

```
In [87]: df_cliente_agregado.count()
```

```
Out[87]: 1563999
```

```
In [81]: from pyspark.ml.feature import PCA, VarianceThresholdSelector
print("\n" + "="*90)
print("🚀 INICIANDO SELEÇÃO DE FEATURES E PCA")
print("="*90)

# 1. DEFINIR COLUNAS PARA O MODELO (Removendo IDs e Targets)
exclude_from_pca = {"msno", "n_meses_observados_all", "n_meses_censurados",
                     "n_meses_observados_no_censor", "n_meses_churn_no_censor",
                     "flag_churn_algum_mes_no_censor", "churn_rate_no_censor"}

features_to_pca = [c for c in df_cliente_agregado.columns if c not in exclude_from_pca]

print(f"✓ Features candidatas ao PCA: {len(features_to_pca)})")

# 2. VECTOR ASSEMBLER
assembler = VectorAssembler(inputCols=features_to_pca, outputCol="features_raw", handleInvalid="keep")
df_vector = assembler.transform(df_cliente_agregado)

# 3. FILTRO DE VARIÂNCIA (Remover colunas com variância zero ou quase zero)
# Threshold de 0.01 remove colunas onde 99% dos valores são iguais
selector = VarianceThresholdSelector(varianceThreshold=0.01, featuresCol="features_raw", outputCol="features_varied")
df_varied = selector.fit(df_vector).transform(df_vector)

# 4. PADRONIZAÇÃO (Z-Score) - CRUCIAL PARA PCA
scaler = StandardScaler(inputCol="features_varied", outputCol="features_scaled", withStd=True, withMean=True)
scaler_model = scaler.fit(df_varied)
df_scaled = scaler_model.transform(df_varied)

# 5. PCA
# Pedir a priori 15 componentes inicialmente para avaliar a variância explicada
k_pca = 15
pca = PCA(k=k_pca, inputCol="features_scaled", outputCol="pca_features")
pca_model = pca.fit(df_scaled)
df_pca = pca_model.transform(df_scaled)

# 6. ANALISAR VARIÂNCIA EXPLICADA
explained_var = pca_model.explainedVariance.toArray()
cumulative_var = explained_var.cumsum()

print("\n📊 RESULTADOS DO PCA:")
for i, var in enumerate(explained_var):
    print(f"  PC{i+1}: {var:.2%} (Acumulado: {cumulative_var[i]:.2%})")

# Selecionar colunas finais para o Clustering
# Mantemos o msno, as métricas de churn (para perfilamento) e os componentes do PCA
df_final_clustering_msno = df_pca.select("msno", "pca_features", "churn_rate_no_censor", "flag_churn_algum_mes_no_censor")

print("\n" + "="*90)
print("✅ PREPARAÇÃO PARA CLUSTERING CONCLUÍDA")
print("="*90)
```

```
=====
🚀 INICIANDO SELEÇÃO DE FEATURES E PCA
=====
✓ Features candidatas ao PCA: 91

📊 RESULTADOS DO PCA:
PC1: 25.55% (Acumulado: 25.55%)
PC2: 14.42% (Acumulado: 39.97%)
PC3: 8.47% (Acumulado: 48.44%)
PC4: 7.59% (Acumulado: 56.03%)
PC5: 5.53% (Acumulado: 61.56%)
PC6: 5.22% (Acumulado: 66.78%)
PC7: 4.54% (Acumulado: 71.32%)
PC8: 3.98% (Acumulado: 75.30%)
PC9: 3.30% (Acumulado: 78.60%)
PC10: 2.52% (Acumulado: 81.12%)
PC11: 2.19% (Acumulado: 83.32%)
PC12: 1.89% (Acumulado: 85.20%)
PC13: 1.84% (Acumulado: 87.04%)
PC14: 1.42% (Acumulado: 88.46%)
PC15: 1.39% (Acumulado: 89.85%)

=====
✅ PREPARAÇÃO PARA CLUSTERING CONCLUÍDA
=====
```

```
In [85]: df_final_clustering_msno.count()
```

```
Out[85]: 1563999
```

Usar 15 valores de PCA fica demais. Vou manter no 10, que já é mais do que 80% do comportamento explicado.

In [82]:

```
# 10 são o suficiente
k_pca = 10
pca = PCA(k=k_pca, inputCol="features_scaled", outputCol="pca_features")
pca_model = pca.fit(df_scaled)
df_pca = pca_model.transform(df_scaled)

# 6. ANALISAR VARIÂNCIA EXPLICADA
explained_var = pca_model.explainedVariance.toArray()
cumulative_var = explained_var.cumsum()

print("\n📊 RESULTADOS DO PCA:")
for i, var in enumerate(explained_var):
    print(f"  PC{i+1}: {var:.2%} (Acumulado: {cumulative_var[i]:.2%})")

# Selecionar colunas finais para o Clustering
# Mantemos o msno, as métricas de churn (para perfilamento) e os componentes do PCA
df_final_clustering_msno = df_pca.select("msno", "pca_features", "churn_rate_no_censor", "flag_churn_algum_mes_no_censor")

print("\n" + "="*90)
print("✅ PREPARAÇÃO PARA CLUSTERING CONCLUÍDA")
print("=*90")
```

```
📊 RESULTADOS DO PCA:
PC1: 25.55% (Acumulado: 25.55%)
PC2: 14.42% (Acumulado: 39.97%)
PC3: 8.47% (Acumulado: 48.44%)
PC4: 7.59% (Acumulado: 56.03%)
PC5: 5.53% (Acumulado: 61.56%)
PC6: 5.22% (Acumulado: 66.78%)
PC7: 4.54% (Acumulado: 71.32%)
PC8: 3.98% (Acumulado: 75.30%)
PC9: 3.30% (Acumulado: 78.60%)
PC10: 2.52% (Acumulado: 81.12%)
```

```
=====
✅ PREPARAÇÃO PARA CLUSTERING CONCLUÍDA
=====
```

Salvando a base de clientes agregados + pca_features

In [83]:

```
print("Salvando ABT final para clustering (cliente agregado + PCA)...")
df_final_clustering_msno.write.mode("overwrite").parquet(gold_path + "df_master_clustering_cliente_agregado_pca")
print("✅ ABT final salva em: " + gold_path + "df_master_clustering_cliente_agregado_pca")
```

```
Salvando ABT final para clustering (cliente agregado + PCA)...
✅ ABT final salva em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_master_clustering_cliente_agregado_pca
```

Execução - msno + safra

In [88]:

```

from pyspark.ml.feature import VectorAssembler, StandardScaler, PCA
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.sql import functions as F

print("\n" + "*90")
print("SEGMENTAÇÃO VALOR + ENGAJAMENTO (MSNO + SAFRA) - PCA + K-MEANS")
print("*90")

# 1. CARREGAR ABT E FILTRAR CENSURA
silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"
df_abt = spark.read.parquet(silver_path + "df_abt_clustering_cliente_mes")

# Criar flag de churn (target_win nulo = churn) e remover censurados
df_prep = (df_abt
    .filter(F.col("is_censored") == 0)
    .withColumn("is_churn_m1", F.when(F.col("target_win").isNull(), 1).otherwise(0))
)

# 2. DEFINIR FEATURES (24 colunas: 19 numéricas + 5 TE)
features_num = [
    'total_secs', 'num_unq', 'total_plays', 'log_total_secs',
    'completed_songs_rate', 'early_drop_rate', 'plays_per_unq_cap',
    'catalog_exploration_ratio_cap', 'margem_liquida_mensal',
    'actual_amount_paid', 'daily_revenue_efficiency',
    'revenue_per_hour_listened_cap', 'usage_intensity_per_tenure_cap',
    'total_secs_ratio_ref_mean_3', 'log_total_plays_mean_3',
    'total_secs_mean_3', 'is_auto_renew', 'flag_plano_mensal',
    'flag_has_transactions'
]

features_cat_te = [f"{c}_te" for c in ['faixa_idade', 'gender_clean', 'tenure_faixa', 'payment_method_group', 'revenue_tier']]
features_all = features_num + features_cat_te

# 3. PIPELINE: ASSEMBLER -> SCALER -> PCA
assembler = VectorAssembler(inputCols=features_all, outputCol="features_raw", handleInvalid="skip")
df_vec = assembler.transform(df_prep)

scaler = StandardScaler(inputCol="features_raw", outputCol="features_scaled", withStd=True, withMean=True)
scaler_model = scaler.fit(df_vec)
df_scaled = scaler_model.transform(df_vec)

print(f"Registros na ABT inicial: {df_abt.count():,}")
print(f"Registros no dataset final: {df_scaled.count():,}")
print(f"Perda de registros (%): {100 * (df_abt.count() - df_scaled.count()) / df_abt.count():.2f}%")


=====
SEGMENTAÇÃO VALOR + ENGAJAMENTO (MSNO + SAFRA) - PCA + K-MEANS
=====
Registros na ABT inicial: 11,242,865
Registros no dataset final: 6,220,460
Perda de registros (%): 44.67%

```

Perdeu-se aproximadamente 5 milhões de registros por valores nulos depois de remover censurados (vimos anteriormente que tem menor impacto) e nulos (naturalmente, maior impacto).

```
In [89]: from pyspark.ml.feature import Imputer # para tratar os nulos

print("\n" + "="*90)
print("✖ RECONSTRUÇÃO ZERO LOSS – PCA + K-MEANS (11.2M REGISTROS)")
print("="*90)

# 1. CARREGAR E FILTRAR APENAS CENSURA (Mantendo todos os outros)
silver_path = "C:/Users/Gustavo/Downloads/datamaster/dados/silver/"
df_abt = spark.read.parquet(silver_path + "df_abt_clustering_cliente_mes")

# Removemos apenas a safra 201612 (censura) para não enviesar o churn
df_prep = (df_abt
    .filter(F.col("is_censored") == 0)
    .withColumn("is_churn_m1", F.when(F.col("target_win").isNull(), 1).otherwise(0))
)

# 2. DEFINIR LISTAS DE FEATURES
features_num = [
    'total_secs', 'num_unq', 'total_plays', 'log_total_secs',
    'completed_songs_rate', 'early_drop_rate', 'plays_per_unq_cap',
    'catalog_exploration_ratio_cap', 'margem_liquida_mensal',
    'actual_amount_paid', 'daily_revenue_efficiency',
    'revenue_per_hour_listened_cap', 'usage_intensity_per_tenure_cap',
    'total_secs_ratio_ref_mean_3', 'log_total_plays_mean_3',
    'total_secs_mean_3', 'is_auto_renew', 'flag_plano_mensal',
    'flag_has_transactions'
]

features_cat_te = [f"{c}_te" for c in ['faixa_idade', 'gender_clean', 'tenure_faixa', 'payment_method_group', 'revenue_tier']]

features_all = features_num + features_cat_te

# 3. IMPUTAÇÃO AUTOMÁTICA (Garante que não haverá NULLs para o Assembler)
# Numéricas: Mediana | Target Encoding: Média (valor neutro do TE)
imputer_num = Imputer(inputCols=features_num, outputCols=features_num, strategy="median")
imputer_te = Imputer(inputCols=features_cat_te, outputCols=features_cat_te, strategy="mean")

df_imputed = imputer_num.fit(df_prep).transform(df_prep)
df_imputed = imputer_te.fit(df_imputed).transform(df_imputed)

# 4. PIPELINE SEM PERDA (handleInvalid="keep")
assembler = VectorAssembler(inputCols=features_all, outputCol="features_raw", handleInvalid="keep")
df_vec = assembler.transform(df_imputed)

scaler = StandardScaler(inputCol="features_raw", outputCol="features_scaled", withStd=True, withMean=True)
scaler_model = scaler.fit(df_vec)
df_scaled = scaler_model.transform(df_vec)

print(f"Registros na ABT inicial: {df_abt.count():,}")
print(f"Registros no dataset final: {df_scaled.count():,}")
print(f"Perda de registros (%): {100 * (df_abt.count() - df_scaled.count()) / df_abt.count():.2f}%")
```

```
=====
✖ RECONSTRUÇÃO ZERO LOSS – PCA + K-MEANS (11.2M REGISTROS)
=====
Registros na ABT inicial: 11,242,865
Registros no dataset final: 10,264,066
Perda de registros (%): 8.71%
```

Agora os registros removidos estão corretos.

```
In [63]: # PCA: Pedindo 15 componentes para avaliar a variância
k_pca = 15
pca = PCA(k=k_pca, inputCol="features_scaled", outputCol="pca_features")
pca_model = pca.fit(df_scaled)
df_pca = pca_model.transform(df_scaled)

# Analisar Variância Explícada
explained_var = pca_model.explainedVariance.toArray()
cumulative_var = explained_var.cumsum()

print("\n📊 VARIÂNCIA EXPLICADA (PCA):")
for i, var in enumerate(explained_var):
    print(f"  PC{i+1}: {var:.2%} (Acumulado: {cumulative_var[i]:.2%})")
```

📊 VARIÂNCIA EXPLICADA (PCA):
 PC1: 24.00% (Acumulado: 24.00%)
 PC2: 12.04% (Acumulado: 36.04%)
 PC3: 10.21% (Acumulado: 46.25%)
 PC4: 9.43% (Acumulado: 55.68%)
 PC5: 7.42% (Acumulado: 63.10%)
 PC6: 5.72% (Acumulado: 68.82%)
 PC7: 5.41% (Acumulado: 74.23%)
 PC8: 4.81% (Acumulado: 79.04%)
 PC9: 4.09% (Acumulado: 83.13%)
 PC10: 3.46% (Acumulado: 86.59%)
 PC11: 3.32% (Acumulado: 89.91%)
 PC12: 2.28% (Acumulado: 92.19%)
 PC13: 1.87% (Acumulado: 94.05%)
 PC14: 1.39% (Acumulado: 95.45%)
 PC15: 1.32% (Acumulado: 96.77%)

Decisão: levar o PC10, com ~86% da variação explicada

```
In [90]: # PCA: Decisão final - levar 10
k_pca = 10
pca = PCA(k=k_pca, inputCol="features_scaled", outputCol="pca_features")
pca_model = pca.fit(df_scaled)
df_pca = pca_model.transform(df_scaled)

# Analisar Variância Explícada
explained_var = pca_model.explainedVariance.toArray()
cumulative_var = explained_var.cumsum()

print("\n📊 VARIÂNCIA EXPLICADA (PCA):")
for i, var in enumerate(explained_var):
    print(f"  PC{i+1}: {var:.2%} (Acumulado: {cumulative_var[i]:.2%})")
```

📊 VARIÂNCIA EXPLICADA (PCA):
 PC1: 24.00% (Acumulado: 24.00%)
 PC2: 12.04% (Acumulado: 36.04%)
 PC3: 10.21% (Acumulado: 46.25%)
 PC4: 9.43% (Acumulado: 55.68%)
 PC5: 7.42% (Acumulado: 63.10%)
 PC6: 5.72% (Acumulado: 68.82%)
 PC7: 5.41% (Acumulado: 74.23%)
 PC8: 4.81% (Acumulado: 79.04%)
 PC9: 4.09% (Acumulado: 83.13%)
 PC10: 3.46% (Acumulado: 86.59%)

```
In [103]: # Verificando novamente
df_pca.count()
```

Out[103]: 10264066

```
In [92]: print("Salvando ABT final para clustering (cliente e safra + PCA)...")
df_pca.write.mode("overwrite").parquet(gold_path + "df_master_clustering_cliente_mes_pca")
print("✅ ABT final salva em: " + gold_path + "df_master_clustering_cliente_mes_pca")
```

Salvando ABT final para clustering (cliente e safra + PCA)...
 ✅ ABT final salva em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_master_clustering_cliente_mes_pca

14.4. Solução Proposta 1: Análise de Volatilidade e Incerteza - Clusterização com GMM vs. K-Means

A premissa é identificar três principais grupos: usuários constantes (dentro do possível), usuários incertos (futuro praticamente imprevisível) e usuários que vão sendo perdidos ao longo do tempo (declínio). Sendo assim, a ideia principal de separação se dará para 3 grupos, mas na execução do K-Means pretendo verificar o quanto mudaria usar um valor de K sugerido por técnicas que determinam o melhor valor.

14.4.1. K-Means

14.4.1.1. Fórmula (K-Means):

Dado o conjunto de pontos $x_i \in \mathbb{R}^d$, definimos os centróides como μ_k e a atribuição de cada ponto como $c_i \in \{1, \dots, K\}$. A função de custo (inércia) é dada por:

$$J(K) = \sum_{i=1}^n \|x_i - \mu_{c_i}\|^2$$

Como interpretar:

- **Quanto menor, melhor** (clusters mais "compactos").
- Mas **sempre cai quando K aumenta** (porque mais centróides = mais flexibilidade). Por isso a gente usa **Elbow**: buscar o ponto onde a melhora passa a ser "marginal".

14.4.1.2. Métricas (K-Means):

- **cost** (Spark: `trainingCost`): é o valor da função objetivo do K-Means (soma dos erros quadráticos intra-cluster). Também aparece como **WSSSE** (Within-Set Sum of Squared Errors). Depende diretamente da escala, do número de pontos (amostra vs full) e da dimensionalidade. Então comparar `cost` entre K s faz sentido **na mesma amostra**, mas comparar `cost` entre amostras diferentes não faz.

- **Silhouette** (K-Means ou qualquer clustering "hard"): mede o quão bem cada ponto está no cluster atual comparado ao cluster "vizinho mais próximo". É uma métrica geométrica de separação vs compactação.

Para cada ponto i :

- $a(i)$: distância média de i aos pontos do **mesmo** cluster (coesão).
- $b(i)$: menor distância média de i aos pontos de **outro** cluster (separação).

Fórmula:

$$s(i) := \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Intervalo:

- Perto de **1**: muito bem alocado (bem dentro do cluster e longe dos outros)
- Perto de **0**: região de sobreposição (fronteira)
- **Negativo**: provavelmente "mal alocado"

Silhouette global = média de $s(i)$ nos pontos.

`n_clientes / count` : tamanho do cluster.

Fórmula:

$$n_k = \sum_{i=1}^n \mathbb{1}[c_i = k]$$

Por que importa: cluster "ótimo" que vira um micro-grupo muito pequeno pode ser pouco acionável (depende do objetivo).

churn_rate_medio (**média de churn_rate_no_censor**): variável contínua entre 0 e 1 por cliente.

Fórmula:

$$\overline{r}_k = \frac{1}{n_k} \sum_{i: c_i=k} r_i$$

onde $r_i = \text{churn_rate_no_censor}$ do cliente i .

Interpretação (sem assumir mais do que foi definido no pipeline): intensidade média de churn (ou "taxa") dentro do cluster. Diferente do "churn aconteceu alguma vez".

pct_churn_algum_mes (**média de flag binária**): proporção de clientes do cluster que churnaram pelo menos uma vez.

Fórmula:

$$\text{pct_churn_algum_mes} = \frac{1}{n_k} \sum_{i: c_i=k} y_i$$

onde $y_i \in \{0, 1\}$ é `flag_churn_algum_mes_no_censor`.

Interpretação: "incidência" (aconteceu ao menos uma vez). Complementa o `churn_rate`: dá pra ter muita gente com churn "em algum mês" (incidência alta), mas com baixa taxa média (eventos raros/episódicos).

14.4.1.3. Busca de K (Elbow + Silhouette)

Definição

Escolha do Número de Clusters (\$K\$): para determinar o número ideal de agrupamentos, aplicamos duas métricas complementares sobre uma amostra dos dados:

1. Método do Cotovelo (Elbow Method): utiliza a métrica de **Custo de Treinamento** (ou *Within-Set Sum of Squared Errors - WSSSE*). O objetivo é encontrar o ponto onde o aumento de \$K\$ para de gerar ganhos significativos na redução da variância interna.

$$J(K) = \sum_{k=1}^K \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

* **O que buscar:** O "ponto de inflexão" na curva onde a queda do custo se torna linear ou menos acentuada.

2. Coeficiente de Silhueta: mede o quanto próximo um objeto está de seu próprio cluster em comparação com outros clusters. O valor varia de -1 a 1:

- * **Próximo a 1:** O ponto está bem alocado no cluster correto.
- * **Próximo a 0:** O ponto está na fronteira entre dois clusters.
- * **Próximo a -1:** O ponto provavelmente foi atribuído ao cluster errado.

Para um ponto x_i , a silhueta $s(i)$ é definida por:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Onde $a(i)$ é a distância média intra-cluster e $b(i)$ é a distância média para o cluster mais próximo.

Execução

```

In [104]: from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

print("\n" + "="*90)
print("💡 K-MEANS – BUSCA DO K (ELBOW + SILHOUETTE) [AMOSTRA]")
print("="*90)

# Amostra para acelerar (ajuste 0.05~0.2 dependendo do seu ambiente)
df_final_clustering = df_final_clustering_msno
sample_frac = 0.10
df_km_sample = df_final_clustering.sample(withReplacement=False, fraction=sample_frac, seed=42).cache()
_ = df_km_sample.count() # Materializar df_km_sample pós cache da base amostral

k_list = list(range(2, 11))
rows = []

evaluator = ClusteringEvaluator(featuresCol="pca_features", metricName="silhouette", distanceMeasure="squaredEuclidean")

for k in k_list:
    km = KMeans(featuresCol="pca_features", k=k, seed=42, maxIter=50)
    km_model = km.fit(df_km_sample)
    pred = km_model.transform(df_km_sample)

    cost = km_model.summary.trainingCost # WSSSE
    sil = evaluator.evaluate(pred)

    rows.append((k, float(cost), float(sil)))
    print(f"K={k} | cost={cost:.2f} | silhouette={sil:.4f}")

df_km_metrics = spark.createDataFrame(rows, ["k", "cost", "silhouette"])
df_km_metrics.orderBy("k").show(truncate=False)

# Heurística simples: maior silhouette (com k>=3) + cotovelo visual no cost
best_k_by_sil = (df_km_metrics
    .filter(F.col("k") >= 3)
    .orderBy(F.col("silhouette").desc(), F.col("k").asc())
    .first()["k"])

print(f"✅ Melhor K por silhouette (>=3): {best_k_by_sil}")

```

```

=====
💡 K-MEANS – BUSCA DO K (ELBOW + SILHOUETTE) [AMOSTRA]
=====

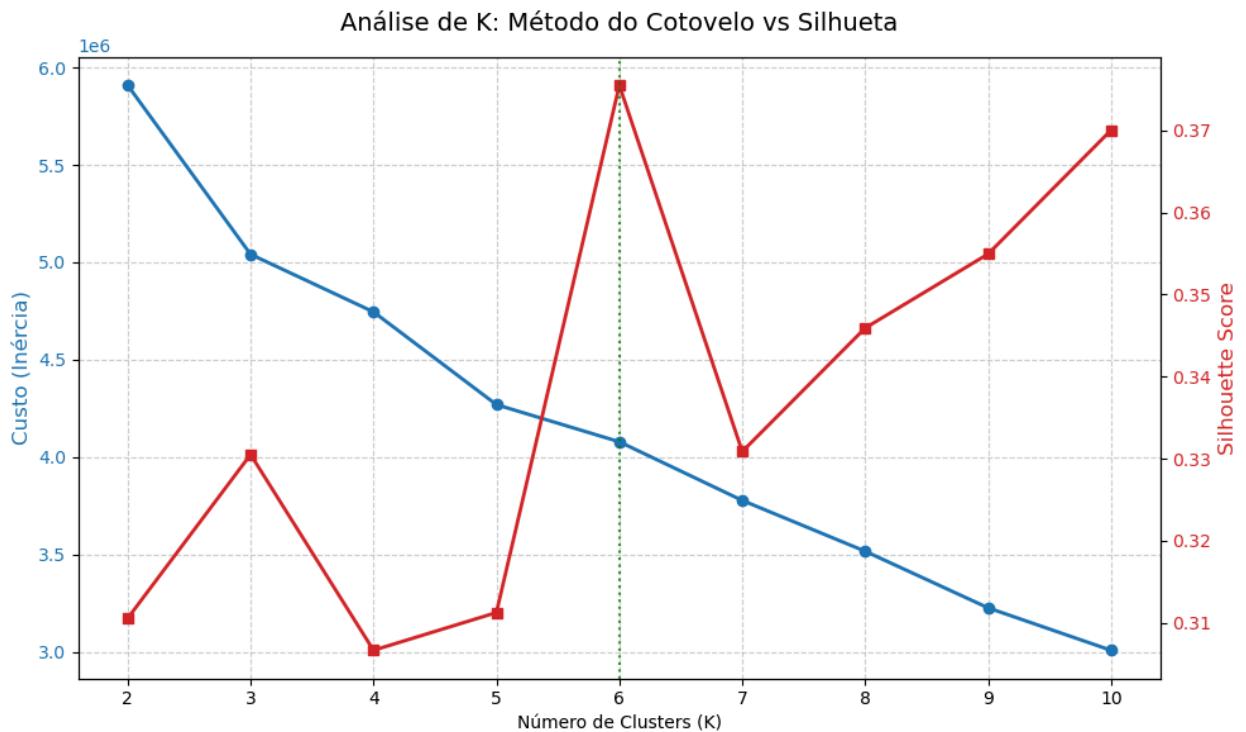
K=2 | cost=5908872.73 | silhouette=0.3106
K=3 | cost=5041024.45 | silhouette=0.3305
K=4 | cost=4745441.18 | silhouette=0.3067
K=5 | cost=4269333.32 | silhouette=0.3113
K=6 | cost=4078589.39 | silhouette=0.3755
K=7 | cost=3777614.93 | silhouette=0.3309
K=8 | cost=3516585.08 | silhouette=0.3459
K=9 | cost=3226056.04 | silhouette=0.3549
K=10 | cost=3008172.41 | silhouette=0.3700
+---+-----+-----+
|k |cost |silhouette |
+---+-----+-----+
|2 |5908872.730259587 |0.31056705558230296|
|3 |5041024.447603902 |0.3305212887129681|
|4 |4745441.1788798375 |0.30667330610476856|
|5 |4269333.32086593 |0.3112677743639848|
|6 |4078589.386593801 |0.37549812471099386|
|7 |3777614.929744531 |0.33088599013523884|
|8 |3516585.082180995 |0.34592867672361177|
|9 |3226056.0353642674 |0.35494823836223005|
|10 |3008172.406531895 |0.37002788010199494|
+---+-----+-----+

```

✅ Melhor K por silhouette (>=3): 6

```
In [106]: # Converter o DataFrame do Spark para Pandas para facilitar o plot  
df_plot = df_km_metrics.orderBy("k").toPandas()
```

```
fig, ax1 = plt.subplots(figsize=(10, 6))  
  
# Configuração do primeiro eixo (Custo / Elbow)  
color_elbow = 'tab:blue'  
ax1.set_xlabel('Número de Clusters (K)')  
ax1.set_ylabel('Custo (Inércia)', color=color_elbow, fontsize=12)  
ax1.plot(df_plot['k'], df_plot['cost'], marker='o', color=color_elbow, linewidth=2, label='Custo')  
ax1.tick_params(axis='y', labelcolor=color_elbow)  
ax1.grid(True, linestyle='--', alpha=0.6)  
  
# Criar o segundo eixo compartilhado  
ax2 = ax1.twinx()  
color_sil = 'tab:red'  
ax2.set_ylabel('Silhouette Score', color=color_sil, fontsize=12)  
ax2.plot(df_plot['k'], df_plot['silhouette'], marker='s', color=color_sil, linewidth=2, label='Silhueta')  
ax2.tick_params(axis='y', labelcolor=color_sil)  
  
# Título e ajustes finais  
plt.title('Análise de K: Método do Cotovelo vs Silhueta', fontsize=14, pad=15)  
plt.xticks(df_plot['k']) # Garante que todos os Ks apareçam no eixo X  
  
# Adicionar uma linha vertical no melhor K que você encontrou (K=6)  
plt.axvline(x=6, color='green', linestyle=':', alpha=0.8, label='Melhor K (6)')  
  
fig.tight_layout()  
plt.show()
```



Salto mais do que nítido para K = 6. K = 10 também poderia ser uma opção, pela silhueta alta. Se buscássemos algo mais granular, talvez poderia ser este o valor escolhido, mas K = 6 se mostra mais eficiente e simplificado.

```
#### 14.4.1.4. Resultados para K = 6 e K = 3
```

```
In [107]: k_kmeans_final = int(best_k_by_sil) # ou fixe 3 se quiser coerência de negócio
# k_kmeans_final = 3

print("\n" + "*90)
print(f"🚀 K-MEANS FINAL - K={k_kmeans_final}")
print("*90)

kmeans = KMeans(featuresCol="pca_features", k=k_kmeans_final, seed=42, maxIter=100)
kmeans_model = kmeans.fit(df_final_clustering)
df_kmeans = kmeans_model.transform(df_final_clustering)

df_kmeans.groupBy("prediction").agg(
    F.count("*").alias("n_clientes"),
    F.mean("churn_rate_no_censor").alias("churn_rate_medio"),
    F.mean("flag_churn_algum_mes_no_censor").alias("pct_churn_algum_mes")
).orderBy("prediction").show(truncate=False)
```

```
=====
🚀 K-MEANS FINAL - K=6
=====

+-----+-----+-----+
|prediction|n_clientes|churn_rate_medio |pct_churn_algum_mes |
+-----+-----+-----+
|0      |619360   |0.04868584035962459 |0.19213381555153708|
|1      |93884    |0.8091569349680423  |0.8587192705892378 |
|2      |127660   |0.08191967570786292 |0.397697007676641071|
|3      |169526   |0.054841031250549496 |0.219370480044358971|
|4      |238425   |0.04332743256650141 |0.172251232043619581|
|5      |315144   |0.6865030636811157  |0.8140786434138045 |
+-----+-----+-----+
```

K-Means com K=6

Clusters **1** e **5** são “claramente” de risco:

- Cluster 1: churn_rate_medio **0.809** e pct_churn_algum_mes **0.859**
- Cluster 5: churn_rate_medio **0.687** e pct_churn_algum_mes **0.814**

Os clusters **0/3/4** parecem “saudáveis/estáveis”:

- churn_rate ~ **0.043–0.055**
- pct churn algum mês ~ **0.172–0.219**

O cluster **2** é o que chama atenção:

- churn_rate_medio **0.082** (não tão alto)
- pct_churn_algum_mes **0.398** (bem alto)

Conclusão (K=6):

- Você não está apenas separando “churn vs não churn”.
- Você está separando pelo menos três perfis:
 - 1) alto churn intenso** (clusters 1 e 5),
 - 2) baixo churn** (0/3/4),
 - 3) churn episódico** (cluster 2: muita gente com pelo menos 1 evento, mas taxa média não explode).

Isso é muito alinhado com a ideia de **volatilidade**: “episódicos” tendem a ser mais instáveis do que “baixo churn consistente”.

```
In [109]: k_kmeans_3 = 3 # Regra de negocio, para comparação com GMM
print("\n" + "="*90)
print(f"📌 K-MEANS 3 - K={k_kmeans_3}")
print("="*90)

kmeans_3 = KMeans(featuresCol="pca_features", k=k_kmeans_3, seed=42, maxIter=100)
kmeans_model_3 = kmeans_3.fit(df_final_clustering)
df_kmeans_k3 = kmeans_model_3.transform(df_final_clustering)

df_kmeans_k3.groupBy("prediction").agg(
    F.count("*").alias("n_clientes"),
    F.mean("churn_rate_no_censor").alias("churn_rate_medio"),
    F.mean("flag_churn_algum_mes_no_censor").alias("pct_churn_algum_mes")
).orderBy("prediction").show(truncate=False)
```

```
=====
📌 K-MEANS 3 - K=3
=====
+-----+-----+-----+
|prediction|n_clientes|churn_rate_medio|pct_churn_algum_mes|
+-----+-----+-----+
|0      |390963   |0.7366543020743022|0.8283392546097712 |
|1      |884903   |0.0569927519093704|0.2330334511240215 |
|2      |288133   |0.048990673744346416|0.19136301638479453|
+-----+-----+-----+
```

K-Means com K=3

Obteve-se:

- Um cluster "alto risco": churn_rate 0.737, pct 0.828
- Dois clusters "baixo risco" com churn_rate 0.057 vs 0.049, e pct 0.233 vs 0.191

Conclusão (K=3): K=3 é mais simples, mas pode estar "colapsando" nuances: especialmente o padrão tipo "episódico" (que em K=6 aparece forte no cluster 2).

14.4.2. GMM - Gaussian Mixture Model

14.4.2.1. Definição

Diferente do K-Means, que realiza uma atribuição rígida (*hard clustering*), o **Gaussian Mixture Model (GMM)** é um modelo probabilístico que assume que todos os pontos de dados são gerados a partir de uma mistura de um número finito de distribuições Gaussianas (normais) com parâmetros desconhecidos.

Fundamentos Matemáticos

O GMM tenta modelar a densidade de probabilidade dos dados $p(x)$ como uma soma ponderada de K distribuições normais multivariadas:

$$p(x) = \sum_{k=1}^K \phi_k \mathcal{N}(x | \mu_k, \Sigma_k)$$

Onde:

- * ϕ_k : É o **peso de mistura** do componente k , sendo que $\sum \phi_k = 1$.
- * μ_k : Representa o **vetor de médias** (centro) do componente k .
- * Σ_k : É a **matriz de covariância**, que define a forma, volume e orientação do cluster.

Expectation-Maximization (EM)

O ajuste do modelo é feito através do algoritmo EM, que busca maximizar a log-verossimilhança:

1. **E-step (Expectation):** Calcula a probabilidade de cada ponto x_i pertencer a cada cluster k (atribuição suave ou *soft assignment*).
2. **M-step (Maximization):** Atualiza os parâmetros ϕ_k , μ_k e Σ_k para maximizar a verossimilhança dos dados dada as atribuições.

Essa abordagem permite que o GMM identifique clusters com diferentes tamanhos e formatos elípticos, o que o torna ideal para capturar a **volatilidade** no comportamento dos clientes.

14.4.2.2. Execução

```
In [110]: from pyspark.ml.clustering import GaussianMixture
from pyspark.ml.evaluation import ClusteringEvaluator

print("\n" + "*90)
print("📝 GMM - CLUSTERIZAÇÃO DE VOLATILIDADE")
print("*90)

gmm = GaussianMixture(
    featuresCol="pca_features",
    k=3, # Consistente, Incerto, Declínio - como declarado na sessão "propostas de agrupamento"
    seed=42,
    maxIter=100
)

gmm_model = gmm.fit(df_final_clustering)
df_gmm = gmm_model.transform(df_final_clustering)

# Ver proporções
df_gmm.groupBy("prediction").count().show()
```

```
=====
📝 GMM - CLUSTERIZAÇÃO DE VOLATILIDADE
=====

+-----+-----+
|prediction| count|
+-----+-----+
|      1|321588|
|      2|832423|
|      0|409988|
+-----+-----+
```

14.4.2.3. Mensuração de Incerteza via Entropia de Shannon

Definição

Uma das maiores vantagens do GMM sobre algoritmos tradicionais é a capacidade de medir a **incerteza da classificação**. Em vez de apenas dizer a qual cluster um cliente pertence, o modelo nos fornece um vetor de probabilidades.

Para quantificar o quanto "confuso" o modelo está em relação a um cliente, calculamos a **Entropia de Shannon**:

$$\$H(x_i) = -\sum_{k=1}^K P(C_k | x_i) \log P(C_k | x_i) \$$$

* Entropia Baixa: O modelo tem alta confiança. Uma das probabilidades está próxima de \$1\$ e as outras de \$0\$.

* Entropia Alta: O modelo está incerto. As probabilidades estão distribuídas entre dois ou mais clusters (ex: o cliente está na fronteira entre "Fiel" e "Risco de Churn").

No código a seguir, adiciona-se um termo constante (10^{-12}) para evitar o cálculo de $\log(0)$, garantindo a estabilidade numérica da operação.

Execução

```
In [111]: from pyspark.sql import functions as F
from pyspark.ml.functions import vector_to_array

print("\n" + "="*90)
print("💡 GMM - ENTROPIA (INCERTEZA) [CORRIGIDO]")
print("="*90)

df_gmm_entropy = (df_gmm
    .withColumn("prob_arr", vector_to_array("probability"))
    .withColumn(
        "entropy",
        F.expr("""
            -aggregate(
                prob_arr,
                cast(0.0 as double),
                (acc, x) -> acc + x * log(x + 1e-12)
            )
        """
    )
)
df_gmm_entropy.select("entropy").describe().show()

df_gmm_entropy.groupBy("prediction").agg(
    F.count("*").alias("n_clientes"),
    F.mean("entropy").alias("entropy_media"),
    F.expr("percentile_approx(entropy, 0.5)").alias("entropy_mediana"),
    F.mean("churn_rate_no_censor").alias("churn_rate_medio"),
    F.mean("flag_churn_algum_mes_no_censor").alias("pct_churn_algum_mes")
).orderBy("prediction").show(truncate=False)
```

```
=====
💡 GMM - ENTROPIA (INCERTEZA) [CORRIGIDO]
=====

+-----+-----+
|summary|      entropy|
+-----+-----+
| count| 1563999|
| mean| 0.12035502243002494|
| stddev| 0.2634747492504076|
| min| 3.995034641576909...|
| max| 1.09861228866511|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

prediction	n_clientes	entropy_media	entropy_mediana	churn_rate_medio	pct_churn_algum_mes
0	409988	0.13732446373980423	0.005906053327759478	0.16798850998644513	0.4552060060294448
1	321588	0.23959473004230852	0.01510178134271038	0.10675849111051938	0.2845068845852457
2	832423	0.06593157858557017	0.00130978322552334	0.2995447506509788	0.368895381314548

Perfil dos Agrupamentos

- Cluster 2 — O "Porto Seguro" (Baixa Volatilidade): é o maior grupo (\$832.423\$ clientes) e o mais estável, com a menor entropia média (\$0.065\$). Embora tenha o maior `churn_rate_medio` (29.9%), o modelo tem muita certeza sobre quem são esses clientes. É um grupo previsível, possivelmente de clientes com comportamento de churn já consolidado ou padrões de uso muito lineares.

- Cluster 1 — A Zona de Incerteza (Alta Volatilidade): apresenta a maior entropia média (\$0.239\$) e mediana (\$0.015\$). Estes são os clientes representados pelos pontos amarelos no gráfico (a seguir). O comportamento deles oscila entre as definições dos outros grupos. Apesar de terem o menor churn rate médio (\$10.6\%\$), o risco reside na instabilidade: eles são os candidatos ideais para monitoramento preventivo, pois o modelo ainda está "aprendendo" a direção que eles tomarão.

- Cluster 0 — Risco Intermediário: possui a maior taxa de churn acumulada (`pct_churn_algum_mes` de \$45.5\%\$). A entropia de \$0.137\$ sugere que, embora o modelo os identifique bem, há uma parcela considerável em transição.

A discrepância entre a Média e a Mediana de entropia (especialmente no Cluster 1, onde a média é 15x maior que a mediana) indica uma distribuição de cauda longa.

Insight: A maioria dos clientes está muito bem classificada (mediana próxima de 0), mas existe uma minoria crítica (os pontos amarelados no scatter plot a seguir) que puxa a média de incerteza para cima. São justamente esses clientes que geram os desvios de performance nas métricas de negócios.

14.4.3. Comparação entre algoritmos

```
In [112]: print("\n" + "="*90)
print("⭐ COMPARAÇÃO – PERFIL DOS CLUSTERS")
print("="*90)

km_summary = (df_kmeans.groupBy(F.col("prediction").alias("cluster_kmeans"))
    .agg(
        F.count("*").alias("n_clientes"),
        F.mean("churn_rate_no_censor").alias("churn_rate_medio"),
        F.mean("flag_churn_algum_mes_no_censor").alias("pct_churn_algum_mes")
    )
)

km_3_summary = (df_kmeans_k3.groupBy(F.col("prediction").alias("cluster_kmeans"))
    .agg(
        F.count("*").alias("n_clientes"),
        F.mean("churn_rate_no_censor").alias("churn_rate_medio"),
        F.mean("flag_churn_algum_mes_no_censor").alias("pct_churn_algum_mes")
    )
)

gmm_summary = (df_gmm_entropy.groupBy(F.col("prediction").alias("cluster_gmm"))
    .agg(
        F.count("*").alias("n_clientes"),
        F.mean("entropy").alias("entropy_media"),
        F.mean("churn_rate_no_censor").alias("churn_rate_medio"),
        F.mean("flag_churn_algum_mes_no_censor").alias("pct_churn_algum_mes")
    )
)

print("◆ KMeans summary - K = 6")
km_summary.orderBy("cluster_kmeans").show(truncate=False)

print("◆ KMeans summary - K = 3")
km_3_summary.orderBy("cluster_kmeans").show(truncate=False)

print("◆ GMM summary")
gmm_summary.orderBy("cluster_gmm").show(truncate=False)
```

```
=====
⭐ COMPARAÇÃO – PERFIL DOS CLUSTERS
=====

◆ KMeans summary - K = 6
+-----+-----+-----+-----+
|cluster_kmeans|n_clientes|churn_rate_medio |pct_churn_algum_mes |
+-----+-----+-----+-----+
|0          |619360    |0.04868584035962459 |0.19213381555153708 |
|1          |93884     |0.8091569349680423  |0.8587192705892378 |
|2          |127660    |0.08191967570786292 |0.39769700767664107 |
|3          |169526    |0.054841031250549496|0.21937048004435897|
|4          |238425    |0.04332743256650141 |0.17225123204361958 |
|5          |315144    |0.6865030636811157  |0.8140786434138045 |
+-----+-----+-----+-----+

◆ KMeans summary - K = 3
+-----+-----+-----+-----+
|cluster_kmeans|n_clientes|churn_rate_medio |pct_churn_algum_mes |
+-----+-----+-----+-----+
|0          |390963    |0.7366543020743022 |0.8283392546097712 |
|1          |884903    |0.0569927519093704 |0.2330334511240215 |
|2          |288133    |0.048990673744346416|0.19136301638479453 |
+-----+-----+-----+-----+

◆ GMM summary
+-----+-----+-----+-----+
|cluster_gmm|n_clientes|entropy_media   |churn_rate_medio |pct_churn_algum_mes |
+-----+-----+-----+-----+
|0          |409988    |0.13732446373980423|0.16798850998644513|0.4552060060294448 |
|1          |321588    |0.23959473004230852|0.10675849111051938|0.2845068845852457 |
|2          |832423    |0.06593157858557017|0.2995447506509788 |0.368895381314548 |
+-----+-----+-----+-----+
```

14.4.4. Gráficos para visualização

14.4.4.1. Extrair PC1/PC2 do vetor pca_features (Spark → colunas)

```
In [113]: from pyspark.ml.functions import vector_to_array

def add_pc1_pc2(df, vec_col="pca_features"):
    return (df
        .withColumn("pc_arr", vector_to_array(vec_col))
        .withColumn("pc1", F.col("pc_arr")[0])
        .withColumn("pc2", F.col("pc_arr")[1])
        .drop("pc_arr")
    )

df_km_plot = add_pc1_pc2(df_kmeans)
df_km_3_plot = add_pc1_pc2(df_kmeans_k3)
df_gmm_plot = add_pc1_pc2(df_gmm_entropy)
```

```
In [114]: sample_n = 50000 # ajuste: 20k~100k

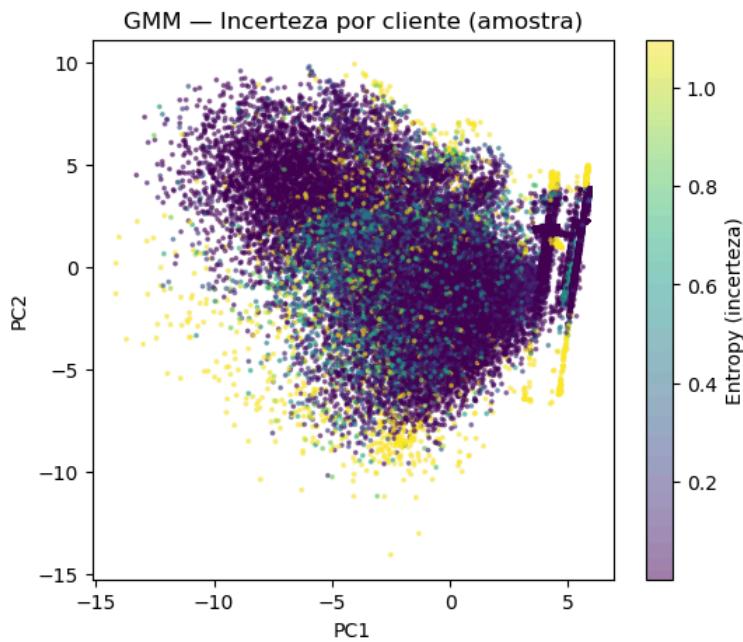
km_pd = (df_km_plot
    .select("pc1", "pc2", F.col("prediction").alias("cluster"))
    .orderBy(F.rand(seed=42))
    .limit(sample_n)
    .toPandas())

km_3_pd = (df_km_3_plot
    .select("pc1", "pc2", F.col("prediction").alias("cluster"))
    .orderBy(F.rand(seed=42))
    .limit(sample_n)
    .toPandas())

gmm_pd = (df_gmm_plot
    .select("pc1", "pc2", F.col("prediction").alias("cluster"), "entropy")
    .orderBy(F.rand(seed=42))
    .limit(sample_n)
    .toPandas())
```

14.4.4.2. Plot "incerteza" (entropia) no GMM

```
In [115]: plt.figure(figsize=(6,5))
plt.scatter(gmm_pd["pc1"], gmm_pd["pc2"], c=gmm_pd["entropy"], cmap="viridis", s=3, alpha=0.5)
plt.colorbar(label="Entropy (incerteza)")
plt.title("GMM – Incerteza por cliente (amostra)")
plt.xlabel("PC1"); plt.ylabel("PC2")
plt.show()
```



1. Zonas de Conflito (Pontos Amarelos): Note que os pontos com maior entropia (amarelos, $\$ > 0.8\$$) concentram-se justamente nas fronteiras de intersecção entre as grandes massas de dados. Isso faz todo sentido estatístico: são clientes cujo comportamento de PC1 e PC2 é ambíguo, não se encaixando perfeitamente em um perfil isolado.

2. Núcleos de Certeza (Pontos Roxos): O "coração" de cada cluster (especialmente no topo à esquerda e na extrema direita) apresenta entropia próxima de $\$0\$$. Isso indica perfis de clientes muito bem definidos — o modelo "tem certeza" do que eles são.

3. A Estrutura Linear à Direita: Existe uma formação linear peculiar no lado direito do gráfico (próximo a $PC1 = 5$). O fato de haver pontos amarelos misturados a roxos nessa "linha" sugere que essa dimensão representa uma transição crítica.

4. Insight de Negócio: Os clientes representados pelos pontos amarelos são os seus "Clientes de Fronteira". Eles são os alvos mais interessantes para testes A/B ou campanhas de retenção, pois um pequeno deslocamento no comportamento deles pode mudar completamente sua classificação de cluster (ex: de "Saudável" para "Risco").

14.4.4.3 Scatter plot (amostra) — KMeans e GMM

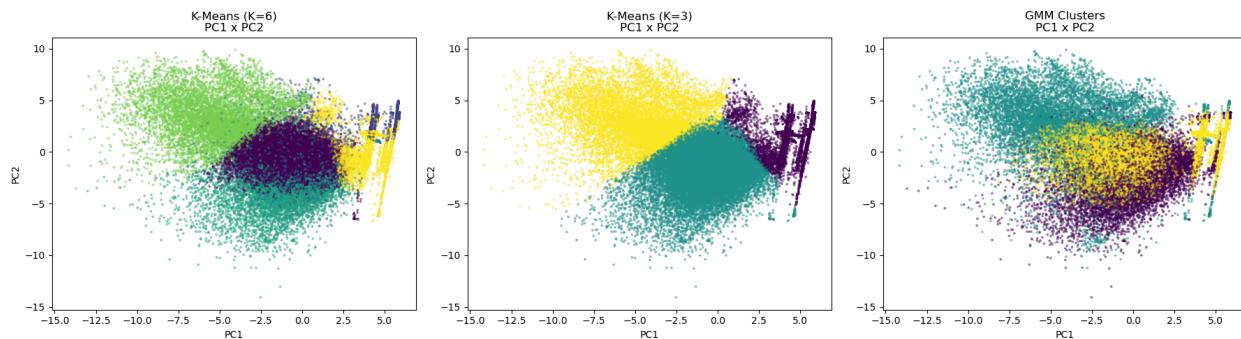
```
In [116]: # Plotar os tres graficos
plt.figure(figsize=(18, 5))

# Gráfico 1: K-Means K=6
plt.subplot(1, 3, 1)
plt.scatter(km_pd["pc1"], km_pd["pc2"], c=km_pd["cluster"], s=3, alpha=0.4, cmap='viridis')
plt.title("K-Means (K=6)\nPC1 x PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")

# Gráfico 2: K-Means K=3
plt.subplot(1, 3, 2)
plt.scatter(km_3_pd["pc1"], km_3_pd["pc2"], c=km_3_pd["cluster"], s=3, alpha=0.4, cmap='viridis')
plt.title("K-Means (K=3)\nPC1 x PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")

# Gráfico 3: GMM
plt.subplot(1, 3, 3)
plt.scatter(gmm_pd["pc1"], gmm_pd["pc2"], c=gmm_pd["cluster"], s=3, alpha=0.4, cmap='viridis')
plt.title("GMM Clusters\nPC1 x PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")

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



Comparação Visual e Estrutural (K-Means vs. GMM)

Ao observar os gráficos de dispersão (PC1 x PC2), é possível observar distinções fundamentais na forma como os algoritmos "enxergam" os dados:

- K-Means (K=3 e K=6): Cria divisões lineares e "limpas", conhecidas como células de Voronoi. No $K=3$, ele separa claramente um grupo de alto churn (amarelo) de dois grupos de baixo churn. No $K=6$, ele refina essa massa, mas as fronteiras continuam rígidas.

- GMM: Observe como as cores no gráfico do GMM se sobrepõem mais suavemente. Isso ocorre porque o GMM permite que os clusters tenham formas elípticas e diferentes orientações. Ele não apenas agrupa, mas modela a densidade. Onde você vê "nuvens" de pontos misturados, é onde a incerteza reside.

14.4.4.4. Unindo resultados

A "Incerteza" como Medida de Transição

No gráfico de Incerteza por Cliente, os pontos amarelos (alta entropia) não são erros; eles são clientes em estado de transição.

* Como a base é composta pelo agrupado por cliente (ou seja, comportamento resumido ao longo das safras), um cliente com alta entropia é aquele cujo comportamento está mudando entre uma safra e outra (ex: ele era muito fiel e começou a oscilar, ou era inativo e está retornando).

* O Cluster 1 do GMM (Entropia Média: \$0.239\$) é o "Laboratório de Volatilidade". Ele tem o menor churn rate imediato (\$10.6\%\$), mas a incerteza indica que o futuro desses clientes é imprevisível.

Conclusão sobre a Dinâmica Temporal

- Estabilidade (Cluster 2): com a menor entropia (\$0.065\$), o grupo representa o comportamento "inercial" da base. Seja "para o bem ou para o mal" (churn de \$29\%\$), esses clientes são consistentes ao longo das safras;

- Risco Silencioso (Cluster 0): possui o maior pct_churn_algum_mes (\$45.5\%\$). A entropia intermediária aqui sugere que o churn neste grupo não é um evento súbito, mas um processo de "derretimento" que o GMM consegue rastrear melhor que o K-Means.

Em resumo...

Enquanto o K-Means fornece uma fotografia estática da segmentação atual, o GMM pela entropia atua como um "termômetro" de volatilidade. A presença de alta incerteza nas fronteiras dos clusters revela que a base de clientes não é estática, mas que existe um fluxo constante de migração comportamental entre as safras, onde o Cluster 1 atua como a principal zona de transição de risco.

14.4.5. Proposta de Estratégia de Estabilização e Resgate (Conclusão)

O **Cluster 0** é identificado como o grupo de maior criticidade. Com um pct_churn_algum_mes de **45.5%**, este segmento não está apenas "em risco", mas já apresenta um comportamento histórico de ruptura consolidado em quase metade da sua base.

14.4.5.1. Perfil de Risco

* Volatilidade: Entropia de \$0.137\$. O modelo tem uma confiança razoável na classificação, o que significa que o padrão de churn é identificável e recorrente;

* Comportamento: É o grupo onde o "derretimento" (attrition) é mais visível nas safras.

14.4.5.2. Plano de Ação

1. Intervenção Preventiva (Churn Control) - Proposta de Churn da solução com algoritmo supervisionado

Como este grupo possui o maior volume de churn acumulado, deve-se aplicar uma régua de comunicação de re-engajamento agressivo:

* Trigger de inatividade: disparar ofertas personalizadas assim que o cliente atingir 50% do tempo médio de ciclo de compra/uso;
*Incentivos financeiros: utilização de cupons de win-back** ou descontos progressivos para quebrar o ciclo de saída.

2. Redução de Atrito no Atendimento

Clientes neste cluster costumam ser sensíveis a falhas de processo.

* Priorização: flag de "alto risco de churn" no CRM para atendimento prioritário no suporte/SAC;

* "Pesquisa de saída": Para os 45% que já churnaram, aplicar pesquisas qualitativas para identificar se o motivo é preço, produto ou concorrência.

3. Monitoramento da Entropia

Para os clientes neste cluster que apresentam entropia acima de 0.5:

* Clientes "em cima do muro": entre o Cluster 0 e o Cluster 1 (Incerteza);

* Ação: Teste A/B com ofertas de fidelização vs. ofertas de desconto pontual para mapear qual estímulo reduz a incerteza do modelo na safra seguinte.

14.4.5.3. KPI de Sucesso Sugerido

Meta: Reduzir o churn_rate_medio de \$16.7\%\$ para \$12\%\$ em 3 safras, focando na conversão de clientes de "Alta Entropia" para o Cluster 2 (Estável). Como os clientes foram agrupados ao longo das safras, o sucesso dessa estratégia será visto quando, numa eventual próxima rodada do GMM, a densidade de pontos se deslocar do Cluster 0 para o Cluster 2, ou quando a entropia média do Cluster 0 diminuir, indicando que as ações de marketing tornaram o comportamento desses clientes mais previsível e estável.

14.4.6. Salvando base agrupada

```
In [118]: print("Salvando dataset clusterizado final (com GMM) para análises futuras de entropia...")
df_gmm_entropy.write.mode("overwrite").parquet(gold_path + "df_clustered_01_uncertainty_and_variability")
print("Dataset salvo em:", gold_path + "df_clustered_01_uncertainty_and_variability")

Salvando dataset clusterizado final (com GMM) para análises futuras de entropia...
Dataset salvo em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_clustered_01_uncertainty_and_variability
```

14.5. Solução Proposta 2: Segmentação de Valor e Engajamento

Enquanto a análise anterior focou em comportamento temporal (volatilidade), esta foca em valor econômico e eficiência operacional. É a diferença entre saber como o cliente se move e saber quanto ele vale.

14.5.1. Melhor K

```
In [ ]: print("\n" + "*90)
print("💡 BUSCA DO K ÓTIMO (ELBOW + SILHOUETTE)")
print("*90)

df_sample = df_pca.sample(False, 0.10, seed=42).cache()

k_list = [3, 4, 5, 6, 7, 8] # Foco nos quadrantes de negócio (VIP, Dreno, etc), então começaremos com 3 (1 antes do esperado)
rows = []
evaluator = ClusteringEvaluator(featuresCol="pca_features", metricName="silhouette")

for k in k_list:
    km = KMeans(featuresCol="pca_features", k=k, seed=42, maxIter=30)
    km_model = km.fit(df_sample)
    pred = km_model.transform(df_sample)

    cost = km_model.summary.trainingCost
    sil = evaluator.evaluate(pred)

    rows.append((k, float(cost), float(sil)))
print(f"K={k} | Cost={cost:.2f} | Silhouette={sil:.4f}")

=====
💡 BUSCA DO K ÓTIMO (ELBOW + SILHOUETTE)
=====
K=3 | Cost=16093946.95 | Silhouette=0.2255
K=4 | Cost=14119459.07 | Silhouette=0.2652
K=5 | Cost=12627313.20 | Silhouette=0.3412
K=6 | Cost=11565923.01 | Silhouette=0.2882
K=7 | Cost=10057427.79 | Silhouette=0.3292
K=8 | Cost=9769583.92 | Silhouette=0.3155
```

```
In [ ]:
# Criar df_km_metrics a partir do rows
df_km_metrics = spark.createDataFrame(rows, ["k", "cost", "silhouette"])

# Escolher o melhor K automaticamente por maior silhouette
best_k = (df_km_metrics
    .orderBy(F.col("silhouette").desc(), F.col("k").asc())
    .first()["k"])

print("✅ Melhor K por silhouette:", best_k)

# converter para Pandas e plotar
df_plot = df_km_metrics.orderBy("k").toPandas()

fig, ax1 = plt.subplots(figsize=(10, 6))

# Eixo 1: Custo (Elbow)
color_elbow = 'tab:blue'
ax1.set_xlabel('Número de Clusters (K)')
ax1.set_ylabel('Custo (WSSSE / Inércia)', color=color_elbow, fontsize=12)
ax1.plot(df_plot['k'], df_plot['cost'], marker='o', color=color_elbow, linewidth=2, label='Custo')
ax1.tick_params(axis='y', labelcolor=color_elbow)
ax1.grid(True, linestyle='--', alpha=0.6)

# Eixo 2: Silhouette
ax2 = ax1.twinx()
color_sil = 'tab:red'
ax2.set_ylabel('Silhouette Score', color=color_sil, fontsize=12)
ax2.plot(df_plot['k'], df_plot['silhouette'], marker='s', color=color_sil, linewidth=2, label='Silhueta')
ax2.tick_params(axis='y', labelcolor=color_sil)

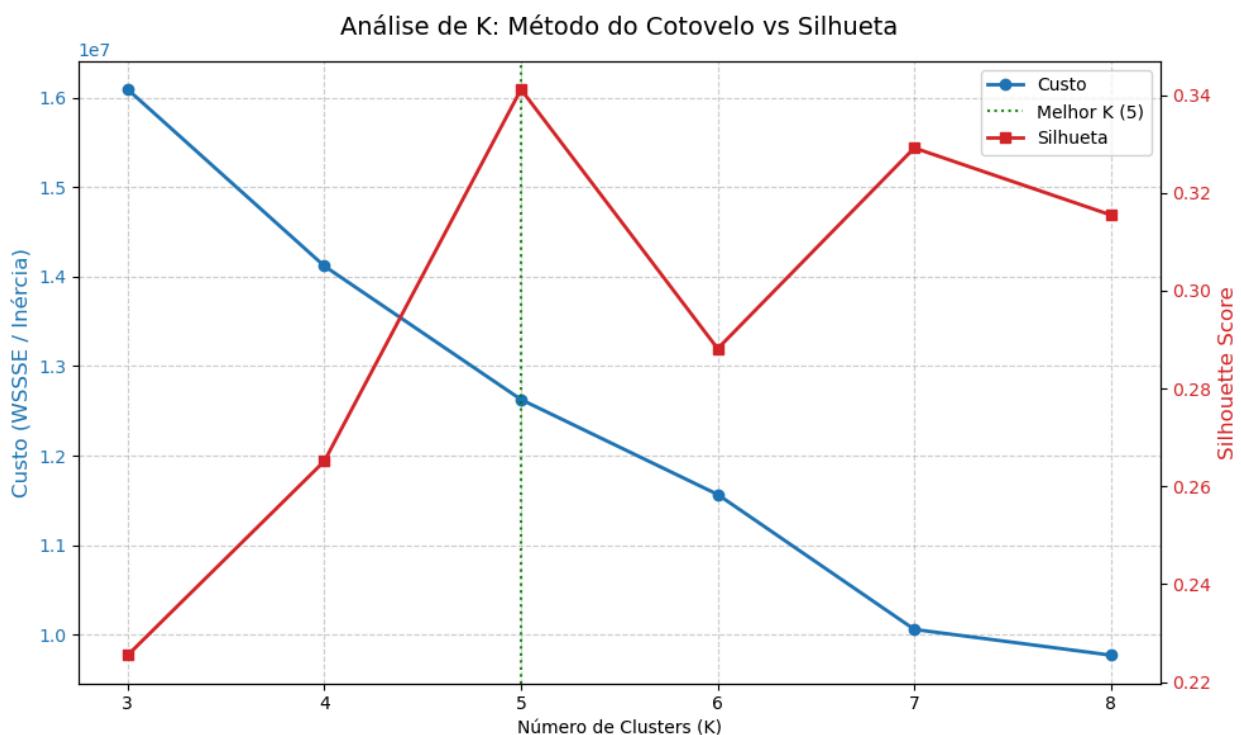
# Linha vertical no melhor K
ax1.axvline(x=best_k, color='green', linestyle=':', alpha=0.9, label=f'Melhor K ({best_k})')

# Título e ticks
plt.title('Análise de K: Método do Cotovelo vs Silhueta', fontsize=14, pad=15)
plt.xticks(df_plot['k'])

# Legenda combinada dos dois eixos
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines1 + lines2, labels1 + labels2, loc='best')

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

✅ Melhor K por silhouette: 5



A segmentação proposta na seção propostas de agrupamento foi:

- "VIPs" (alta margem, alto uso);
- "Eficientes" (alta margem, baixo uso - baixo custo operacional);
- "Drenos" (baixo lucro, mas usam muito a plataforma); e
- "Ausentes" (baixo lucro, baixo uso da plataforma).

Porém, pelo resultado de `best_k`, vou considerar um grupo a mais, o qual logo descobrirei se é de fato um novo grupo ou se poderia ser nomeado subgrupo de um maior.

14.5.2. Treinamento Final - K-Means + Visualização inicial

```
In [ ]: print("\n" + "*90")
print(f"🚀 TREINAMENTO FINAL K-MEANS (K={best_k})")
print("*90")

kmeans_final = KMeans(featuresCol="pca_features", k=best_k, seed=42, maxIter=50)
model_final = kmeans_final.fit(df_pca)
df_clustered = model_final.transform(df_pca)

summary = (df_clustered
    .groupBy("prediction")
    .agg(
        F.count("*").alias("n_cliente_mes"),
        F.mean("is_churn_M1").alias("taxa_churn_M1"),
        F.mean("margem_liquida_mensal").alias("margem_media"),
        F.expr("percentile_approx(margem_liquida_mensal, 0.5)").alias("margem_mediana"),
        F.mean("total_secs").alias("uso_medio_segundos"),
        F.expr("percentile_approx(total_secs, 0.5)").alias("uso_mediano_segundos"),
        F.mean("log_total_secs").alias("log_uso_medio"),
        F.mean("num_unq").alias("num_unq_medio"),
        F.mean("completed_songs_rate").alias("completed_rate_medio"),
        F.mean("early_drop_rate").alias("early_drop_medio"),
        F.mean("is_auto_renew").alias("pct_auto_renew"),
        F.mean("flag_plano_mensal").alias("pct_plano_mensal"),
        F.mean("flag_has_transactions").alias("pct_has_transactions")
    )
    .orderBy(F.desc("taxa_churn_M1"))
)
summary.show(truncate=False)
```

```
=====
🚀 TREINAMENTO FINAL K-MEANS (K=5)
=====
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
-----+
|prediction|n_cliente_mes|taxa_churn_M1      |margem_media      |margem_mediana
|uso_medio_segundos|uso_mediano_segundos|log_uso_medio      |num_unq_medio      |completed_rate_medio|early_drop_medio
|pct_auto_renew   |pct_plano_mensal   |pct_has_transactions|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
-----+
|2      |1818929      |0.23704828500727626 | -60.366412164296676 |-57.6811985      |97031.0089565934  |82108.481
|10.874924982014477|398.05242535580004|0.5928820408521552
|0.22741402572654704|0.8312501477517814|0.007169603651379465|0.1756423697681438 |
|1      |1017367      |0.02145833128556363|3.0608064554369054 |24.63150200000001|499646.6108034221 |443255.871
|13.048937355481428|1645.9543566874097|0.8208156205021695 |0.14047952028580737|0.8400184004395661|0.7797766194500116
|0.7851542265475487 |
|0      |3206286      |0.018478077127243173|66.19641586278289 |49.0          |41074.81006028663 |32473.584000000003
|9.936729637779768|187.89118968176888|0.4274206978489006 |0.26028969896856236|0.9726605798734111|0.9967311088280958
|0.9968240512543173 |
|3      |4108391      |0.017756099650690502|79.24855130911307 |84.6876545     |122258.57256681232|107199.605
|11.468173384699131|481.01687132505157|0.7064193179148567 |0.201496468847406 |0.8161871642694184|0.9984938142450415 |1.0
|
|4      |113093       |0.0          |761.9137672051475 |807.8115841     |148792.2685552058 |103690.98700000001
|11.383572402856295|582.0337775105444|0.6751988324326927 |0.2532301760898847|0.0          |0.0          |1.0
|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
-----+
```

Primeiras observações:

Cluster 4 — Os "Super VIPs" (Os 1%)

- Perfil: Margem média altíssima (761.91) e uso consistente;
- Diferencial: Curiosamente, têm 0% de auto-renovação e plano mensal. Isso sugere que são clientes de planos Anuais ou Corporativos (pagamento único antecipado), o que explica a margem gigante;
- Papel do negócio: Taxa de churn M+1 é 0.0. São a base mais estável do ecossistema.

Cluster 3 — "Subgrupo": Os "VIPs Operacionais" (Engajados e Lucrativos)

- Perfil: Alta margem (79.24) e uso robusto (122k segundos);
- Diferencial: 100% têm transações recentes. É o maior grupo em volume (\$4.1M\$ clientes-mês);
- Papel no negócio: É a "vaca leiteira" (Cash Cow). Pagam bem e usam muito o serviço.

Cluster 0 — Os "Eficientes" (Baixo Custo, Margem Saudável)

- Perfil: Margem positiva (66.19), mas com o menor uso da plataforma (41k segundos);
- Diferencial: Quase 100% em planos mensais e auto-renovação;
- Papel no negócio: São clientes lucrativos porque não "oneram" a infraestrutura (uso baixo), mas mantêm a assinatura ativa.

Cluster 1 — Os "Drenos" (Alto Engajamento, Baixa Margem)

- Perfil: O uso mais extremo de todos (499k segundos), mas com margem quase nula (3.06);
- Diferencial: São usuários "Heavy Users", que provavelmente estão em planos promocionais ou antigos;
- Papel no negócio: Risco. O custo operacional de servir esses clientes é proporcionalmente alto em relação ao que eles pagam.

Cluster 2 — Os "Ausentes em Risco" (Margem Negativa e Churn)

- Perfil: Margem negativa (-60.36) e maior taxa de churn (23.7%);

Diagnóstico: Este é o "grupo de risco" identificado. Eles rendem prejuízo e têm baixa probabilidade de renovação.

14.5.3. Gráficos

```
In [72]: print("\n" + "="*90)
print("📊 VISUALIZAÇÕES (AMOSTRAIS) – para performance")
print("="*90)

# =====
# 7. VISUALIZAÇÕES (AMOSTRA)
# =====
sample_frac_vis = 0.02 # 2% geralmente já dá muitos pontos em 11M; ajuste se precisar
seed = 42

df_vis = (df_clustered
          .select("prediction", "margem_liquida_mensal", "total_secs", "log_total_secs", "is_churn_m1", "safra", "pca_features")
          .sample(False, sample_frac_vis, seed=seed)
          )

# Converter pca_features (vector) -> colunas PC1, PC2 para scatter
from pyspark.ml.functions import vector_to_array
df_vis = df_vis.withColumn("pca_arr", vector_to_array("pca_features")) \
    .withColumn("PC1", F.col("pca_arr")[0]) \
    .withColumn("PC2", F.col("pca_arr")[1])

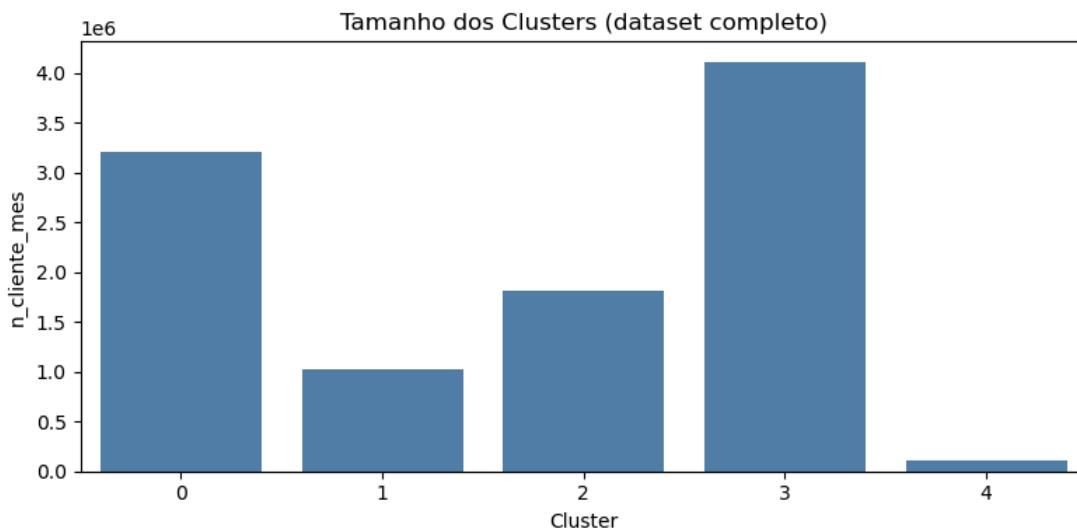
pdf = df_vis.select("prediction", "margem_liquida_mensal", "total_secs", "log_total_secs", "is_churn_m1", "safra", "PC1",
"PC2").toPandas()

# Garantir tipos
pdf["prediction"] = pdf["prediction"].astype(int)
pdf["is_churn_m1"] = pdf["is_churn_m1"].astype(int)

=====
📊 VISUALIZAÇÕES (AMOSTRAIS) – para performance
=====
```

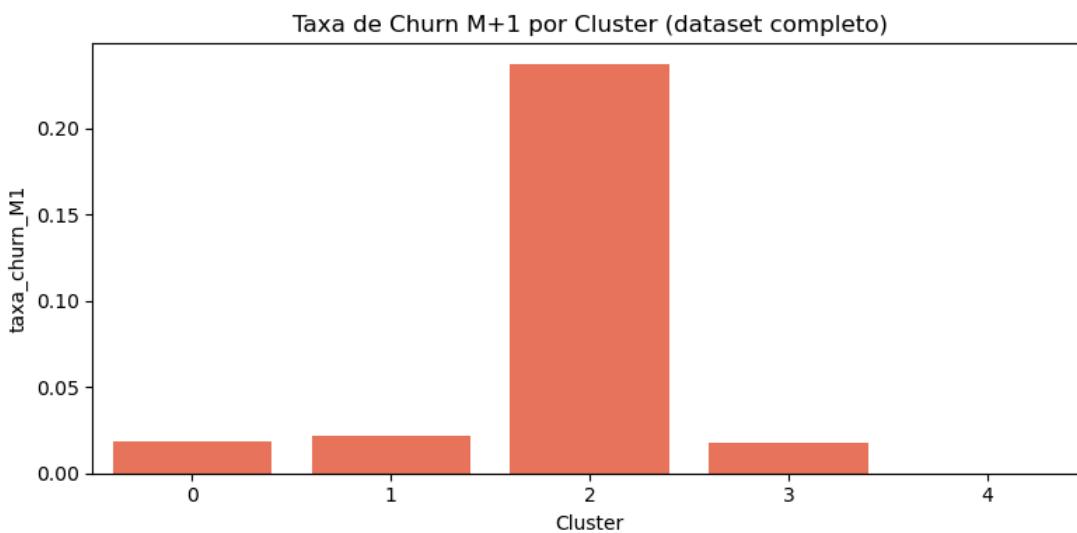
14.5.3.1. Tamanho amostras + churn por cluster

```
In [ ]: sizes_pdf = (df_clustered.groupBy("prediction").count().orderBy("prediction").toPandas())
plt.figure(figsize=(8, 4))
sns.barplot(data=sizes_pdf, x="prediction", y="count", color="steelblue")
plt.title("Tamanho dos Clusters (dataset completo)")
plt.xlabel("Cluster")
plt.ylabel("n_cliente_mes")
plt.tight_layout()
plt.show()
```



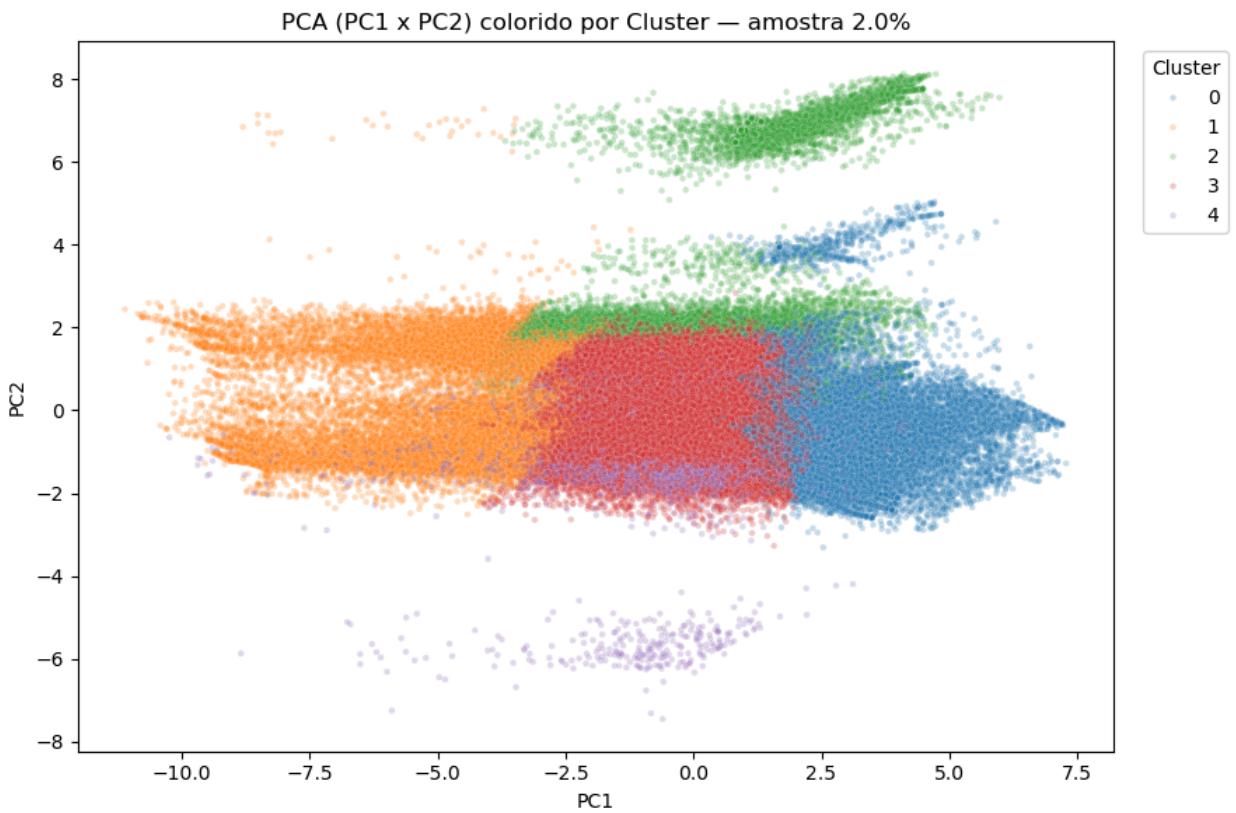
```
In [ ]: churn_pdf = (df_clustered.groupBy("prediction")
                    .agg(F.mean("is_churn_m1").alias("taxa_churn_M1"))
                    .orderBy("prediction")
                    .toPandas())

plt.figure(figsize=(8, 4))
sns.barplot(data=churn_pdf, x="prediction", y="taxa_churn_M1", color="tomato")
plt.title("Taxa de Churn M+1 por Cluster (dataset completo)")
plt.xlabel("Cluster")
plt.ylabel("taxa_churn_M1")
plt.tight_layout()
plt.show()
```



14.5.3.2. Scatter-Plot: storytelling do PCA + KMeans

```
In [ ]: plt.figure(figsize=(9, 6))
sns.scatterplot(data=pdf, x="PC1", y="PC2", hue="prediction", alpha=0.25, s=10, palette="tab10")
plt.title(f"PCA (PC1 x PC2) colorido por Cluster — amostra {sample_frac_vis*100:.1f}%")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="Cluster", bbox_to_anchor=(1.02, 1), loc="upper left")
plt.tight_layout()
plt.show()
```



14.5.3.3. Distribuição de Valor e Engajamento por cluster

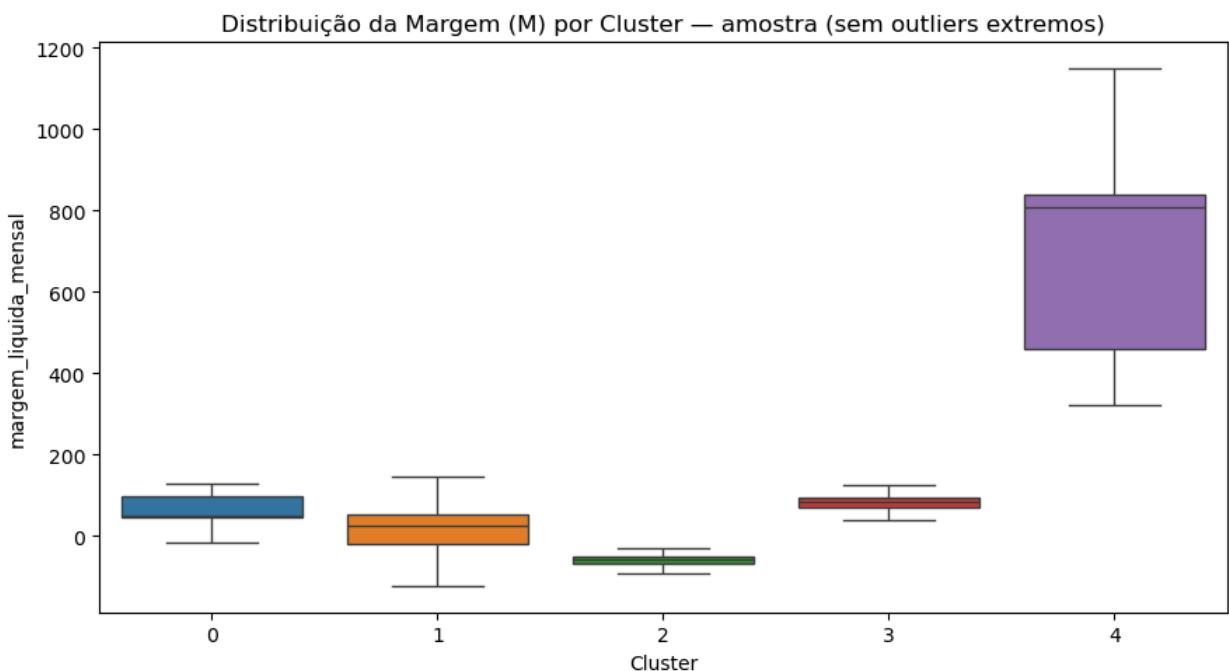
```
In [ ]:
# Margem
plt.figure(figsize=(9, 5))
sns.boxplot(data=pdf, x="prediction", y="margem_liquida_mensal", showfliers=False, palette="tab10")
plt.title("Distribuição da Margem (M) por Cluster – amostra (sem outliers extremos)")
plt.xlabel("Cluster")
plt.ylabel("margem_liquida_mensal")
plt.tight_layout()
plt.show()

# Uso (log_total_secs é mais legível)
plt.figure(figsize=(9, 5))
sns.boxplot(data=pdf, x="prediction", y="log_total_secs", showfliers=False, palette="tab10")
plt.title("Distribuição do Engajamento (log_total_secs) por Cluster – amostra")
plt.xlabel("Cluster")
plt.ylabel("log_total_secs")
plt.tight_layout()
plt.show()
```

C:\Users\Gustavo\AppData\Local\Temp\ipykernel_22964\1974209455.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

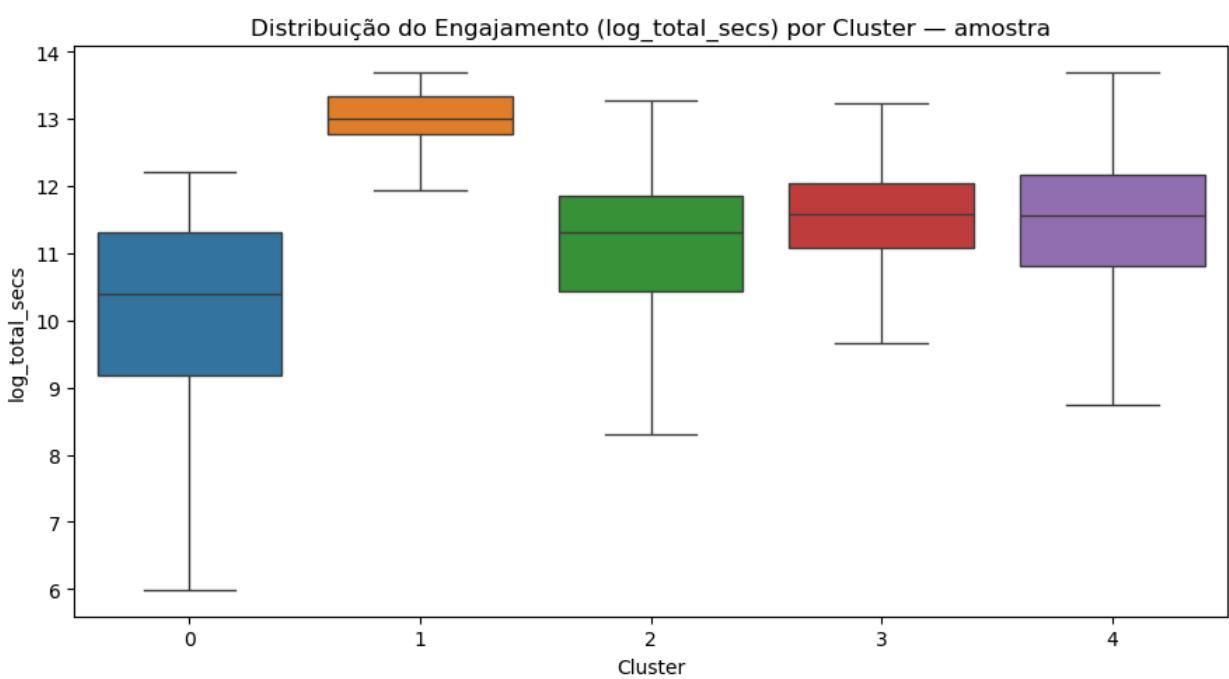
```
sns.boxplot(data=pdf, x="prediction", y="margem_liquida_mensal", showfliers=False, palette="tab10")
```



C:\Users\Gustavo\AppData\Local\Temp\ipykernel_22964\1974209455.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=pdf, x="prediction", y="log_total_secs", showfliers=False, palette="tab10")
```

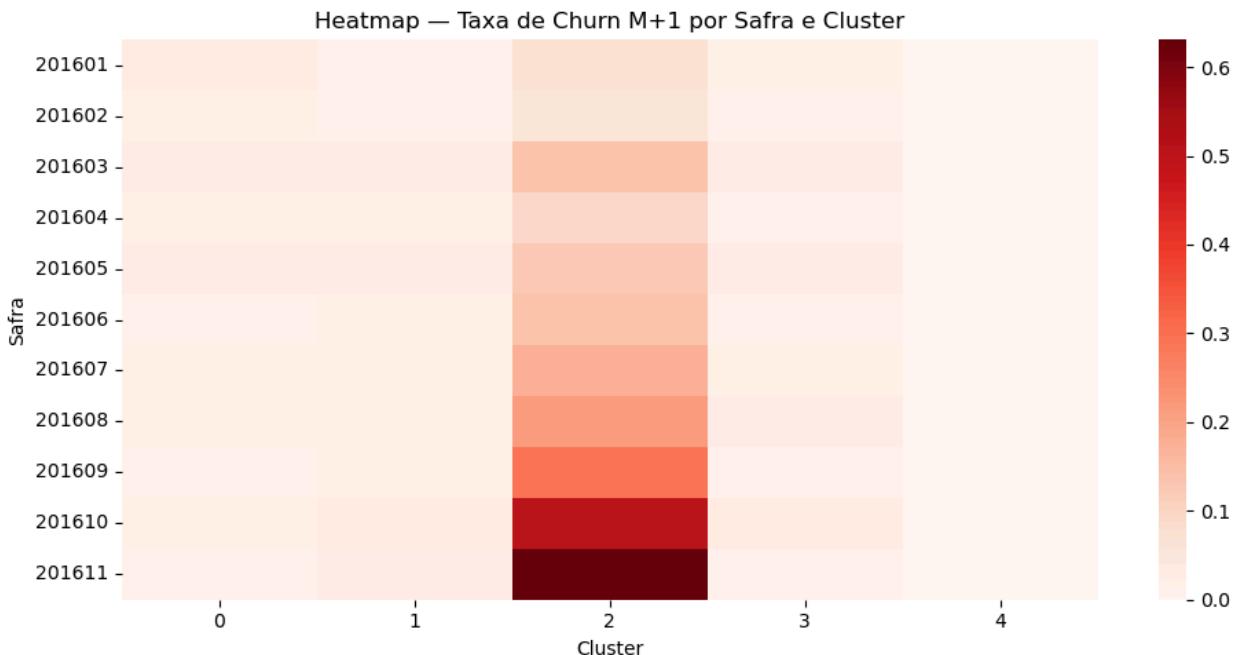


14.5.3.4. Heatmap

```
In [ ]:
heat = (df_clustered
         .groupBy("safra", "prediction")
         .agg(F.count("*").alias("n"), F.mean("is_churn_m1").alias("taxa_churn_M1"))
         .toPandas())

# pivôs para heatmap
heat_pivot = heat.pivot(index="safra", columns="prediction", values="taxa_churn_M1").sort_index()

plt.figure(figsize=(10, 5))
sns.heatmap(heat_pivot, cmap="Reds", annot=False)
plt.title("Heatmap — Taxa de Churn M+1 por Safra e Cluster")
plt.xlabel("Cluster")
plt.ylabel("Safra")
plt.tight_layout()
plt.show()
```



14.5.3.5. Conclusão com base na análise conjunta dos gráficos

Análise dos Grupos

- Cluster 4 (Super VIPs - Roxo): O boxplot de margem isola este grupo no topo da escala (\$> 800\$), enquanto o gráfico de dispersão (PCA) o coloca em "ilhas" distintas na parte inferior do gráfico. Eles são poucos em volume, mas representam a elite financeira da base;

- Cluster 1 (Drenos - Laranja): É o campeão de engajamento no boxplot de `log_total_secs`, mas sua margem é achataada próxima a zero. No PCA, ele ocupa a extrema esquerda, indicando um comportamento de uso massivo que o distancia dos demais;

- Cluster 2 (Grupo de Risco/Ausentes - Verde): O gráfico de barras de churn e o heatmap são implacáveis: este grupo detém quase todo o risco da operação. No PCA, eles estão dispersos no topo, longe do "centro de gravidade" dos usuários pagantes.

Por que o Churn aumenta no Cluster 2 ao longo das safras?

No Heatmap, o Cluster 2 começa com um churn moderado em janeiro e termina em um vermelho profundo em novembro (\$> 0.6\$). Existem três razões técnicas e de negócio para isso:

1. **Efeito de Seleção (Deterioração da Safras)**: Clientes bons permanecem nos clusters de valor (0, 3 ou 4). O Cluster 2 atua como um "filtro de decantação". Quem sobra nele a cada mês são os clientes que perderam o interesse e estão apenas esperando o fim do ciclo para cancelar;

2. **Sazonalidade de Final de Ano**: O aumento drástico no final de 2016 sugere que o comportamento de churn nesse grupo é sazonal. Clientes com margem negativa ou baixo uso tendem a cortar gastos supérfluos no encerramento do ano fiscal ou períodos de renovação de contrato;

3. **Inércia Negativa**: Como a ABT agrupa clientes ao longo das safras, o Cluster 2 captura a "morte lenta". O modelo identifica que, quanto mais tempo um cliente passa com margem negativa (Cluster 2), maior a probabilidade estatística de ele finalmente efetivar o churn.

14.5.4. Conclusão Geral: Valor vs. Engajamento

A análise prova que engajamento nem sempre é valor. O Cluster 1 (Drenos) usa muito a plataforma, mas não gera caixa. Já o Cluster 0 (Eficientes) usa pouco, mas é altamente lucrativo. O sucesso da estratégia de retenção depende de tratar esses grupos de forma oposta.

Estratégia de Ação por Perfil

Cluster 4 - Elite/Anuais

Blindagem: Atendimento white-glove. Não precisam de descontos, mas de reconhecimento e novas funcionalidades. Trazer benefícios a mais, diferenciais.

Cluster 3 - Engajados VIP

Fidelização: Manter o uso alto. São os candidatos perfeitos para programas de pontos ou até mesmo cashback.

Cluster 0 - Eficientes

Manutenção "silenciosa": "Não acorde a fera". Eles pagam e não usam. Evitar comunicações excessivas que os façam lembrar de cancelar a assinatura é uma boa abordagem.

Cluster 1 - Drenos

Revisão: Tentar migrá-los para planos que limitem o uso ou aumentem a margem. Usam do produto, mas pagam pouco por ele.

Cluster 2 - Risco Crítico

Recuperação ou Descarte: Campanhas de "Última Chance". Se não reativarem com ofertas agressivas, o churn é inevitável e aceitável (já que a margem é negativa).

14.5.5. Salvando base agrupada

```
In [79]: print("Salvando dataset clusterizado final (com PCA e clusters) para análises futuras...")
df_clustered.write.mode("overwrite").partitionBy("safra").parquet(gold_path + "df_clustered_02_value_and_engagement")
print("Dataset salvo em:", gold_path + "df_clustered_02_value_and_engagement")
```

```
Salvando dataset clusterizado final (com PCA e clusters) para análises futuras...
Dataset salvo em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_clustered_02_value_and_engagement
```

14.6.1. Carregando as previsões do LightGBM

```
In [124]: df_cliente_mes_pca = spark.read.parquet(gold_path + "df_master_clustering_cliente_mes_pca")
```

```
In [119]: df_lightm_predictions = spark.read.parquet(gold_path + "df_predictions_lightgbm")
```

```
In [95]: df_lightm_predictions.groupBy("partition").count().show()
```

partition	count
oot	1850732
train	6262831
test	1565303

```
In [120]: df_err = (df_lightm_predictions
    .filter(F.col("partition").isin(["test", "oot"])) # apenas fora de treino
    .withColumn("residual", F.col("target_win") - F.col("prediction_lgbm"))
    .withColumn("abs_residual", F.abs(F.col("residual"))))
```

```
In [102]: df_err.printSchema()
```

```
root
|-- msno: string (nullable = true)
|-- partition: string (nullable = true)
|-- set: string (nullable = true)
|-- target_win: double (nullable = true)
|-- prediction_lgbm: double (nullable = true)
|-- residual_lgbm: double (nullable = true)
|-- daily_revenue_efficiency: double (nullable = true)
|-- daily_revenue_efficiency_min_3: double (nullable = true)
|-- daily_revenue_efficiency_min_3_is_sentinel: long (nullable = true)
|-- flag_plano_mensal: long (nullable = true)
|-- flag_plano_mensal_max_3: long (nullable = true)
|-- flag_valid_fee_max_3: long (nullable = true)
|-- log_total_plays: double (nullable = true)
|-- log_total_plays_mean_3: double (nullable = true)
|-- log_total_plays_mean_3_is_sentinel: long (nullable = true)
|-- num_25: double (nullable = true)
|-- num_50: double (nullable = true)
|-- num_75: double (nullable = true)
|-- num_985: double (nullable = true)
|-- plays_per_unq_cap_min_6: double (nullable = true)
|-- plays_per_unq_cap_min_6_is_sentinel: long (nullable = true)
|-- revenue_per_hour_listened_cap: double (nullable = true)
|-- total_secs_mean_3: double (nullable = true)
|-- total_secs_mean_3_is_sentinel: long (nullable = true)
|-- total_secs_ratio_ref_max_6: double (nullable = true)
|-- total_secs_ratio_ref_max_6_is_sentinel: long (nullable = true)
|-- usage_intensity_per_tenure_cap: double (nullable = true)
|-- avg_secs_per_unq_cap_group_te: double (nullable = true)
|-- early_drop_rate_group_te: double (nullable = true)
|-- faixa_idade_te: double (nullable = true)
|-- gender_clean_te: double (nullable = true)
|-- plays_behavior_vs_completion_collapsed_te: double (nullable = true)
|-- plays_per_unq_behavior_te: double (nullable = true)
|-- registered_via_group_te: double (nullable = true)
|-- revenue_per_hour_tier_te: double (nullable = true)
|-- total_plays_group_te: double (nullable = true)
|-- usage_intensity_tier_te: double (nullable = true)
|-- safra: integer (nullable = true)
|-- residual: double (nullable = true)
|-- abs_residual: double (nullable = true)
```

```
In [121]: df_err.select(
    F.mean("abs_residual"),
    F.expr("percentile_approx(abs_residual, 0.9)"),
    F.expr("percentile_approx(abs_residual, 0.95)")
).show()
```

avg(abs_residual) percentile_approx(abs_residual, 0.9, 10000) percentile_approx(abs_residual, 0.95, 10000)
15.829586953854628 31.238655434048034 75.3930518805964

```
In [131]: # Threshold top 5%
threshold = df_err.approxQuantile("abs_residual", [0.95], 0.001)[0]

df_err = df_err.withColumn("is_worst", (F.col("abs_residual") >= F.lit(threshold)).cast("int"))

print("Total registros:", df_err.count())
print("Worst registros:", df_err.filter("is_worst = 1").count())
```

```
Total registros: 3416035
Worst registros: 172501
```

14.6.2. Unindo público de previsões errôneas + ABT msno + safra

```
In [142]: df_audit = (df_err
    .select("msno", "safra", "partition", "prediction_lgbm", "residual", "abs_residual", "is_worst")
    .join(df_cliente_mes_pca, on=["msno", "safra"], how="inner"))
```

```
In [133]: df_audit.count()
```

```
Out[133]: 3416035
```

```
In [134]: df_audit.printSchema()

root
 |-- msno: string (nullable = true)
 |-- safra: integer (nullable = true)
 |-- partition: string (nullable = true)
 |-- target_win: double (nullable = true)
 |-- prediction_lgbm: double (nullable = true)
 |-- residual: double (nullable = true)
 |-- abs_residual: double (nullable = true)
 |-- total_secs: double (nullable = true)
 |-- num_unq: double (nullable = true)
 |-- total_plays: double (nullable = true)
 |-- log_total_secs: double (nullable = true)
 |-- completed_songs_rate: double (nullable = true)
 |-- early_drop_rate: double (nullable = true)
 |-- plays_per_unq_cap: double (nullable = true)
 |-- catalog_exploration_ratio_cap: double (nullable = true)
 |-- margem_liquida_mensal: double (nullable = true)
 |-- actual_amount_paid: float (nullable = true)
 |-- daily_revenue_efficiency: double (nullable = true)
 |-- revenue_per_hour_listened_cap: double (nullable = true)
 |-- usage_intensity_per_tenure_cap: double (nullable = true)
 |-- total_secs_ratio_ref_mean_3: double (nullable = true)
 |-- log_total_plays_mean_3: double (nullable = true)
 |-- total_secs_mean_3: double (nullable = true)
 |-- is_auto_renew: integer (nullable = true)
 |-- flag_plano_mensal: integer (nullable = true)
 |-- flag_has_transactions: integer (nullable = true)
 |-- faixa_idade_te: double (nullable = true)
 |-- gender_clean_te: double (nullable = true)
 |-- tenure_faixa_te: double (nullable = true)
 |-- payment_method_group_te: double (nullable = true)
 |-- revenue_tier_te: double (nullable = true)
 |-- target_win: double (nullable = true)
 |-- is_censored: integer (nullable = true)
 |-- is_churn_m1: integer (nullable = true)
 |-- features_raw: vector (nullable = true)
 |-- features_scaled: vector (nullable = true)
 |-- pca_features: vector (nullable = true)
```

```
In [137]: df_audit.groupBy("is_worst").agg(
    F.count("*").alias("n"),
    F.mean("abs_residual").alias("erro_medio"),
    F.mean("margem_liquida_mensal").alias("margem_media"),
    F.mean("total_secs").alias("uso_medio"),
    F.mean("is_auto_renew").alias("pct_auto_renew"),
    F.mean("tenure_faixa_te").alias("tenure_te_medio")
).show()
```

is_worst	n	erro_medio	margem_media	uso_medio	pct_auto_renew	tenure_te_medio
1	172501	130.498375951717	41.84253711625912	129974.57992598023	0.5632141262949201	0.8602553514304666
0	3243534	9.731152107532457	55.959379943886056	133709.6338967611	0.9090165233353497	0.8611294998648371

Métrica	Worst (Top 5%)	Resto	Interpretação
Erro médio	130.50	9.73	Worst erra 13x mais (esperado por construção)
Margem média	41.84	55.96	Worst tem 25% menos margem → clientes de menor valor
Uso médio	129.9k seg	133.7k seg	Uso ligeiramente menor (não é o principal driver)
Auto-renew	56%	91%	Worst tem 38% menos renovação automática → instabilidade contratual
Tenure TE	0.860	0.861	Praticamente igual (tenure não explica)

Perfil de Falha Identificado:

O modelo falha desproporcionalmente em clientes de menor valor econômico e sem renovação automática (maior volatilidade comportamental). Esses clientes representam 5% da base, mas concentram erros 13x maiores. A partir deste ponto, vamos focar somente nos casos de `is_worst == 1`

14.6.3. K-Means

14.6.3.1. Melhor K

```
In [143]: print("\n" + "="*90)
print("💡 CLUSTERING DOS PIORES ERROS – BUSCA DE K ÓTIMO")
print("="*90)

# 1. FILTRAR APENAS OS WORST
df_worst = df_audit.filter(F.col("is_worst").isin(1))
print(f"✓ Registros para clustering (Top 5% erros): {df_worst.count():,}")

# 2. BUSCA DE K (SILHOUETTE)
k_list = [3, 4, 5, 6]
rows = []
evaluator = ClusteringEvaluator(featuresCol="pca_features", metricName="silhouette")

for k in k_list:
    km = KMeans(featuresCol="pca_features", k=k, seed=42, maxIter=50)
    model = km.fit(df_worst)
    pred = model.transform(df_worst)

    sil = evaluator.evaluate(pred)
    cost = model.summary.trainingCost

    rows.append((k, float(cost), float(sil)))
    print(f"K={k} | Cost={cost:.2f} | Silhouette={sil:.4f}")

# DataFrame com métricas
df_k_metrics = spark.createDataFrame(rows, ["k", "cost", "silhouette"])

# Melhor K por silhouette
best_k = (df_k_metrics
    .orderBy(F.col("silhouette").desc(), F.col("k").asc())
    .first()["k"])

print(f"\n💡 Melhor K por silhouette: {best_k}")
```

```
=====
💡 CLUSTERING DOS PIORES ERROS – BUSCA DE K ÓTIMO
=====

✓ Registros para clustering (Top 5% erros): 172,501
K=3 | Cost=2618759.11 | Silhouette=0.3678
K=4 | Cost=2238241.94 | Silhouette=0.3664
K=5 | Cost=1864410.70 | Silhouette=0.4258
K=6 | Cost=1717845.48 | Silhouette=0.4259
```

💡 Melhor K por silhouette: 6

```
In [144]: pdf_k = df_k_metrics.orderBy("k").toPandas()

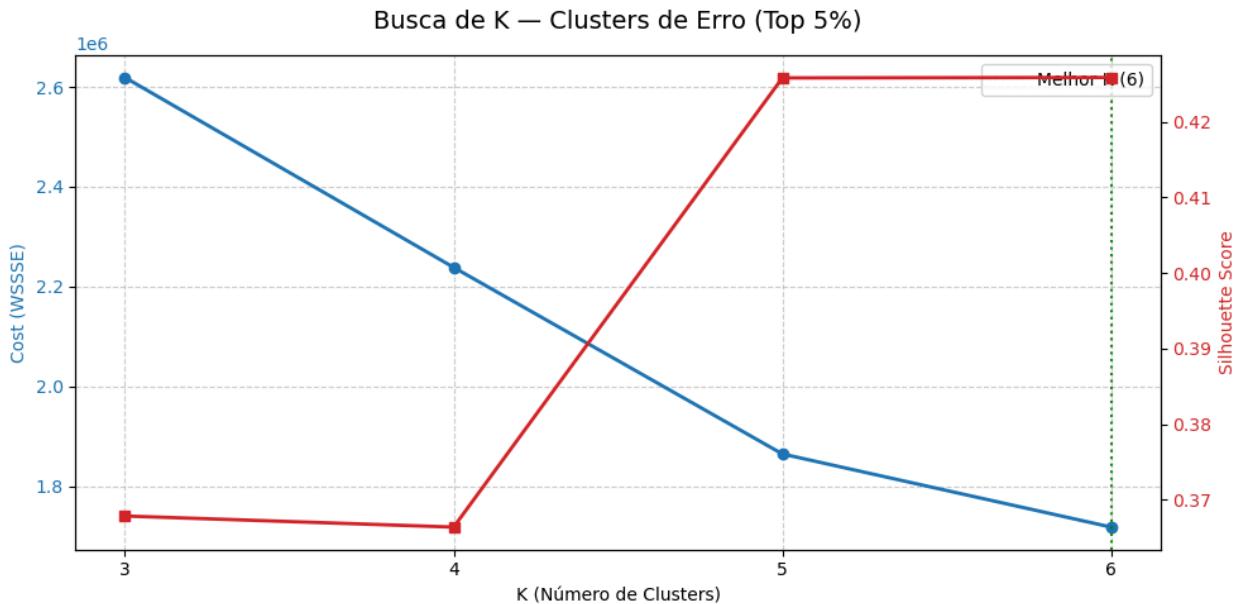
fig, ax1 = plt.subplots(figsize=(10, 5))

# Eixo 1: Cost
ax1.set_xlabel('K (Número de Clusters)')
ax1.set_ylabel('Cost (WSSSE)', color='tab:blue')
ax1.plot(pdf_k['k'], pdf_k['cost'], marker='o', color='tab:blue', linewidth=2)
ax1.tick_params(axis='y', labelcolor='tab:blue')
ax1.grid(True, linestyle='--', alpha=0.6)

# Eixo 2: Silhouette
ax2 = ax1.twinx()
ax2.set_ylabel('Silhouette Score', color='tab:red')
ax2.plot(pdf_k['k'], pdf_k['silhouette'], marker='s', color='tab:red', linewidth=2)
ax2.tick_params(axis='y', labelcolor='tab:red')

# Linha no melhor K
ax1.axvline(x=best_k, color='green', linestyle=':', alpha=0.9, label=f'Melhor K ({best_k})')

plt.title('Busca de K — Clusters de Erro (Top 5%)', fontsize=14, pad=15)
plt.xticks(pdf_k['k'])
ax1.legend(loc='upper right')
fig.tight_layout()
plt.show()
```



14.6.3.2. Execução com melhor K

```
In [145]: # 4. TREINAMENTO FINAL COM MELHOR K
print("\n" + "="*90)
print(f"🚀 TREINAMENTO FINAL — K={best_k}")
print("=*90")

kmeans_final = KMeans(featuresCol="pca_features", k=best_k, seed=42, maxIter=100)
model_final = kmeans_final.fit(df_worst)
df_worst_clustered = model_final.transform(df_worst)
```

```
=====
🚀 TREINAMENTO FINAL — K=6
=====
```

```
In [146]: # 5. PERFIL DOS CLUSTERS DE ERRO
summary_err = (df_worst_clustered
    .groupBy("prediction")
    .agg(
        F.count("*").alias("n"),
        F.mean("abs_residual").alias("erro_medio_abs"),
        F.expr("percentile_approx(abs_residual, 0.5)").alias("erro_p50"),
        F.mean("residual").alias("bias_medio"), # + subestima / - superestima
        F.mean("target_win").alias("margem_real_media"),
        F.mean("prediction_lgbm").alias("predicao_media"),
        F.mean("margem_liquida_mensal").alias("margem_M_media"),
        F.mean("total_secs").alias("uso_medio_seg"),
        F.mean("log_total_secs").alias("log_uso_medio"),
        F.mean("num_unq").alias("num_unq_medio"),
        F.mean("is_auto_renew").alias("pct_auto_renew"),
        F.mean("flag_plano_mensal").alias("pct_plano_mensal"),
        F.mean("flag_has_transactions").alias("pct_has_transactions"),
        F.mean("tenure_faixa_te").alias("tenure_te_medio"),
        F.mean("faixa_idade_te").alias("idade_te_medio"),
        F.mean("revenue_tier_te").alias("revenue_tier_te_medio")
    )
    .orderBy(F.desc("erro_medio_abs"))
)
.print("\n📊 PERFIL DOS CLUSTERS DE ERRO:")
summary_err.show(truncate=False)
```

📊 PERFIL DOS CLUSTERS DE ERRO:							
	prediction	erro_medio_abs	erro_p50	bias_medio	margem_real_media	predicao_media	
	margem_M_media	uso_medio_seg	log_uso_medio	num_unq_medio	pct_auto_renew	pct_plano_mensal	
	pct_has_transactions	tenure_te_medio	idade_te_medio	revenue_tier_te_medio			
1	30500	144.52974491132795	141.4875733687274	86.74001440611731	69.21531739260695		
-17.524697013510732	-63.22762608075084	116145.46838606577	11.263849792874305	473.46481967213117	1.0		0.0
0.0		0.8616143460181624	0.8627687263676878	0.8312695595601343			
5	53936	136.01854468323754	138.1117436958918	-131.54181780844024	-55.81482774735612	75.72699006108405	
95.69343894567638		100135.93621720205	11.13060843664151	407.0423279442302	0.001075348561257787	0.995550281815485	1.0
0.8589802840485966	0.8580047600219429	0.8728588093583344					
4	14932	130.25521642178825	128.00708858137	-48.75479517130582	-34.359128576634134	14.395666594671745	
-4.48267671075543	536939.4280023441	13.129496565714577	1733.4174926332707	0.5898071256362175			
0.7164478971336726	0.7237476560407179	0.8604926349729577	0.8606644546944164	0.8666790562260266			
3	11657	125.78995782093662	128.70272046498712	-110.51907589623433	-42.43008101649653	68.08899487973778	
86.46342964175173	15102.557575791374	8.52858780842944	143.29098395813674	0.4410225615509994			
0.9462983614995282	0.9476709273397959	0.8587590360530744	0.858789666298146	0.8644052875946032			
0	12257	120.85108639024314	120.51915720234736	35.50802023233124	43.34973508621196	7.841714853880714	
-55.011314464583506	55909.98601142199	9.960660423809927	231.7850208044383	0.28220608631802235	0.2846536672921596	1.0	
0.8591181186490132	0.8577703044885654	0.2249948114184195					
2	49219	119.34559231818004	119.38108246599197	-116.51422908831428	-55.09771628293949	61.4165128053748	
75.54614230854348	93428.50563993635	10.9765346524568	369.76610658485544	0.9994107966435726	1.0		1.0
0.8613760810690081	0.8621963506834899	0.8945032242581593					

Pontos de atenção:

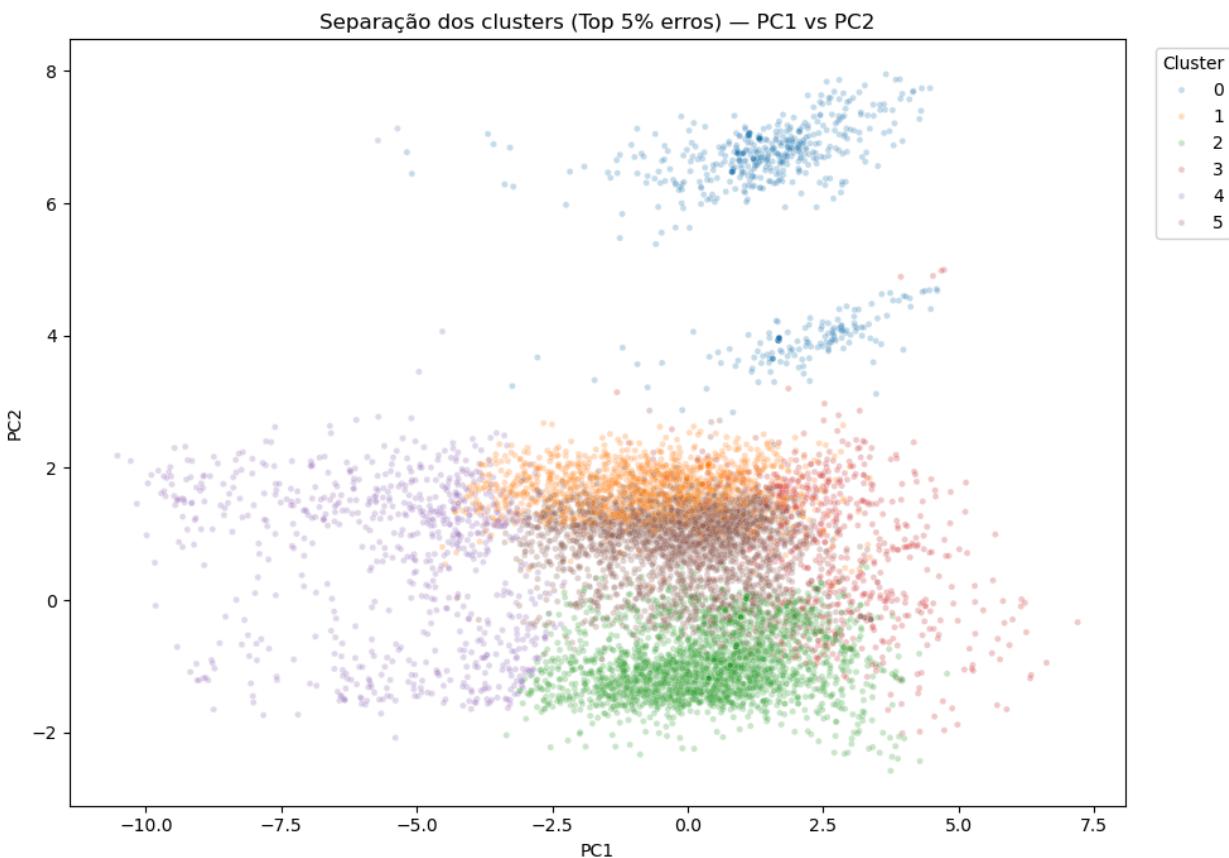
- bias_medio indo de valores positivos até negativos, com valores bem dispersos;
- feat_te : nenhuma categórica com target encoding entrega separação clara dos usuários de maior erro.

14.6.3.3. Resultados

```
In [155]: df_plot = (df_worst_clustered
    .withColumn("pca_arr", vector_to_array(F.col("pca_features")))
    .withColumn("pc1", F.col("pca_arr")[0])
    .withColumn("pc2", F.col("pca_arr")[1])
    .select("pc1", "pc2", "prediction", "partition", "abs_residual")
)
```

```
pdf = df_plot.sample(withReplacement=False, fraction=0.05, seed=42).toPandas()

plt.figure(figsize=(10, 7))
sns.scatterplot(
    data=pdf, x="pc1", y="pc2",
    hue="prediction",
    alpha=0.25, s=12,
    palette="tab10"
)
plt.title("Separação dos clusters (Top 5% erros) — PC1 vs PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="Cluster", bbox_to_anchor=(1.02, 1), loc="upper left")
plt.tight_layout()
plt.show()
```



- Isolamento do Erro Tipo 0: Note que o Cluster 0 (azul claro) aparece em "ilhas" isoladas no topo do gráfico. Isso indica que esses erros ocorrem em perfis de clientes muito específicos e geograficamente (matematicamente) distantes da massa principal. É um erro de "nicho";

- A Massa Central de Incerteza: Os Clusters 5, 1 e 2 estão densamente sobrepostos no centro. Isso explica por que o modelo se confunde: os atributos desses clientes (PC1 e PC2) são muito parecidos, mas seus resultados de margem real são opostos. O modelo não tem variáveis suficientes para separar o "joio do trigo" nessa zona central.

- Dispersão do Cluster 4: O Cluster 4 (roxo) está muito espalhado à esquerda. Isso corrobora o fato de serem os Heavy Users (alto uso). O modelo erra neles por causa da alta variabilidade do comportamento de uso intenso.

```
In [147]: # 6. ANÁLISE POR PARTIÇÃO (TEST vs OOT)
summary_part = (df_worst_clustered
    .groupBy("partition", "prediction")
    .agg(
        F.count("*").alias("n"),
        F.mean("abs_residual").alias("erro_medio_abs"),
        F.mean("residual").alias("bias_medio")
    )
    .orderBy("partition", "prediction")
)

print("\n📊 ERRO POR PARTIÇÃO (TEST vs OOT):")
summary_part.show(truncate=False)
```

📊 ERRO POR PARTIÇÃO (TEST vs OOT):				
partition	prediction	n	erro_medio_abs	bias_medio
oot	0	6699	124.33344481389751	106.69686642709442
oot	1	10332	137.27447049680757	72.81934302412537
oot	2	22614	117.16609025733842	-114.13719302800754
oot	3	5572	127.09060710687567	-114.10432307874716
oot	4	6784	127.41324935552377	-68.70352024894646
oot	5	28024	138.7344702508955	-136.04613426544327
test	0	5558	116.65383574611558	-50.2951608865458
test	1	20168	148.24659810702528	93.87152852346854
test	2	26605	121.19814858970321	-118.53468740932908
test	3	6085	124.59896064406692	-107.23608537840985
test	4	8148	132.62142955108857	-32.14554738943121
test	5	25912	133.08125315390564	-126.67036966120901

A análise das partições revela que os perfis de erro identificados não são aleatórios, mas sim **deficiências estruturais** do modelo que se repetem ao longo do tempo.

- * **Consistência Sistêmica:** os Clusters 1 (Subestimação) e 5 (Superestimação) apresentam erros persistentes em ambas as bases, sugerindo a criação de possíveis **regras de compensação pós-modelo**;
- * **Alerta de Instabilidade (Cluster 0):** inversão de viés no Cluster 0, passando de superestimação no teste para uma subestimação severa no OOT. É interessante monitorar bem ao utilizar previsões para este grupo em safras recentes;
- * **Confiabilidade do MAE:** a variação do Erro Médio Absoluto entre as bases é inferior a 10% na maioria dos clusters, o que valida a utilização desta auditoria como base para o diagnóstico de falhas com os stakeholders.

```
In [148]: # 7. APlicar MODELO NO DATASET COMPLETO (para calcular prob_worst por cluster)
print("\n" + "="*90)
print("🔍 PROBABILIDADE DE ERRO ALTO POR CLUSTER (BASE COMPLETA)")
print("="*90)

df_audit_clustered = model_final.transform(df_audit)

prob_worst = (df_audit_clustered
    .groupBy("prediction")
    .agg(
        F.count("*").alias("n_total"),
        F.sum("is_worst").alias("n_worst"),
        F.mean("is_worst").alias("prob_worst"),
        F.mean("abs_residual").alias("erro_medio_abs"),
        F.mean("margem_liquida_mensal").alias("margem_media"),
        F.mean("is_auto_renew").alias("pct_auto_renew")
    )
    .orderBy(F.desc("prob_worst"))
)

print("\n📊 PROBABILIDADE DE ESTAR NO TOP 5% DE ERRO:")
prob_worst.show(truncate=False)
```

=====							
🔍 PROBABILIDADE DE ERRO ALTO POR CLUSTER (BASE COMPLETA)							
=====							
📊 PROBABILIDADE DE ESTAR NO TOP 5% DE ERRO:							
prediction	n_total	n_worst	prob_worst	erro_medio_abs	margem_media	pct_auto_renew	
0	25163	12257	0.4871040813893417	74.85322577120148	-57.00804101218457	0.35842308150856417	
5	288467	53936	0.18697459328103389	37.54356923362797	152.0913032886376	0.005900154957066146	
1	401697	30500	0.07592787598612885	25.745314739085906	-63.76402386472962	1.0	
3	224509	11657	0.051922194655893525	13.36319652406007	72.12768313656163	0.9024894324949111	
4	332126	14932	0.04495884092181883	20.685212518093678	12.394906518561191	0.8626816328742706	
2	2144073	49219	0.02295584152218698	9.863815312143746	70.70143351972291	0.9999785455066129	

Ao expandir a análise para a base completa, é possível quantificar a **confiabilidade** das previsões de acordo com o perfil do cliente:

- * **Incerteza Crítica (Cluster 0):** quase **50% das previsões** para este grupo resultam em erros severos (Top 5%). Decisões automáticas de negócios devem ser suspensas para este perfil até o ajuste das features;
- * **Vulnerabilidade Contratual (Cluster 5):** clientes fora do regime de renovação automática apresentam uma probabilidade de erro **8x maior** que a média da base estável. O modelo superestima a retenção de valor onde não há recorrência garantida;
- * **Eficiência Operacional (Cluster 2): 85% da base total** (Cluster 2) opera em uma zona de segurança, com erro médio absoluto inferior a 10 unidades. O modelo é altamente recomendável para automações em decisões de negócios neste segmento.

Gráficos

In [153]:

```
# 8. VISUALIZAÇÕES
print("\n" + "="*90)
print("📊 VISUALIZAÇÕES")
print("="*90)

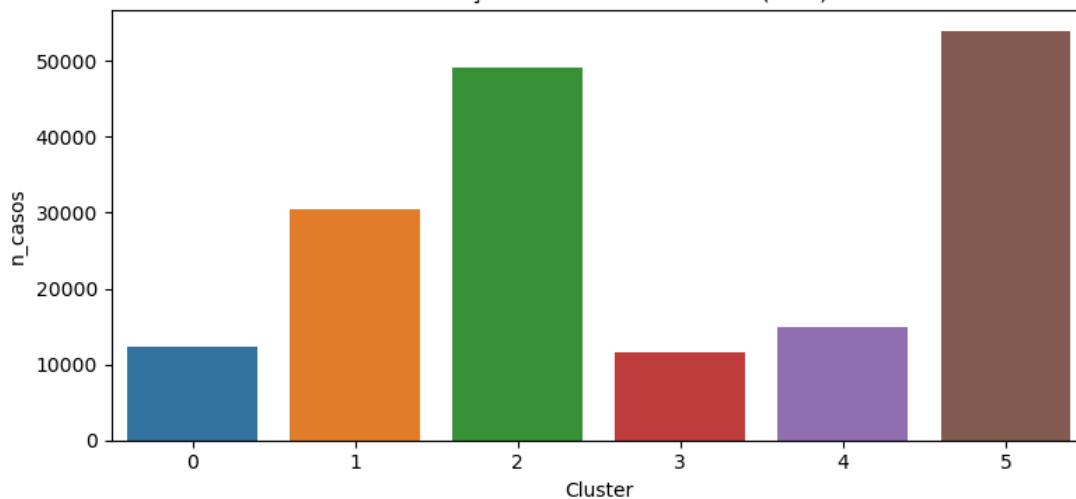
# 8.1 Tamanho dos clusters de erro
pdf_size = df_worst_clustered.groupBy("prediction").count().orderBy("prediction").toPandas()
plt.figure(figsize=(8, 4))
sns.barplot(data=pdf_size, x="prediction", y="count", palette="tab10")
plt.title(f"Distribuição dos Clusters de Erro (K={best_k})")
plt.xlabel("Cluster")
plt.ylabel("n_casos")
plt.tight_layout()
plt.show()

# 8.2 Erro médio por cluster
pdf_err = summary_err.toPandas()
plt.figure(figsize=(8, 4))
sns.barplot(data=pdf_err, x="prediction", y="erro_medio_abs", palette="Reds_r")
plt.title("Erro Médio Absoluto por Cluster")
plt.xlabel("Cluster")
plt.ylabel("Erro Médio Abs")
plt.tight_layout()
plt.show()

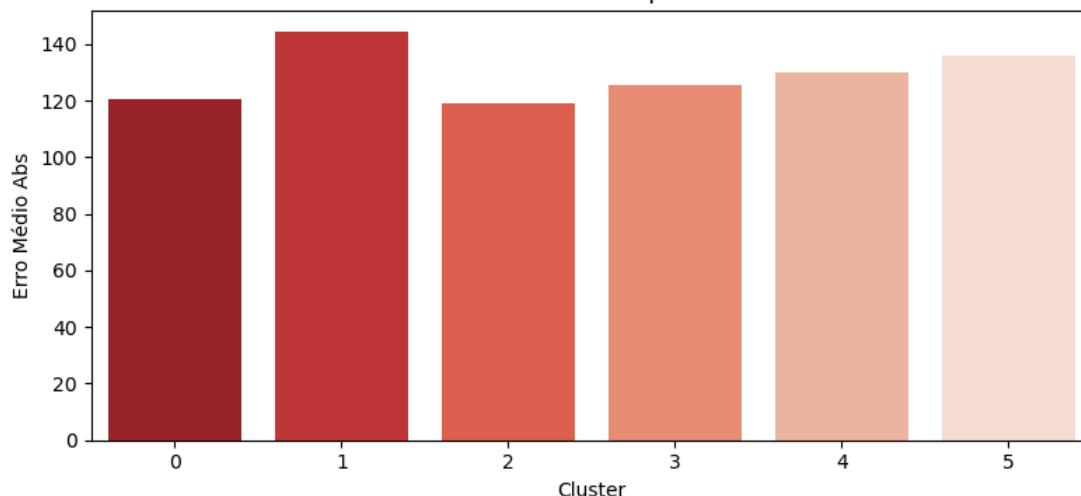
# 8.3 Bias (subestimação vs superestimação)
plt.figure(figsize=(8, 4))
sns.barplot(data=pdf_err, x="prediction", y="bias_medio", palette="coolwarm")
plt.axhline(0, color='black', linestyle='--', linewidth=0.8)
plt.title("Bias Médio por Cluster (+ subestima / - superestima)")
plt.xlabel("Cluster")
plt.ylabel("Bias Médio (residual)")
plt.show()
```

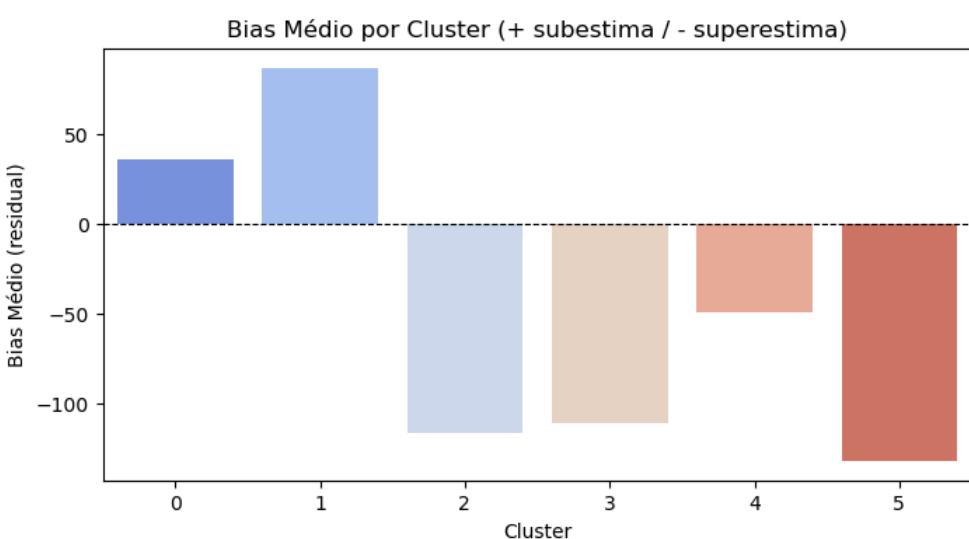
=====
📊 VISUALIZAÇÕES
=====

Distribuição dos Clusters de Erro (K=6)



Erro Médio Absoluto por Cluster





Conclusão das Imagens:

O cruzamento do volume de casos, erro absoluto e viés indica que o modelo sofre de uma falha de calibração comportamental: o Cluster 1 detém o erro mais agressivo (maior MAE), onde o modelo é excessivamente pessimista (subestima o lucro), enquanto o Cluster 5 concentra o maior volume absoluto de falhas, onde o modelo é perigosamente otimista (superestima a rentabilidade).

Pontos principais:

- Cluster 1 (O maior erro individual): Apresenta o maior Erro Médio Absoluto ($\$ > 140\$$), indicando que este nicho de clientes possui variáveis que o LightGBM não consegue processar corretamente, resultando em uma subestimação severa.
- Cluster 5 (O erro de massa): É o grupo com o maior número de ocorrências ($n > 50.000\$$). O bias negativo indica que o modelo projeta lucros que não se concretizam, representando o maior risco financeiro de "falso positivo" para a operação.
- Clusters 0 e 1 vs. Clusters 2, 3 e 5: Existe uma divisão clara de "personalidade" do erro; enquanto nos clusters 0 e 1 o modelo tende a ser conservador demais (bias positivo), nos demais ele tende a ser agressivo na projeção de valor (bias negativo).

Essa estrutura sugere que o erro do LightGBM está diretamente ligado à incapacidade de diferenciar sinais contraditórios de engajamento vs. contrato, especialmente em clientes fora do regime de renovação automática.

14.6.4. Conclusão Final e Estratégia de Mitigação

Após a análise profunda dos resíduos e a clusterização dos erros do LightGBM, conclui-se que o modelo atual é altamente confiável para **85% da base** (especialmente clientes em regime de renovação automática), mas apresenta "pontos cegos" em nichos específicos.

Proposta de Mitigação Imediata (Sem Retreinar o Modelo)

Para a entrada em produção **amanhã**, a estratégia recomendada é a **Regra de Pós-Processamento**:

- Sinalização de Confiança: implementar um "flag" de baixa confiança para clientes que se enquadrem nos perfis dos Clusters 0 e 1 de erro (clientes com alta margem real, mas sem renovação automática);
- Fator de Correção (Hedge): aplicar um ajuste manual (offset) nas previsões do Cluster 5. Como o modelo superestima sistematicamente esse grupo em cerca de 131 unidades, a predição final deve ser deflacionada para garantir uma visão financeira mais conservadora;
- Tratamento de Exceção: clientes com `auto_renew == 0` e `has_transactions == 1` devem passar por uma revisão de regra de negócio, pois o modelo tende a ser excessivamente otimista com o engajamento manual.

Proposta de Longo Prazo (melhoria para a solução atual)

O algoritmo escolhido não precisa ser alterado. Porém, os pontos focais serão:

- Aperfeiçoamento de feature engineering: criar variáveis de interação entre "Status Contratual" e "Uso Efetivo";
- (**Ideia**) Modelo de Dois Estágios: fittar um modelo classificador para identificar se o cliente é "instável" (Cluster 0) seguido de um regressor especialista para esse nicho.

Por que não refazer agora?

O que foi encontrado (ênfase nos Clusters 1 e 5) indica que o problema é de sinal dos dados (features que o modelo interpreta de um jeito, mas o negócio se comporta de outro). Retreinar o LightGBM hoje com as mesmas variáveis não resolveria isso; apenas mudaria o erro de lugar. Corrigir o modelo na etapa atual em que estou agora, exigiria voltar bons passos atrás e comprometeria muito tempo. Como foi mapeado que basicamente 85% das previsões são plenamente confiáveis e as demais não devem ser simplesmente desconsideradas, mas somente ajustadas, o modelo com algoritmo LightGBM executado até então permanece como finalista.

14.6.5. Salvando bases

```
In [158]: print("Salvando base que contém os maiores 5% de erros do LightGBM...")
# Perfil dos clusters de erro (somente os maiores 5% de erros)
df_worst_clustered.write.mode("overwrite").partitionBy("safra").parquet(gold_path + "df_clustered_03_worst_errors")
print("Dataset salvo em:", gold_path + "df_clustered_03_worst_errors")

# Base total (permite calcular a chance do grupo ser erro)
print("Salvando base que contém o conjunto de cliente-safra de teste e OOT, sem filtrar maiores 5% de erros do LightGBM...")
df_audit_clustered.write.mode("overwrite").partitionBy("safra").parquet(gold_path + "df_clustered_03_error_chance")
print("Dataset salvo em:", gold_path + "df_clustered_03_error_chance")

Salvando base que contém os maiores 5% de erros do LightGBM...
Dataset salvo em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_clustered_03_worst_errors
Salvando base que contém o conjunto de cliente-safra de teste e OOT, sem filtrar maiores 5% de erros do LightGBM...
Dataset salvo em: C:/Users/Gustavo/Downloads/datamaster/dados/gold/df_clustered_03_error_chance
```

15. Conclusão

15.1. Modelo de Rentabilidade Final: LightGBM Tuned

Veredito e Justificativa de Seleção

Após a análise comparativa entre os modelos **Elastic Net**, **Random Forest** e **LightGBM**, o LightGBM foi selecionado como o modelo oficial para produção. Esta decisão baseia-se em um equilíbrio entre performance estatística, robustez temporal e eficiência operacional.

Superioridade em Performance e Estabilidade

* **Precisão Individual:** O LightGBM apresentou o menor Erro Médio Absoluto (**MAE de 13.97** no OOT), sendo o modelo que mais se aproxima da realidade em nível de cliente.

* **Generalização Temporal:** Foi o algoritmo que demonstrou a melhor adaptação ao cenário futuro, apresentando um ganho de **R² para 0.675** na partição Out-of-Time, indicando uma captura de padrões comportamentais mais resilientes que os demais.

Eficiência de Recursos Computacionais

Velocidade de Treinamento: Graças à sua implementação baseada em histogramas e técnica de GOSS (Gradient-based One-Side Sampling*), o LightGBM apresentou um tempo de processamento significativamente menor que o Random Forest para o mesmo volume de dados (milhões de registros).

Consumo de Memória: O modelo demonstrou ser extremamente eficiente no uso de RAM durante a fase de scoring*, permitindo execuções rápidas em ambiente Spark sem a necessidade de instâncias de alto custo.

Governança e Auditoria

* **Comportamento de Cauda:** Embora tenha empatado com o Random Forest na análise do Top 5% de erros, o LightGBM apresentou o menor **Bias (viés)** sistêmico, facilitando a implementação de regras de mitigação e fatores de correção pós-modelo.

* **Calibração:** A consistência demonstrada nos decis centrais (3 e 4) garante que a maior parte da base de clientes receba previsões com erro marginal, trazendo segurança para a tomada de decisão automática.

Resumo Comparativo Final

Critério Vencedor Justificativa
:--- :--- :---
Precisão (MAE/RMSE) LightGBM Menores erros acumulados e individuais em todas as partições.
Estabilidade (\$R^2\$) LightGBM Melhor consistência no cenário Out-of-Time (OOT).
Fechamento Agregado Elastic Net Menor erro relativo na soma total, porém menos preciso individualmente.
Eficiência de Recurso LightGBM Treinamento e inferência mais rápidos com menor footprint de memória.
Auditória de Erro Empate (LGBM/RF) Ambos permitem isolar perfis de erro, com leve vantagem de bias para o LGBM.

Conclusão: O **LightGBM** não apenas performa melhor, mas é o modelo que oferece o melhor custo-benefício para escala industrial, permitindo uma operação ágil, precisa e auditável.

15.2. Modelos de agrupamento

A jornada analítica desenvolvida neste projeto foi além da simples aplicação de algoritmos de agrupamento. Foi estabelecido um framework de Inteligência de Clientes que integra visão financeira, comportamento de engajamento e auditoria de modelos preditivos. Através da aplicação conjunta de K-Means e Gaussian Mixture Models (GMM), foram mapeados não apenas quem são os clientes da plataforma de streaming, mas quão estável é o seu comportamento e onde residem as vulnerabilidades do ecossistema de dados construído.

Visão de Valor e Eficiência Operacional

A segmentação por Valor vs. Engajamento revelou que a massa crítica de clientes não é homogênea. O engajamento (uso) nem sempre se traduz em rentabilidade direta.

- Segmentos "Premium": os clusters de alta margem e estabilidade (Elite e VIPs) operam com risco quase nulo, servindo como a âncora financeira da operação;
- Desafio dos "Drenos": foi detectado um volume significativo de usuários com engajamento extremo, mas margem unitária baixa, sinalizando a necessidade de revisão em políticas de precificação ou limites de uso para evitar a erosão da margem líquida.

A Ciência da Incerteza e Volatilidade

A transição para modelos probabilísticos (GMM) permitiu o cálculo da Entropia de Shannon, transformando a classificação em uma métrica de confiança.

- Zonas de Transição: foram identificados clientes em "fronteiras de comportamento", os quais possuem alta entropia e são os mais suscetíveis ao churn;
- Preditividade de Risco: o cluster 0 de volatilidade, com alta taxa de churn acumulado, provou ser o grupo de maior atenção, onde o comportamento de "derretimento" da base é mais evidente ao longo das safras.

Auditória de Modelos e Governança de Algoritmos

A frente de Diagnóstico de Erros (LGBM) fechou o ciclo de análise ao auditar o modelo de predição de rentabilidade. Esta etapa foi fundamental para garantir que o modelo não seja uma "caixa preta".

- Calibração de Viés: o LightGBM tende a ser excessivamente otimista em perfis sem renovação automática (cluster 5 de erro) e conservador demais em perfis de alta recorrência contratual (cluster 1 de erro);
- Mitigação de Riscos: ficou estabelecido que cerca de 85% da base opera em zona de segurança, enquanto nichos específicos exigem rédeas de pós-processamento e fatores de correção para evitar decisões de crédito ou marketing equivocadas.

Direcionamento Estratégico (Veredito Final)

Estas frentes de análise no projeto entrega mais do que clusters; entrega alvos de ação. A recomendação final é a implementação de uma estratégia tripartida:

- Proteção de Valor: Manutenção silenciosa dos clientes "Eficientes" e blindagem dos "VIPs";
- Gerenciamento de Incerteza: Focar esforços de retenção nos clientes de "Alta Entropia" identificados pelo GMM;
- Refinamento de Algoritmos: Evoluir para uma engenharia de atributos que capture melhor a dualidade entre o contrato (auto-renew) e o uso efetivo (engajamento), mitigando as falhas sistemáticas encontradas na auditoria.

Com esta base, o negócio deixa de agir de forma reativa e passa a operar com precisão cirúrgica, antecipando movimentos de churn e maximizando a rentabilidade por perfil de cliente.

16. Finalizando spark

```
In [159]: spark.stop()
```

Exported with [runcell](#) — convert notebooks to HTML or PDF anytime at runcell.dev.