**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**
**NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**
**Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data**

**Mariana Silva Costa**

# CATEGORIZAÇÃO DE PRODUTOS EM MARKETPLACES

Belo Horizonte
2023

**Mariana Silva Costa**

# CATEGORIZAÇÃO DE PRODUTOS EM MARKETPLACES

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2023

**SUMÁRIO**

# 1. Introdução

## 1.1. Contextualização

A categorização de produtos em *marketplaces* é fundamental para organizar e estruturar a vasta quantidade de produtos disponíveis. Esse processo é importante para melhorar a experiência do usuário, facilitar a navegação, aumentar a descoberta de produtos, melhorar os resultados de pesquisa e possibilitar a comparação e filtragem. Ao organizar os produtos de forma eficiente, os marketplaces podem oferecer uma experiência de compra mais agradável e aumentar as chances de sucesso tanto para os clientes quanto para os vendedores.

## 1.2. O Problema Proposto

O objetivo da categorização de produtos com uso de *machine learning* é automatizar e aprimorar o processo de atribuição de categorias aos produtos em um *marketplace*. Em vez de depender exclusivamente de classificações manuais feitas por especialistas humanos, o *machine learning* utiliza algoritmos e técnicas para analisar dados e aprender padrões a fim de atribuir categorias de forma mais eficiente e precisa.

As informações utilizadas neste projeto foram coletadas da plataforma *Kaggle* - uma comunidade voltada para ciência de dados, que apresenta tutoriais, competições, rankings, cursos, dicas, fóruns, *datasets* entre outros conteúdos. O material escolhido foi o conjunto de dados públicos de comércio eletrônico brasileiro de pedidos feitos na *Olist Store* - um canal de vendas digital que está presente nos maiores marketplaces do país. Esse material contém informações de 100 mil pedidos de 2016 a 2018 feitos em vários marketplaces no Brasil.

Nesse trabalho serão analisados os *datasets* com detalhes sobre os produtos vendidos pela *Olist,* com o intuito de criar um modelo de *machine learning* capaz de categorizá-los por meio das informações disponíveis no conjunto de dados.

### 1.3. Linguagem de Programação e Ambiente Computacional

Esse projeto foi elaborado utilizando a linguagem de programação *Python*, na versão 3.8.8, no ambiente de desenvolvimento *Jupyter Notebook,* na versão 6.3.0 (figura 1).

**Figura 1 –** Versão do *Python* e do *Jupyter Notebook*


Foram importadas algumas bibliotecas básicas, além das nativas do *Python*, para coleta, análise, processamento e tratamento dos dados. Para criação dos modelos de ML foi utilizada a biblioteca *scikit-learn*. Para otimização bayesiana de hiper parâmetros foi utilizada a biblioteca *scikit-optimize*. As importações de todas elas são apresentadas na figura 2.

**Figura 2 –** Bibliotecas Python importadas

```
# Importando bibliotecas básicas
import pandas as pd   #processamento de dados
from ydata_profiling import ProfileReport #análise de dados
import numpy as np   #álgebra linear
import seaborn as sns #visualização baseada em matplotlib
sns.set_style("white") #gráficos sem grade
import matplotlib.pyplot as plt #biblioteca base de visualização
n = '\033[1m' #codificação de fonte em negrito
r = '\033[22m' #redefenir a formatação em negrito
```

```
# Importando a biblioteca scikit-learn e xgboost
# Algoritimos de aprendizado de máquina, pré-processamento, seleção de modelos, métricas de desempenho, ...
from sklearn.model_selection import KFold, cross_validate, cross_val_score, train_test_split, GridSearchCV
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, recall_score
from sklearn.metrics import precision_score, f1_score, make_scorer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
import xgboost as xgb
```

```
# Importando a biblioteca scikit-optimize
# Utilizada para Otimização Bayesiana de hiper parâmetros
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer
```

## 2. Coleta de Dados

### 2.1. Conjunto de Dados

O material escolhido para elaboração deste trabalho foi o conjunto de dados públicos de comércio eletrônico brasileiro da Olist Store, baixado do site da kaggle no dia 17/04/2023, pelo link (https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce).

O conjunto de dados contém informações de 100 mil pedidos de 2016 a 2018 feitos em vários marketplaces no Brasil. Seus recursos permitem visualizar um pedido de várias dimensões: desde o status do pedido, preço, pagamento e desempenho do frete até a localização do cliente, atributos do produto e, finalmente, avaliações escritas pelos clientes. Também lançamos um conjunto de dados de geolocalização que relaciona os códigos postais brasileiros às coordenadas lat/lng.

Estes são dados comerciais reais, que foram anonimizados e as referências às empresas e parceiros no texto da revisão foram substituídas por códigos. Ao todo são nove datasets descritos a seguir:

i. *Customers Dataset* - Conjunto de dados de clientes

Este conjunto de dados contém informações sobre o cliente e sua localização. Usado para identificar clientes únicos no conjunto de dados de pedidos e para encontrar o local de entrega dos pedidos. No sistema da Olist, cada pedido é atribuído a um customer_id exclusivo. Isso significa que o mesmo cliente receberá IDs diferentes para pedidos diferentes. O objetivo de ter um customer_unique_id no conjunto de dados é permitir a identificação de clientes que fizeram recompras na loja.

ii. *Geolocation Dataset - Conjunto de dados de geolocalização*

Este conjunto de dados contém informações sobre códigos postais brasileiros e suas coordenadas lat/lng. Usado para traçar mapas e encontrar distâncias entre vendedores e clientes.

iii. *Order Items Dataset - Conjunto de dados de itens de pedido*

Este conjunto de dados inclui dados sobre os itens comprados em cada pedido. Exemplo: O *order_id* = 00143d0f86d6fbd9f9b38ab440ac16f5 possui 3 itens (mesmo produ-

to). Cada item tem o frete calculado de acordo com suas medidas e peso. Para obter o valor total do frete de cada pedido basta somar.

iv. *Payments Dataset - Conjunto de dados de pagamentos*

Este conjunto de dados inclui dados sobre as opções de pagamento dos pedidos.

v. *Order Reviews Dataset - Conjunto de dados de avaliações de pedidos*

Este conjunto de dados inclui dados sobre as avaliações feitas pelos clientes. Depois que um cliente compra o produto da *Olist Store*, um vendedor é notificado para atender a esse pedido. Assim que o cliente recebe o produto, ou a data prevista de entrega está prevista, o cliente recebe um inquérito de satisfação por e-mail onde pode dar nota da experiência de compra e deixar alguns comentários.

vi. *Order Dataset - Conjunto de dados do pedido*

Este é o conjunto de dados principal. De cada pedido pode-se encontrar todas as outras informações.

vii. *Products Dataset - Conjunto de dados de produtos*

Este conjunto de dados inclui dados sobre os produtos vendidos pela *Olist*.

viii. *Sellers Dataset - Conjunto de dados de vendedores*

Este conjunto de dados inclui dados sobre os vendedores que atenderam aos pedidos feitos na *Olist*. Usado para encontrar a localização do vendedor e identificar qual vendedor entregou cada produto.

ix. *Category Name Translation - Tradução do nome da categoria*

Traduz o *product_category_name* para inglês.

### 2.2. Escolha dos Datasets

Os nove datasets da Olist disponíveis na página da kaggle foram importados no Python e tiveram seus atributos identificados. Porém, como o objetivo desse trabalho é categorizar os produtos, optou-se pela utilização de apenas dois deles, os que possuem informações relevantes sobre os produtos em si, Products Dataset e Order Items Dataset. A tabela abaixo apresenta todas as colunas desses dois datasets, com descrição e tipo de cada uma, bem como a informação se o atributo será aproveitado ou não no estudo.

**Tabela 1:** Descrição dos campos/colunas dos *datasets*

| Dataset | Nome da coluna | Descrição | Tipo | Aproveitar Atributo |
|---|---|---|---|---|
| **products** | **product_id** | **id do pedido** | **object** | **Sim** |
| products | product_category_name | categoria do produto | object | Sim |
| products | product_name_lenght | comprimento do nome | float64 | Não |
| products | product_description_lenght | comprimento da descrição | float64 | Não |
| products | product_photos_qty | quantidade de fotos | float64 | Não |
| products | product_weight_g | peso do produto em gramas | float64 | Sim |
| products | product_length_cm | comprimento em cm | float64 | Sim |
| products | product_height_cm | altura em cm | float64 | Sim |
| products | product_width_cm | largura em cm | float64 | Sim |
| items | order_id | id do pedido por produto | object | Não |
| items | order_item_id | Qtde de produtos iguais | int64 | Não |
| **items** | **product_id** | **id do pedido** | **object** | **Sim** |
| items | seller_id | id do vendedor | object | Não |
| items | shipping_limit_date | data limite de envio do pedido | object | Não |
| items | price | preço unitário do produto | float64 | Sim |
| items | freight_value | valor unitário do frete | float64 | Não |

## 2.3. Importação e Leitura dos Dados

Os *datasets* foram baixados da plataforma *Kaggle*, importados para o projeto no Python, e já criado um *dataframe* compilado (df) com o *Products Dataset* (products_df) e *Order Items Dataset* (*items_df*), através da função *merge(),* pela coluna '*product_id*', como mostra a figura 3.

**Figura 3 –** Importação e compilação dos *datasets*

```
# Importando os datasets de produtos e itens da Olist
path = 'C://Users//user//Desktop//Pos_Ciencia_de_Dados\//TCC//Projeto'
products_df = pd.read_csv(f'{path}//olist_products_dataset.csv')
items_df = pd.read_csv(f'{path}//olist_order_items_dataset.csv')
```

```
# Criando um dataframe compilado
df = products_df
df = pd.merge(df, items_df,  how='left', on='product_id')
df.head()
```

As primeiras linhas do *dataframe* compilado podem ser observadas na figura abaixo, gerada a partir da função *head()* chamada na figura 3.

**Figura 4 –** Primeiras linhas do *dafatrame* compilado

| | product_id | product_category_name | product_name_lenght | product_description_lenght | product_photos_qty |
|---|---|---|---|---|---|
| 0 | 1e9e8ef04dbcff4541ed26657ea517e5 | perfumaria | 40.0 | 287.0 | 1.0 |
| 1 | 3aa071139cb16b67ca9e5dea641aaa2f | artes | 44.0 | 276.0 | 1.0 |
| 2 | 96bd76ec8810374ed1b65e291975717f | esporte_lazer | 46.0 | 250.0 | 1.0 |
| 3 | cef67bcfe19066a932b7673e239eb23d | bebes | 27.0 | 261.0 | 1.0 |
| 4 | 9dc1a7de274444849c219cff195d0b71 | utilidades_domesticas | 37.0 | 402.0 | 4.0 |

| product_weight_g | product_length_cm | product_height_cm | product_width_cm | order_id | order_item_id |
|---|---|---|---|---|---|
| 225.0 | 16.0 | 10.0 | 14.0 | e17e4f88e31525f7deef66779844ddce | 1 |
| 1000.0 | 30.0 | 18.0 | 20.0 | 5236307716393b7114b53ee991f36956 | 1 |
| 154.0 | 18.0 | 9.0 | 15.0 | 01f66e58769f84129811d43eefd187fb | 1 |
| 371.0 | 26.0 | 4.0 | 26.0 | 143d00a4f2dde4e0364ee1821577adb3 | 1 |
| 625.0 | 20.0 | 17.0 | 13.0 | 86cafb8794cb99a9b1b77fc8e48fbbbb | 1 |

| seller_id | shipping_limit_date | price | freight_value |
|---|---|---|---|
| 5670f4db5b62c43d542e1b2d56b0cf7c | 2018-04-30 17:33:54 | 10.91 | 7.39 |
| b561927807645834b59ef0d16ba55a24 | 2018-02-06 19:11:15 | 248.00 | 17.99 |
| 7b07b3c7487f0ea825fc6df75abd658b | 2018-07-11 21:30:20 | 79.80 | 7.82 |
| c510bc1718f0f2961eaa42a23330681a | 2018-08-07 09:10:13 | 112.30 | 9.54 |
| 0be8ff43f22e456b4e0371b2245e4d01 | 2018-04-17 01:30:23 | 37.90 | 8.29 |

## 3. Processamento/Tratamento de Dados

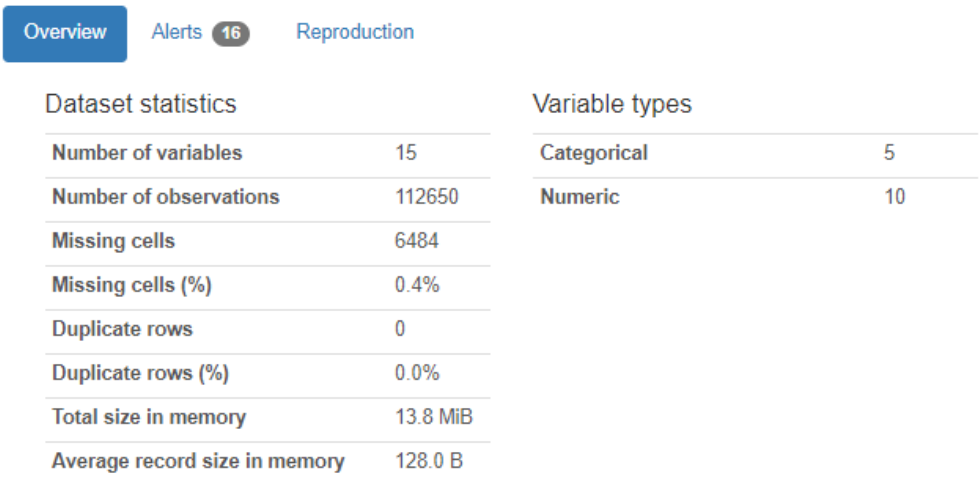
### 3.1 Análise Exploratória Pré-Processamento de Dados

Antes de se iniciar qualquer tratamento nos dados é importante realizar uma análise exploratória dos dados, para se identificar que tipos de tratamentos serão necessários, tendo em vista a base de dados com a qual se está trabalhando, bem como os objetivos que se espera alcançar.

Existem diversas funções que revelam informações úteis para essa etapa, como *shape(), display(), info(), describe(),* entre outras. Mas, nesse trabalho optou-se por gerar um relatório de análise de dados com o *Pandas Profiling Report*, que entrega uma visão geral completa do conjunto de dados.

Observou-se então, com o relatório gerado pelo ProfileReport(), que o *dataframe* compilado (df) possui 15 variáveis/colunas, sendo 5 categóricas e 10 numéricas, 112.650 linhas com nenhuma linha duplicada e 6.484 células ausentes, representando apenas 0,4% do total de células da base. Essas informações podem ser vistas na figura 5.

**Figura 5 –** Visão geral do *dataframe* (df)




Para complementar a visão geral do *Profiling Report* foi chamada a propriedade *DataFrame.dtypes*, para se identificar os tipos de dados de cada coluna e os métodos *isnull*() e, sum() agregados, para soma dos valores nulos dessas colunas. Como pode ser visto na figura abaixo, e concordando com o relatório analítico já extraído, há 5 variáveis categóricas (*object*) contra 10 numéricas (*float* ou *int*) e uma quantidade muito pequena de valores nulos na base, considerando a quantidade total de registros disponível.

**Figura 6 –** Tipos de dados e valores nulos do *dataframe* (df)

```
# Visualizando os tipos de colunas
df.dtypes
```

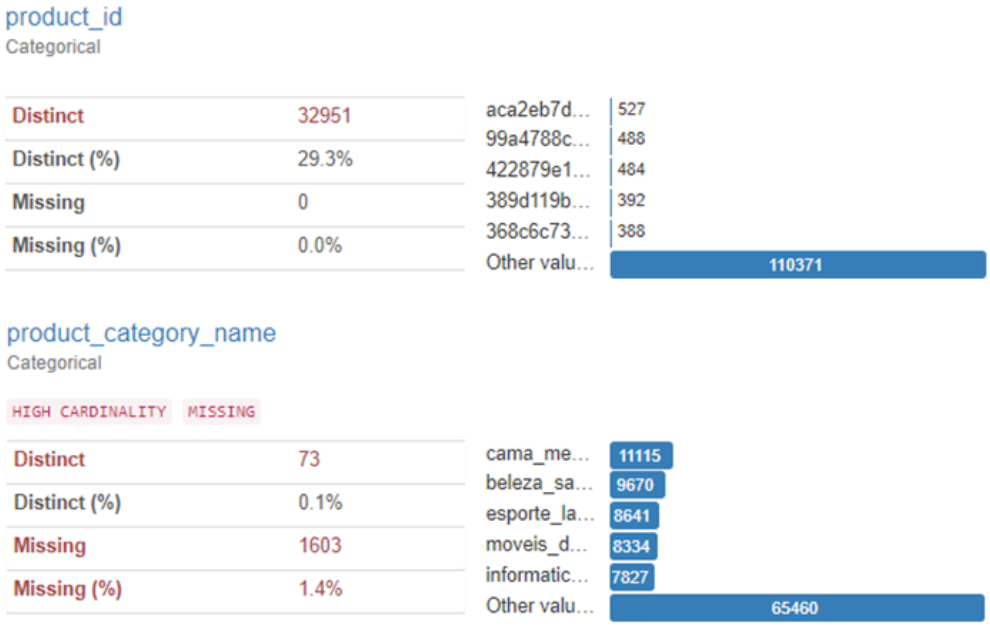
```
product_id                     object
product_category_name          object
product_name_lenght           float64
product_description_lenght    float64
product_photos_qty            float64
product_weight_g              float64
product_length_cm             float64
product_height_cm             float64
product_width_cm              float64
order_id                       object
order_item_id                   int64
seller_id                      object
shipping_limit_date            object
price                         float64
freight_value                 float64
dtype: object
```

```
# Calculando os valores ausentes por colunas
df.isnull().sum()
```

```
product_id                        0
product_category_name          1603
product_name_lenght            1603
product_description_lenght     1603
product_photos_qty             1603
product_weight_g                 18
product_length_cm                18
product_height_cm                18
product_width_cm                 18
order_id                          0
order_item_id                     0
seller_id                         0
shipping_limit_date               0
price                             0
freight_value                     0
dtype: int64
```

Voltando para o relatório extraído foi possível identificar (figura 7) que a base possui ao todo 32.951 produtos distintos, distribuídos em 73 categorias.

**Figura 7 –** Informações sobre as variáveis produto e categoria




## 3.2 Tratamento do Conjunto de Dados

Nesta etapa, os dados já coletados e analisados passam por um tratamento. Isso pode incluir a limpeza dos dados, tratamento de valores ausentes, normalização de variáveis e outras técnicas de pré-processamento.

Tendo analisado previamente o conjunto de dados pôde se iniciar a etapa de tratamento dos dados. A primeira ação executada foi a de converter a unidade da variável peso de grama (g) para quilograma (Kg) (figura 8), pois é a unidade padrão de massa no Sistema Internacional de unidades (SI) e tem uma melhor visualização nesse conjunto de dados.

**Figura 8 –** Conversão de unidades

```
# Convertendo a unidade de peso de grama para quilo
df['product_weight_g'] = df['product_weight_g'].apply(lambda x: x/1000)
df = df.rename(columns={'product_weight_g': 'product_weight_kg'})
```

O objetivo do projeto é classificar os produtos da base na categoria correta. Para isso, é necessário que os códigos (chave primária) tenham valores exclusivos em toda a base de dados e que, todas as variáveis estejam com seus valores preenchidos corretamente. Sendo assim, a segunda ação realizada foi a de eliminar os elementos duplicados da coluna *product_id* e de eliminar as linhas que contém elementos nulos, como mostrado na figura 9.

**Figura 9 –** Eliminação de linhas duplicadas e com valores nulos

```
# Eliminando duplicadas da coluna chave (product_id)
df = df.drop_duplicates('product_id')
# Eliminando linhas que contém elementos nulos
df = df.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32340 entries, 0 to 112649
Data columns (total 15 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   product_id                  32340 non-null  object
 1   product_category_name       32340 non-null  object
 2   product_name_lenght         32340 non-null  float64
 3   product_description_lenght  32340 non-null  float64
 4   product_photos_qty          32340 non-null  float64
 5   product_weight_g            32340 non-null  float64
 6   product_length_cm           32340 non-null  float64
 7   product_height_cm           32340 non-null  float64
 8   product_width_cm            32340 non-null  float64
 9   order_id                    32340 non-null  object
 10  order_item_id               32340 non-null  int64
 11  seller_id                   32340 non-null  object
 12  shipping_limit_date         32340 non-null  object
 13  price                       32340 non-null  float64
 14  freight_value               32340 non-null  float64
dtypes: float64(9), int64(1), object(5)
```

Trabalhar com 73 categorias de uma só vez pode ser extremamente complexo. Então optou-se por categorizar os produtos de apenas duas categorias. Para isso identificou-se as que mais produtos possuem, sendo cama_mesa_banho, esporte_lazer, moveis_decoracao, beleza_saude e utilidades_domesticas. A quantidade de produtos de cada uma delas pode ser vista na figura 10.

**Figura 10 –** Quantidade de produtos por categoria

```
# Indentificando a quantidade de produtos em cada categoria
df.product_category_name.value_counts()
```

```
cama_mesa_banho           3029
esporte_lazer             2867
moveis_decoracao          2657
beleza_saude              2444
utilidades_domesticas     2335
                          ...
```

As duas categorias escolhidas para a análise foram cama_mesa_banho e beleza_saude. Essa decisão se deu pelo fato de as variáveis disponíveis na base apresentarem peso, dimensões e preço dos produtos. Logo, entendeu-se que essas duas categorias seriam suficientemente distintas para serem categorizadas pela observação dessas variáveis.

**Figura 11 –** Criação de um *dataframe* enxuto

```
# Criando um dataframe enxuto
df1 = df
df1.drop(columns=['product_name_lenght', 'product_description_lenght', 'product_photos_qty', 'order_id', 'order_item_id', 'seller_id', 'shipping_limit_date', 'freight_value'], inplace=True)

# Filtrando apenas 2 categorias de produtos
df1 = df1[(df1['product_category_name'] == 'cama_mesa_banho') | (df1['product_category_name'] == 'beleza_saude')]

# Transformando a coluna de categoria em tipo numérico
df1 = df1.replace({'cama_mesa_banho':0, 'beleza_saude':1})

# Simplificando os nomes das colunas no português
df1.rename(columns={'product_category_name':'categoria', 'product_weight_kg':'peso_kg', 'product_length_cm':'comprimento_cm', 'product_height_cm':'altura_cm', 'product_width_cm':'largura_cm', 'price':'preco'}, inplace=True)
display(df1)
```

| | product_id | categoria | peso_kg | comprimento_cm | altura_cm | largura_cm | preco |
|---|---|---|---|---|---|---|---|
| 19 | 14aa47b7fe5c25522b47b4b29c98dcb9 | 0 | 1.100 | 16.0 | 10.0 | 16.0 | 71.99 |
| 34 | f53103a77d9cf245e579ea37e5ec51f0 | 0 | 0.500 | 16.0 | 10.0 | 16.0 | 41.99 |
| 39 | 518ef5de2c2b3a255e326a4594ba15d9 | 0 | 0.800 | 36.0 | 8.0 | 16.0 | 145.00 |
| 40 | e3e020af31d4d89d2602272b315c3f6e | 1 | 0.075 | 21.0 | 7.0 | 13.0 | 29.90 |
| 60 | 47859fca9dd7485cbd93c3e8993bb74f | 0 | 0.650 | 16.0 | 10.0 | 16.0 | 41.99 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 112531 | c1cf541d5b33a4b04ddc1c3be7aa1c86 | 1 | 0.150 | 35.0 | 2.0 | 26.0 | 38.00 |
| 112547 | 5bde14b0ba42a788655c3ebf4ba54597 | 0 | 1.500 | 31.0 | 21.0 | 26.0 | 75.61 |
| 112574 | dfec64aac9b864b2807a7be33222b75f | 0 | 0.850 | 38.0 | 7.0 | 28.0 | 84.90 |
| 112637 | 9a7c6041fa9592d9d9ef6cfe62a71f8c | 0 | 1.400 | 27.0 | 7.0 | 27.0 | 127.00 |
| 112649 | 106392145fca363410d287a815be6de4 | 0 | 2.083 | 12.0 | 2.0 | 7.0 | 107.50 |

5473 rows × 7 columns

As próximas etapas de transformação dos dados podem ser vistas na figura acima, onde:

i. Foi criado o *dataframe* df1 apenas com as variáveis código do produto (*product_id*), categoria, peso em Kg, dimensões do produto (comprimento, altura e largura, todas em cm, devido ao provável tamanho dos produtos - em metros não ficaria boa a visualização) e preço. As demais variáveis não interessavam ao objetivo desse trabalho, então foram dropadas (eliminadas) através da função *drop().*

ii. Nesse novo *dataframe* foram filtradas apenas as duas categorias escolhidas para se trabalhar.

iii. A coluna categoria, que é um atributo categórico foi transformada em variável *dummy*, com 0 (zero) para cama, mesa e banho, e 1 (um) para beleza e saúde. Para isto foi utilizada a função *replace().*

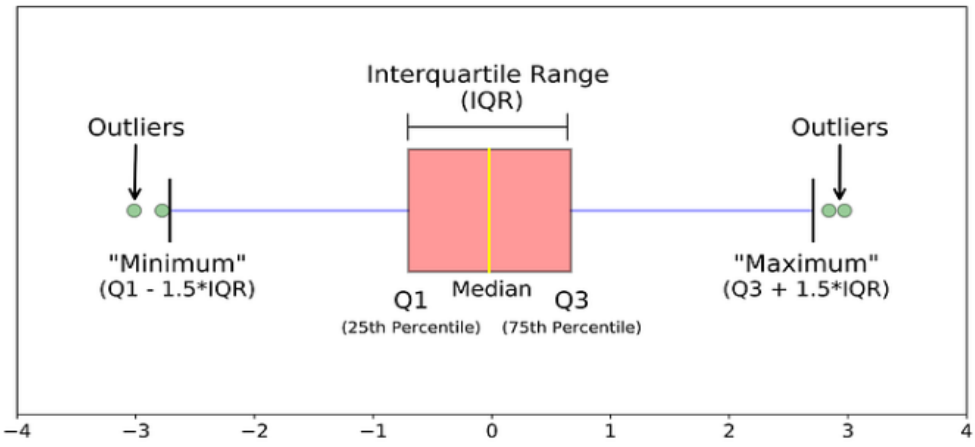
iv. Os nomes das colunas foram simplificados e traduzidos para o português com a ajuda da função *rename().*

As 5 primeiras e 5 últimas linhas desse novo *dataframe* (df1) foram exibidas através do comando display(df1).

Prosseguindo com o tratamento dos dados, visualizou-se a distribuição dos dados das variáveis por meio de diagramas de caixa (*Boxplots*).

Com o *Boxplot* é possível identificar 5 medidas estatísticas: o mínimo, o primeiro quartil (Q1), a mediana, o terceiro quartil (Q3) e o máximo. Ele também acaba informando a presença de valores discrepantes (outliers) dos dados, como bem apresentado na figura abaixo. Importante lembrar que os quartis dividem o conjunto de dados em quatro, cada um contendo 25% (um quarto) dos dados, sendo o valor central a mediana ou segundo quartil. Outro conceito relevante sobre a análise dos *boxplots* é a amplitude ou intervalo interquartil (IQR), calculado pela distância entre o primeiro quartil (Q1) e o terceiro quartil (Q3), onde localizam-se 50% dos dados.

**Figura 12 –** Elementos que compõem um *Boxplot*

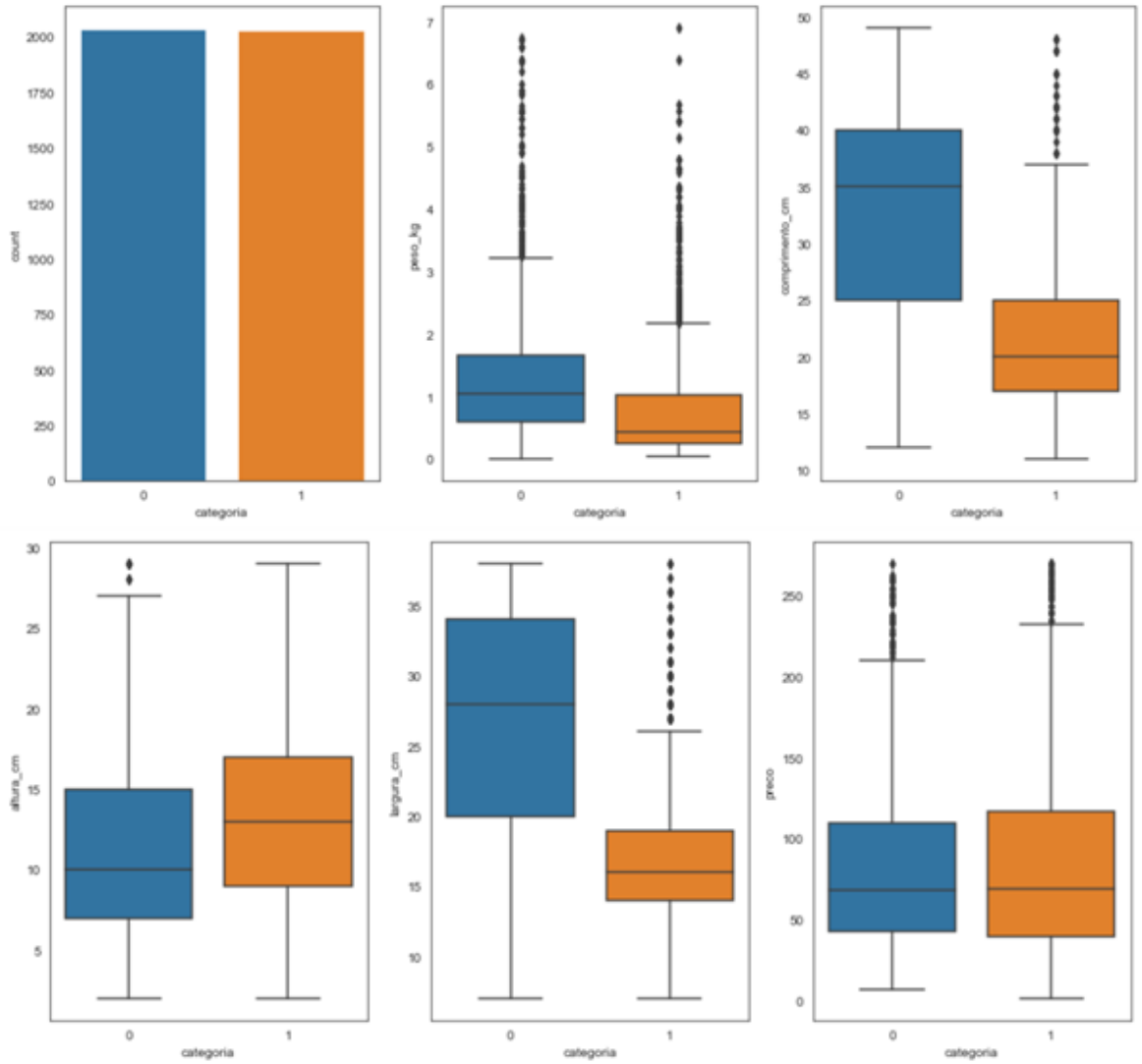



Fonte: Fernando Matsumoto – Medium.com

**Figura 13 –** *Boxplot* das variáveis

```
# Plotando as variáveis
fig = plt.figure(figsize=(15, 15))
# Gráfico 1 - Barras de contagem
fig.add_subplot(231)
sns.countplot(df1['categoria'])
# Gráfico 2 - Boxplot de peso
fig.add_subplot(232)
sns.boxplot(data=df1, x='categoria', y='peso_kg')
# Gráfico 2 - Boxplot de comprimento
fig.add_subplot(233)
sns.boxplot(data=df1, x='categoria', y='comprimento_cm')
# Gráfico 3 - Boxplot de altura
fig.add_subplot(234)
sns.boxplot(data=df1, x='categoria', y='altura_cm')
# Gráfico 4 - Boxplot de largura
fig.add_subplot(235)
sns.boxplot(data=df1, x='categoria', y='largura_cm')
# Gráfico 4 - Boxplot de preço
fig.add_subplot(236)
sns.boxplot(data=df1, x='categoria', y='preco')
print(f'{n}Boxplot das variáveis analisadas')
plt.show()
```

```
Boxplot das variáveis analisadas
```



Sendo assim, podemos concluir, pelo *boxplot* da figura 13, que, neste estudo, comprimento e largura são boas variáveis para agrupar os produtos dentro de cada categoria, pois pelo menos 50% dos produtos se diferem bastante nesses quesitos. Porém, apenas essas duas variáveis não devem ser suficientes para uma perfeita categorização, tendo em vista que muitos produtos se enquadram no mesmo intervalo dessas dimensões.

As categorias abrangem muitos tipos de produtos. Cama mesa e banho pode contemplar *sousplats*, toalhas ou edredons, por exemplos. Enquanto beleza e saúde podem englobar batons ou kits completos de tratamento capilar. Ou seja, há uma considerável variação de dimensões, peso e preço dentro de cada categoria e as duas acabam possuindo muitas interseções. Além disso, há muitos dados discrepantes por todos os atributos, para as duas classes.

Para tratar desses outliers, foi utilizada a função *dataframe.quantile(), retornando* valores no quartil 95 para todas as variáveis independentes.

**Figura 14 –** Remoção de outliers

```
# Removendo outliers das variáveis
peso_q95 = df1['peso_kg'].quantile(0.95)
df1 = df1[df1['peso_kg'] < peso_q95]
comprimento_q95 = df1['comprimento_cm'].quantile(0.95)
df1 = df1[df1['comprimento_cm'] < comprimento_q95]
altura_q95 = df1['altura_cm'].quantile(0.95)
df1 = df1[df1['altura_cm'] < altura_q95]
largura_q95 = df1['largura_cm'].quantile(0.95)
df1 = df1[df1['largura_cm'] < largura_q95]
preco_q95 = df1['preco'].quantile(0.95)
df1 = df1[df1['preco'] < preco_q95]

# Verificando como ficou o dataframe
df1.info()
df1.categoria.value_counts()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4062 entries, 19 to 112649
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   product_id      4062 non-null   object
 1   categoria       4062 non-null   int64
 2   peso_kg         4062 non-null   float64
 3   comprimento_cm  4062 non-null   float64
 4   altura_cm       4062 non-null   float64
 5   largura_cm      4062 non-null   float64
 6   preco           4062 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 253.9+ KB

0    2035
1    2027
Name: categoria, dtype: int64
```

Em seguida, foi feita a normalização/redimensionamento dos dados com utilização da técnica *RobustScaler*, que estima a mediana e o intervalo interquartil (IQR) dos dados e, em seguida, dimensiona os dados subtraindo a mediana e dividindo pelo IQR. Nesse modelo os outliers ainda estão presentes no conjunto de dados, porém estão representados dentro de uma escala em que o seu impacto negativo é reduzido.

A normalização é uma técnica usada para pré-processar os dados, ajustando-os para que eles estejam em uma escala comum, geralmente entre 0 e 1. Isso é feito para evitar que certas variáveis tenham um peso excessivo sobre o algoritmo de aprendizado de máquina apenas por causa de suas magnitudes originais.

O RobustScaler permite que os dados sejam ajustados em uma escala comum, tornando-os adequados para análise e modelagem usando algoritmos de aprendizado de máquina. Além disso, o *RobustScaler* lida de forma robusta com *outliers*, preservando a distribuição original dos dados. Dessa forma, ele é menos sensível a valores discrepantes do que outros métodos de normalização, como o Z-score.

A construção do processo de normalização dos dados, bem como as informações estatísticas descritivas das colunas normalizadas estão apresentadas na figura abaixo.

**Figura 15 –** Normalização dos dados com *RobustScaler*

```
# Criando um dataframe apenas com as colunas numéricas e categoria
numeric_df = df1.drop(['product_id'],axis=1)
# Criando o scaler
scaler = RobustScaler()
# Fazendo o fit com os dados
scaler = scaler.fit(numeric_df[['peso_kg']])
scaler = scaler.fit(numeric_df[['comprimento_cm']])
scaler = scaler.fit(numeric_df[['altura_cm']])
scaler = scaler.fit(numeric_df[['largura_cm']])
scaler = scaler.fit(numeric_df[['preco']])
# Fazendo a transformação e criando columns_robust
numeric_df['peso_kg_robust'] = scaler.transform(numeric_df[['peso_kg']])
numeric_df['comprimento_cm_robust'] = scaler.transform(numeric_df[['comprimento_cm']])
numeric_df['altura_cm_robust'] = scaler.transform(numeric_df[['altura_cm']])
numeric_df['largura_cm_robust'] = scaler.transform(numeric_df[['largura_cm']])
numeric_df['preco_robust'] = scaler.transform(numeric_df[['preco']])
# Visualizando novamente os dados
numeric_df.describe()
```

| | categoria | peso_kg_robust | comprimento_cm_robust | altura_cm_robust | largura_cm_robust | preco_robust |
|---|---|---|---|---|---|---|
| count | 4062.000000 | 4062.000000 | 4062.000000 | 4062.000000 | 4062.000000 | 4062.000000 |
| mean | 0.499015 | -0.953077 | -0.588840 | -0.797572 | -0.665751 | 0.198600 |
| std | 0.500061 | 0.013371 | 0.137547 | 0.082916 | 0.112355 | 0.777556 |
| min | 0.000000 | -0.967561 | -0.813088 | -0.939475 | -0.869260 | -0.950709 |
| 25% | 0.000000 | -0.962646 | -0.714787 | -0.855217 | -0.756916 | -0.405982 |
| 50% | 0.000000 | -0.957029 | -0.616486 | -0.813088 | -0.700744 | 0.000000 |
| 75% | 1.000000 | -0.947901 | -0.476057 | -0.742873 | -0.560315 | 0.594018 |
| max | 1.000000 | -0.870664 | -0.279455 | -0.560315 | -0.433928 | 2.824042 |

## 4. Análise e Exploração dos Dados

Análise e exploração dos dados é a fase em que são utilizadas técnicas de visualização e exploração de dados para obter insights iniciais. Gráficos, tabelas, histogramas e outras representações visuais são criados para identificar padrões, tendências, relações e anomalias nos dados. Isso ajuda a entender melhor a natureza dos dados e a formular hipóteses iniciais.

A primeira análise realizada foi aplicada no *dataframe df1* (que possui os valores originais – não normalizados) para cada classe, a partir da função *describe(),* que apresenta informações estatísticas descritivas, como contagem de valores, média, desvio padrão, quartis, mínimos e máximos.

As Figuras 16 e 17 indicam que a categoria cama, mesa e banho possui um tamanho amostral de 2.035 produtos, enquanto a categoria beleza e saúde possui 2.027. Ou seja, a base total está balanceada. Além disso, em toda a base não há valores ausentes/nulos.

**Figura 16 –** Análise de funções estatísticas da categoria cama, mesa e banho

```
# Analisando as informações estatísticas do df1-cama_mesa_banho
print(f'{n}Informações Estatísticas da categoria cama, mesa e banho:')
cmb = pd.DataFrame(data=df1)
cmb.loc[cmb['categoria'] == 0].describe().round(2)
```

```
Informações Estatísticas da categoria cama, mesa e banho:
```

| | categoria | peso_kg | comprimento_cm | altura_cm | largura_cm | preco |
|---|---|---|---|---|---|---|
| count | 2035.0 | 2035.00 | 2035.00 | 2035.00 | 2035.00 | 2035.00 |
| mean | 0.0 | 1.30 | 32.28 | 11.04 | 26.37 | 81.70 |
| std | 0.0 | 0.99 | 9.99 | 5.64 | 7.76 | 52.40 |
| min | 0.0 | 0.00 | 12.00 | 2.00 | 7.00 | 6.99 |
| 25% | 0.0 | 0.60 | 25.00 | 7.00 | 20.00 | 42.90 |
| 50% | 0.0 | 1.05 | 35.00 | 10.00 | 28.00 | 68.50 |
| 75% | 0.0 | 1.66 | 40.00 | 15.00 | 34.00 | 109.90 |
| max | 0.0 | 6.75 | 49.00 | 29.00 | 38.00 | 270.00 |

**Figura 17 –** Análise de funções estatísticas da categoria beleza e saúde

```
# Analisando as informações estatísticas do df1-beleza_saude
print(f'{n}Informações Estatísticas da categoria beleza e saúde:')
bs = pd.DataFrame(data=df1)
bs.loc[bs['categoria'] == 1].describe().round(2)
```

```
Informações Estatísticas da categoria beleza e saúde:
```

| | categoria | peso_kg | comprimento_cm | altura_cm | largura_cm | preco |
|---|---|---|---|---|---|---|
| count | 2027.0 | 2027.00 | 2027.00 | 2027.00 | 2027.00 | 2027.00 |
| mean | 1.0 | 0.76 | 21.64 | 13.18 | 16.59 | 84.39 |
| std | 0.0 | 0.83 | 5.94 | 5.97 | 4.45 | 58.18 |
| min | 1.0 | 0.05 | 11.00 | 2.00 | 7.00 | 1.20 |
| 25% | 1.0 | 0.25 | 17.00 | 9.00 | 14.00 | 39.75 |
| 50% | 1.0 | 0.44 | 20.00 | 13.00 | 16.00 | 68.90 |
| 75% | 1.0 | 1.03 | 25.00 | 17.00 | 19.00 | 116.94 |
| max | 1.0 | 6.90 | 48.00 | 29.00 | 38.00 | 269.90 |

Calculando-se os valores em torno da média de cada atributo, para as duas categorias, considerado o desvio padrão, chega-se na tabela abaixo. Nela é possível notar que os atributos peso, comprimento e largura possuem uma boa faixa de valores exclusivos entre as categorias. Por exemplo, um produto que tenha de 1,60Kg até 2,29Kg, de 27,59cm até 42,27cm de comprimento ou 21,05cm até 34,13cm de largura deve pertencer a categoria cama, mesa e banho. Já os atributos altura e preço apresentam faixas de valores muito semelhantes para as duas categorias. Sendo assim, na fase de treinamento dos algoritmos de aprendizado de máquina será testado se todas as *features* podem ou não contribuir para a predição das categorias dos produtos.

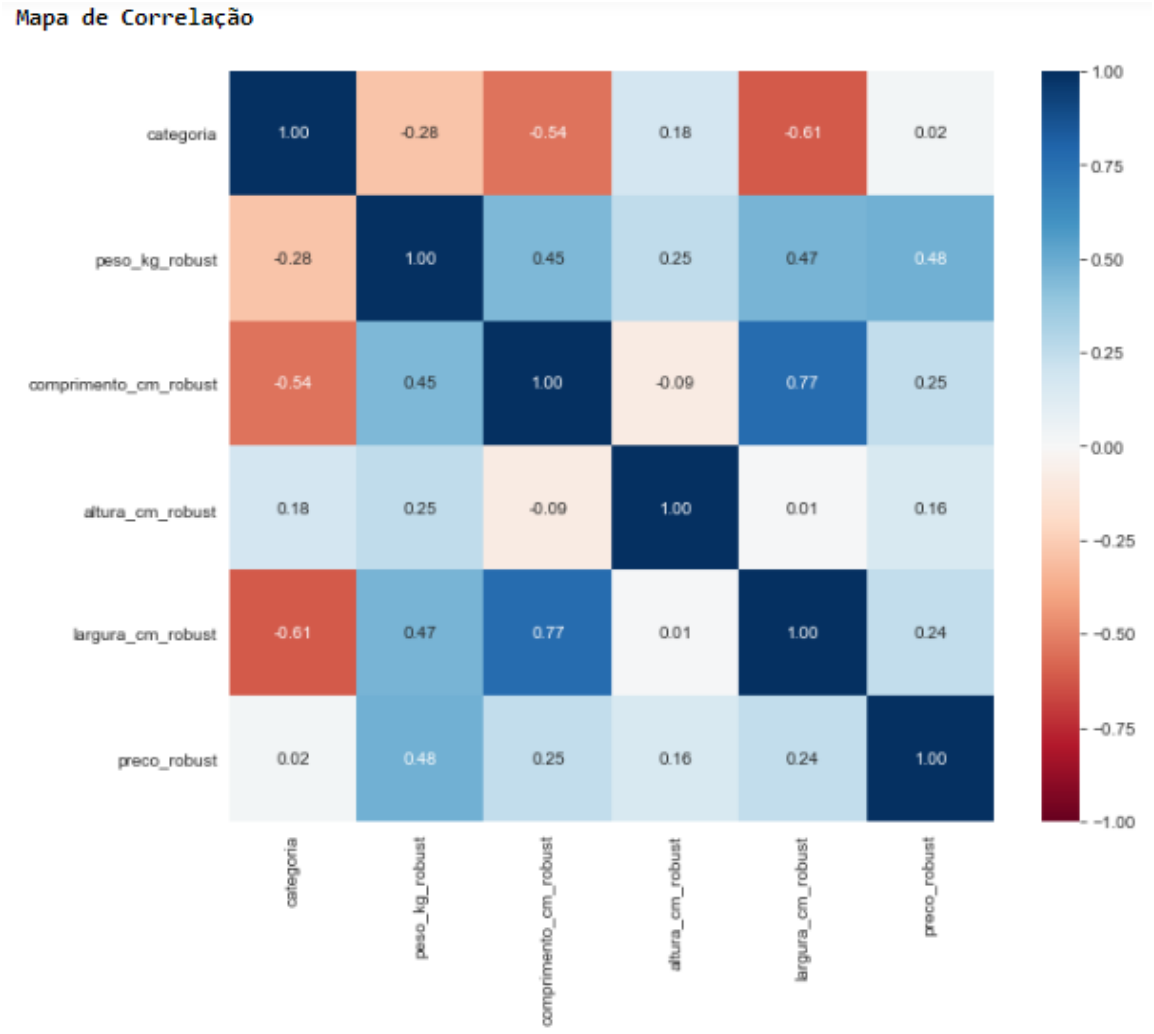
**Tabela 2:** Medidas em torno das médias de cada atributo para as duas categorias

| Atributo | categoria 0 cama_mesa_banho | | categoria 1 beleza_saude | |
|---|---|---|---|---|
| | De | Até | De | Até |
| peso_kg | 0,31 | 2,29 | -0,07 | 1,59 |
| comprimento_cm | 22,29 | 42,27 | 15,7 | 27,58 |
| altura_cm | 5,4 | 16,68 | 7,21 | 19,15 |
| largura_cm | 18,61 | 34,13 | 12,14 | 21,04 |
| preco | 29,3 | 134,1 | 26,21 | 142,57 |

Dando continuidade à análise e exploração dos dados, criou-se o *dataframe robust_df* (idêntico ao *numeric_df,* mas apenas com as *columns_robust* e categoria). Foi chamada a função *dataframe. corr()* para cálculo da correlação emparelhada das colunas. E, em seguida, avaliada a correlação entre as variáveis a partir de um *heatmap* (mapa de calor, nesse caso um mapa de correlação).

Um mapa de correlação é uma representação visual das correlações entre diferentes variáveis de um conjunto de dados. Ele pode destacar a presença de correlações positivas, indicando que o aumento de uma variável está associado ao aumento de outra variável. Por outro lado, as correlações negativas indicam que o aumento de uma variável está associado à diminuição de outra variável. Outra indicação importante a ser observada é a intensidade das correlações entre as variáveis. Quanto mais próximo o valor estiver de 1 (positivo) ou -1 (negativo), mais forte é a correlação. Valores próximos a 0 indicam uma correlação fraca.

**Figura 18 –** Mapa de correlação em *heatmap*



Conforme exibido na figura 18, o mapa de calor desse estudo indica que a variável largura possui alta correlação com o comprimento (0,77) e uma correlação de 0,47 com o peso. O mapa indica ainda uma correlação de 0,48 entre peso e preço, que não havia sido pensada. Além disso, os campos mais avermelhados indicam uma correlação negativa entre comprimento e largura com a variável categoria. Isso quer dizer que quanto maior for o produto nessas dimensões há uma maior tendência de a categoria ser zero (cama_mesa_banho), como já avaliado na tabela 2.

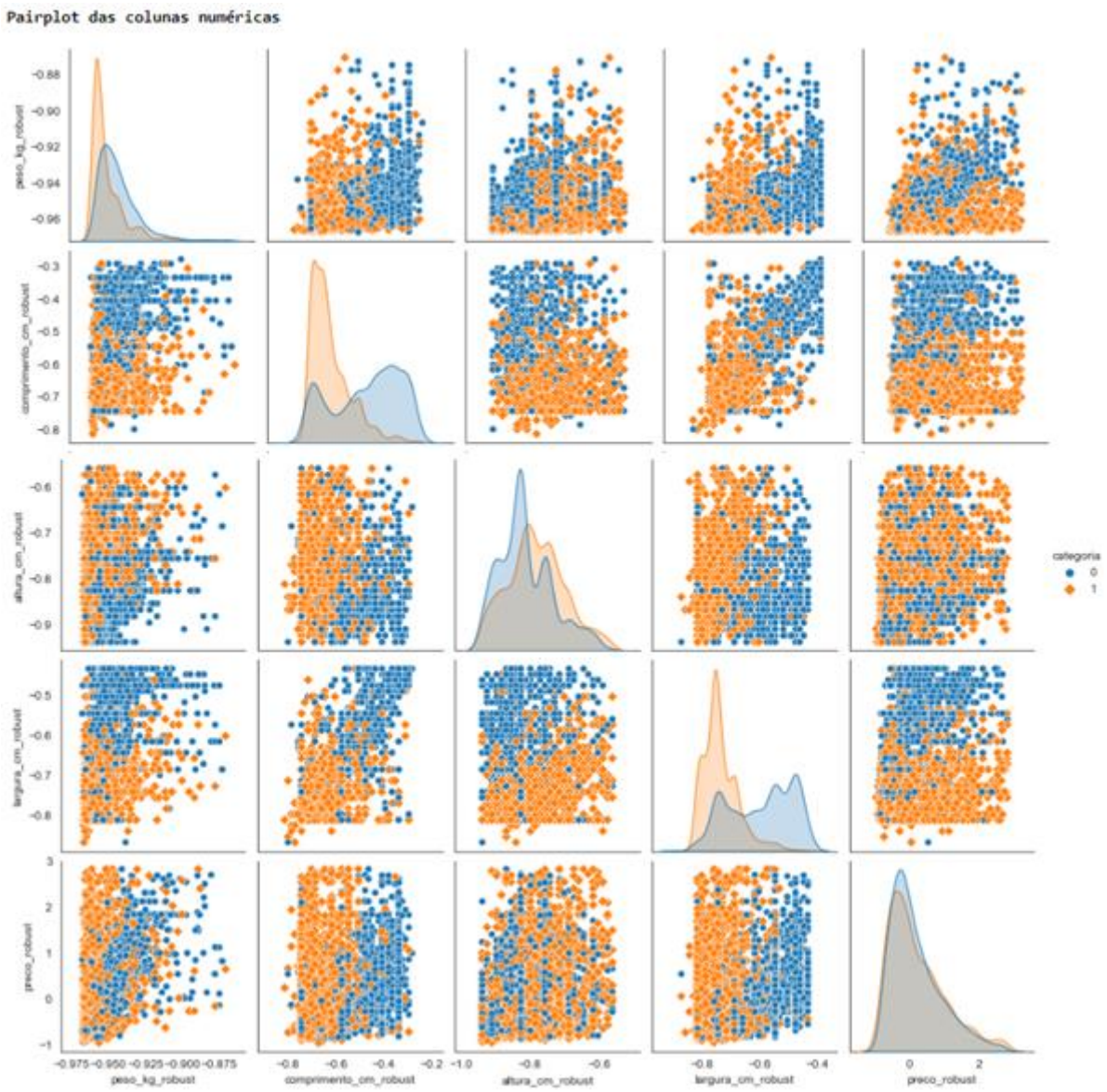
Outra ferramenta poderosa utilizada nesse projeto é o *pairplot(),* uma função da biblioteca *seaborn*, que plota a distribuição de cada atributo, assim como as relações entre pares de atributos (na forma de gráficos de dispersão - *scatter plots*) com visualização em forma de matriz. Essa representação visual permite identificar rapidamente padrões e relações entre múltiplas variáveis.

Ao observar os gráficos de dispersão em um *pairplot*, é possível identificar agrupamentos ou clusters de pontos que sugerem a existência de subgrupos ou padrões no conjunto de dados. Se houver uma variável de classe ou uma variável que define grupos, um *pairplot* pode revelar *insights* sobre a separabilidade dos grupos ou como as variáveis estão relacionadas dentro de cada categoria.

Além das relações entre pares de variáveis, um *pairplot* também exibe histogramas ou KDE (estimativa de densidade de kernel em camadas) na diagonal principal da matriz. Esses gráficos mostram a distribuição de cada variável individualmente, fornecendo informações sobre sua forma e características, como simetria, assimetria ou presença de outliers.

No *pairplot* elaborado nesse projeto (figura 19) nota-se, a partir dos gráficos de dispersão, uma alta complexidade para agrupamento dos pontos/produtos nas duas categorias (cama_mesa_banho e beleza_saude). Tal separação pode ser dita impossível de ser realizada visualmente por humanos. Enquanto isso, as densidades de Kernel apresentam visualmente uma esperança de sucesso nessa categorização, se aproveitadas as distinções de todas as categorias para abranger a diversidade da base que está sendo trabalhada.

**Figura 19 –** Pairplot das robust_columns




## 5. Criação de Modelos de Machine Learning

A criação de modelos de *machine learning* é um processo que envolve a construção de algoritmos e técnicas que permitem a máquina aprender a partir de dados e tomar decisões ou fazer previsões sem ser explicitamente programada. Esses modelos são construídos utilizando-se de técnicas estatísticas e algoritmos de aprendizado de máquina, que permitem identificar padrões nos dados e construir um modelo preditivo ou descritivo.

O processo de criação de modelos de *machine learning* envolve as seguintes etapas:

A escolha do algoritmo de aprendizado de máquina - Existem diversos algoritmos de aprendizado de máquina disponíveis, cada um com suas próprias características e adequado para diferentes tipos de problemas. A escolha do algoritmo depende da natureza do problema e dos dados disponíveis.

O treinamento do modelo - O modelo é alimentado com os dados de treinamento, que consistem em pares de entrada-saída, para que ele possa aprender a relação entre os recursos e as respostas desejadas. O algoritmo ajusta os parâmetros do modelo de acordo com os dados de treinamento para minimizar um erro ou função de perda.

A avaliação do modelo - O desempenho do modelo é avaliado utilizando-se de dados de teste, que são diferentes dos dados de treinamento e permitem verificar como o modelo generaliza para novas amostras. Métricas de desempenho, como acurácia, precisão, recall ou erro quadrático médio, são utilizadas para avaliar o quão bem o modelo está fazendo previsões ou tomando decisões.

Após o treinamento e validação, o modelo pode ser implantado em produção, onde é usado para fazer previsões ou tomar decisões em tempo real com base em novos dados de entrada. Isso pode envolver a integração do modelo em sistemas existentes ou o desenvolvimento de interfaces para interação com o usuário.

### 5.1. Escolha do Algoritmo de Aprendizado de Máquina

Existem vários algoritmos de aprendizado de máquina que podem ser aplicados à categorização. A escolha do algoritmo mais adequado depende do conjunto de dados específico e dos objetivos do problema. Os algoritmos comumente usados para essa tarefa são:

Árvores de Decisão (*Decision Trees*): As árvores de decisão são algoritmos de aprendizado supervisionado que dividem o conjunto de dados em subconjuntos com base em características relevantes. Elas podem ser usadas para classificar instâncias em diferentes categorias com base em regras de decisão.

Random Forest: Uma *random forest* é uma coleção de árvores de decisão. Cada árvore é construída usando um subconjunto aleatório dos dados e produz uma classificação. A classificação final é determinada pela maioria das classificações das árvores individuais.

*Extreme Gradient Boosting*: O extreme *gradient boosting (XGBoost)* é um método de aprendizado de máquina que constrói um modelo preditivo forte combinando vários mode-

los mais fracos em uma sequência. Ele é adequado para a categorização de variáveis numéricas, pois pode lidar com interações complexas entre os recursos.

Regressão Logística: A regressão logística é um método de aprendizado de máquina supervisionado que utiliza a função logística (também conhecida como função sigmoide) para modelar a probabilidade de uma observação pertencer a uma determinada classe. Essa função transforma os valores reais em um intervalo de 0 a 1, representando a probabilidade de pertencer à classe positiva. A regressão logística estima os coeficientes que ponderam as variáveis preditoras e ajusta o modelo aos dados de treinamento.

Máquinas de Vetores de Suporte (SVM): As SVMs são algoritmos de aprendizado supervisionado que podem ser usados tanto para classificação quanto para regressão. Elas mapeiam os dados em um espaço de alta dimensão e encontram uma fronteira de decisão ótima para separar diferentes classes.

Redes Neurais: As redes neurais artificiais são modelos inspirados no funcionamento do cérebro humano. Elas são compostas por camadas de neurônios interconectados que podem aprender a mapear os dados de entrada para as classes de saída desejadas. Redes neurais profundas, como as redes neurais convolucionais (CNNs) ou as redes neurais recorrentes (RNNs), são especialmente eficazes em tarefas de categorização de variáveis numéricas.

*Naive Bayes*: O classificador *Naive Bayes* é um método probabilístico que assume independência entre os recursos. Apesar dessa suposição simplificadora, ele pode fornecer resultados eficazes em muitos casos e é rápido de treinar e aplicar.

*K-Nearest Neighbors* (KNN): O algoritmo KNN atribui rótulos a novas instâncias com base na maioria dos rótulos das instâncias vizinhas mais próximas no espaço de recursos. Ele é uma abordagem não paramétrica e pode ser usado para classificação em tarefas de categorização.

Para esse projeto optou-se por trabalhar com os algoritmos Árvores de Decisão, Regressão Logística, Naive *Bayes* e *XGBoost*. Cada um desses três algoritmos será detalhado nos próximos tópicos.

### 5.1.1 Árvores de Decisão

#### 5.1.1.1 Árvores de Decisão – Conceito e Etapas de Criação do Modelo

As árvores de decisão são algoritmos de aprendizado de máquina que podem ser usados para tarefas de classificação e regressão. Elas são construídas com base em uma estrutura de árvore, onde cada nó interno representa uma decisão baseada em um atributo ou característica dos dados, e cada folha representa uma classe ou um valor de resposta.

O primeiro passo na construção de uma árvore de decisão é dividir os dados de treinamento em subconjuntos com base em diferentes características. O objetivo é encontrar a divisão que melhor separa as instâncias das diferentes classes ou valores de resposta.

O próximo passo é selecionar o atributo que será usado para a divisão. Isso pode ser feito usando várias métricas, como o ganho de informação, a razão de ganho ou o índice de Gini. Essas métricas avaliam a qualidade da divisão com base na pureza dos subconjuntos resultantes.

Após selecionar o atributo, um nó é criado na árvore. Esse nó representa a decisão a ser tomada com base no valor desse atributo. Por exemplo, se o atributo for "Idade" e a divisão for "Idade < 30", o nó terá uma regra de decisão associada a ele.

O processo de divisão é aplicado recursivamente aos subconjuntos resultantes em cada ramo da árvore. Isso continua até que uma condição de parada seja atendida, como alcançar um número mínimo de instâncias em um nó ou uma profundidade máxima predefinida.

Quando uma condição de parada é atendida, as folhas da árvore são criadas. Cada folha representa uma classe ou um valor de resposta. Por exemplo, se estivermos realizando uma tarefa de classificação binária, uma folha pode representar a classe "Sim" e outra folha a classe "Não".

Uma vez que a árvore de decisão é construída, ela pode ser usada para prever ou classificar novos dados. Isso é feito percorrendo a árvore a partir da raiz até chegar a uma folha. Cada nó interno é avaliado com base no valor do atributo correspondente nos novos dados, seguindo o ramo apropriado até alcançar uma folha. A classe ou o valor da folha é então atribuído à nova instância.

#### 5.1.1.2 Árvores de Decisão – Vantagens e Desvantagens

As árvores de decisão têm várias vantagens e desvantagens. Aqui estão algumas das principais:

Vantagens do algoritmo de árvores de decisão:

Interpretabilidade: As árvores de decisão são altamente interpretáveis e podem ser facilmente compreendidas e visualizadas. A estrutura em forma de árvore com nós e ramificações torna fácil acompanhar o processo de tomada de decisão do algoritmo.

Lida com dados mistos: As árvores de decisão podem lidar com conjuntos de dados que possuem uma combinação de variáveis categóricas e numéricas. Elas são capazes de tratar ambos os tipos de variáveis sem a necessidade de pré-processamento adicional.

Não requer normalização de atributos: Ao contrário de alguns algoritmos de aprendizado de máquina, como regressão logística ou redes neurais, as árvores de decisão não exigem que os atributos sejam normalizados ou padronizados. Elas são insensíveis à escala dos atributos.

Requer menos preparação de dados: As árvores de decisão podem lidar diretamente com dados que possuem valores ausentes. Elas também são menos sensíveis a outliers, pois são baseadas em divisões recursivas que não dependem diretamente de medidas de centralidade ou dispersão dos dados.

Eficiência computacional: As árvores de decisão podem ser computacionalmente eficientes, especialmente para conjuntos de dados menores ou com poucas características. O tempo de treinamento e a velocidade de previsão são geralmente rápidos.

Desvantagens do algoritmo de árvores de decisão:

Tendência ao *overfitting*: As árvores de decisão têm uma tendência a se ajustarem demais aos dados de treinamento, especialmente se não houver restrições ou técnicas de poda adequadas. Isso pode levar a um desempenho ruim em dados de teste ou novos dados.

Sensibilidade a pequenas variações nos dados: As árvores de decisão podem ser sensíveis a pequenas variações nos dados de treinamento, o que pode levar a diferentes estruturas de árvore e resultados inconsistentes.

Dificuldade em capturar relações complexas: As árvores de decisão têm limitações em capturar relações complexas entre as variáveis. Elas são mais adequadas para problemas

onde as relações são mais simples e podem ter dificuldades em modelar relações não lineares ou interações complexas entre os atributos.

Instabilidade com dados ruidosos: As árvores de decisão são suscetíveis a *overfitting* em conjuntos de dados com ruído ou com informações irrelevantes. A presença de ruído pode levar a decisões errôneas em certos ramos da árvore, resultando em uma árvore menos generalizável.

Dificuldade em lidar com classes desbalanceadas: As árvores de decisão podem ter dificuldade em lidar com conjuntos de dados onde as classes estão desbalanceadas, ou seja, quando há uma grande diferença na quantidade de instâncias entre as classes. Isso pode resultar em uma tendência a favorecer a classe majoritária.

### 5.1.2 Regressão Logística

#### 5.1.2.1 Regressão Logística – Conceito e Etapas de Criação do Modelo

A regressão logística é um algoritmo adequado para lidar com variáveis numéricas como características preditoras e uma variável de resposta categórica binária. Para aplicar a regressão logística é necessário codificar a variável de resposta categórica em uma representação numérica adequada. Isso pode ser feito atribuindo um valor binário (0 ou 1) para cada classe. Em seguida, as variáveis numéricas podem ser usadas como recursos preditores.

A regressão logística utiliza a função de log-verossimilhança para estimar os coeficientes do modelo. O processo de treinamento envolve a minimização de uma função de custo, como a função de custo de entropia cruzada. Existem várias abordagens para ajustar os parâmetros do modelo, incluindo o algoritmo de descida do gradiente.

Após o treinamento, o modelo de regressão logística pode ser usado para fazer previsões em novos dados. A partir dos coeficientes estimados e dos valores das variáveis preditoras, é calculada a probabilidade de pertencer a cada classe. A classe com a maior probabilidade é então atribuída à observação.

#### 5.1.2.2 Regressão Logística – Vantagens e Desvantagens

O algoritmo de regressão logística tem várias vantagens e desvantagens. Aqui estão algumas das principais:

Vantagens do algoritmo de regressão logística:

Interpretabilidade: A regressão logística fornece coeficientes que representam a contribuição relativa de cada atributo para a variável dependente. Isso torna o modelo altamente interpretável, permitindo entender como cada variável influencia a probabilidade de uma determinada classe.

Eficiência computacional: A regressão logística é computacionalmente eficiente, especialmente em comparação com algoritmos mais complexos, como redes neurais. O treinamento e a previsão são geralmente rápidos, mesmo em grandes conjuntos de dados.

Requer menos dados: A regressão logística requer menos dados de treinamento em comparação com alguns outros algoritmos de aprendizado de máquina. Ela pode fornecer bons resultados mesmo quando o número de observações é limitado, tornando-a adequada para problemas com amostras pequenas.

Lida com atributos categóricos: A regressão logística pode lidar facilmente com atributos categóricos, uma vez que eles são codificados adequadamente. Não é necessário transformar os atributos categóricos em variáveis *dummy*, pois a regressão logística pode trabalhar diretamente com eles.

Probabilidades estimadas: Além de prever a classe de uma instância, a regressão logística também pode fornecer a probabilidade estimada de pertencer a uma determinada classe. Isso pode ser útil em cenários em que é necessário avaliar a incerteza das previsões.

Desvantagens do algoritmo de regressão logística:

Pressupostos lineares: A regressão logística assume uma relação linear entre os atributos independentes e a variável dependente transformada logit. Se houver uma relação não linear entre as variáveis, a regressão logística pode não a capturar adequadamente. Nesses casos, podem ser necessárias transformações nos atributos ou o uso de modelos não lineares.

Sensibilidade a outliers: A regressão logística pode ser sensível a outliers, que podem distorcer os coeficientes estimados e afetar as previsões. É importante identificar e tratar os outliers antes de aplicar o algoritmo para obter resultados mais robustos.

Dados balanceados: A regressão logística pode ser afetada pelo desequilíbrio de classes nos dados de treinamento. Se houver uma grande diferença na quantidade de instâncias

entre as classes, o modelo pode ficar enviesado em direção à classe majoritária. Nesses casos, técnicas de balanceamento de dados, como *oversampling* ou *undersampling*, podem ser necessárias.

Multicolinearidade: A multicolinearidade ocorre quando há alta correlação entre os atributos independentes. Isso pode dificultar a interpretação dos coeficientes estimados e levar a resultados instáveis. É importante realizar uma análise exploratória dos dados para identificar e tratar a multicolinearidade antes de aplicar a regressão logística.

Limite na complexidade do modelo: A regressão logística tem uma capacidade limitada de modelar relações complexas entre as variáveis. Ela pode não ser adequada para problemas com interações não lineares ou relações.

### 5.1.3 Naive Bayes

#### 5.1.3.1 Naive Bayes – Conceito e Etapas de Criação do Modelo

O algoritmo *Naive Bayes* é um método de aprendizado de máquina baseado no teorema de *Bayes*, que utiliza a probabilidade condicional para fazer previsões. Ele é conhecido como "*Naive*" (ingênuo) porque assume a independência condicional entre os atributos, ou seja, considera que os atributos são independentes entre si, dado o valor da classe. Essa é uma suposição simplificadora, mas muitas vezes eficaz na prática.

O primeiro passo é preparar os dados de treinamento. Isso envolve a codificação dos atributos em formato numérico e a divisão dos dados em classes ou categorias.

O próximo passo é calcular as probabilidades a priori de cada classe. Isso é feito determinando a frequência relativa de cada classe nos dados de treinamento.

Em seguida, são estimadas as probabilidades condicionais de cada atributo, dada a classe. Para cada atributo, o algoritmo calcula a frequência relativa de cada valor do atributo para cada classe.

Com as probabilidades a priori e condicionais calculadas, o *Naive Bayes* pode calcular as probabilidades posteriores para cada classe, dada uma nova instância de teste. Isso é feito multiplicando as probabilidades condicionais relevantes para cada atributo e multiplicando-as pela probabilidade a priori da classe correspondente.

Após calcular as probabilidades posteriores para cada classe, o *Naive Bayes* atribui a nova instância à classe com a maior probabilidade posterior. Isso significa que a classe com a maior probabilidade é considerada a classe prevista para a instância de teste.

Se o conjunto de dados contiver atributos contínuos, o *Naive Bayes* assume que eles seguem uma distribuição de probabilidade específica, como a distribuição normal (Gaussiana). É usado o cálculo de densidade de probabilidade para estimar as probabilidades condicionais nesses casos.

Para evitar problemas de probabilidade zero quando um valor de atributo não está presente em uma determinada classe nos dados de treinamento, é comum aplicar uma técnica de suavização, como a suavização de Laplace. Essa técnica adiciona uma pequena quantidade às contagens de frequência dos valores do atributo, garantindo que todas as probabilidades sejam maiores que zero.

O algoritmo *Naive Bayes* é rápido e eficiente, especialmente em grandes conjuntos de dados. Embora a suposição de independência condicional seja simplificadora e nem sempre verdadeira na prática, o *Naive Bayes* pode fornecer resultados eficazes em muitos cenários, como classificação de texto, filtragem de spam e detecção de sentimentos.

É importante destacar que o desempenho do *Naive Bayes* depende da adequação dessa suposição de independência aos dados específicos do problema. Portanto, é sempre recomendável realizar uma análise exploratória dos dados antes de aplicar o algoritmo e avaliar sua eficácia em relação aos resultados desejados.

#### 5.1.3.2 Naive Bayes – Vantagens e Desvantagens

O algoritmo *Naive Bayes* tem várias vantagens e desvantagens. Aqui estão algumas das principais:

Vantagens do algoritmo *Naive Bayes*:

Simplicidade e eficiência: O *Naive Bayes* é um algoritmo simples e fácil de implementar. Ele possui um tempo de treinamento e previsão rápido, mesmo em conjuntos de dados grandes. Isso torna o *Naive Bayes* adequado para problemas com restrições de recursos computacionais.

Lida bem com dados de alta dimensionalidade: O *Naive Bayes* funciona bem mesmo quando o número de atributos é alto em relação ao número de observações. Ele é eficaz em

problemas de classificação com dados de alta dimensionalidade, como análise de texto ou classificação de documentos.

Trata de forma eficiente atributos categóricos: O *Naive Bayes* lida naturalmente com atributos categóricos e funciona bem em conjuntos de dados que contêm variáveis discretas. Ele usa a suposição de independência condicional entre os atributos para calcular as probabilidades condicionais.

Lida bem com dados ausentes: O *Naive Bayes* pode lidar de forma robusta com dados que possuem valores ausentes. Ele simplesmente ignora os valores ausentes durante o cálculo das probabilidades condicionais, o que evita a necessidade de imputação de dados.

Poucos parâmetros para estimar: O *Naive Bayes* requer apenas a estimativa das probabilidades a priori e condicionais. Isso significa que, em comparação com outros algoritmos, o *Naive Bayes* possui um número reduzido de parâmetros a serem estimados, o que pode ser uma vantagem em problemas com conjuntos de dados pequenos.

Desvantagens do algoritmo *Naive Bayes*:

Suposição de independência: O *Naive Bayes* assume a independência condicional entre os atributos, o que nem sempre reflete a realidade. Essa suposição pode levar a uma perda de informações sobre as interações entre os atributos e resultar em previsões menos precisas.

Sensibilidade a *features* irrelevantes: O *Naive Bayes* considera todos os atributos igualmente importantes na classificação, mesmo aqueles que podem ser irrelevantes ou ter pouco impacto na previsão. Isso pode afetar negativamente o desempenho do modelo, especialmente se houver atributos irrelevantes presentes no conjunto de dados.

Dados de treinamento escassos: O desempenho do *Naive Bayes* pode ser prejudicado quando há uma quantidade limitada de dados de treinamento, especialmente quando as classes são desbalanceadas. Como o algoritmo depende de estimativas de probabilidade, se houver poucas observações em uma classe específica, as estimativas podem não ser confiáveis.

Incapacidade de lidar com problemas de regressão: O *Naive Bayes* é amplamente utilizado para problemas de classificação, mas não é adequado para problemas de regressão, onde a variável de saída é contínua. O algoritmo não fornece uma estrutura nativa para lidar com previsões de valores contínuos.

Sensibilidade a violações das premissas: O *Naive Bayes* assume que os atributos são independentes e que a distribuição das classes é a mesma nos dados de treinamento e teste. Quando essas premissas são violadas, o desempenho do *Naive Bayes* pode ser comprometido.

É importante ressaltar que, embora o *Naive Bayes* tenha suas limitações, ele ainda pode ser um algoritmo eficaz em muitos casos, especialmente quando as suposições subjacentes são razoáveis e o conjunto de dados é adequado.

### 5.1.4 XGBoost

#### 5.1.4.1 XGBoost – Conceito e Etapas de Criação do Modelo

O *XGBoost (Extreme Gradient Boosting)* é um algoritmo de aprendizado de máquina baseado em árvores de decisão que se destaca por sua eficácia e desempenho em competições de ciência de dados. Ele utiliza o método de boosting para criar um modelo preditivo robusto e preciso.

O *XGBoost* começa treinando um modelo de árvore de decisão de base. Essas árvores são construídas recursivamente a partir dos dados de treinamento, dividindo o conjunto de dados em subconjuntos com base nos valores dos atributos. A construção da árvore ocorre de forma iterativa, onde os nós são adicionados até atingir uma profundidade máxima ou atender a um critério de parada.

O *XGBoost* utiliza uma função objetivo para medir a diferença entre as previsões do modelo e os rótulos reais dos dados de treinamento. A função objetivo combina duas partes: uma função de perda que mede o erro em cada previsão e uma penalidade de regularização que controla a complexidade do modelo para evitar o *overfitting*. A função objetivo é otimizada durante o treinamento para encontrar os melhores parâmetros do modelo.

O *XGBoost* utiliza o método de boosting para melhorar gradualmente o desempenho do modelo. O boosting envolve a criação de um conjunto de árvores de decisão de base, onde cada árvore é treinada para corrigir os erros cometidos pelas árvores anteriores. Durante o treinamento, o *XGBoost* ajusta os pesos das instâncias de treinamento, dando mais importância às instâncias que foram previstas incorretamente nas iterações anteriores.

O *XGBoost* aplica regularização para evitar o overfitting e melhorar a capacidade de generalização do modelo. Duas formas comuns de regularização usadas pelo *XGBoost* são a

regularização L1 (*lasso*) e a regularização L2 (*ridge*). Essas técnicas adicionam uma penalidade aos pesos das árvores de decisão durante o treinamento, desencorajando pesos excessivamente altos e reduzindo a complexidade do modelo.

O *XGBoost* utiliza uma técnica chamada "*gradient boosting*" para ajustar os parâmetros do modelo. Durante o treinamento, as árvores de decisão de base são ajustadas iterativamente para minimizar a função objetivo. Em cada iteração, o *XGBoost* calcula os gradientes da função objetivo em relação às previsões atuais do modelo, e então ajusta as árvores de decisão de base para minimizar esses gradientes.

Para melhorar a eficiência e evitar o *overfitting,* o *XGBoost* realiza a poda das árvores de decisão durante o treinamento. A poda remove ramos da árvore que não contribuem significativamente para a melhoria do modelo. A poda ajuda a simplificar o modelo e reduzir o número total de nós e divisões, resultando em um modelo mais compacto.

#### 5.1.4.2 XGBoost – Vantagens e Desvantagens

O algoritmo *XGBoost* (*Extreme Gradient Boosting*) possui diversas vantagens e desvantagens. Aqui estão algumas das principais:

Vantagens do algoritmo XGBoost:

Alta precisão: O *XGBoost* é conhecido por sua capacidade de fornecer resultados altamente precisos. Ele utiliza técnicas avançadas de boosting e regularização, o que ajuda a reduzir o viés e a variância do modelo, resultando em previsões mais acuradas.

Lida bem com dados desbalanceados: O *XGBoost* é eficaz em lidar com conjuntos de dados desbalanceados, onde as classes têm tamanhos muito diferentes. Ele usa estratégias como ponderação de instâncias e amostragem por gradient boosting para tratar esse desequilíbrio, o que melhora o desempenho na classe minoritária.

Flexibilidade e versatilidade: O *XGBoost* é um algoritmo versátil que pode ser aplicado tanto para problemas de classificação quanto para problemas de regressão. Além disso, ele suporta diferentes funções de perda e critérios de avaliação, permitindo que você escolha a métrica mais adequada para o seu problema.

Lida bem com dados de alta dimensionalidade: O *XGBoost* é capaz de lidar eficientemente com conjuntos de dados de alta dimensionalidade, onde o número de atributos é grande em relação ao número de observações. Ele utiliza técnicas como poda de

árvores e regularização para evitar *overfitting* e melhorar o desempenho em conjuntos de dados com muitos atributos.

Tratamento automático de missing values: O *XGBoost* tem a capacidade de lidar com dados que possuem valores ausentes (*missing values*) sem a necessidade de imputação. Ele pode aprender a melhor maneira de lidar com os valores ausentes durante o treinamento, tornando o processo mais conveniente.

Desvantagens do algoritmo XGBoost:

Sensibilidade a hiper parâmetros: O XGBoost possui vários hiper parâmetros que precisam ser ajustados corretamente para obter um desempenho ideal. A seleção adequada dos hiperparâmetros requer experimentação e pode ser um processo demorado e complexo.

Requer mais recursos computacionais: O *XGBoost* pode exigir mais recursos computacionais, como memória e poder de processamento, em comparação com algoritmos mais simples, devido à sua complexidade e ao treinamento de múltiplas árvores de decisão.

Interpretabilidade limitada: À medida que o *XGBoost* utiliza conjuntos de árvores de decisão complexas, a interpretabilidade do modelo pode ser comprometida. A relação entre as variáveis de entrada e a saída pode não ser facilmente compreendida ou explicada.

Sensibilidade a outliers: O *XGBoost* pode ser sensível a outliers em conjuntos de dados. *Outliers* extremos podem ter um impacto desproporcional no treinamento do modelo, afetando negativamente a performance.

Dados categóricos exigem pré-processamento: O *XGBoost* lida melhor com dados numéricos, então, se o conjunto de dados contiver atributos categóricos, eles geralmente precisam ser pré-processados e convertidos em variáveis numéricas antes de aplicar o algoritmo.

## 5.2 Treinamento do Modelo de Aprendizado de Máquina

Na etapa de treinamento o modelo de *machine learning* é alimentado com os dados de treino preparados. O objetivo é ajustar os parâmetros internos do modelo (fazer "*fit*" dos dados) de forma que ele seja capaz de fazer previsões precisas. Durante o treinamento, o modelo faz atualizações iterativas nos parâmetros com base nas informações fornecidas pelos dados.

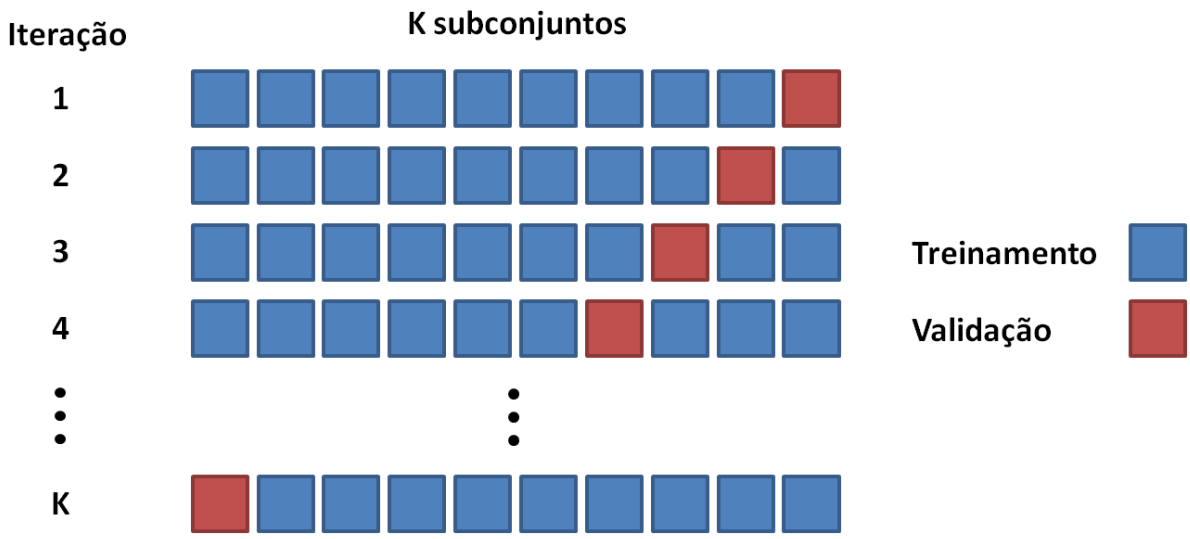
### 5.2.1 Validação Cruzada K-Fold

*K-Fold* é uma técnica de validação cruzada *(cross-validation*) usada em aprendizado de máquina e análise estatística. É usado para avaliar o desempenho de um modelo de forma mais confiável, dividindo os dados em conjuntos de treinamento e teste em várias iterações.

A validação cruzada é usada para mitigar problemas como *overfitting* e garantir que o modelo seja capaz de generalizar bem para novos dados. O *K-Fold* é uma abordagem específica de validação cruzada que divide o conjunto de dados em K "dobras" ou partições. Cada partição é usada como conjunto de teste em uma das iterações, enquanto as outras K-1 partições são usadas como conjunto de treinamento.

A validação cruzada envolve dividir o conjunto de dados disponível em várias partições, normalmente chamadas de "*folds*". Em seguida, o modelo é treinado em uma combinação desses *folds* e avaliado nos *folds* restantes. Esse processo é repetido várias vezes, alternando os *folds* usados para treinamento e avaliação, de modo que todos os folds sejam usados tanto para treinamento quanto para avaliação em algum momento (Figura 19). Ao final, os resultados são agregados e uma métrica de desempenho é calculada, como acurácia, precisão, recall, F1-score, entre outras.

**Figura 20 –** k-fold cross validation




Fonte: Eric Couto – ericcouto.wordpress.com

Ao escolher o melhor algoritmo para um problema de *Machine Learning* com base na validação cruzada, é possível comparar e avaliar diferentes modelos ou algoritmos, levando em consideração a performance média em várias divisões dos dados. Isso é importante porque diferentes algoritmos podem ter desempenhos variados em diferentes conjuntos de dados, e a validação cruzada permite uma avaliação mais robusta e imparcial.

Através da validação cruzada, é possível identificar se um modelo está sofrendo de *overfitting* (sobreajuste) ou underfitting (subajuste). O *overfitting* ocorre quando o modelo se ajusta excessivamente aos dados de treinamento, mas tem um desempenho ruim em dados não vistos. Já o *underfitting* ocorre quando o modelo não consegue capturar adequadamente a relação subjacente nos dados e apresenta desempenho ruim. A validação cruzada ajuda a identificar esses problemas e a selecionar o algoritmo que melhor se adapte ao problema em questão, levando em consideração o equilíbrio entre viés (bias) e variância do modelo.

### 5.2.2 Otimização de Hiper Parâmetros

A otimização de hiper parâmetros, também conhecida como 'tunagem', é uma técnica utilizada para encontrar a combinação ideal de hiper parâmetros de um modelo de *machine learning*, a fim de obter o melhor desempenho possível. Existem diferentes métodos de otimização de hiper parâmetros, sendo os mais comuns:

Busca em grade (Grid Search): Esse método consiste em definir uma grade de valores para cada hiper parâmetro e realizar uma busca exaustiva por todas as combinações possíveis. O Grid Search avalia o desempenho do modelo para cada combinação de hiper parâmetros e seleciona aquela que obtém os melhores resultados.

Busca aleatória (*Random Search*): Nesse método, as combinações de hiper parâmetros são selecionadas aleatoriamente dentro de um espaço definido. Diferentemente do Grid Search, que explora todas as combinações possíveis, o Random Search seleciona um subconjunto aleatório de combinações para avaliação.

Otimização Bayesiana (*Bayesian Optimization*): A otimização bayesiana é um método que utiliza o aprendizado de máquina para encontrar a combinação ideal de hiper parâmetros. Ele modela uma função objetivo desconhecida que mapeia os hiper parâmetros para o desempenho do modelo e usa inferência bayesiana para encontrar a próxima combinação de hiper parâmetros a ser avaliada. Esse método é capaz de explorar de forma mais eficiente

o espaço de busca e geralmente requer menos avaliações do modelo em comparação com o Grid Search e Random Search.

Otimização baseada em gradiente (Gradient-Based Optimization): Nesse método, os hiper parâmetros são tratados como variáveis contínuas e a otimização é realizada usando técnicas de otimização baseadas em gradiente, como descida de gradiente. O objetivo é encontrar a direção do gradiente que maximize o desempenho do modelo.

Não existe um método de otimização de hiper parâmetros que seja universalmente recomendado, pois a escolha depende do conjunto de dados, do modelo e do problema em questão. No entanto, a busca em grade é uma abordagem simples e direta que pode ser útil quando o espaço de busca não é muito grande. A otimização bayesiana é uma técnica mais avançada que pode ser eficiente para espaços de busca maiores e desconhecidos, mas pode exigir mais recursos computacionais. A busca aleatória é uma boa opção quando há recursos computacionais limitados, mas não é tão eficiente quanto os outros métodos. A otimização baseada em gradiente é mais adequada quando os hiper parâmetros são contínuos e a função objetivo é diferenciável.

Em resumo, a escolha do método de otimização de hiper parâmetros depende das restrições computacionais, do espaço de busca e do conhecimento prévio sobre os hiper parâmetros. Neste projeto, após algumas experimentações, optou-se por adotar a otimização Bayesiana de hiper parâmetros.

### 5.2.3 Treinamento Aplicado ao Projeto

Com base nas hipóteses formuladas durante a fase de exploração, é realizada uma análise mais aprofundada dos dados. Isso pode envolver a aplicação de técnicas estatísticas, modelagem matemática, algoritmos de aprendizado de máquina, mineração de dados e outras abordagens para extrair informações significativas dos dados. O objetivo é responder às perguntas de pesquisa e validar as hipóteses levantadas.

Antes de realizar os treinos com os algoritmos escolhidos para esse trabalho, x e y foram identificados, a base foi separada aleatoriamente em conjunto de dados de treino (com 70% dos dados) e em conjunto de dados de treino de teste (com 30% dos dados) e ajustada a validação cruzada (cv) com k-fold. Como pode ser visto na figura abaixo.

**Figura 21 –** Preparação dos dados para treino/teste

```
# Separando X e y
X = robust_df.drop(['categoria'],axis=1)
y = robust_df.categoria
```

```
# Selecionando os conjuntos de treinamento (70%) e teste (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

```
# Tamanhos de x e y de treino e de teste
X_train.shape, X_test.shape
```

```
((2843, 5), (1219, 5))
```

```
# Ajustes da validação cruzada sem otimização de hiper parâmetros
cv = KFold(n_splits=10,random_state=42,shuffle=True)
```

Em seguida, para cada algoritmo (Árvores de Decisão, Regressão Logística, *Naive Bayes* e *XGBoost*), foi criado um classificador do modelo sem otimização de hiper parâmetros e calculados a acurácia e o desvio padrão. Para os algoritmos Árvores de Decisão e *XGBoost*, que apresentaram melhores resultados foi aplicada a otimização bayesiana com ajuste dos seus principais hiper parâmetros, além de "fit" dos dados, medição de performance e pesquisa de melhores parâmetros e estimadores.

Os processos executados no Python para esses dois algoritmos serão então detalhados em breve. Mas antes disso, é importante explicar quais são os principais hiper parâmetros dos algoritmos Árvores de Decisão e XGBoost, que devem ser ajustados.

Principais hiper parâmetros – Árvores de Decisão (Decision Tree)

criterion: É um hiper parâmetro categórico em que pode ser usado "entropia" ou impureza de "gini" (padrão). Ambos medem a pureza de uma partição em um mecanismo diferente. A entropia varia entre 0 e 1, mas a impureza de Gini varia entre 0 e 0,5.

*splitter*: É a estratégia de como dividir um nó, podendo ser "best" (padrão) ou "random". O melhor divisor passa por todos os conjuntos possíveis de divisões em cada feição no conjunto de dados e seleciona a melhor divisão. Ele sempre dá o mesmo resultado, escolhe o mesmo recurso e limite para dividir porque sempre procura a melhor divisão.

max_depth: Determina a profundidade máxima da árvore. Se *None* for fornecido, a divisão continuará até que todas as folhas sejam puras (ou até atingir o limite especificado no parâmetro *min_samples_split).*

*max_features*: O número de recursos a serem considerados ao dividir. Podemos determinar o número máximo de recursos a serem usados fornecendo números inteiros e proporcionalmente fornecendo números flutuantes. Se for fornecido um float, então *max_features* será *int* (*max_features* * *n_features* ). *sqrt* é o número da raiz quadrada do total de recursos e *log2* é o log2 do número total de recursos.

Principais hiper parâmetros – XGBoost (Xtreme Gradient Boosting)

max_depth: Decide a complexidade de cada árvore em seu modelo. Refere-se à profundidade máxima que uma árvore pode atingir. Uma árvore mais profunda significa mais caminhos de decisão e captura potencialmente padrões mais complexos nos dados. No entanto, aumentar a profundidade também pode fazer com que o modelo superajuste os dados de treinamento, pois tornará a árvore mais complexa.

min_child_weight: É um hiperparâmetro que define a soma mínima dos pesos das instâncias que devem estar presentes em um nó filho em cada árvore. Este parâmetro ajuda a controlar a complexidade do modelo e evita o overfitting. Isso não significa que se deva definir min_child_weightum valor alto, pois isso tornará seu modelo menos flexível e mais propenso a subajuste.

subsample: Esse hiperparâmetro desempenha um papel no controle da quantidade de dados usados para construir cada árvore em seu modelo. É uma fração que varia de 0 a 1, representando a proporção do conjunto de dados a ser selecionada aleatoriamente para o treinamento de cada árvore.

colsample_bytree: Ele determina a proporção de feições a serem consideradas para cada árvore. Este valor varia de 0 a 1, onde um valor de 1 significa que todos os recursos serão considerados para cada árvore, e um valor menor indica que apenas um subconjunto de recursos será escolhido aleatoriamente antes de construir cada árvore.

Concluída a explicação sobre os hiper parâmetros que foram ajustados nos modelos dos algoritmos Árvores de Decisão e XGBoost, agora serão apresentadas as codificações e saídas do Python para todo o processo de treinamento desses dois algoritmos.

**Figura 22 –** Árvores de decisão – Sem otimização de hiper parâmetros

```
# Criando o classificador
clf_dt = tree.DecisionTreeClassifier(random_state=1)
# Calculando a acurácia e o desvio padrão
score_dt = cross_val_score(clf_dt, X_train, y_train, scoring='accuracy', cv=cv, n_jobs=-1)
print('acuráia: %.2f | desvio padrão: %.3f' % (np.mean(score_dt)*100, np.std(score_dt)))
```

```
acuráia: 88.85 | desvio padrão: 0.013
```

**Figura 23 –** Árvores de decisão – Com otimização de hiper parâmetros

```
# Criando o classificador
clf_dt_tunado = tree.DecisionTreeClassifier(random_state=2)
# Fazendo o k-fold Cross-Validation
score_dt_tunado = cross_val_score(clf_dt_tunado, X_train, y_train, scoring='accuracy', cv=5, n_jobs=-1)
# Definindo o range de parâmetros
dt_search_space = {"criterion": Categorical(['gini','entropy']),
                   "splitter": Categorical(['best', 'random']),
                   "max_depth": Integer(3, 15),
                   "max_features": Categorical(['auto', 'sqrt', 'log2']),
                  }
# Fitando o Modelo e Medindo a Performance
dt_bayes_search = BayesSearchCV(clf_dt_tunado, dt_search_space, scoring='accuracy', cv=5)
dt_bayes_search.fit(X_train, y_train)
```

```
BayesSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=2),
              scoring='accuracy',
              search_spaces={'criterion': Categorical(categories=('gini', 'entropy'), prior=None),
                             'max_depth': Integer(low=3, high=15, prior='uniform', transform='normalize'),
                             'max_features': Categorical(categories=('auto', 'sqrt', 'log2'), prior=None),
                             'splitter': Categorical(categories=('best', 'random'), prior=None)})
```

```
# Configuração de parâmetro que deu os melhores resultados nos dados de espera
dt_bayes_search.best_params_
```

```
OrderedDict([('criterion', 'entropy'),
             ('max_depth', 12),
             ('max_features', 'sqrt'),
             ('splitter', 'best')])
```

```
# Estimador que deu maior pontuação ou menor perda nos dados deixados de fora
dt_bayes_search.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=12, max_features='sqrt',
                       random_state=2)
```

```
# Calculando a acurácia
print('acuráia: %.2f' % (np.mean(dt_bayes_search.best_score_)*100))
```

```
acuráia: 89.13
```

A otimização bayesiana melhorou a acurácia do algoritmo de Árvores de Decisão neste projeto em 0,28% (de 88,85 para 89,13).

**Figura 24 –** *XGBoost* – Sem otimização de hiper parâmetros

```
# Criando o classificador
clf_xgb = xgb.XGBClassifier(random_state=5)
# Calculando a acurácia e o desvio padrão
scores_xgb = cross_val_score(clf_xgb, X_train, y_train, scoring='accuracy', cv=cv)
print('acuráia: %.2f | desvio padrão: %.3f' % (np.mean(scores_xgb)*100, np.std(scores_xgb)))
```

```
acuráia: 92.72 | desvio padrão: 0.010
```

**Figura 25 –** *XGBoost* – Com otimização de hiper parâmetros

```
# Criando o classificador
clf_xgb_tunado = xgb.XGBClassifier(random_state=6)
# Fazendo o k-fold Cross-Validation
score_xgb_tunado = cross_val_score(clf_xgb_tunado, X_train, y_train, scoring='accuracy', cv=5, n_jobs=-1).mean()
# Definindo o range de parâmetros
xgb_search_space = {"max_depth": Integer(1,15),
                    "min_child_weight": Integer(0, 3),
                    "subsample": Real(0.1, 1.0),
                    "colsample_bytree": Real(0.1, 1.0),
                   }
# Fitando o Modelo e Medindo a Performance
xgb_bayes_search = BayesSearchCV(clf_xgb_tunado, xgb_search_space, scoring='accuracy', cv=5)
xgb_bayes_search.fit(X_train, y_train)
```

```
BayesSearchCV(cv=5,
              estimator=XGBClassifier(base_score=None, booster=None,
                                      callbacks=None, colsample_bylevel=None,
                                      colsample_bynode=None,
                                      colsample_bytree=None,
                                      early_stopping_rounds=None,
                                      enable_categorical=False,
                                      eval_metric=None, feature_types=None,
                                      gamma=None, gpu_id=None, grow_policy=None,
                                      importance_type=None,
                                      interaction_constraints=None,
                                      learning_rate=None...
                                      random_state=6, ...),
              scoring='accuracy',
              search_spaces={'colsample_bytree': Real(low=0.1, high=1.0, prior='uniform', transform='normalize'),
                             'max_depth': Integer(low=1, high=15, prior='uniform', transform='normalize'),
                             'min_child_weight': Integer(low=0, high=3, prior='uniform', transform='normalize'),
```

```
# Configuração de parâmetro que deu os melhores resultados nos dados de espera
xgb_bayes_search.best_params_
```

```
OrderedDict([('colsample_bytree', 0.8354316780693142),
             ('max_depth', 8),
             ('min_child_weight', 0),
             ('subsample', 0.7139862179864905)])
```

```
# Estimador que deu maior pontuação ou menor perda nos dados deixados de fora
xgb_bayes_search.best_estimator_
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8354316780693142, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=8, max_leaves=None,
              min_child_weight=0, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=6, ...)
```

```
# Calculando a acurácia
print('acuráia: %.2f' % (np.mean(xgb_bayes_search.best_score_)*100))
```

```
acuráia: 92.75
```

A otimização bayesiana melhorou a acurácia do algoritmo de XGBoost neste projeto em 0,03% (de 92,72 para 92,75).

Finalizando a etapa de treino dos modelos de ML, identificou-se que o melhor algoritmo de aprendizado de máquina para esse projeto é o XGBoost e que há ganho de performance com a otimização de seus hiper parâmetros. A melhor acurácia obtida no treino foi de 92,75% (erro de 7,25%) com o classificador xgb_bayes_search.best_estimator_, como destacado na tabela abaixo.

**Tabela 3:** Resumo dos valores de acurácia e erro obtidos nos treinos

| Algoritmo | Classificador | Acurácia | Erro |
|---|---|---|---|
| Árvores de Decisão | clf_dt=tree.DecisionTreeClassifier() | 88,85 | 11,15 |
| Árvores de Decisão | dt_bayes_search.best_estimator_ | 89,13 | 10,87 |
| Regressão Logística | clf_lr = LogisticRegression() | 80,44 | 19,56 |
| Naive Bayes | clf_nb = GaussianNB() | 78,72 | 21,28 |
| XGBoost | clf_xgb = xgb.XGBClassifier() | 92,72 | 7,28 |
| **XGBoost** | **xgb_bayes_search.best_estimator_** | **92,75** | **7,25** |

## 5.3 Avaliação do Modelo de Aprendizado de Máquina

Neste trabalho adotou-se o aprendizado supervisionado, que consiste em aprender a ligação entre dois conjuntos de dados: os dados observados (X) e uma variável externa (y) a qual se deseja prever, geralmente chamada de “alvo” ou “rótulos”. Nesse caso, os dados observados são os valores das colunas (atributos relacionados aos produtos: peso, as três dimensões e preço) e a saída é a categoria de cada produto (cama, mesa e banho ou beleza e saúde). As amostras pertencem a duas classes e queremos aprender com os dados já rotulados como prever uma classe de dados não rotulados.

Então, após ajustar os modelos e treiná-los, chegou o momento de avaliar o desempenho de ambos os modelos com dados diferentes daqueles nos quais foram treinados, ou seja, com os 30% de dados da base geral reservados para o teste. A figura 26 apresenta a previsão das classes sendo realizada com o objeto do *scikit-learn,* “*.fit(X, y)predict(T)”.*

**Figura 26 –** Previsão das classes

```
# Realizando a predição da base de teste utilizando o melhor estimador encontrado
xgb_bayes_search.best_estimator_.fit(X_train, y_train)
y_pred = xgb_bayes_search.best_estimator_.predict(X_test)
y_pred[:10]
```

```
array([1, 0, 0, 1, 0, 1, 0, 1, 0, 1])
```

Uma análise mais detalhada do modelo passa pela Matriz de Confusão (*Confusion Matrix*), em que é utilizada uma classificação binária (Sim ou Não) dos rótulos do modelo e dos dados reais observados. Dessa maneira, ao se utilizar o modelo de classificação em dados já observados, é possível verificar o quanto que o modelo conseguiu predizer corretamente. Os 4 valores que compões a Matriz de Confusão são detalhados a seguir e podem ser visualizados na figura 27, gerada para o melhor modelo deste trabalho. É preciso atenção para o fato de que a classe “0” (cama, mesa e banho) entra nesse projeto como negativa e a classe “1” (beleza e saúde) como positiva. Trata-se, portanto, de uma classificação de classes categóricas, em que não há maior relevância de uma das classes.

True Positive (TP): 93% dos valores classificados como classe “1” (beleza e saúde) nos dados originais foram corretamente previstos como classe “1” no modelo.

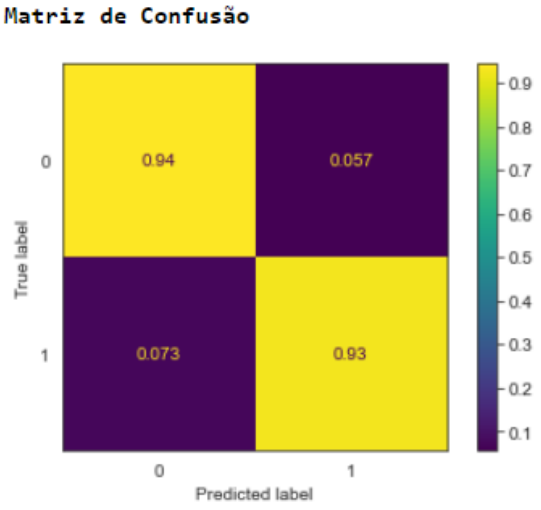
True Negatives (TN): 94% dos valores classificados como classe “0” (cama, mesa e banho) nos dados originais foram corretamente previstos como classe “0” no modelo.

False Positives (FP): 5,7% dos valores classificados como classe “0” (cama, mesa e banho) nos dados originais foram erroneamente previstos como classe “1” (beleza e saúde) no modelo.

False Negatives (FN): 7,3% dos valores classificados como classe “1” (beleza e saúde) nos dados originais foram erroneamente previstos como classe “0” (cama, mesa e banho) no modelo.

**Figura 27 –** Matriz de Confusão

```
# Visualizando a matriz de confusão
cm = confusion_matrix(y_test,y_pred, normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=xgb_bayes_search.best_estimator_.classes_)
print(f"{n}Matriz de Confusão")
disp.plot()
plt.show()
```




Por fim, é possível observar o desempenho dos modelos pelas métricas de acurácia, erro, precisão, *recall* e F1 Score. Para a classificação, a acurácia e o erro são as medidas básicas de desempenho. Quanto maior a acurácia do modelo, maior serão os acertos e menores serão os erros cometidos. A acurácia é calculada pelo total de observações classificadas corretamente dividido pelo total de observações classificadas. O erro é o complementar da acurácia.

Na figura 28 são apresentadas diversas métricas de avalição do modelo vencedor, a saber, o XGBoost (*Xtreme Gradient Boosting***)** com otimização bayesiana (detalhado no tópico 5.2.3).

**Figura 28 –** Avaliação do modelo

```
# Avaliação do modelo
total = len(y_test)
acertos = (y_pred == y_test).sum()
erros = total-acertos
accuracy = (100*acertos/total).round(2)
precision = (100*precision_score(y_test, y_pred)).round(2)
recall = (100*recall_score(y_test, y_pred)).round(2)
f1 = (100*f1_score(y_test, y_pred)).round(2)

print(f"{n}Avaliação do modelo\n{r}")
print("Tamanho total da base de teste:" +str(total)+ " produtos")
print("Acertos da predição: " +str(acertos)+ " produtos, ou seja, " +str((100*(acertos/total)).round(2))+ "%.")
print("Total de erros na predição: " +str(erros)+ " produtos, ou seja, " +str((100*(erros/total)).round(2))+ "%.\n")
# Acurácia: soma dos acertos, dividido pela soma de todos os resultados, que é também o número total de amostras.
print("Acurácia:", accuracy)
# Precisão: razão entre o número de verdadeiros positivos e o número total de previsões positivas feitas pelo modelo.
print("Precisão:", precision)
# Recall: razão entre o número de verdadeiros positivos e o número total de casos positivos.
print("Recall:", recall)
# F1 Score: média harmônica entre a precisão e o recall.
print("F1 Score:", f1)
```

```
Avaliação do modelo

Tamanho total da base de teste:1219 produtos
Acertos da predição: 1140 produtos, ou seja, 93.52%.
Total de erros na predição: 79 produtos, ou seja, 6.48%.

Acurácia: 93.52
Precisão: 94.41
Recall: 92.73
F1 Score: 93.56
```

## 6. Interpretação dos Resultados

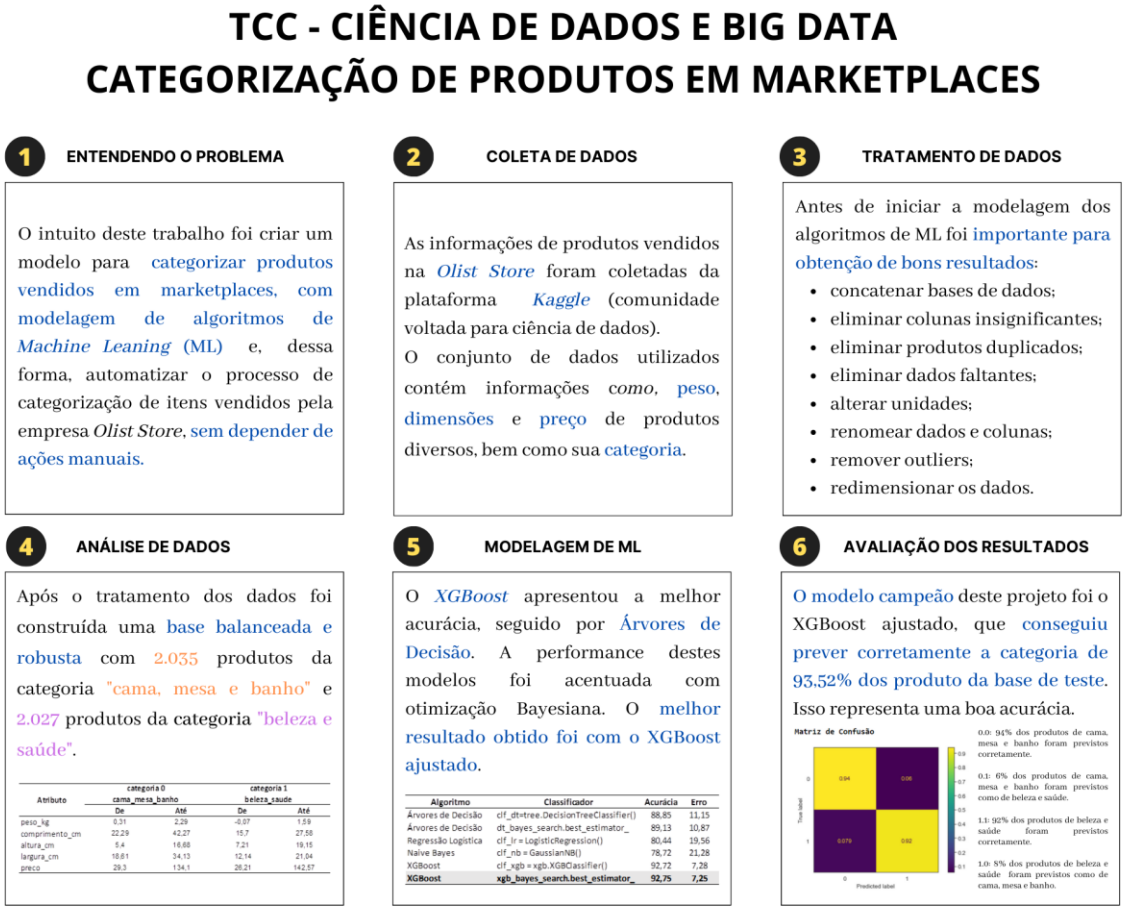
O objetivo deste projeto foi encontrar um modelo de algoritmo de *Machine Learning* capaz de prever uma classe de dados não rotulados, a partir do treinamento supervisionado de uma base de dados rotulada. O conjunto de dados utilizado foi o de produtos vendidos pela *Olist Store*, com características de peso, dimensões e preço e com os rótulos de categoria (neste caso treinamos apenas as categorias cama_mesa_banho e beleza_saude).

Tendo em vista o alvo proposto neste trabalho, conclui-se que ele foi bem alcançado pelo algoritmo XGBoost (Xtreme Gradient Boosting) com otimização bayesiana. Esse modelo apresentou 92,75% de acurácia contra 7,25% de erro no treino e 93,52% de acurácia contra 6,48% de erro na predição. Para relembrar, a tabela 4 apresenta a melhor parametrização encontrada pelo modelo para o “xgb_bayes_search.best_estimator_”, que teve os hiper parâmetros “colsample_bytree”, “max_depth”, “min_child_weight” e “subsample“ treinados pelo modelo e identificados pelo BayesSearchCV.

**Tabela 4 –** Melhor parametrização encontrada para o modelo

| Parâmetros | Melhor valor encontrado | Parâmetros | Melhor valor encontrado |
|---|---|---|---|
| base_score | None | learning_rate | None |
| booster | None | max_bin | None |
| callbacks | None | max_cat_threshold | None |
| colsample_bylevel | None | max_cat_to_onehot | None |
| colsample_bynode | None | max_delta_step | None |
| colsample_bytree | 0,835431678 | max_depth | 8 |
| early_stopping_rounds | None | max_leaves | None |
| enable_categorical | False | min_child_weight | 0 |
| eval_metric | None | missing | nan |
| feature_types | None | monotone_constraints | None |
| gamma | None | n_estimators | 100 |
| gpu_id | None | n_jobs | None |
| grow_policy | None | num_parallel_tree | None |
| importance_type | None | predictor | None |
| interaction_constraints | None | subsample | 0,713986218 |

## 7. Apresentação dos Resultados



## 8. Links

Aqui estão os links para o vídeo com a minha apresentação de 5 minutos e para o repositório contendo os materiais elaborados no projeto.

Link para o vídeo: https://youtu.be/blkPAIrJGbM

Link para o repositório: https://github.com/MarianaSilvaCosta/Ciencia-de-Dados

## REFERÊNCIAS


Kaggle.PlataformadaKaggle,2023. URL https://olist.com/pt-br/

OlistStore. SitedaOlistStore,2023. URL https://olist.com/pt-br/

Python. Pythondocumentation,2023. URL https://docs.python.org/3/

Pandas-pydata.Pandasdocumentation,2023. URL https://pandas.pydata.org/docs/index.html

Numpy.Numpydocumentation,2023. URL https://numpy.org/doc/

Seaborn. Seaborndocumentation,2023. URL https://seaborn.pydata.org/index.html

scikit-learn.scikit-learndocumentation,2023. URL https://scikit-learn.org/stable/#

scikit-optmize.scikit-optmizedocumentation,2023. URL
https://scikit-optimize.github.io/stable/index.html

XGBoost.Xgboostdocumentation,2023. URL
https://xgboost.readthedocs.io/en/stable/index.html

data-to-viz-.data-to-viz,2023. URL https://www.data-to-viz.com/

UFAC.cursoPythonUFACgithub,2023. URL https://cursopythonufac.github.io/

StackOverflow.SiteStackOverflow,2023. URL https://stackoverflow.com/questions/

HashtagTreinamentos.BlogHashtagTreinamentos,2023. URL
https://www.hashtagtreinamentos.com/blog

ThiagoNunes.PortaldoThiagoNunes,2023. URL https://www.thiagocarmonunes.com.br/

TatianaEscovedo.BlogMedium-TatianaEscovedo,2023. URL https://tatianaesc.medium.com/

MarioFilho.SitedoMarioFilho-forecastegy,2023. URL https://forecastegy.com/

InPlainEnglish.BlogdaInPlainEnglish,2023. URL https://plainenglish.io/blog

# APÊNDICE

**Programação/Scripts**

<h1 style="font-size:24px;">TCC - CIÊNCIA DE DADOS E BIG DATA</h1>
<h1 style="font-size:24px;">CATEGORIZAÇÃO DE PRODUTOS - DATASET DA OLIST</h1>

__Nome da aluna__: Mariana Silva Costa
__Matrícula__: 1162946
__Telefone__: (21) 9 8862-6509
__E-mail__: 1358597@sga.pucminas.br

**1. IMPORTAÇÃO DE BIBLIOTECAS**

```
# Importando bibliotecas básicas
import pandas as pd   #processamento de dados
from ydata_profiling import ProfileReport #análise de dados
import numpy as np   #álgebra linear
import seaborn as sns #visualização baseada em matplotlib
sns.set_style("white") #gráficos sem grade
import matplotlib.pyplot as plt #biblioteca base de visualização
n = '\033[1m' #codificação de fonte em negrito
r = '\033[22m' #redefenir a formatação em negrito

# Importando a biblioteca scikit-learn e xgboost
# Algoritimos de aprendizado de máquina, pré-processamento, seleção de modelos, métricas de desempenho, ...
from sklearn.model_selection import KFold, cross_validate, cross_val_score, train_test_split, GridSearchCV
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score,
recall_score
from sklearn.metrics import precision_score, f1_score, make_scorer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
import xgboost as xgb

# Importando a biblioteca scikit-optimize
# Utilizada para Otimização Bayesiana de hiper parâmetros
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer

**2. COLETA DE DADOS**

# Importando os datasets de produtos e itens da Olist
path = 'C://Users//user//Desktop//Pos_Ciencia_de_Dados\//TCC//Projeto'
products_df = pd.read_csv(f'{path}//olist_products_dataset.csv')
items_df = pd.read_csv(f'{path}//olist_order_items_dataset.csv')

# Criando um dataframe compilado
df = products_df
df = pd.merge(df, items_df,  how='left', on='product_id')
df.head()

**3. PROCESSAMENTO/TRATAMENTO DE DADOS**

# Análises do dataframe compilado
profile = ProfileReport(df, title="Análise do dataframe compilado")
profile
```

```
# Exportando a análise
profile.to_file("Relatorio_base.html")

# Visualizando os tipos de colunas
df.dtypes

# Calculando os valores ausentes por colunas
df.isnull().sum()

# Eliminando duplicadas da coluna chave (product_id)
df = df.drop_duplicates('product_id')
# Eliminando linhas que contém elementos nulos
df = df.dropna()
df.info()

# Convertendo a unidade de peso de grama para quilo
df['product_weight_g'] = df['product_weight_g'].apply(lambda x: x/1000)
df = df.rename(columns={'product_weight_g': 'product_weight_kg'})

# Indentificando a quantidade de produtos em cada categoria
df.product_category_name.value_counts()

# Criando um dataframe enxuto
df1 = df
df1.drop(columns=['product_name_lenght', 'product_description_lenght',
'product_photos_qty', 'order_id', 'order_item_id', 'seller_id', 'shipping_limit_date',
'freight_value'], inplace=True)

# Filtrando apenas 2 categorias de produtos
df1 = df1[(df1['product_category_name'] == 'cama_mesa_banho') |
(df1['product_category_name'] == 'beleza_saude')]
```

```
# Transformando a coluna de categoria em tipo numérico
df1 = df1.replace({'cama_mesa_banho':0, 'beleza_saude':1})

# Simplificando os nomes das colunas no português
df1.rename(columns={'product_category_name':'categoria', 'product_weight_kg':'peso_kg',
'product_length_cm':'comprimento_cm', 'product_height_cm':'altura_cm',
'product_width_cm':'largura_cm', 'price':'preco'}, inplace=True)
display(df1)

# Plotando as variáveis
fig = plt.figure(figsize=(15, 15))
# Gráfico 1 - Barras de contagem
fig.add_subplot(231)
sns.countplot(df1['categoria'])
# Gráfico 2 - Boxplot de peso
fig.add_subplot(232)
sns.boxplot(data=df1, x='categoria', y='peso_kg')
# Gráfico 2 - Boxplot de comprimento
fig.add_subplot(233)
sns.boxplot(data=df1, x='categoria', y='comprimento_cm')
# Gráfico 3 - Boxplot de altura
fig.add_subplot(234)
sns.boxplot(data=df1, x='categoria', y='altura_cm')
# Gráfico 4 - Boxplot de largura
fig.add_subplot(235)
sns.boxplot(data=df1, x='categoria', y='largura_cm')
# Gráfico 4 - Boxplot de preço
fig.add_subplot(236)
sns.boxplot(data=df1, x='categoria', y='preco')
print(f'{n}Boxplot das variáveis analisadas')
plt.show()
```

```
# Removendo outliers das variáveis
peso_q95 = df1['peso_kg'].quantile(0.95)
df1 = df1[df1['peso_kg'] < peso_q95]
comprimento_q95 = df1['comprimento_cm'].quantile(0.95)
df1 = df1[df1['comprimento_cm'] < comprimento_q95]
altura_q95 = df1['altura_cm'].quantile(0.95)
df1 = df1[df1['altura_cm'] < altura_q95]
largura_q95 = df1['largura_cm'].quantile(0.95)
df1 = df1[df1['largura_cm'] < largura_q95]
preco_q95 = df1['preco'].quantile(0.95)
df1 = df1[df1['preco'] < preco_q95]

# Verificando como ficou o dataframe
df1.info()
df1.categoria.value_counts()

**Normalização - Redimensionamento / Scaler**

# Criando um dataframe apenas com as colunas numéricas e categoria
numeric_df = df1.drop(['product_id'],axis=1)
# Criando o scaler
scaler = RobustScaler()
# Fazendo o fit com os dados
scaler = scaler.fit(numeric_df[['peso_kg']])
scaler = scaler.fit(numeric_df[['comprimento_cm']])
scaler = scaler.fit(numeric_df[['altura_cm']])
scaler = scaler.fit(numeric_df[['largura_cm']])
scaler = scaler.fit(numeric_df[['preco']])
# Fazendo a transformação e criando columns_robust
numeric_df['peso_kg_robust'] = scaler.transform(numeric_df[['peso_kg']])
```

```
numeric_df['comprimento_cm_robust'] =
scaler.transform(numeric_df[['comprimento_cm']])
numeric_df['altura_cm_robust'] = scaler.transform(numeric_df[['altura_cm']])
numeric_df['largura_cm_robust'] = scaler.transform(numeric_df[['largura_cm']])
numeric_df['preco_robust'] = scaler.transform(numeric_df[['preco']])
# Visualizando novamente os dados
numeric_df.describe()

**4. ANÁLISE E EXPLORAÇÃO DOS DADOS**

# Analisando as informações estatísticas do df1-cama_mesa_banho
print(f'{n}Informações Estatísticas da categoria cama, mesa e banho:')
cmb = pd.DataFrame(data=df1)
cmb.loc[cmb['categoria'] == 0].describe().round(2)

# Analisando as informações estatísticas do df1-beleza_saude
print(f'{n}Informações Estatísticas da categoria beleza e saúde:')
bs = pd.DataFrame(data=df1)
bs.loc[bs['categoria'] == 1].describe().round(2)

# Criando um dataframe apenas com as colunas robust e categoria
robust_df = numeric_df.drop(['peso_kg', 'comprimento_cm', 'altura_cm', 'largura_cm',
'preco'], axis=1)

# Calculando a correlação emparelhada de colunas
corr = robust_df.corr()

# Plotando um Mapa de Correlação das robust_columns
plt.figure(figsize=(10, 8))
sns.heatmap(corr, cmap='RdBu', norm=plt.Normalize(-1,1), annot=True, fmt='.2f')
print(f'{n}Mapa de Correlação')
plt.show()
```

```
# Pairplot das robust_columns
sns.pairplot(robust_df, hue="categoria", markers=["o", "D"])
print(f"{n}Pairplot das colunas numéricas")
plt.show()
```

**5. CRIAÇÃO E AVALIAÇÃO DE MODELOS DE MACHINE LEARNING**

**5.1 TREINAMENTO DO MODELO**

```
# Separando X e y
X = robust_df.drop(['categoria'],axis=1)
y = robust_df.categoria

# Selecionando os conjuntos de treinamento (70%) e teste (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

# Tamanhos de x e y de treino e de teste
X_train.shape, X_test.shape

# Ajustes da validação cruzada sem otimização de hiper parâmetros
cv = KFold(n_splits=10,random_state=42,shuffle=True)
```

**I. Árvore de decisão**

Sem Otimização de Hiperparâmetros

```
# Criando o classificador
clf_dt = tree.DecisionTreeClassifier(random_state=1)
# Calculando a acurácia e o desvio padrão
score_dt = cross_val_score(clf_dt, X_train, y_train, scoring='accuracy', cv=cv, n_jobs=-1)
print('acuráia: %.2f | desvio padrão: %.3f' % (np.mean(score_dt)*100, np.std(score_dt)))
```

Com Otimização Bayesiana

```
# Criando o classificador
clf_dt_tunado = tree.DecisionTreeClassifier(random_state=2)
# Fazendo o k-fold Cross-Validation
score_dt_tunado = cross_val_score(clf_dt_tunado, X_train, y_train, scoring='accuracy', cv=5,
n_jobs=-1)
# Definindo o range de parâmetros
dt_search_space = {"criterion": Categorical(['gini','entropy']),
                   "splitter": Categorical(['best', 'random']),
                   "max_depth": Integer(3, 15),
                   "max_features": Categorical(['auto', 'sqrt', 'log2']),
                  }
# Fitando o Modelo e Medindo a Performance
dt_bayes_search = BayesSearchCV(clf_dt_tunado, dt_search_space, scoring='accuracy',
cv=5)
dt_bayes_search.fit(X_train, y_train)

# Configuração de parâmetro que deu os melhores resultados nos dados de espera
dt_bayes_search.best_params_

# Estimador que deu maior pontuação ou menor perda nos dados deixados de fora
dt_bayes_search.best_estimator_

# Calculando a acurácia
print('acuráia: %.2f' % (np.mean(dt_bayes_search.best_score_)*100))
```

**II. Regressão Logística**

Sem Otimização de Hiperparâmetros

```
# Criando o classificador
clf_lr = LogisticRegression(random_state=3, max_iter=1000)
# Calculando a acurácia e o desvio padrão
score_lr = cross_val_score(clf_lr, X_train, y_train, scoring='accuracy', cv=cv)
print('acuráia: %.2f | desvio padrão: %.3f' % (np.mean(score_lr)*100, np.std(score_lr)))
```

**III. Naive Bayes**

Sem Otimização de Hiperparâmetros

```
# Criando o classificador
clf_nb = GaussianNB()
# Calculando a acurácia e o desvio padrão
score_nb = cross_val_score(clf_nb, X_train, y_train, scoring='accuracy', cv=cv)
print('acuráia: %.2f | desvio padrão: %.3f' % (np.mean(score_nb)*100, np.std(score_nb)))
```

**IV. XGBoost - Xtreme Gradient Boosting**

Sem Otimização de Hiperparâmetros

```
# Criando o classificador
clf_xgb = xgb.XGBClassifier(random_state=5)
# Calculando a acurácia e o desvio padrão
scores_xgb = cross_val_score(clf_xgb, X_train, y_train, scoring='accuracy', cv=cv)
print('acuráia: %.2f | desvio padrão: %.3f' % (np.mean(scores_xgb)*100,
np.std(scores_xgb)))
```

Com Otimização Bayesiana

```
# Criando o classificador
clf_xgb_tunado = xgb.XGBClassifier(random_state=6)
# Fazendo o k-fold Cross-Validation
```

```
score_xgb_tunado = cross_val_score(clf_xgb_tunado, X_train, y_train, scoring='accuracy',
cv=5, n_jobs=-1).mean()
# Definindo o range de parâmetros
xgb_search_space = {"max_depth": Integer(1,15),
                    "min_child_weight": Integer(0, 3),
                    "subsample": Real(0.1, 1.0),
                    "colsample_bytree": Real(0.1, 1.0),
                   }
# Fitando o Modelo e Medindo a Performance
xgb_bayes_search = BayesSearchCV(clf_xgb_tunado, xgb_search_space, scoring='accuracy',
cv=5)
xgb_bayes_search.fit(X_train, y_train)


# Configuração de parâmetro que deu os melhores resultados nos dados de espera
xgb_bayes_search.best_params_

# Estimador que deu maior pontuação ou menor perda nos dados deixados de fora
xgb_bayes_search.best_estimator_

# Calculando a acurácia
print('acuráia: %.2f' % (np.mean(xgb_bayes_search.best_score_)*100))

**5.2 AVALIAÇÃO DO MODELO**

O MELHOR CLASSIFICADOR >> xgb_bayes_search.best_estimator_

# Realizando a predição da base de teste utilizando o melhor estimador encontrado
xgb_bayes_search.best_estimator_.fit(X_train, y_train)
y_pred = xgb_bayes_search.best_estimator_.predict(X_test)
y_pred[:10]
```

```
# y de teste(real)
y_test[:10]

# Visualizando onde a predição não foi correta.
X_test[y_test != y_pred]

# Visualizando a matriz de confusão
cm = confusion_matrix(y_test,y_pred, normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=xgb_bayes_search.best_estimator_.classes_)
print(f"{n}Matriz de Confusão")
disp.plot()
plt.show()

# Avaliação do modelo
total = len(y_test)
acertos = (y_pred == y_test).sum()
erros = total-acertos
accuracy = (100*acertos/total).round(2)
precision = (100*precision_score(y_test, y_pred)).round(2)
recall = (100*recall_score(y_test, y_pred)).round(2)
f1 = (100*f1_score(y_test, y_pred)).round(2)

print(f"{n}Avaliação do modelo\n{r}")
print("Tamanho total da base de teste:" +str(total)+ " produtos")
print("Acertos da predição: " +str(acertos)+ " produtos, ou seja, "
+str((100*(acertos/total)).round(2))+ "%.")
print("Total de erros na predição: " +str(erros)+ " produtos, ou seja, "
+str((100*(erros/total)).round(2))+ "%.\n")
# Acurácia: soma dos acertos, dividido pela soma de todos os resultados, que é também o
número total de amostras.
print("Acurácia:", accuracy)
```

```
# Precisão: razão entre o número de verdadeiros positivos e o número total de previsões positivas feitas pelo modelo.
print("Precisão:", precision)
# Recall: razão entre o número de verdadeiros positivos e o número total de casos positivos.
print("Recall:", recall)
# F1 Score: média harmônica entre a precisão e o recall.
print("F1 Score:", f1)
```