

## ▼ Tech Challenge 4 - Data viz and production models

### Desafio

#### 🎯 Objetivo do Projeto

Desenvolver um modelo preditivo de Machine Learning para auxiliar uma equipe médica a diagnosticar níveis de obesidade em pacientes, com assertividade mínima de 75%, deployado em uma aplicação Streamlit, acompanhado de dashboard analítico e vídeo de apresentação.

#### ✳️ Entregáveis Principais

- Pipeline de Machine Learning
- Feature engineering, treinamento e avaliação do modelo.
- Modelo com acurácia  $\geq 75\%$
- Deploy no Streamlit
- Aplicação funcional acessível por link público.
- Dashboard analítico
- Visualização dos principais insights.
- Documentação (.doc ou .txt)
- Links do app, dashboard e repositório GitHub.
- Vídeo de apresentação (4 a 10 minutos)

## ▼ Bibliotecas

```
import pandas as pd
import re
import json
import os
import requests
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import numpy as np
from scipy.stats import chisquare
from scipy.stats import chi2_contingency
import scipy.stats as stats
from typing import List, Optional
from IPython.display import display, HTML
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder
```

## ▼ Funções

### Dicionario dados

```
def criar_novo_dicionario_dados(df: pd.DataFrame) -> dict:
    """
    Cria um novo dicionário de dados com uma estrutura inicial para cada coluna do DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame cujas colunas serão usadas para criar o dicionário.

    Returns:
        dict: O novo dicionário de dados.
    """
    try:
        novo_dicionario = {}
        for coluna in df.columns:
```

```

novo_dicionario[coluna] = {
    'nome_original': coluna,
    'tipo_coluna': 'original',
    'novo_nome': None,
    'descricao': None,
    'transformacao': {}
}
print("Novo dicionário de dados criado.")
return novo_dicionario
except Exception as e:
    print(f"Ocorreu um erro ao criar o novo dicionário de dados: {e}")
    return None

```

## ▼ Atualiza de forma manual das colunas do dicionário de dados

```

def atualizar_campo_dicionario(dicionario: dict, coluna: str, campo: str, novo_valor):
    """
    Atualiza um campo específico para uma coluna no dicionário de dados.

    Args:
        dicionario (dict): O dicionário de dados a ser atualizado.
        coluna (str): O nome da coluna (chave no dicionário) cujo campo será atualizado.
        campo (str): O nome do campo a ser atualizado ('nome_original', 'novo_nome', 'descricao', 'transformacao').
        novo_valor: O novo valor para o campo especificado.

    """
    try:
        if coluna in dicionario:
            if campo == 'nome_original':
                print(f"Erro: Não é permitido atualizar o campo 'nome_original'.")
            elif campo in dicionario[coluna]:
                if campo == 'transformacao' and isinstance(novo_valor, dict):
                    if not isinstance(dicionario[coluna][campo], dict):
                        print(f"Aviso: O campo 'transformacao' para a coluna '{coluna}' não é um dicionário. Inicializar"
                            "dicionario[{coluna}][{campo}] = {}")
                    dicionario[coluna][campo].update(novo_valor)
                    print(f"Campo '{campo}' da coluna '{coluna}' atualizado com novos itens.")
                else:
                    dicionario[coluna][campo] = novo_valor
                    print(f"Campo '{campo}' da coluna '{coluna}' atualizado para: '{novo_valor}'")
            else:
                print(f"Erro: O campo '{campo}' não existe para a coluna '{coluna}' no dicionário de dados.")
        else:
            print(f"Erro: A coluna '{coluna}' não foi encontrada no dicionário de dados.")
    except Exception as e:
        print(f"Ocorreu um erro ao atualizar o campo '{campo}' para a coluna '{coluna}': {e}")

```

## ▼ Atualiza nomes dicionario

```

def atualizar_dicionario_renomeacao(dicionario: dict, mapa_colunas: dict):
    """
    Atualiza o dicionário de dados com base nas renomeações de colunas aplicadas.

    Args:
        dicionario (dict): O dicionário de dados a ser atualizado.
        mapa_colunas (dict): Dicionário com o mapeamento de renomeação de colunas (nome_original: novo_nome).

    """
    try:
        print("Iniciando a atualização do dicionário de dados com renomeações...")
        chaves_originais = list(dicionario.keys())

        for chave_original in chaves_originais:
            if chave_original in dicionario:
                info = dicionario[chave_original]
                nome_original = info.get('nome_original')
                novo_nome_existente = info.get('novo_nome')

                if nome_original in mapa_colunas:
                    atualiza_novo_nome = mapa_colunas[nome_original]

                    if chave_original != atualiza_novo_nome and (novo_nome_existente is None or novo_nome_existente != atualiza_novo_nome):
                        dicionario[atualiza_novo_nome] = dicionario.pop(chave_original)
                        info = dicionario[atualiza_novo_nome] #
                        info['novo_nome'] = atualiza_novo_nome

```

```

        if not isinstance(info.get('transformacao'), dict):
            info['transformacao'] = {}

        renome_desc = f"Renomeada de '{nome_original}' para '{atualiza_novo_nome}'."
        if renome_desc not in info['transformacao'].values():
            transform_key = f"transformacao_{len(info['transformacao']) + 1}"
            info['transformacao'][transform_key] = renome_desc

    elif novo_nome_existente is not None and chave_original != novo_nome_existente:
        if novo_nome_existente in dicionario:

            print(f"Aviso: Conflito ao renomear. '{chave_original}' tem novo nome '{novo_nome_existente}', mas"
            else:
                dicionario[novo_nome_existente] = dicionario.pop(chave_original)
                info = dicionario[novo_nome_existente]

    print("Dicionário de dados atualizado com renomeações.")

except Exception as e:
    print(f"Ocorreu um erro ao atualizar o dicionário de dados com renomeações: {e}")

```

## ▼ Atualiza Dicionario com as novas colunas criadas

```

def atualizar_dicionario_novas_colunas(dicionario: dict, df: pd.DataFrame):
    """
    Atualiza o dicionário de dados com informações sobre colunas recém-criadas no DataFrame.

    Args:
        dicionario (dict): O dicionário de dados a ser atualizado.
        df (pd.DataFrame): O DataFrame com as colunas recém-criadas.
    """

    print("Iniciando a atualização do dicionário de dados com novas colunas...")
    try:
        colunas_atuais = df.columns.tolist()

        existing_keys_and_new_names = set()
        for key, info in dicionario.items():
            existing_keys_and_new_names.add(key)
            if info.get('novo_nome'):
                existing_keys_and_new_names.add(info['novo_nome'])

        novas_colunas = [col for col in colunas_atuais if col not in existing_keys_and_new_names]

        for nova_coluna in novas_colunas:

            dicionario[nova_coluna] = {
                'nome_original': nova_coluna,
                'tipo_coluna' : 'criada',
                'novo_nome': None,
                'descricao': None,
                'transformacao': {}
            }
            print(f"Adicionada nova coluna '{nova_coluna}' ao dicionário de dados.")

        print("Dicionário de dados atualizado com novas colunas.")
        return dicionario
    except Exception as e:
        print(f"Ocorreu um erro ao atualizar o dicionário com novas colunas: {e}")
        return None

```

## ▼ Atualiza campo descricao do dicionario em lote

```

def atualizar_descricoes_dicionario(dicionario: dict, descricoes: dict):
    """
    Atualiza o campo 'descricao' no dicionário de dados com base em um dicionário de descrições.

    Args:
        dicionario (dict): O dicionário de dados a ser atualizado.
        descricoes (dict): O dicionário contendo as descrições (chave: nome da coluna, valor: descrição).
    """

```

```

Returns:
    int: A quantidade de descrições atualizadas.
"""

quantidade_linhas_atualizadas = 0
try:

    if not isinstance(descricaoes, dict):
        print("Erro: O argumento 'descricaoes' deve ser um dicionário.")
        return 0

    for chave in dicionario:
        if chave in descricaoes:

            atualizar_campo_dicionario(dicionario, chave, 'descricao', descricaoes[chave])
            quantidade_linhas_atualizadas += 1

    print(f'Quantidade de descrições atualizadas: {quantidade_linhas_atualizadas}.')
    return quantidade_linhas_atualizadas
except Exception as e:
    print(f'Ocorreu um erro ao atualizar as descrições no dicionário: {e}')
    return 0

```

## ▼ Atualiza Transformação one hot encoding

```

def atualizar_dicionario_ohe(dicionario: dict, df: pd.DataFrame, coluna_original: str) -> None:
    """
    ohe = one hot encoding
    Atualiza o dicionário de dados com informações sobre colunas criadas por One-Hot Encoding.

    Args:
        dicionario (dict): O dicionário de dados a ser atualizado.
        df (pd.DataFrame): O DataFrame com as colunas One-Hot Encoded.
        coluna_original (str): O nome da coluna original que foi transformada (e também o prefixo usado para as novas colunas).
    """
    print(f"Iniciando a atualização do dicionário de dados para colunas OHE da variável '{coluna_original}'...")
    try:

        atualizar_dicionario_novas_colunas(dicionario, df)

        novas_colunas_ohe = [col for col in df.columns if col.startswith(f'{coluna_original}_')]
        info_coluna_original = dicionario.get(coluna_original, {})

        for coluna_ohe in novas_colunas_ohe:
            if coluna_ohe in dicionario:

                dicionario[coluna_ohe]['nome_original'] = coluna_original
                dicionario[coluna_ohe]['novo_nome'] = coluna_ohe
                dicionario[coluna_ohe]['tipo_coluna'] = 'criada_ohe'

                categoria = coluna_ohe.replace(f'{coluna_original}_', '')
                if info_coluna_original and info_coluna_original.get('descricao'):
                    dicionario[coluna_ohe]['descricao'] = f'Representa a categoria "{categoria}" da variável original "{coluna_original}"'
                else:
                    dicionario[coluna_ohe]['descricao'] = f'Variável dummy para a categoria "{categoria}" da coluna original "{coluna_original}"'

            if 'transformacao' not in dicionario[coluna_ohe] or not isinstance(dicionario[coluna_ohe]['transformacao'], dict):
                dicionario[coluna_ohe]['transformacao'] = {}

            transform_key_number = len(dicionario[coluna_ohe]['transformacao']) + 1
            transformacao_info = {
                f'transformacao_{transform_key_number}': f'Criada via One-Hot Encoding da coluna "{coluna_original}" ({len(novas_colunas_ohe)})'
            }
            atualizar_campo_dicionario(dicionario, coluna_ohe, 'transformacao', transformacao_info)
    except:
        print(f'Aviso: Coluna "{coluna_ohe}" não encontrada no dicionário de dados após adicionar novas colunas.')

    print(f'Dicionário de dados atualizado para colunas OHE da variável "{coluna_original}".')
except Exception as e:
    print(f'Ocorreu um erro ao atualizar o dicionário de dados com informações de OHE para a variável "{coluna_original}"')

```

## ✓ Atualiza dicionario alteracao ordinalEncoder

```

def atualizar_dicionario_ordinal_encoding(dicionario: dict, coluna_processada: str, coluna_original: str, mapa_encoding: dict):
    """
    Atualiza o dicionário de dados com informações sobre uma coluna processada por Ordinal Encoding.

    Args:
        dicionario (dict): O dicionário de dados a ser atualizado.
        coluna_processada (str): O nome da coluna no DataFrame após o Ordinal Encoding.
        coluna_original (str): O nome da coluna original antes do Ordinal Encoding.
        mapa_encoding (dict): O dicionário de mapeamento usado pelo OrdinalEncoder (valor original: valor codificado).
    """
    print(f"Iniciando a atualização do dicionário de dados para a coluna '{coluna_processada}' (Ordinal Encoding)...")
    try:
        if coluna_processada not in dicionario:
            dicionario[coluna_processada] = {
                'nome_original': coluna_original,
                'tipo_coluna': 'criada_encoded_ordinal',
                'novo_nome': coluna_processada,
                'descricao': f"Coluna codificada numericamente via Ordinal Encoding da coluna original '{coluna_original}'",
                'transformacao': {}
            }
            print(f"Adicionada nova entrada para a coluna '{coluna_processada}' ao dicionário.")
        else:
            dicionario[coluna_processada]['nome_original'] = coluna_original
            if dicionario[coluna_processada].get('tipo_coluna') != 'criada_encoded_ordinal':
                dicionario[coluna_processada]['tipo_coluna'] = 'transformada_encoded_ordinal' # Tipo se a coluna original

            if dicionario[coluna_processada].get('descricao') is None:
                dicionario[coluna_processada]['descricao'] = f"Coluna codificada numericamente via Ordinal Encoding da co"

        if 'transformacao' not in dicionario[coluna_processada] or not isinstance(dicionario[coluna_processada]['transformacao'], dict):
            dicionario[coluna_processada]['transformacao'] = {}

        transform_key_number = len(dicionario[coluna_processada]['transformacao']) + 1
        transformacion_info = {
            f'transformacao_{transform_key_number}': "Aplicado Ordinal Encoding",
            "tipo": "ordinal_encoding",
            "mapa_aplicado": mapa_encoding
        }
        atualizar_campo_dionario(dicionario, coluna_processada, 'transformacao', transformacion_info)

        print(f"Dicionário de dados atualizado para a coluna '{coluna_processada}'.")

    except Exception as e:
        print(f"Ocorreu um erro ao atualizar o dicionário de dados com informações de Ordinal Encoding para a coluna '{coluna_processada}'")

```

## ✓ Funções estatísticas

### ✓ Análise bi-variada para variável categórica

```

def analisar_variavel_categorica_bi_variada(
    df: pd.DataFrame,
    coluna_feature: str,
    coluna_alvo: str,
    ordem_linhas: Optional[List[str]] = None,
    ordem_colunas: Optional[List[str]] = None
) -> None:
    """
    Realiza a análise bivariada completa para uma feature categórica
    em relação a uma variável alvo categórica.

    A função irá:
    1. Executar e interpretar o Teste Qui-Quadrado de Independência.
    2. Plotar um Gráfico de Barras Empilhado 100% da feature vs. alvo.

    Argumentos:
        df (pd.DataFrame): O DataFrame completo.
        coluna_feature (str): O nome da coluna da feature (eixo X, ex: 'genero').
        coluna_alvo (str): O nome da coluna alvo (legendas, ex: 'classificacao_peso_corporal').
    """

```

```

ordem_linhas (Optional[List[str]]): Lista opcional para forçar uma ordem
específica nas barras do eixo X
(ex: ['Baixo', 'Medio', 'Alto']).
ordem_colunas (Optional[List[str]]): Lista opcional para forçar uma ordem
específica nas categorias da legenda
(ex: ['peso_insuficiente', 'peso_normal', ...]).
"""

print(f"--- Análise da Feature: '{coluna_feature}' vs. Alvo: '{coluna_alvo}' ---")
print("\n")

try:

    if coluna_feature not in df.columns:
        raise KeyError(f"Coluna da feature '{coluna_feature}' não encontrada no DataFrame.")
    if coluna_alvo not in df.columns:
        raise KeyError(f"Coluna alvo '{coluna_alvo}' não encontrada no DataFrame.")

    crosstab_contagem = pd.crosstab(df[coluna_feature], df[coluna_alvo])

    print("--- 1. Resultados do Teste Qui-Quadrado ---")

    try:
        chi2_stat, p_value, dof, expected_freqs = stats.chi2_contingency(crosstab_contagem)

        print(f"Estatística Qui-Quadrado ( $\chi^2$ ): {chi2_stat:.4f}")
        print(f"P-valor (p-value): {p_value:.4g}")
        print(f"Graus de Liberdade (dof): {dof}")
        print("\n")

        alpha = 0.05
        if p_value < alpha:
            print(f"Interpretação (p < {alpha}): REJEITA H0.")
            print("CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.")
        else:
            print(f"Interpretação (p >= {alpha}): FALHA EM REJEITAR H0.")
            print("CONCLUSÃO: NÃO há evidência de associação estatística entre as variáveis.")

    except ValueError as chi_err:
        print(f"AVISO: Não foi possível calcular o Teste Qui-Quadrado: {chi_err}")
        print("Isso pode ocorrer se alguma categoria tiver contagens esperadas nulas.")

    print("\n" + "="*80 + "\n")

    print("--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---")

    crosstab_pct = crosstab_contagem.apply(lambda r: r / r.sum(), axis=1)
    crosstab_plot = crosstab_pct.copy()

    if ordem_linhas:

        linhas_validas = [l for l in ordem_linhas if l in crosstab_plot.index]
        linhas_restantes = [l for l in crosstab_plot.index if l not in linhas_validas]

        crosstab_plot = crosstab_plot.reindex(index=linhas_validas + linhas_restantes)
        print(f"Ordenação de linhas (eixo X) aplicada: {linhas_validas + linhas_restantes}")

    if ordem_colunas:

        colunas_validas = [c for c in ordem_colunas if c in crosstab_plot.columns]
        colunas_restantes = [c for c in crosstab_plot.columns if c not in colunas_validas]

        crosstab_plot = crosstab_plot[colunas_validas + colunas_restantes]
        print(f"Ordenação de colunas (legenda) aplicada: {colunas_validas + colunas_restantes}")

    print('\n')
    display(round(crosstab_plot * 100,2))
    print('\n')

    num_cores = len(crosstab_plot.columns)
    colors = sns.color_palette("tab20", n_colors=num_cores)

    ax = crosstab_plot.plot(
        kind='bar'
    )

```

```

        stacked=True,
        figsize=(12, 8),
        color=colors
    )

    for c in ax.containers:
        labels = [f'{v*100:.1f}%' if (v > 0.01) else '' for v in c.datavalues]
        ax.bar_label(c,
                     labels=labels,
                     label_type='center',
                     color='white',
                     fontsize=9,
                     fontweight='bold')

    ax.yaxis.set_major_formatter(mtick.PercentFormatter(1.0))
    ax.set_ylim(0, 1)

    ax.set_title(f"Distribuição de '{coluna_alvo}' por '{coluna_feature}'", fontsize=16, pad=20)
    ax.set_xlabel(coluna_feature, fontsize=12)
    ax.set_ylabel(f"Proporção de '{coluna_alvo}'", fontsize=12)

    plt.xticks(rotation=0)

    ax.legend(title=coluna_alvo, bbox_to_anchor=(1.02, 1), loc='upper left')

    plt.tight_layout()
    plt.show()

except KeyError as ke:
    print(f"ERRO DE CHAVE: {ke}. Verifique se os nomes das colunas estão corretos.")
except Exception as e:
    print(f"ERRO INESPERADO ao analisar '{coluna_feature}': {e}")

```

## ▼ Teste ANOVA

```

def teste_anova(df: pd.DataFrame, agrupa_col: str, nome_col: str):
    """
    Realiza o teste de Análise de Variância (ANOVA) de uma via.

    Args:
        df (pd.DataFrame): O DataFrame contendo os dados.
        agrupa_col (str): O nome da coluna categórica que define os grupos.
        nome_col (str): O nome da coluna numérica cujos valores serão comparados entre os grupos.

    Returns:
        tuple: Uma tupla contendo a estatística F e o valor-p do teste ANOVA,
               ou (None, None) se ocorrer um erro ou as colunas não forem válidas.
    """
    if agrupa_col not in df.columns:
        print(f"Erro: A coluna de grupo '{agrupa_col}' não existe no DataFrame.")
        return None, None
    if nome_col not in df.columns:
        print(f"Erro: A coluna de valor '{nome_col}' não existe no DataFrame.")
        return None, None
    if not pd.api.types.is_numeric_dtype(df[nome_col]):
        print(f"Erro: A coluna de valor '{nome_col}' não é do tipo numérico. Não é possível realizar o teste ANOVA.")
        return None, None

    try:
        groups = df.groupby(agrupa_col)[nome_col].apply(list).dropna()

        if len(groups) < 2:
            print(f"Erro: É necessário ter pelo menos 2 grupos na coluna '{agrupa_col}' com dados na coluna '{nome_col}' para realizar o teste ANOVA.")
            return None, None

        fvalue, pvalue = stats.f_oneway(*groups)

        print(f"--- Resultados do Teste ANOVA ({nome_col} vs {agrupa_col}) ---")
        print(f"Estatística F: {fvalue:.4f}")
        print(f"P-valor: {pvalue:.4g}")

        alpha = 0.05
        if pvalue < alpha:
            print(f"Interpretação (p < {alpha}): REJEITA H0.")
            print(f"CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de '{nome_col}' entre os grupos.")
        else:
            print(f"Interpretação (p >= {alpha}): NÃO SE PODE REJEITAR H0.")
            print(f"CONCLUSÃO: Não existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de '{nome_col}' entre os grupos.")

    except Exception as e:
        print(f"Erro: Ocorreu um erro ao realizar o teste ANOVA: {e}")
        return None, None

```

```

print(f"Interpretação (p >= {alpha}): FALHA EM REJEITAR H0.")
print(f"CONCLUSÃO: NÃO há evidência de diferença estatística significativa nas médias de '{nome_col}' entre os

return fvalue, pvalue

except Exception as e:
    print(f"Ocorreu um erro durante o teste ANOVA: {e}")
    return None, None

```

## Função qui-quadrado

```

def realizar_teste_qui_quadrado(frequencias_observadas, f_esperadas=None, alpha=0.05):
    """
    Realiza o teste de Qui-Quadrado (Goodness-of-Fit ou Independência).

    Args:
        frequencias_observadas (array-like): As frequências observadas.
        f_esperadas (array-like, optional): As frequências esperadas.
            Se None, o teste é de aderência a uma distribuição uniforme.
        alpha (float, optional): O nível de significância para a interpretação. Padrão é 0.05.

    Returns:
        tuple: Uma tupla contendo a estatística Qui-Quadrado e o valor-p.
    """
    try:
        estatistica_qui2, valor_p = chisquare(f_obs=frequencias_observadas, f_exp=f_esperadas)

        print("\n--- Interpretação ---")
        print(f"Estatística Qui-Quadrado ( $\chi^2$ ): {estatistica_qui2:.4f}")
        print(f"Valor-p (p-value): {valor_p:.4f}")

        if valor_p <= alpha:
            print(f"Como p ({valor_p:.4f}) <= {alpha}, rejeitamos a Hipótese Nula (H0).")
            print("Conclusão: A distribuição NÃO é estatisticamente balanceada.")
        else:
            print(f"Como p ({valor_p:.4f}) > {alpha}, não podemos rejeitar a Hipótese Nula (H0).")
            print("Conclusão: A distribuição PODE ser considerada estatisticamente balanceada.")

        return estatistica_qui2, valor_p
    except ValueError as e:
        print(f"Erro nos dados de entrada: {e}")
        return None, None
    except Exception as e:
        print(f"Ocorreu um erro ao realizar o teste de Qui-Quadrado: {e}")
        return None, None

```

## Razão de balanceamento

```

def calcular_razao_desbalanceamento(quantidade_maxima: int, quantidade_minima: int) -> float:
    """
    Calcula a Razão de Desbalanceamento (Imbalance Ratio - IR) e fornece uma interpretação.

    Args:
        quantidade_maxima (int): A quantidade de ocorrências da classe majoritária.
        quantidade_minima (int): A quantidade de ocorrências da classe minoritária.

    Returns:
        float: A Razão de Desbalanceamento. Retorna float('inf') se quantidade_minima for zero.
    """
    if quantidade_minima <= 0:
        print("Erro: A quantidade mínima de ocorrências deve ser maior que zero para calcular a Razão de Desbalanceamento.")
        razao_desbalanceamento = float('inf')
    else:
        razao_desbalanceamento = quantidade_maxima / quantidade_minima

    print("\n--- Razão de Desbalanceamento (IR) ---")
    print(f"Quantidade da classe majoritária: {quantidade_maxima}")
    print(f"Quantidade da classe minoritária: {quantidade_minima}")
    print(f"Razão de Desbalanceamento (IR): {razao_desbalanceamento:.2f}")

    print("\n--- Interpretação da Razão de Desbalanceamento ---")

    if razao_desbalanceamento <= 2:
        print(f"🟢 Balanceado. Não faça nada.")
    elif 3 <= razao_desbalanceamento <= 10:
        print(f"🟡 Moderado. Fique atento, use F1-Score, talvez precise de class_weight.")

```

```

        elif razao_desbalanceamento > 10:
            print(f"🔴 Severo. Precisa de técnicas de balanceamento (ex: SMOTE).")
        else:
            # casos 2 < IR < 3
            print("Aviso: A Razão de Desbalanceamento está entre 2 e 3. Considere como Moderado.")
            print("3:1 a 10:1: (IR entre 3 e 10)🟡 Moderado. Fique atento, use F1-Score, talvez precise de class_weight.")

    return razao_desbalanceamento

```

## ▼ Funções de Transformação

### ▼ Renomear Colunas

```

def renomear_colunas(df: pd.DataFrame, mapa_renomeacao: dict = None, dicionario: dict = None) -> None:
    """
    Renomeia as colunas de um DataFrame utilizando um dicionário fornecido
    ou aplicando um padrão de formatação (minúsculas, espaços por '_').
    Se o padrão de formatação for aplicado, o dicionário de mapeamento gerado
    é salvo em um arquivo JSON chamado 'renaming_map_pattern.json'.
    Opcionalmente, atualiza um dicionário de dados com as renomeações.

    Args:
        df (pd.DataFrame): O DataFrame cujas colunas serão renomeadas.
        mapa_renomeacao (dict, optional): Um dicionário mapeando nomes de colunas originais para novos nomes.
                                            Se None, um padrão de formatação será aplicado.
        dicionario (dict, optional): O dicionário de dados a ser atualizado. Defaults to None.
    """
    colunas_originais = df.columns.tolist()
    num_linhas_antes, num_colunas_antes = df.shape

    print("Número de linhas e colunas antes da transformação:")
    print(f"Linhas: {num_linhas_antes}, Colunas: {num_colunas_antes}")
    print(f"\nNomes das colunas originais:\n{colunas_originais}\n")

    try:
        if mapa_renomeacao:
            if any(col in mapa_renomeacao for col in colunas_originais):
                df.rename(columns=mapa_renomeacao, inplace=True)
                print("Utilizado dicionário de renomeação fornecido.")
            else:
                print("Dicionário de renomeação fornecido, mas nenhuma chave corresponde às colunas existentes.")
                print("Aplicando padrão de formatação como alternativa.")

                mapa_nova_coluna = {}
                nova_coluna = []
                for col in colunas_originais:
                    col_caixa_baixa = col.lower()
                    col_limpa = col_caixa_baixa.strip()
                    col_formatada = re.sub(r'\s+', '_', col_limpa)
                    nova_coluna.append(col_formatada)
                    mapa_nova_coluna[col] = col_formatada
                df.columns = nova_coluna
                print("Padrão de formatação aplicado.")

                with open(f'mapeamento_coluna_{df}.json', 'w') as f:
                    json.dump(mapa_nova_coluna, f, indent=4)
                print(f"Dicionário de renomeação gerado pelo padrão salvo em f'mapeamento_coluna_{df}.json'.")

                mapa_renomeacao_aplicado = mapa_nova_coluna
            else:
                print("Nenhum dicionário de renomeação fornecido. Aplicando padrão de formatação.")
                mapa_nova_coluna = {}
                nova_coluna = []
                for col in colunas_originais:
                    col_caixa_baixa = col.lower()
                    col_limpa = col_caixa_baixa.strip()
                    col_formatada = re.sub(r'\s+', '_', col_limpa)
                    nova_coluna.append(col_formatada)
                    mapa_nova_coluna[col] = col_formatada
                df.columns = nova_coluna
                print("Padrão de formatação aplicado.")

                with open(f'mapeamento_coluna_{df}.json', 'w') as f:
                    json.dump(mapa_nova_coluna, f, indent=4)
                print(f"Dicionário de renomeação gerado pelo padrão salvo em f'mapeamento_coluna_{df}.json'.")

    except Exception as e:
        print(f"Ocorreu um erro ao renomear as colunas: {e}")

```

```

    mapa_renomeacao_aplicado = mapa_nova_coluna

    colunas_modificadas = df.columns.tolist()
    num_linhas_depois, num_colunas_depois = df.shape

    print(f"\nNomes das colunas modificadas:\n{colunas_modificadas}\n")
    print("Número de linhas e colunas após a transformação:")
    print(f"Linhas: {num_linhas_depois}, Colunas: {num_colunas_depois}")

    if dicionario is not None:

        mapa_para_atualizar_dicionario = mapa_renomeacao if mapa_renomeacao and any(col in mapa_renomeacao for col in colunas_modificadas) else {}
        atualizar_dicionario_renomeacao(dicionario, mapa_para_atualizar_dicionario)

    except NameError as e:
        print(f"Ocorreu um erro de nome ao renomear as colunas: {e}")
    except Exception as e:
        print(f"Ocorreu um erro inesperado ao renomear as colunas: {e}")

```

## ▼ Transforma valores de uma coluna de string para string

```

def transformar_valores_string(df: pd.DataFrame, coluna: str, mapa_transformacao: dict, dicionario: dict = None):
    """
    Transforma os valores de uma coluna do tipo string para string utilizando um mapeamento,
    salva a alteração e atualiza o dicionário de dados.

    Args:
        df (pd.DataFrame): O DataFrame a ser modificado.
        coluna (str): O nome da coluna a ser transformada.
        mapa_transformacao (dict): Um dicionário contendo o mapeamento dos valores originais para os novos valores.
        dicionario (dict, optional): O dicionário de dados a ser atualizado. Defaults to None.
    """

    if coluna not in df.columns:
        print(f"Erro: A coluna '{coluna}' não existe no DataFrame.")
        return

    if df[coluna].dtype != 'object':
        print(f"Erro: A coluna '{coluna}' não é do tipo 'object'. Nenhuma transformação será aplicada.")
        return

    try:

        valores_unicos_na_coluna = df[coluna].dropna().unique()
        valores_nao_mapeados = [valor for valor in valores_unicos_na_coluna if valor not in mapa_transformacao]

        if valores_nao_mapeados:
            print(f"Erro: Os seguintes valores únicos na coluna '{coluna}' não foram encontrados no mapeamento:")
            print(valores_nao_mapeados)
            print("A transformação não será realizada pois nem todos os valores possuem um mapeamento.")
            return

        linhas_anteriores = df.shape[0]

        df[coluna] = df[coluna].map(mapa_transformacao)

        linhas_depois = df.shape[0]
        linhas_alteradas = linhas_anteriores

        print(f"Coluna '{coluna}' transformada com sucesso.")
        print(f"Tipo de alteração: Mapeamento de valores string para string.")
        print(f"Número de linhas processadas: {linhas_alteradas}")

        print(f"\nValores únicos e contagem depois da transformação para a coluna '{coluna}':")
        display(df[coluna].value_counts(dropna=False))

        if dicionario is not None:
            alteracao = {
                'tipo': 'mapeamento_string_para_string',

```

```

        'mapa_aplicado': mapa_transformacao
    }
    atualizar_campo_dicionario(dicionario, coluna, 'transformacao', alteracao)

except Exception as e:
    print(f'Ocorreu um erro ao transformar a coluna '{coluna}': {e}')

```

## ▼ standardScale

```

def padronizar_colunas_standardscaler(df: pd.DataFrame, colunas_para_padronizar: list, dicionario: dict = None, nome_novo_df: str = None):
    """
    Aplica o StandardScaler às colunas especificadas de um DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame a ser modificado.
        colunas_para_padronizar (list): Lista de nomes das colunas a serem padronizadas.
        dicionario (dict, optional): O dicionário de dados a ser atualizado. Defaults to None.
        nome_novo_df (str, optional): Nome para o novo DataFrame se uma cópia for desejada.
            Se None, a padronização é aplicada no DataFrame original.

    Returns:
        pd.DataFrame: O DataFrame com as colunas padronizadas.
        Retorna uma cópia se nome_novo_df for fornecido, caso contrário, retorna o DataFrame original modificado.

    """
    if nome_novo_df:
        df_processado = df.copy()
        print(f'Criada uma cópia do DataFrame original com o nome '{nome_novo_df}'')
    else:
        df_processado = df
        print("Aplicando padronização diretamente no DataFrame original.")

    colunas_inexistentes = [col for col in colunas_para_padronizar if col not in df_processado.columns]
    if colunas_inexistentes:
        print(f'Erro: As seguintes colunas para padronizar não existem no DataFrame: {colunas_inexistentes}')
        return df_processado

    scaler = StandardScaler()

    try:
        df_processado[colunas_para_padronizar] = scaler.fit_transform(df_processado[colunas_para_padronizar])

        print("DataFrame após padronização (primeiras 5 linhas das colunas padronizadas):")
        display(df_processado[colunas_para_padronizar].head())

        print("\nEstatísticas Descritivas das Colunas Padronizadas:")
        display(df_processado[colunas_para_padronizar].describe())

        if dicionario is not None:
            print("\nAtualizando dicionário de dados com informações de padronização...")
            for coluna in colunas_para_padronizar:
                if coluna in dicionario:
                    if 'transformacao' not in dicionario[coluna] or not isinstance(dicionario[coluna]['transformacao'], dict):
                        dicionario[coluna]['transformacao'] = {}

                    transform_key_number = len(dicionario[coluna]['transformacao']) + 1
                    transformacao_para_adicionar = {
                        f'transformacao_{transform_key_number}': "Padronizado usando StandardScaler"
                    }

                    atualizar_campo_dicionario(dicionario, coluna, 'transformacao', transformacao_para_adicionar)
                else:
                    print(f'Aviso: Coluna '{coluna}' não encontrada no dicionário de dados para atualização.')

            print("Atualização do dicionário de dados concluída.")

        return df_processado

    except Exception as e:
        print(f'Ocorreu um erro durante a padronização com StandardScaler: {e}')
        return df_processado

```

## ▼ label encoding

```
def aplicar_label_encoding(df: pd.DataFrame, coluna: str, dicionario: dict = None) -> pd.DataFrame:
    """
    Aplica Label Encoding a uma coluna de um DataFrame usando scikit-learn LabelEncoder
    e atualiza o dicionário de dados.

    Args:
        df (pd.DataFrame): O DataFrame a ser modificado.
        coluna (str): O nome da coluna a ser codificada.
        dicionario (dict, optional): O dicionário de dados a ser atualizado. Defaults to None.

    Returns:
        pd.DataFrame: O DataFrame com a coluna codificada, ou o DataFrame original em caso de erro.
    """

    if coluna not in df.columns:
        print(f"Erro: A coluna '{coluna}' não existe no DataFrame.")
        return df

    try:
        label_encoder = LabelEncoder()

        nova_coluna_nome = f'{coluna}_rcod'
        df[nova_coluna_nome] = label_encoder.fit_transform(df[coluna])

        print(f"Coluna origina: '{coluna}'")
        display(df[coluna].head())

        print(f"\nColuna codificada '{nova_coluna_nome}'")
        display(df[nova_coluna_nome].head())

        if dicionario is not None:
            atualizar_dicionario_novas_colunas(dicionario, df)
            descricao_codificada = f"Variável '{coluna}' codificada numericamente via scikit-learn LabelEncoder."
            atualizar_campo_dicionario(dicionario, nova_coluna_nome, 'descricao', descricao_codificada)
            transformacao_info = {
                'tipo': 'label_encoding_sklearn',
                'classe_codificada': label_encoder.classes_.tolist()
            }
            atualizar_campo_dicionario(dicionario, nova_coluna_nome, 'transformacao', transformacao_info)

        return df

    except Exception as e:
        print(f"An error occurred during Label Encoding or dictionary update for column '{coluna}': {e}")
        return df
```

## one-hot encoding

```
def aplicar_one_hot_encoding(df: pd.DataFrame, colunas_para_ohe: list, dicionario: dict = None, drop_first: bool = True) ->
    """
    Aplica One-Hot Encoding a uma ou mais colunas de um DataFrame e atualiza o dicionário de dados.

    Args:
        df (pd.DataFrame): O DataFrame a ser modificado.
        colunas_para_ohe (list): Uma lista de nomes das colunas para aplicar One-Hot Encoding.
        dicionario (dict, optional): O dicionário de dados a ser atualizado. Defaults to None.
        drop_first (bool, optional): Se True, remove a primeira categoria para evitar multicolinearidade. Defaults to True.

    Returns:
        pd.DataFrame: O DataFrame com as colunas One-Hot Encoded.
        Retorna o DataFrame original em caso de erro.
    """

    df_processado = df.copy()

    colunas_inexistentes = [col for col in colunas_para_ohe if col not in df_processado.columns]
    if colunas_inexistentes:
        print(f"Erro: As seguintes colunas para One-Hot Encoding não existem no DataFrame: {colunas_inexistentes}")
        return df_processado

    try:
        print(f"Aplicando One-Hot Encoding nas colunas: {colunas_para_ohe}")

        for coluna in colunas_para_ohe:

            if df_processado[coluna].dtype == 'object':

                df_processado = pd.get_dummies(df_processado, columns=[coluna], prefix=coluna, drop_first=drop_first, dtype=
```

```

print(f"One-Hot Encoding aplicado na coluna '{coluna}'.")

if dicionario is not None:

    if coluna in dicionario:
        if 'transformacao' not in dicionario[coluna] or not isinstance(dicionario[coluna]['transformacao']):
            dicionario[coluna]['transformacao'] = {}
        transform_key_original = len(dicionario[coluna]['transformacao']) + 1
        transformacao_original = {
            f'transformacao_{transform_key_original}': f"Aplicado One-Hot Encoding (coluna original '{coluna}'"
        }
        atualizar_campo_dicionario(dicionario, coluna, 'transformacao', transformacao_original)

    atualizar_dicionario_ohe(dicionario, df_processado, coluna)

else:
    print(f"Aviso: A coluna '{coluna}' não é do tipo 'object' e não terá One-Hot Encoding aplicado.")

print("\nDataFrame após One-Hot Encoding (primeiras 5 linhas):")
display(df_processado.head())

print("\nColunas do DataFrame após One-Hot Encoding:")
print(df_processado.columns)

return df_processado

except Exception as e:
    print(f"Ocorreu um erro durante a aplicação do One-Hot Encoding: {e}")
    return df

```

## ▼ Transforma ordinalEncoder

```

def aplicar_ordinal_encoding(df: pd.DataFrame, coluna: str, ordem_categorias: list, dicionario: dict = None) -> pd.DataFrame:
    """
    Aplica Ordinal Encoding a uma coluna de um DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame a ser modificado.
        coluna (str): O nome da coluna a ser codificada.
        ordem_categorias (list): Uma lista com a ordem desejada das categorias.
        dicionario (dict, optional): O dicionário de dados a ser atualizado. Defaults to None.

    Returns:
        pd.DataFrame: O DataFrame com a coluna codificada, ou o DataFrame original em caso de erro.
    """
    if coluna not in df.columns:
        print(f"Erro: A coluna '{coluna}' não existe no DataFrame.")
        return df

    try:
        print(f"Aplicando Ordinal Encoding na coluna '{coluna}' com a ordem: {ordem_categorias}")

        encoder = OrdinalEncoder(categories=[ordem_categorias])

        df_processado[f'{coluna}_ocod'] = encoder.fit_transform(df_processado[[coluna]])

        print(f"Coluna '{coluna}' codificada com sucesso na nova coluna '{coluna}_ocod'.")

        print(f"\nContagem de valores na coluna '{coluna}_ocod':")
        display(df_processado[f'{coluna}_ocod'].value_counts().sort_index())

        if dicionario is not None:

            mapa_encoding = {categoria: i for i, categoria in enumerate(ordem_categorias)}

            atualizar_dicionario_ordinal_encoding(dicionario, f'{coluna}_ocod', coluna, mapa_encoding)

    return df_processado

except Exception as e:
    print(f"Ocorreu um erro durante a aplicação do Ordinal Encoding na coluna '{coluna}': {e}")
    return df

```

## Funções de Análise

### Analisar Colunas do Tipo 'object' em um DataFrame

```
def analisar_colunas_objeto(df: pd.DataFrame):
    """
    Identifica e analisa colunas do tipo 'object' em um DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame a ser analisado.

    """
    objetos_colunas = df.select_dtypes(include='object').columns
    print("Colunas do tipo 'object' restantes:")
    print(objetos_colunas)

    for col in objetos_colunas:
        print(f"\nValores únicos para a coluna '{col}':")
        display(df[col].unique())
        print(f"Contagem de valores para a coluna '{col}':")
        display(df.groupby(col).size())
```

### Cria Tabela de frequencia uma variavel

```
def criar_dataframe_frequencia(df: pd.DataFrame, coluna: str, ordem_categorias: list = None) -> pd.DataFrame:
    """
    Cria um DataFrame de frequência para uma coluna específica, incluindo contagem e porcentagem.

    Args:
        df (pd.DataFrame): O DataFrame de entrada.
        coluna (str): O nome da coluna para a qual criar o DataFrame de frequência.
        ordem_categorias (list, optional): Uma lista com a ordem desejada das categorias.
            Se None, a ordem será baseada na contagem decrescente.

    Returns:
        pd.DataFrame: Um DataFrame com as colunas 'categoria', 'quantidade' e 'porcentagem'.
    """
    if coluna not in df.columns:
        print(f"Erro: A coluna '{coluna}' não existe no DataFrame.")
        return pd.DataFrame()

    frequencia_contagem = df[coluna].value_counts().reset_index()
    frequencia_contagem.columns = ['categoria', 'quantidade']

    if ordem_categorias:
        frequencia_contagem['categoria'] = pd.Categorical(
            frequencia_contagem['categoria'],
            categories=ordem_categorias,
            ordered=True
        )
        frequencia_contagem = frequencia_contagem.sort_values('categoria').reset_index(drop=True)
    else:
        frequencia_contagem = frequencia_contagem.sort_values('quantidade', ascending=False).reset_index(drop=True)

    total_linhas = df.shape[0]
    frequencia_contagem['porcentagem'] = round((frequencia_contagem['quantidade'] / total_linhas) * 100, 2)

    return frequencia_contagem
```

### Gráfico barras

```
def gera_grafico_barras(df: pd.DataFrame, x_col: str, y_col: str = None, hue_col: str = None, title: str = "Gráfico de Barras"):
    """
    Gera um gráfico de barras usando Seaborn com opções para ordem e exibição de porcentagens.

    Args:
        df (pd.DataFrame): O DataFrame contendo os dados.
        x_col (str): O nome da coluna para o eixo X.
        y_col (str, optional): O nome da coluna para o eixo Y (para barplot).
            Se None, gera um countplot (contagem de ocorrências).
        hue_col (str, optional): O nome da coluna para dividir as barras por cor. Defaults to None.
    """
    # Implementação do gráfico de barras
```

```

title (str, optional): O título do gráfico. Defaults to "Gráfico de Barras".
order (list, optional): Uma lista com a ordem desejada das categorias no eixo X. Defaults to None.
"""
plt.figure(figsize=(12, 6))

if y_col:

    ax = sns.barplot(data=df, x=x_col, y=y_col, hue=hue_col, palette='colorblind', order=order)
    plt.ylabel(y_col)

    if 'porcentagem' in df.columns:
        for container in ax.containers:
            ax.bar_label(container, fmt='%.1f%%', label_type='edge')

    else:
        if hue_col is None:
            ax = sns.countplot(data=df, x=x_col, hue=x_col, palette='colorblind', order=order, legend=False)
        else:
            ax = sns.countplot(data=df, x=x_col, hue=hue_col, palette='colorblind', order=order)

        plt.ylabel("Contagem")

    total = len(df)
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height() / total)
        x = p.get_x() + p.get_width() / 2.
        y = p.get_height()
        ax.annotate(percentage, (x, y), ha='center', va='bottom')

plt.title(title)
plt.xlabel(x_col)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

```

## ▼ Histograma

```

def gera_histograma(df: pd.DataFrame, coluna: str, titulo: str, cor: str = None, cor_kde: str = None):
    """
    Gera um histograma para uma coluna específica de um DataFrame com opção de cor para as barras e para a linha do KDE.
    """
    if coluna not in df.columns:
        print(f"Erro: A coluna '{coluna}' não existe no DataFrame.")
        return

    if not pd.api.types.is_numeric_dtype(df[coluna]):
        print(f"A coluna '{coluna}' não é do tipo numérico. Não é possível gerar um histograma.")
        return

    try:
        plt.figure(figsize=(10, 6))
        ax = sns.histplot(data=df, x=coluna, kde=True, color=cor)
        ax.lines[0].set_color(cor_kde)
        plt.title(titulo)
        plt.xlabel(coluna)
        plt.xticks(rotation=45, ha='right')
        plt.ylabel("Frequência")
        plt.show()
    except Exception as e:
        print(f"Ocorreu um erro ao gerar o histograma para a coluna '{coluna}': {e}")

```

## ▼ Boxplot

```

def plotar_boxplot_numerica(df: pd.DataFrame, coluna: str, titulo: str = None, cor: str = None):
    """
    Gera um boxplot para uma coluna numérica específica de um DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame contendo os dados.
        coluna (str): O nome da coluna numérica para gerar o boxplot.
        titulo (str, optional): O título do gráfico. Se None, um título padrão será usado.
        cor (str, optional): A cor a ser usada no boxplot. Defaults to None.
    """
    if coluna not in df.columns:

```

```

print(f"Erro: A coluna '{coluna}' não existe no DataFrame.")
return

if not pd.api.types.is_numeric_dtype(df[coluna]):
    print(f"A coluna '{coluna}' não é do tipo numérico. Não é possível gerar um boxplot.")
    return

try:
    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df, y=coluna, color=cor) # Added color parameter
    if titulo is None:
        plt.title(f'Boxplot da Variável {coluna}')
    else:
        plt.title(titulo)
    plt.ylabel(coluna)
    plt.show()
except Exception as e:
    print(f"Ocorreu um erro ao gerar o boxplot para a coluna '{coluna}': {e}")

```

## ▼ Boxplot agrupado vertical

```

def plotar_boxplot_agrupado_vertical(df: pd.DataFrame, variavel: str, variavel_alvo: str, titulo: str = None, ordem: list =
""):
    Gera um boxplot agrupado mostrando a distribuição de uma variável numérica (target_col)
    para cada categoria de uma variável categórica (feature_col).

Args:
    df (pd.DataFrame): O DataFrame contendo os dados.
    feature_col (str): O nome da coluna categórica (eixo X).
    target_col (str): O nome da coluna numérica (eixo Y).
    order_feature (list, optional): Uma lista com a ordem desejada das categorias
        da feature_col no eixo X. Defaults to None.
    cor (str, optional): A cor a ser usada nos boxplots. Defaults to None.
    titulo (str, optional): O título do gráfico. Se None, um título padrão será usado.
"""

if variavel not in df.columns:
    print(f"Erro: A coluna da feature '{variavel}' não existe no DataFrame.")
    return
if variavel_alvo not in df.columns:
    print(f"Erro: A coluna alvo '{variavel_alvo}' não existe no DataFrame.")
    return
if not pd.api.types.is_numeric_dtype(df[variavel]):
    print(f"Erro: A coluna alvo '{variavel}' não é do tipo numérico. Não é possível gerar um boxplot agrupado.")
    return

try:
    plt.figure(figsize=(12, 8))
    sns.boxplot(data=df, x=variavel, y=variavel_alvo, order=ordem, color=cor)

    if titulo is None:
        plt.title(f'Distribuição da variável "{variavel}" pela "{variavel_alvo}"')
    else:
        plt.title(titulo)
    plt.xlabel(variavel)
    plt.ylabel(variavel_alvo)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
except Exception as e:
    print(f"Ocorreu um erro ao gerar o boxplot agrupado: {e}")

```

## ▼ Boxplot agrupado horizontal

```

def plotar_boxplot_agrupado_horizontal(df: pd.DataFrame, variavel: str, variavel_alvo: str, titulo: str = None, ordem: list =
""):
    Gera um boxplot agrupado mostrando a distribuição de uma variável numérica (target_col)
    para cada categoria de uma variável categórica (feature_col).

Args:
    df (pd.DataFrame): O DataFrame contendo os dados.
    feature_col (str): O nome da coluna categórica (eixo X).
    target_col (str): O nome da coluna numérica (eixo Y).
    order_feature (list, optional): Uma lista com a ordem desejada das categorias
        da feature_col no eixo X. Defaults to None.
    cor (str, optional): A cor a ser usada nos boxplots. Defaults to None.
    titulo (str, optional): O título do gráfico. Se None, um título padrão será usado.
"""

if variavel not in df.columns:

```

```

        print(f"Erro: A coluna da feature '{variavel}' não existe no DataFrame.")
        return
    if variavel_alvo not in df.columns:
        print(f"Erro: A coluna alvo '{variavel_alvo}' não existe no DataFrame.")
        return
    if not pd.api.types.is_numeric_dtype(df[variavel]):
        print(f"Erro: A coluna alvo '{variavel}' não é do tipo numérico. Não é possível gerar um boxplot agrupado.")
        return

    try:
        plt.figure(figsize=(12, 8))
        sns.boxplot(data=df, x=variavel_alvo, y=variavel, order=ordem, color=cor)

        if titulo is None:
            plt.title(f'Distribuição da variável "{variavel}" pela "{variavel_alvo}"')
        else:
            plt.title(titulo)
        plt.xlabel(variavel_alvo)
        plt.ylabel(variavel)
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
    except Exception as e:
        print(f'Ocorreu um erro ao gerar o boxplot agrupado: {e}')

```

## ▼ Funções gerais

### ▼ Leitura de arquivo json de uma URL

```

def ler_json_de_url(url: str) -> dict:
    """
    Lê um arquivo JSON de uma URL fornecida.

    Args:
        url (str): A URL do arquivo JSON.

    Returns:
        dict: O dicionário de dados carregado do arquivo JSON, ou Nenhum se ocorrer um erro.
    """
    try:
        response = requests.get(url)
        response.raise_for_status()
        data_dict = response.json()
        print(f"Leitura com sucesso do caminho {url}")
        return data_dict
    except requests.exceptions.RequestException as e:
        print(f"Erro ao acessar a URL {url}: {e}")
        return None
    except json.JSONDecodeError:
        print(f"Erro: Não foi possível decodificar JSON da URL {url}. Verifique o formato do arquivo.")
        return None
    except Exception as e:
        print(f"Ocorreu um erro inesperado ao ler de {url}: {e}")
        return None

```

### ▼ Função de classificação para coluna

```

def classificar_coluna_por_config(df: pd.DataFrame, coluna_original: str, nova_coluna: str, config: dict) -> pd.DataFrame:
    """
    Classifica os valores de uma coluna numérica em categorias usando um dicionário de configuração.

    Args:
        df (pd.DataFrame): O DataFrame a ser modificado.
        coluna_original (str): O nome da coluna numérica a ser classificada.
        nova_coluna (str): O nome da nova coluna que conterá as categorias.
        config (dict): O dicionário de configuração com 'bins', 'labels', 'right', 'include_lowest'.

    Returns:
        pd.DataFrame: O DataFrame com a nova coluna de classificação adicionada, ou o DataFrame original em caso de erro.
    """
    if coluna_original not in df.columns:
        print(f"Erro: A coluna original '{coluna_original}' não existe no DataFrame.")
        return df

    if not pd.api.types.is_numeric_dtype(df[coluna_original]):
        print(f"A coluna '{coluna_original}' não é do tipo numérico. Nenhuma classificação será aplicada.")

```

```

        return df

    try:

        if 'bins' not in config or 'labels' not in config:
            print("Erro: O dicionário de configuração deve conter as chaves 'bins' e 'labels'.")
            return df

        bins = config['bins']
        labels = config['labels']
        right = config.get('right', True)
        include_lowest = config.get('include_lowest', False)

        bins_processados = []
        for b in bins:
            if isinstance(b, str):
                if b.lower() == '-inf':
                    bins_processados.append(-np.inf)
                elif b.lower() == 'inf':
                    bins_processados.append(np.inf)
                else:
                    try:
                        bins_processados.append(float(b))
                    except ValueError:
                        print(f"Aviso: Valor inválido em 'bins': '{b}'. Mantendo como está ou pode causar erro.")
                        bins_processados.append(b)
            else:
                bins_processados.append(b)

        if not all(bins_processados[i] <= bins_processados[i+1] for i in range(len(bins_processados)-1)):
            print("Erro: Os bins fornecidos no dicionário de configuração não estão em ordem crescente.")
            return df

        if len(bins_processados) != len(labels) + 1:
            print(f"Erro: O número de bins ({len(bins_processados)}) deve ser um a mais que o número de labels ({len(labels)}).")
            return df

        df[nova_coluna] = pd.cut(df[coluna_original], bins=bins_processados, labels=labels, right=right, include_lowest=include_lowest)

        print(f"Coluna '{coluna_original}' classificada com sucesso na nova coluna '{nova_coluna}'.")

        print(f"\nContagem de valores para a nova coluna '{nova_coluna}':")
        display(df[nova_coluna].value_counts(dropna=False))

    return df
except Exception as e:
    print(f"Ocorreu um erro ao classificar a coluna '{coluna_original}': {e}")
    return df

```

## Função para verificar valores nulos

```

def verificar_nulos_por_coluna(df: pd.DataFrame) -> None:
    """
    Verifica e imprime a quantidade de valores nulos por coluna em um DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame a ser verificado.
    """
    print("Verificando valores nulos por coluna:")
    try:
        nulos_por_coluna = df.isnull().sum()
        colunas_com_nulos = nulos_por_coluna[nulos_por_coluna > 0]

        if colunas_com_nulos.empty:
            print("Nenhuma coluna possui valores nulos.")
        else:
            print("\nColunas com valores nulos:")
            display(colunas_com_nulos)

    except Exception as e:
        print(f"Ocorreu um erro ao verificar valores nulos: {e}")

```

## ▼ Criacao de CSV a partir de um dataframe

```
def criar_csv_de_dataframe(df: pd.DataFrame, nome_arquivo: str) -> None:
    """
    Gera um arquivo CSV a partir de um DataFrame.

    Args:
        df (pd.DataFrame): O DataFrame a ser salvo.
        nome_arquivo (str): O nome do arquivo CSV de saída (incluindo a extensão .csv).
    """
    if not isinstance(df, pd.DataFrame):
        print("Erro: O primeiro argumento deve ser um DataFrame do pandas.")
        return

    if not isinstance(nome_arquivo, str) or not nome_arquivo.lower().endswith('.csv'):
        print("Erro: O nome do arquivo deve ser uma string e terminar com '.csv'.")
        return

    try:
        df.to_csv(nome_arquivo, index=False)
        print(f"DataFrame salvo com sucesso em '{nome_arquivo}'")
    except Exception as e:
        print(f"Ocorreu um erro ao salvar o DataFrame em '{nome_arquivo}': {e}")
```

## ▼ Criação de arquivo json a partir de um dicionario

```
def criar_json_de_dicionario(dicionario: dict, nome_arquivo: str) -> None:
    """
    Gera um arquivo JSON a partir de um dicionário Python.

    Args:
        dicionario (dict): O dicionário a ser salvo.
        nome_arquivo (str): O nome do arquivo JSON de saída (incluindo a extensão .json).
    """
    if not isinstance(dicionario, dict):
        print("Erro: O primeiro argumento deve ser um dicionário Python.")
        return

    if not isinstance(nome_arquivo, str) or not nome_arquivo.lower().endswith('.json'):
        print("Erro: O nome do arquivo deve ser uma string e terminar com '.json'.")
        return

    try:
        with open(nome_arquivo, 'w') as f:
            json.dump(dicionario, f, indent=4)
        print(f"Dicionário salvo com sucesso em '{nome_arquivo}'")
    except TypeError as e:
        print(f"Erro: O conteúdo do dicionário não é serializável para JSON. Detalhes: {e}")
    except Exception as e:
        print(f"Ocorreu um erro ao salvar o dicionário em '{nome_arquivo}': {e}")
```

## ▼ Dados

### Data frame principal

```
df = pd.read_csv('https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main/dados/Obesidade.csv')
```

### arquivo json mapa\_colunas

```
mapa_colunas = ler_json_de_url('https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main/mapping_colunas.json')
Leitura com sucesso do caminho https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main/mapping_colunas.json
```

```
mapa_colunas.keys()
```

```
dict_keys(['Gender', 'Age', 'Height', 'Weight', 'family_history', 'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH20', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRANS', 'Obesity'])
```

### arquivo JSON mapa\_valores\_colunas

```
mapa_valores_colunas = ler_json_de_url('https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main')
Leitura com sucesso do caminho https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main

mapa_valores_colunas.keys()

dict_keys(['mapeamento_genero', 'mapeamento_sim_nao', 'mapeamento_frequencia', 'mapeamento_mtrans',
'mapeamento_classificacao_peso_corporal'])
```

### Arquivo JSON descricao\_dados\_obesidade

```
caminho_arquivo_descricao_dados_obesidade = 'https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main'
descricao_dados = ler_json_de_url(caminho_arquivo_descricao_dados_obesidade)

Leitura com sucesso do caminho https://raw.githubusercontent.com/bernardo-seg/tech-challenge-fase4-obesidade/refs/heads/main

descricao_dados

{'Gender': 'Genero',
'Age': 'Idade',
'Height': 'Altura em metros',
'Weight': 'Peso em kgs.',
'family_history': 'Algum membro da familia sofreu ou sofre de excesso de peso?',
'FAVC': 'Voce come alimentos altamente caloricos com frequencia?',
'FCVC': 'Voce costuma comer vegetais nas suas refeicoes?',
'NCP': 'Quantas refeicoes principais voce faz diariamente?',
'CAEC': 'Voce come alguma coisa entre as refeicoes?',
'SMOKE': 'Voce fuma?',
'CH20': 'Quanta agua voce bebe diariamente?',
'SCC': 'Voce monitora as calorias que ingere diariamente?',
'FAF': 'Com que frequencia voce pratica atividade fisica?',
'TUE': 'Quanto tempo voce usa dispositivos tecnologicos como celular,',
'CALC': 'Com que frequencia voce bebe alcool?',
'MTRANS': 'Qual meio de transporte voce costuma usar?',
'Obesity': 'Classifica o nivel de obesidade da amostra.'}
```

```
descricao_dados.keys()

dict_keys(['Gender', 'Age', 'Height', 'Weight', 'family_history', 'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH20', 'SCC',
'FAF', 'TUE', 'CALC', 'MTRANS', 'Obesity'])
```

## ✓ Dicionário de Dados

- Criação de um dicionário de dados básicos com as colunas contidas no csv `Obesity.csv`. Para isso foi utilizado a função `criar_novo_dicionario_dados()`.

```
dicionario_dados_nivel_obesidade = criar_novo_dicionario_dados(df)
atualizar_descricoes_dicionario(dicionario_dados_nivel_obesidade, descricao_dados)
```

Novo dicionário de dados criado.

Campo 'descricao' da coluna 'Gender' atualizado para: 'Genero'  
 Campo 'descricao' da coluna 'Age' atualizado para: 'Idade'  
 Campo 'descricao' da coluna 'Height' atualizado para: 'Altura em metros'  
 Campo 'descricao' da coluna 'Weight' atualizado para: 'Peso em kgs.'  
 Campo 'descricao' da coluna 'family\_history' atualizado para: 'Algum membro da familia sofreu ou sofre de excesso de peso?'  
 Campo 'descricao' da coluna 'FAVC' atualizado para: 'Voce come alimentos altamente caloricos com frequencia?'  
 Campo 'descricao' da coluna 'FCVC' atualizado para: 'Voce costuma comer vegetais nas suas refeicoes?'  
 Campo 'descricao' da coluna 'NCP' atualizado para: 'Quantas refeicoes principais voce faz diariamente?'  
 Campo 'descricao' da coluna 'CAEC' atualizado para: 'Voce come alguma coisa entre as refeicoes?'  
 Campo 'descricao' da coluna 'SMOKE' atualizado para: 'Voce fuma?'  
 Campo 'descricao' da coluna 'CH20' atualizado para: 'Quanta agua voce bebe diariamente?'  
 Campo 'descricao' da coluna 'SCC' atualizado para: 'Voce monitora as calorias que ingere diariamente?'  
 Campo 'descricao' da coluna 'FAF' atualizado para: 'Com que frequencia voce pratica atividade fisica?'  
 Campo 'descricao' da coluna 'TUE' atualizado para: 'Quanto tempo voce usa dispositivos tecnologicos como celular,'  
 Campo 'descricao' da coluna 'CALC' atualizado para: 'Com que frequencia voce bebe alcool?'  
 Campo 'descricao' da coluna 'MTRANS' atualizado para: 'Qual meio de transporte voce costuma usar?'  
 Campo 'descricao' da coluna 'Obesity' atualizado para: 'Classifica o nivel de obesidade da amostra.'

Quantidade de descrições atualizadas: 17.

```
print(json.dumps(dicionario_dados_nivel_obesidade, indent=4))
```

```

        "novo_nome": null,
        "descricao": "Voce fuma?",
        "transformacao": {}
    },
    "CH20": {
        "nome_original": "CH20",
        "tipo_coluna": "original",
        "novo_nome": null,
        "descricao": "Quanta agua voce bebe diariamente?",
        "transformacao": {}
    },
    "SCC": {
        "nome_original": "SCC",
        "tipo_coluna": "original",
        "novo_nome": null,
        "descricao": "Voce monitora as calorias que ingere diariamente?",
        "transformacao": {}
    },
    "FAF": {
        "nome_original": "FAF",
        "tipo_coluna": "original",
        "novo_nome": null,
        "descricao": "Com que frequencia voce pratica atividade fisica?",
        "transformacao": {}
    },
    "TUE": {
        "nome_original": "TUE",
        "tipo_coluna": "original",
        "novo_nome": null,
        "descricao": "Quanto tempo voce usa dispositivos tecnologicos como celular,",
        "transformacao": {}
    },
    "CALC": {
        "nome_original": "CALC",
        "tipo_coluna": "original",
        "novo_nome": null,
        "descricao": "Com que frequencia voce bebe alcool?",
        "transformacao": {}
    },
    "MTRANS": {
        "nome_original": "MTRANS",
        "tipo_coluna": "original",
        "novo_nome": null,
        "descricao": "Qual meio de transporte voce costuma usar?",
        "transformacao": {}
    },
    "Obesity": {
        "nome_original": "Obesity",
        "tipo_coluna": "original",
        "novo_nome": null,
        "descricao": "Classifica o nivel de obesidade da amostra.",
        "transformacao": {}
    }
}

```

df.head()

	Gender	Age	Height	Weight	family_history	FAVC	FCVC	NCP	CAEC	SMOKE	CH20	SCC	FAF	TUE	CALC	
0	Female	21.0	1.62	64.0		yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no Public_Tran
1	Female	21.0	1.52	56.0		yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes Public_Tran
2	Male	23.0	1.80	77.0		yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently Public_Tran
3	Male	27.0	1.80	87.0		no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently
4	Male	22.0	1.78	89.8		no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes Public_Tran

Próximas etapas: [Gerar código com df](#) [New interactive sheet](#)

## ▼ Alteração Nomes Colunas

Utilizada a função `renomear_colunas()` para aplicar o novo padrão de nomes contido no arquivo [/content/mapa\\_colunas.json](#). Essa função também atualiza o dicionário de dados.

```
renomear_colunas(df, mapa_colunas, dionario_dados_nivel_obesidade)
```

Número de linhas e colunas antes da transformação:  
Linhas: 2111, Colunas: 17

Nomes das colunas originais:  
['Gender', 'Age', 'Height', 'Weight', 'family\_history', 'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH20', 'SCC', 'FAF', 'TUE',

Utilizado dicionário de renomeação fornecido.

```
Nomes das colunas modificadas:  
['genero', 'idade', 'altura', 'peso', 'historico_familiar', 'favc', 'fcvc', 'ncp', 'caec', 'fumante', 'ch20', 'scc', 'faf',  
Número de linhas e colunas após a transformação:  
Linhas: 2111, Colunas: 17  
Iniciando a atualização do dicionário de dados com renomeações...  
Dicionário de dados atualizado com renomeações.
```

## ✓ Análise Exploratória

### ✓ Verificação/ análise colunas e valores

#### ✓ Análise:

Variáveis categóricas nominais:

```
genero, historico_familiar, favc, fumante, scc, mtrans
```

Variáveis categóricas ordinais:

```
caec, calc
```

Variável Alvo:

```
classificacao_peso_corporal
```

Variáveis numéricas

```
idade, altura, peso, fcvc, ncp, ch20, favc, tue
```

```
display(HTML(df.head().to_html(index = False)))
```

genero	idade	altura	peso	historico_familiar	favc	fcvc	ncp	caec	fumante	ch20	scc	faf	tue	calc	
Female	21.0	1.62	64.0		yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no Public_T
Female	21.0	1.52	56.0		yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes Public_T
Male	23.0	1.80	77.0		yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently Public_T
Male	27.0	1.80	87.0		no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently
Male	22.0	1.78	89.8		no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes Public_T

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   genero          2111 non-null   object 
 1   idade           2111 non-null   float64
 2   altura          2111 non-null   float64
 3   peso            2111 non-null   float64
 4   historico_familiar  2111 non-null   object 
 5   favc            2111 non-null   object 
 6   fcvc            2111 non-null   float64
 7   ncp             2111 non-null   float64
 8   caec            2111 non-null   object 
 9   fumante         2111 non-null   object 
 10  ch20            2111 non-null   float64
 11  scc             2111 non-null   object 
 12  favf            2111 non-null   float64
 13  tue             2111 non-null   float64
 14  calc            2111 non-null   object 
 15  mtrans           2111 non-null   object 
 16  classificacao_peso_corporal 2111 non-null   object 
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

```
df.describe()
```

	idade	altura	peso	fcvc	ncp	ch20	faf	tue	grid
<b>count</b>	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	grid
<b>mean</b>	24.312600	1.701677	86.586058	2.419043	2.685628	2.008011	1.010298	0.657866	grid
<b>std</b>	6.345968	0.093305	26.191172	0.533927	0.778039	0.612953	0.850592	0.608927	grid
<b>min</b>	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	0.000000	0.000000	grid
<b>25%</b>	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812	0.124505	0.000000	grid
<b>50%</b>	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000	1.000000	0.625350	grid
<b>75%</b>	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420	1.666678	1.000000	grid
<b>max</b>	61.000000	1.980000	173.000000	3.000000	4.000000	3.000000	3.000000	2.000000	grid

```
verificar_nulos_por_coluna(df)
```

Verificando valores nulos por coluna:  
Nenhuma coluna possui valores nulos.

```
analisar_colunas_objeto(df)
```



```
Colunas do tipo 'object' restantes:
Index(['genero', 'historico_familiar', 'favc', 'caec', 'fumante', 'scc',
       'calc', 'mtrans', 'classificacao_peso_corporal'],
      dtype='object')
```

Valores únicos para a coluna 'genero':  
array(['Female', 'Male'], dtype=object)

Contagem de valores para a coluna 'genero':

0

**genero**

Female	1043
--------	------

Male	1068
------	------

**dtype:** int64

Valores únicos para a coluna 'historico\_familiar':  
array(['yes', 'no'], dtype=object)

Contagem de valores para a coluna 'historico\_familiar':

0

**historico\_familiar**

no	385
----	-----

yes	1726
-----	------

**dtype:** int64

Valores únicos para a coluna 'favc':  
array(['no', 'yes'], dtype=object)

Contagem de valores para a coluna 'favc':

0

**favc**

no	245
----	-----

yes	1866
-----	------

**dtype:** int64

Valores únicos para a coluna 'caec':

array(['Sometimes', 'Frequently', 'Always', 'no'], dtype=object)

Contagem de valores para a coluna 'caec':

0

**caec**

Always	53
--------	----

Frequently	242
------------	-----

Sometimes	1765
-----------	------

no	51
----	----

## ▼ Tradução/criação de coluna

Valores únicos para a coluna 'fumante':

array(['no', 'yes'], dtype=object)

## ▼ Alteração do nome dos valores da variável `classificacao_peso_corporal`

0

A variável `classificacao_peso_corporal` é a variável alvo do presente estudo e no momento apenas foi realizada uma tradução dos valores que estavam em inglês.

```
transformar_valores_string(df,'classificacao_peso_corporal' , mapa_valores_colunas['mapeamento_classificacao_peso_corporal']
                           dtype: int64
```

Valores únicos para a coluna 'scc':

array(['no', 'yes'], dtype=object)

Contagem de valores para a coluna 'scc':

0

**scc**

no	2015
----	------

yes	96
-----	----

**dtype:** int64

Valores únicos para a coluna 'calc':

array(['no', 'Sometimes', 'Frequently', 'Always'], dtype=object)

```

Contagem das saídas após o mapeamento foi realizada com sucesso.
Tipo de alteração: Mapeamento de valores string para string.
Número de linhas processadas: 2111
    calc
Valores únicos e contagem depois da transformação para a coluna 'classificacao_peso_corporal':
  Always      1      count
  Frequently   70
  classificacao_peso_corporal
    obesidade_tipo_1      351
    no                   639
    obesidade_tipo_3      324
    obesidade_tipo_2      297
    sobrepeso_tipo_1      290
    sobrepeso_tipo_2      290
Contagem de valores para a coluna 'mtrans':
  peso_normal      287
  peso_insuficiente      272
  Bike                 7
  Insufficient_Weight      1
dtype: int64
Automobile      457
Campo 'transformacao' da coluna 'classificacao_peso_corporal' atualizado com novos itens.

```

### ordenação de nível classificacao\_peso\_corporal

```

Public_Transportation      1580
ordem_classificacao_peso_corporal = mapa_valores_colunas['mapeamento_classificacao_peso_corporal']['ordem_classificacao_peso_corporal']
print(ordem_classificacao_peso_corporal)

```

```

dtype: int64
  peso_insuficiente, 'peso_normal', 'sobrepeso_tipo_1', 'sobrepeso_tipo_2', 'obesidade_tipo_1', 'obesidade_tipo_2', 'obesidade_tipo_3'
Valores únicos para a coluna 'classificacao_peso_corporal':
array(['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Obesity_Type_II', 'Obesity_Type_III'], dtype=object)

```

### Alteração de nome dos valores da variável mtrans

```

Contagem de valores para a coluna 'classificacao_peso_corporal'.
transformar_valores_string(df,'mtrans' , mapa_valores_colunas['mapeamento_mtrans']['valores_novos_mtrans'], dicionario_dados)

```

classificacao\_peso\_corporal com sucesso.

Tipo de alteração: Mapeamento de valores string para string.

Número de linhas processadas: 2111

Normal\_Weight 287

Valores únicos e contagem depois da transformação para a coluna 'mtrans':

	count
Obesity_Type_I	351
Obesity_Type_II	297

	count
transporte_publico	324

	count
Overweight_Level_I	290

	count
caminhando	56

	count
moto	11

	count
bicicleta	7

**dtype: int64**

Campo 'transformacao' da coluna 'mtrans' atualizado com novos itens.

### Alterações do nomes das variaveis que possuem valores de frequência

#### Variável caec

```

transformar_valores_string(df,'caec' , mapa_valores_colunas['mapeamento_frequencia']['valores_novos_frequencia'], dicionario_dados)

```

```
Coluna 'caec' transformada com sucesso.
Tipo de alteração: Mapeamento de valores string para string.
Número de linhas processadas: 2111

Valores únicos e contagem depois da transformação para a coluna 'caec':
count
caec
as_vezes    1765
frequentemente   242
sempre      53
nunca       51

dtype: int64
Campo 'transformacao' da coluna 'caec' atualizado com novos itens.
```

**Variável calc**

```
transformar_valores_string(df,'calc' , mapa_valores_colunas['mapeamento_frequencia']['valores_novos_frequencia'], dicionar:

Coluna 'calc' transformada com sucesso.
Tipo de alteração: Mapeamento de valores string para string.
Número de linhas processadas: 2111

Valores únicos e contagem depois da transformação para a coluna 'calc':
count
calc
as_vezes    1401
nunca       639
frequentemente   70
sempre      1

dtype: int64
Campo 'transformacao' da coluna 'calc' atualizado com novos itens.
```

**ordenação de frequencia**

```
ordem_frequencia = mapa_valores_colunas['mapeamento_frequencia']['ordem_frequencia']
```

**✓ Criação coluna IMC**

Criada a coluna `imc` para comparacao para estudo do relacionamento com a variável alvo `classificacao_peso_corporal`.

```
#criacao da coluna imc
df['imc'] = round(df['peso'] / (df['altura'] * df['altura']),2)
display(df.head())
print(f'Total de linhas: {df.shape[0]}\nTotal de colunas: {df.shape[1]} ')
```

	genero	idade	altura	peso	historico_familiar	favc	fcvc	ncp	caec	fumante	ch20	scc	faf	tue	calc
0	Female	21.0	1.62	64.0		yes	no	2.0	3.0	as_vezes	no	2.0	no	0.0	1.0
1	Female	21.0	1.52	56.0		yes	no	3.0	3.0	as_vezes	yes	3.0	yes	3.0	0.0
2	Male	23.0	1.80	77.0		yes	no	2.0	3.0	as_vezes	no	2.0	no	2.0	frequentemente
3	Male	27.0	1.80	87.0		no	no	3.0	3.0	as_vezes	no	2.0	no	2.0	frequentemente
4	Male	22.0	1.78	89.8		no	no	2.0	1.0	as_vezes	no	2.0	no	0.0	0.0

Total de linhas: 2111

Total de colunas: 18

```
atualizar_dicionario_novas_colunas(dicionario_dados_nivel_obesidade, df)
```

```
descricao_imc = ''
IMC(Indice de Massa Corporal)
Medida internacional usada para avaliar se uma pessoa esta no seu peso ideal em relacao a sua altura.
Indicador simples da OMS que ajuda a identificar quadros de magreza, peso normal,sobrepeso ou obesidade.
Calculo: imc = peso/(altura*altura)
... .
```

```
    '''.strip()

    atualizar_campo_dicionario(dicionario_dados_nivel_obesidade, 'imc', 'descricao', descricao_imc)
```

Iniciando a atualização do dicionário de dados com novas colunas...  
 Adicionada nova coluna 'imc' ao dicionário de dados.  
 Dicionário de dados atualizado com novas colunas.  
 Campo 'descricao' da coluna 'imc' atualizado para: 'IMC(Indice de Massa Corporal)  
 Medida internacional usada para avaliar se uma pessoa está no seu peso ideal em relação a sua altura.  
 Indicador simples da OMS que ajuda a identificar quadros de magreza, peso normal,sobre peso ou obesidade.  
 Calculo: imc = peso/(altura\*altura)'

## ✓ Criacão de coluna de classificação para IMC

O intuito dessa coluna é verificar se existe diferença entre o padrão de classificação da OMS e da variável classificação peso corporal.

```
config_classificacao_imc = {
    "bins": [0, 18.50, 25, 30, 35, 40, np.inf],
    "labels": ['abaixo_do_peso', 'peso_normal', 'sobre peso', 'obesidade_grau_1', 'obesidade_grau_2', 'obesidade_grau_3'],
    "right": False,
    "include_lowest": True
}

classificar_coluna_por_config(df, 'imc', 'classificacao_imc', config_classificacao_imc)

atualizar_dicionario_novas_colunas(dicionario_dados_nivel_obesidade, df)

descricao_classificacao_imc = '''
Classificacao do IMC baseada nas faixas de valores:
- abaixo_do_peso: IMC < 18.50
- peso_normal: IMC >= 18.50 e IMC < 25
- sobre peso: IMC >= 25 e IMC < 30
- obesidade_grau_1: IMC >= 30 e IMC < 35
- obesidade_grau_2: IMC >= 35 e IMC < 40
- obesidade_grau_3: IMC >= 40
''''.strip()
atualizar_campo_dicionario(dicionario_dados_nivel_obesidade, 'classificacao_imc', 'descricao', descricao_classificacao_imc)

Coluna 'imc' classificada com sucesso na nova coluna 'classificacao_imc'.

Contagem de valores para a nova coluna 'classificacao_imc':
   count
   classificacao_imc
   _____
      sobre peso      566
      obesidade_grau_1  368
      obesidade_grau_2  338
      peso_normal      301
      abaixo_do_peso    270
      obesidade_grau_3  268

  dtype: int64
Iniciando a atualização do dicionário de dados com novas colunas...
Adicionada nova coluna 'classificacao_imc' ao dicionário de dados.
Dicionário de dados atualizado com novas colunas.
Campo 'descricao' da coluna 'classificacao_imc' atualizado para: 'Classificacao do IMC baseada nas faixas de valores:
- abaixo_do_peso: IMC < 18.50
- peso_normal: IMC >= 18.50 e IMC < 25
- sobre peso: IMC >= 25 e IMC < 30
- obesidade_grau_1: IMC >= 30 e IMC < 35
- obesidade_grau_2: IMC >= 35 e IMC < 40
- obesidade_grau_3: IMC >= 40'
```

## ✓ Análise de variáveis

### ✓ Pesquisa teórica

Existem vários métodos para avaliar a gordura corporal. A densitometria por absorciometria radiológica de dupla energia (DEXA) é o padrão ouro, utilizando raios X para medir com precisão a composição corporal, incluindo percentual de gordura, massa muscular,

massa óssea e gordura visceral. Outros métodos incluem circunferência da cintura, relação cintura-quadril, exames de bioimpedância e o Body Roundness Index (BRI).

O Índice de Massa Corporal (IMC) é prático e econômico para triagem populacional, mas pode superestimar a gordura em pessoas ativas e subestimar em sedentários, obesos, idosos e em certas condições clínicas. Por isso, é recomendado associar o IMC a outras medidas de gordura, como a circunferência da cintura, e, se possível, a exames de composição corporal.

Estudos mostram que combinar o IMC com informações comportamentais e padrões alimentares melhora a previsão da obesidade. Essa abordagem integrada, que considera hábitos alimentares, atividade física, sono e estilo de vida, permite identificar padrões, orientar intervenções preventivas e promover mudanças de comportamento. Modelos que associam dados antropométricos (IMC, peso e altura) a hábitos alimentares ajudam a prever não apenas o diagnóstico atual, mas também a propensão ao ganho de peso e obesidade futura, oferecendo uma estratégia mais completa para prevenção e tratamento, com benefícios para a saúde pública e clínica.

## ▼ Análise variável alvo: classificacao\_peso\_corporal

### Técnicas aplicadas:

- Tabela de frequencia,
- Gráfico de barras
- O Teste de Aderência, utilizando qui-quadrado
- Razão de Desbalanceamento

### Análise:

A amostra se mostra estatisticamente desbalanceada em relação as classes da variável alvo classificação\_peso\_corporal, no entanto ao calcular a razão de desbalanceamento verificou-se que a amostra é suficientemente balanceada para o treinamento inicial de um modelo de machine learning.

### Tabela de frequência

```
tab_freq_classificacao_peso_corporal = criar_dataframe_frequencia(df, 'classificacao_peso_corporal')
display(tab_freq_classificacao_peso_corporal)
print('\n')
print('Porcentagens:\n')
print(f"Porcentagem mínima de uma categoria: {round(tab_freq_classificacao_peso_corporal['porcentagem'].min(),1)}")
print(f"Porcentagem máxima de uma categoria: {round(tab_freq_classificacao_peso_corporal['porcentagem'].max(),1)}")
```

	categoria	quantidade	porcentagem	
0	obesidade_tipo_1	351	16.63	
1	obesidade_tipo_3	324	15.35	
2	obesidade_tipo_2	297	14.07	
3	sobrepeso_tipo_1	290	13.74	
4	sobrepeso_tipo_2	290	13.74	
5	peso_normal	287	13.60	
6	peso_insuficiente	272	12.88	

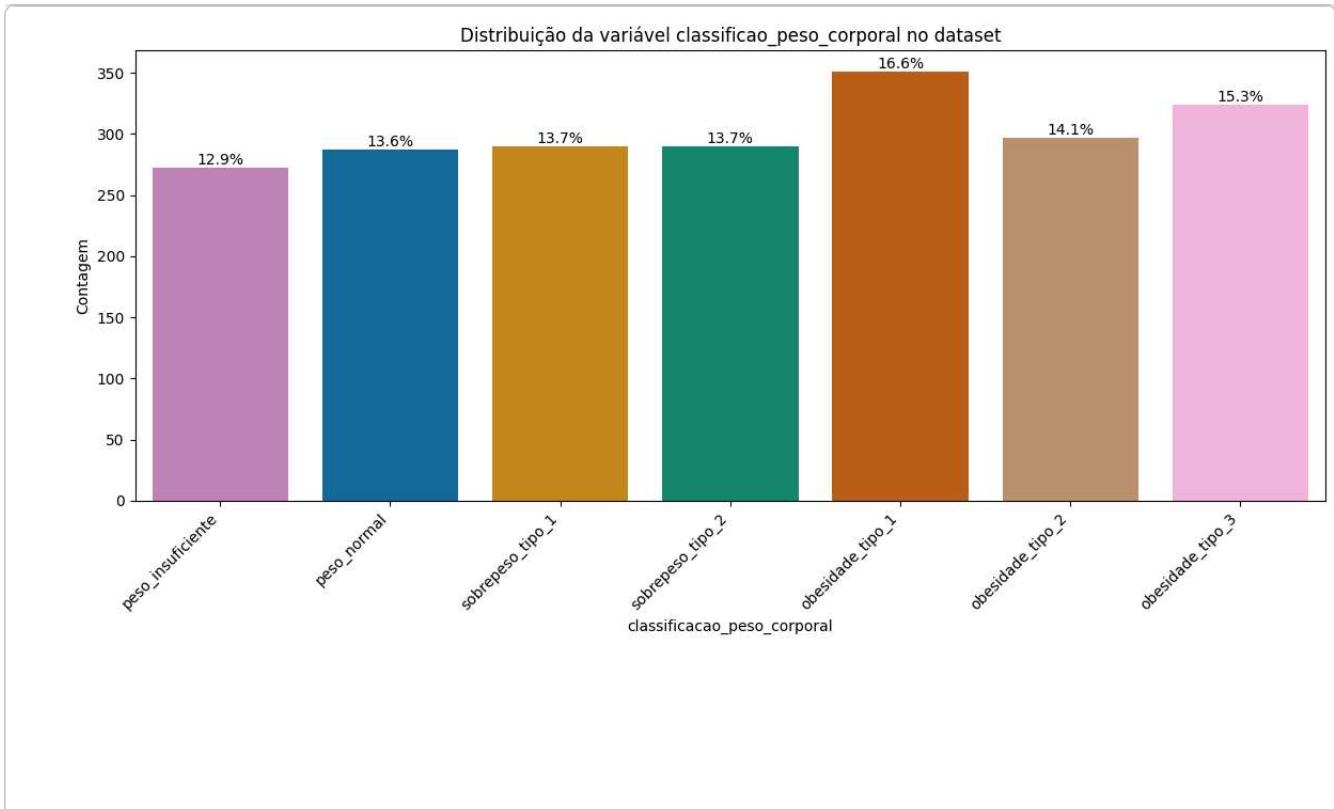
### Porcentagens:

Porcentagem mínima de uma categoria: 12.9  
Porcentagem máxima de uma categoria: 16.6

Próximas etapas: ([Gerar código com tab\\_freq\\_classificacao\\_peso\\_corporal](#)) [New interactive sheet](#)

### Gráfico de barras

```
gera_grafico_barras(df, 'classificacao_peso_corporal', title = "Distribuição da variável classificacao_peso_corporal no dataset")
```



### Teste de Aderência utilizando Qui-Quadrado

```
frequencias_observadas_classificacao_peso = tab_freq_classificacao_peso_corporal['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_classificacao_peso, f_esperadas=None)
```

--- Interpretação ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 14.3306  
 Valor-p (p-value): 0.0262  
 Como p (0.0262) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).  
 Conclusão: A distribuição NÃO é estatisticamente balanceada.  
 (np.float64(14.330648981525345), np.float64(0.02615299868612885))

### Razão de desbalanceamento

```
qtd_min_classificacao_peso_corporal = tab_freq_classificacao_peso_corporal['porcentagem'].min()
qtd_max_classificacao_peso_corporal = tab_freq_classificacao_peso_corporal['porcentagem'].max()
razao_desbalanceamento_classificacao_peso_corporal = calcular_razao_desbalanceamento(quantidade_maxima = qtd_max_classificacao_
```

--- Razão de Desbalanceamento (IR) ---  
 Quantidade da classe majoritária: 16.63  
 Quantidade da classe minoritária: 12.88  
 Razão de Desbalanceamento (IR): 1.29

--- Interpretação da Razão de Desbalanceamento ---  
● Balanceado. Não faça nada.

## ▼ Análise variáveis categóricas nominais

`genero`, `historico_familiar`, `favc`, `fumante`, `scc`, `mtrans`

### Técnicas utilizadas:

#### Análise univariada:

- Tabela de frequência
- Gráfico de barras
- Teste de aderência (univariado) utilizando qui-quadrado

#### Análise bi-varuada:

- Tabela cruzada
- Gráfico de barras empilhado

- Teste de Independência (Bivariado) utilizando qui-quadrado

## ▼ Análise variável `genero`

```
print(f"Descrição da variavel {dicionario_dados_nivel_obesidade['genero'][novo_nome]}: {dicionario_dados_nivel_obesidade[novo_nome]}")
Descrição da variavel genero: Genero
```

### Conclusão:

A análise univariada mostrou que a amostra é estatisticamente balanceada em relação aos grupos que compõe a variável gênero.

A análise bi-variaada apresenta um gráfico empilhado onde as barras de feminino e masculino da variável gênero estão distribuídas de forma diferente o que mostra um poder preditivo em relação a variável alvo. Isto é confirmado pelo retorno de estatisticamente relevante a associação entre as variáveis do teste qui-quadrado.

Essa feature é candidata para ser utilizada no modelo preditivo.

### Sugestão para a feature `genero`:

- Realizar a transformação dos valores que atualmente são do tipo objeto, em inteiro utilizando a técnica de `label_encoding`.

## Análise univariada

### Tabela de Freqüência

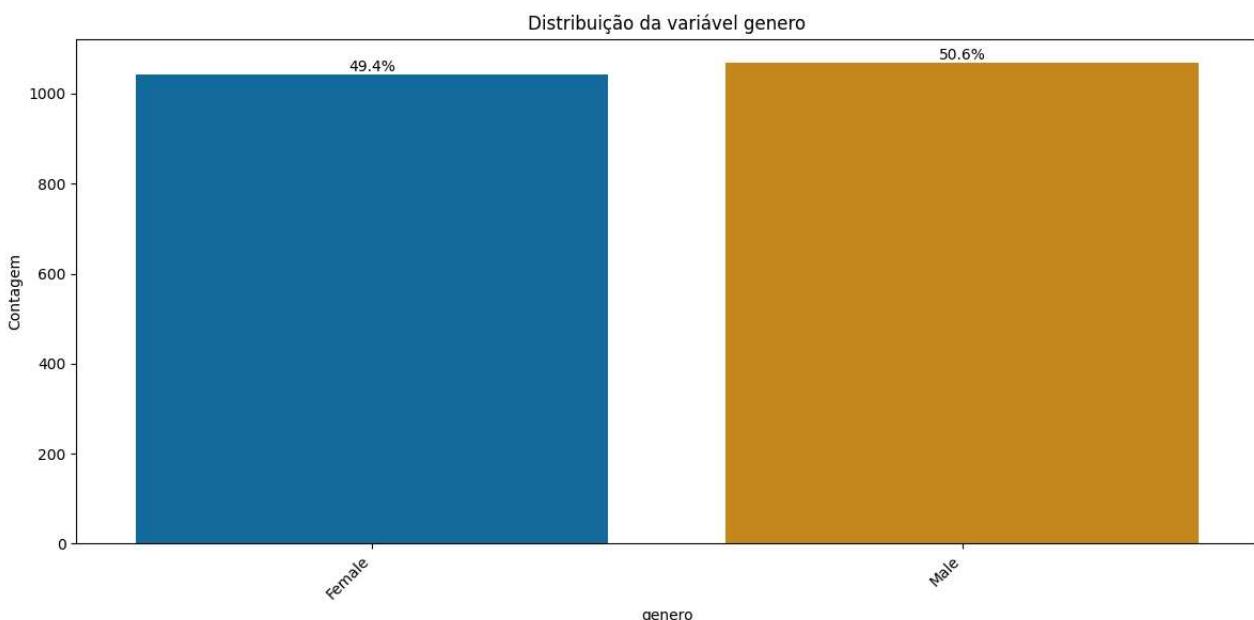
```
tab_freq_classificacao_genero = criar_dataframe_frequencia(df = df, coluna = 'genero')
display(tab_freq_classificacao_genero)
```

	categoria	quantidade	porcentagem	
0	Male	1068	50.59	
1	Female	1043	49.41	

Próximas etapas: ([Gerar código com tab\\_freq\\_classificacao\\_genero](#) | [New interactive sheet](#))

### Gráfico de barras

```
gera_grafico_barras(df = df , x_col = 'genero', title = "Distribuição da variável genero")
```



### Teste de aderência (univariado)

```
frequencias_observadas_genero = tab_freq_classificacao_genero['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_genero)
```

```
--- Interpretação ---
Estatística Qui-Quadrado ( $\chi^2$ ): 0.2961
Valor-p (p-value): 0.5864
Como p (0.5864) > 0.05, não podemos rejeitar a Hipótese Nula ( $H_0$ ).
Conclusão: A distribuição PODE ser considerada estatisticamente balanceada.
(np.float64(0.29606821411653245), np.float64(0.5863578616481406))
```

## Análise Bi-variada

genero vs classificacao\_peso\_alvo

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'genero',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_colunas = ordem_classificacao_peso_corporal
)
```

--- Análise da Feature: 'genero' vs. Alvo: 'classificacao\_peso\_corporal' ---

--- 1. Resultados do Teste Qui-Quadrado ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 657.7462  
 P-valor (p-value): 8.089e-139  
 Graus de Liberdade (dof): 6

Interpretação (p < 0.05): REJEITA H0.  
 CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

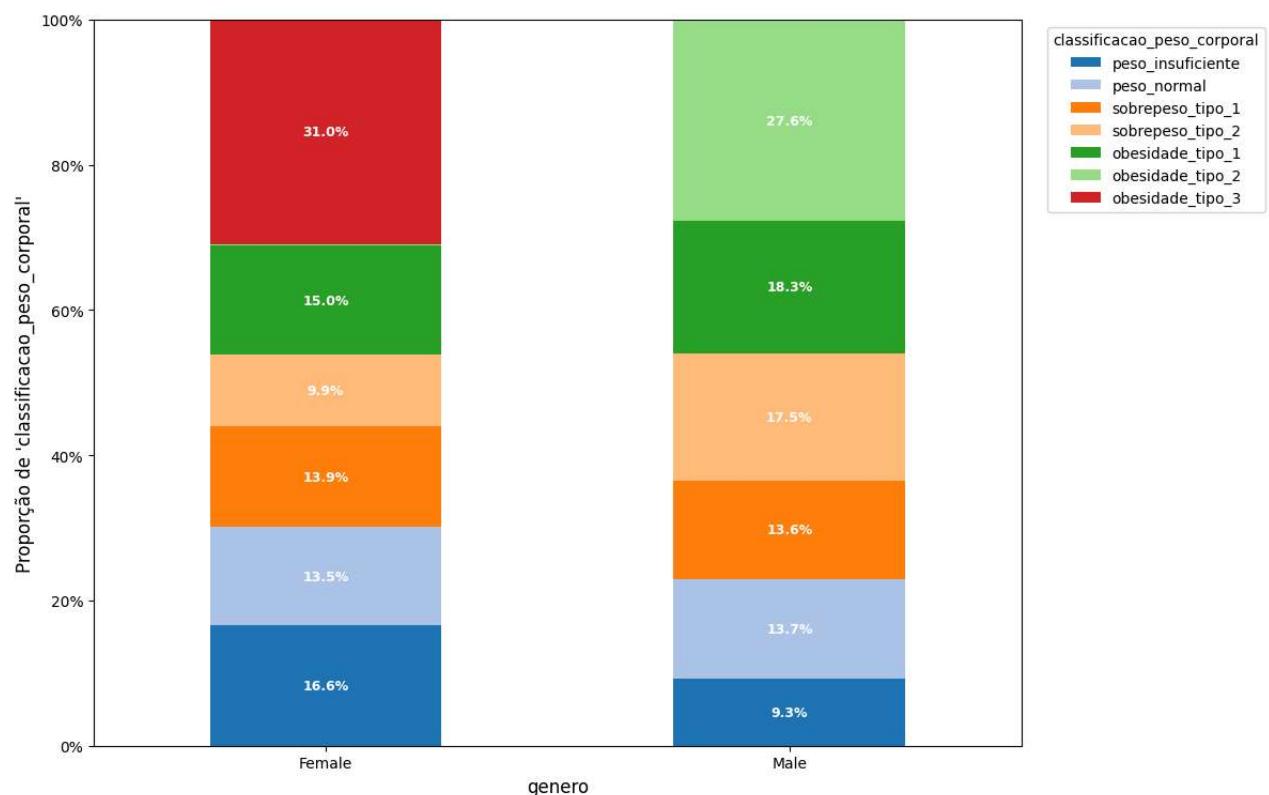
=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---

Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobrepeso\_tipo\_1', 'sobrepeso\_tipo\_2', 'obesidade\_tipo\_1', 'obesidade\_tipo\_2', 'obesidade\_tipo\_3']

genero	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobrepeso_tipo_1	sobrepeso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
Female		16.59	13.52	13.90	9.88	14.96		
Male		9.27	13.67	13.58	17.51	18.26		

Distribuição de 'classificacao\_peso\_corporal' por 'genero'



## ▼ Análise variável historico\_familiar

```
print(f"Descrição da variavel {dicionario_dados_nivel_obesidade['historico_familiar']['novo_nome']}: {dicionario_dados_nivel_obesidade['historico_familiar']}
```

Descrição da variavel historico\_familiar: Algum membro da familia sofreu ou sofre de excesso de peso?

### Conclusão:

A análise univariada mostrou que a amostra é estatisticamente desbalanceada em relação aos grupos que compõe a variável historico\_familiar. No entanto, ainda temos uma amostra grande o suficiente do grupo minoritário para alimentar o modelo de machine learning.

learning.

A análise bi-variaada apresenta um gráfico empilhado onde as barras de sim e não da variável historico\_familiar estão distribuídas de forma diferente o que mostra um poder preditivo em relação a variável alvo. Isto é confirmado pelo retorno de estatisticamente relevante a associação entre as variáveis do teste qui-quadrado.

Essa feature é uma candidata para ser utilizada no modelo preditivo.

#### Sugestão para a feature `historico_familiar`:

- Realizar a transformação dos valores que atualmente são do tipo objeto, em inteiro utilizando a técnica de `label_encoding`.

#### Análise univariada

##### Tabela de Frequência

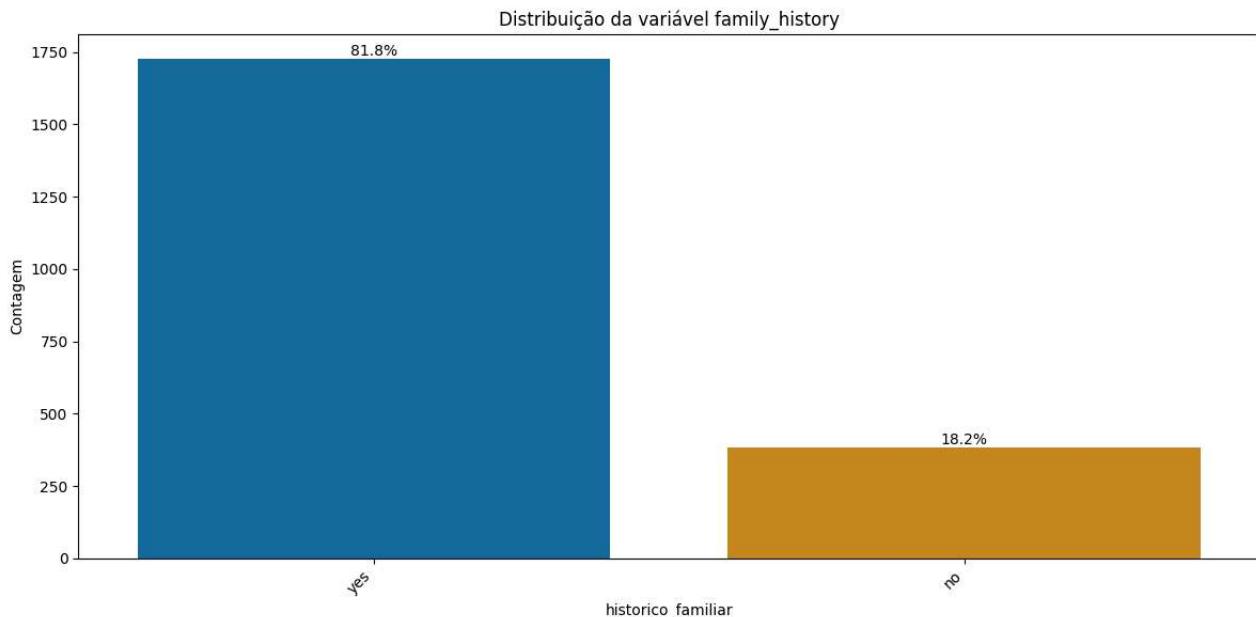
```
tab_freq_classificacao_historico_familiar = criar_dataframe_frequencia(df = df, coluna = 'historico_familiar')
display(tab_freq_classificacao_historico_familiar)
```

	categoria	quantidade	porcentagem	
0	yes	1726	81.76	█
1	no	385	18.24	██████

Próximas etapas: [Gerar código com tab\\_freq\\_classificacao\\_historico\\_familiar](#) [New interactive sheet](#)

#### Gráfico de barras

```
gera_grafico_barras(df = df , x_col = 'historico_familiar', title = f"Distribuição da variável {dicionario_dados_nivel_obe":
```



#### Teste de aderência (univariado)

```
frequencias_observadas_historico_familiar = tab_freq_classificacao_historico_familiar['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_historico_familiar)
```

```
--- Interpretação ---
Estatística Qui-Quadrado ( $\chi^2$ ): 851.8622
Valor-p (p-value): 0.0000
Como p (0.0000) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).
Conclusão: A distribuição NÃO é estatisticamente balanceada.
(np.float64(851.8621506395074), np.float64(2.862407015229573e-187))
```

## Análise Bi-variada

frequencias\_observadas\_historico\_familiar vs classificacao\_peso\_alvo

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'historico_familiar',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_colunas = ordem_classificacao_peso_corporal
)
```

--- Análise da Feature: 'historico\_familiar' vs. Alvo: 'classificacao\_peso\_corporal' ---

--- 1. Resultados do Teste Qui-Quadrado ---

Estatística Qui-Quadrado ( $\chi^2$ ): 621.9794

P-valor (p-value): 4.228e-131

Graus de Liberdade (dof): 6

Interpretação (p < 0.05): REJEITA H0.

CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

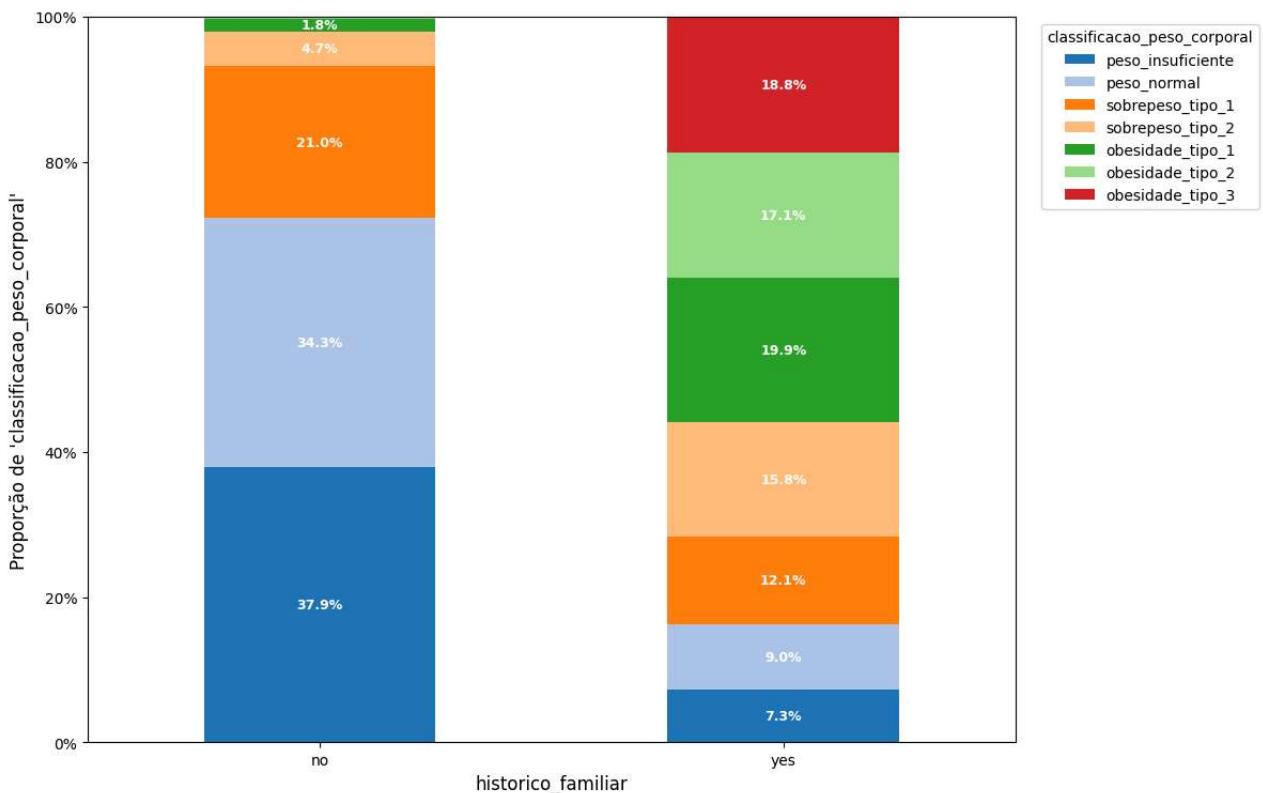
=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---

Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobrepeso\_tipo\_1', 'sobrepeso\_tipo\_2', 'obesidade\_tipo\_1', 'obesidade\_tipo\_2', 'obesidade\_tipo\_3']

	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobrepeso_tipo_1	sobrepeso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
	historico_familiar	no	yes	no	yes	no	yes	no
no		37.92	34.29	21.04	4.68	1.82		
yes		7.30	8.98	12.11	15.76	19.93		

Distribuição de 'classificacao\_peso\_corporal' por 'historico\_familiar'



## Análise variável favc

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['fvc']['nome_original']}: {dicionario_dados_nivel_obesidade['fvc']['desc']}
```

Descrição da variável FAVC: Voce come alimentos altamente calóricos com frequencia?

### Conclusão:

A análise univariada mostrou que a amostra é estatisticamente desbalanceada em relação aos grupos que compõe a variável favc. No entanto, ainda temos uma amostra grande o suficiente do grupo minoritário para alimentar o modelo de machine learning.

A análise bi-variaada apresenta um gráfico empilhado onde as barras de sim e não da variável favc estão distribuídas de forma diferente o que mostra um poder preditivo em relação a variável alvo. Isto é confirmado pelo retorno de estatisticamente relevante a associação entre as variáveis do teste qui-quadrado.

Essa feature é uma candidata para ser utilizada no modelo preditivo.

### Sugestão para a feature favc:

- Realizar a transformação dos valores que atualmente são do tipo objeto, em inteiro utilizando uma técnica de `label_encoding`.

### Análise univariada

#### Tabela de Frequência

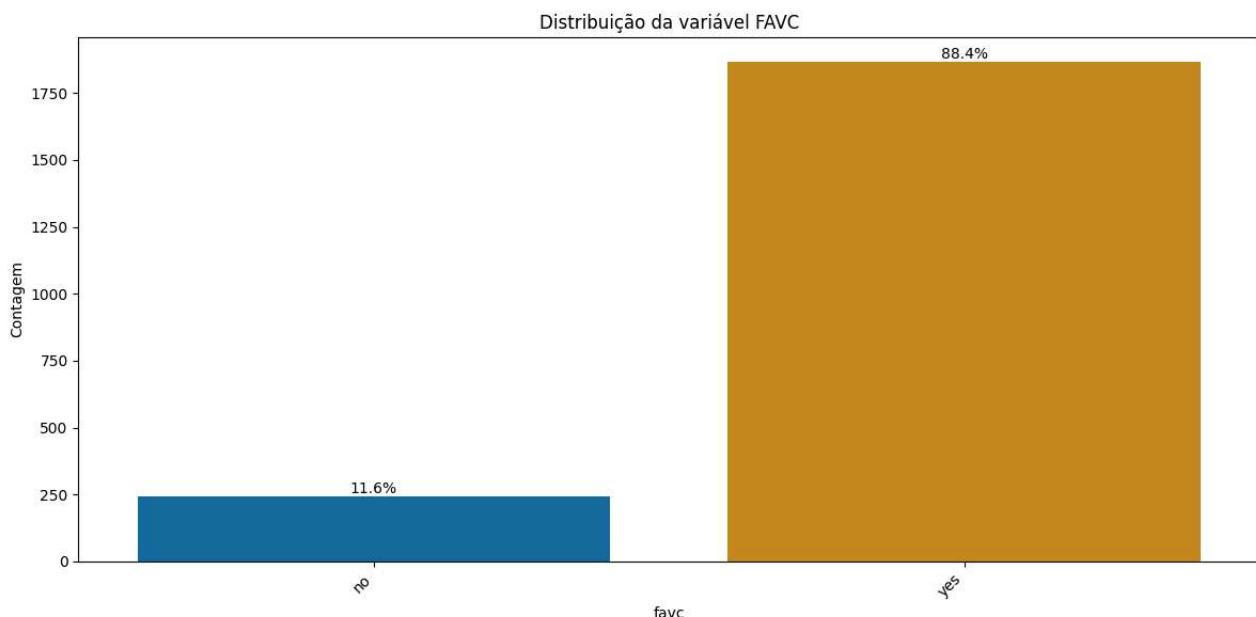
```
tab_freq_classificacao_favc = criar_dataframe_frequencia(df = df, coluna = 'fvc')
display(tab_freq_classificacao_favc)
```

	categoria	quantidade	porcentagem	
0	yes	1866	88.39	
1	no	245	11.61	

Próximas etapas: [Gerar código com tab\\_freq\\_classificacao\\_favc](#) [New interactive sheet](#)

### Gráfico de barras

```
gera_grafico_barras(df = df , x_col = 'fvc', title = f"Distribuição da variável {dicionario_dados_nivel_obesidade['fvc']}")
```



### Teste de aderência (univariado)

```
frequencias_observadas_favc = tab_freq_classificacao_favc['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_favc)
```

```
--- Interpretação ---
Estatística Qui-Quadrado ( $\chi^2$ ): 1244.7376
Valor-p (p-value): 0.0000
Como p (0.0000) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).
Conclusão: A distribuição NÃO é estatisticamente balanceada.
(np.float64(1244.7375651350071), np.float64(1.1553803549669082e-272))
```

## Análise Bi-variada

`favc` vs `classificacao_peso_alvo`

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'favc',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_colunas = ordem_classificacao_peso_corporal
)
```

--- Análise da Feature: 'favc' vs. Alvo: 'classificacao\_peso\_corporal' ---

--- 1. Resultados do Teste Qui-Quadrado ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 233.3413  
 P-valor (p-value): 1.482e-47  
 Graus de Liberdade (dof): 6

Interpretação (p < 0.05): REJEITA H0.

CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

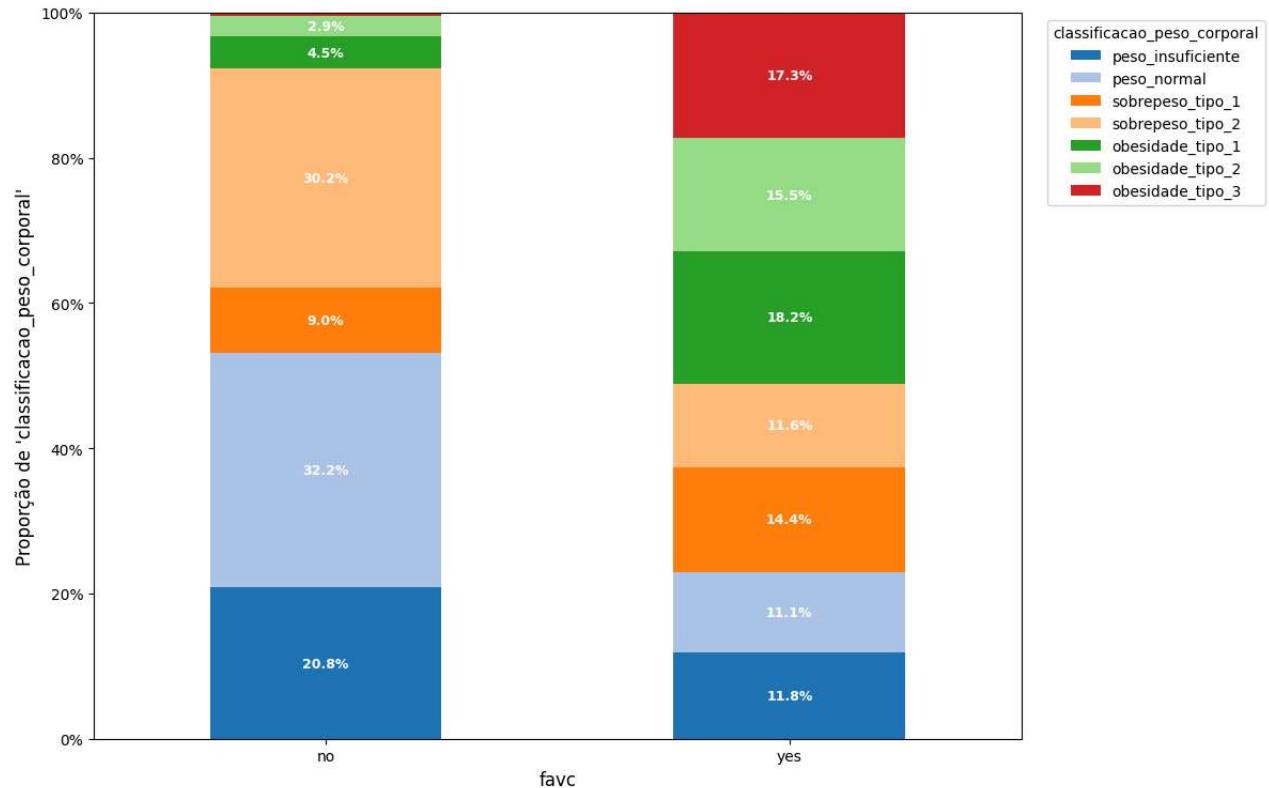
=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---

Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobrepeso\_tipo\_1', 'sobrepeso\_tipo\_2', 'obesidade\_tipo\_1', 'obesidade\_tipo\_2', 'obesidade\_tipo\_3']

	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobrepeso_tipo_1	sobrepeso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
<b>favc</b>								
<b>no</b>		20.82	32.24	8.98	30.20	4.49		
<b>yes</b>		11.84	11.15	14.36	11.58	18.22		

Distribuição de 'classificacao\_peso\_corporal' por 'favc'



## ▼ Análise variável fumante

```
print(f"Descrição da variavel {dicionario_dados_nivel_obesidade['fumante']['novo_nome']}: {dicionario_dados_nivel_obesidade['fumante']}
```

Descrição da variavel fumante: Voce fuma?

### Conclusão:

A análise univariada mostrou que a amostra é estatisticamente desbalanceada em relação aos grupos que compõe a variável fumante e temos uma amostra pequena do grupo sim.

Devido a esse motivo, apesar da análise bi-variada uma associação estatisticamente relevante no teste qui-quadrado, a recomendação é de não usar essa feature no modelo preditivo pois o risco de overfitting é maior que o possível ganho.

#### Sugestão para a feature `fumante`:

- Excluir a feature do modelo preditivo.
- Caso optem por utilizar a feature é necessário realizar a transformação dos valores que atualmente são do tipo objeto, em inteiro utilizando uma técnica de `label_encoding`.

#### Análise univariada

##### Tabela de Freqüência

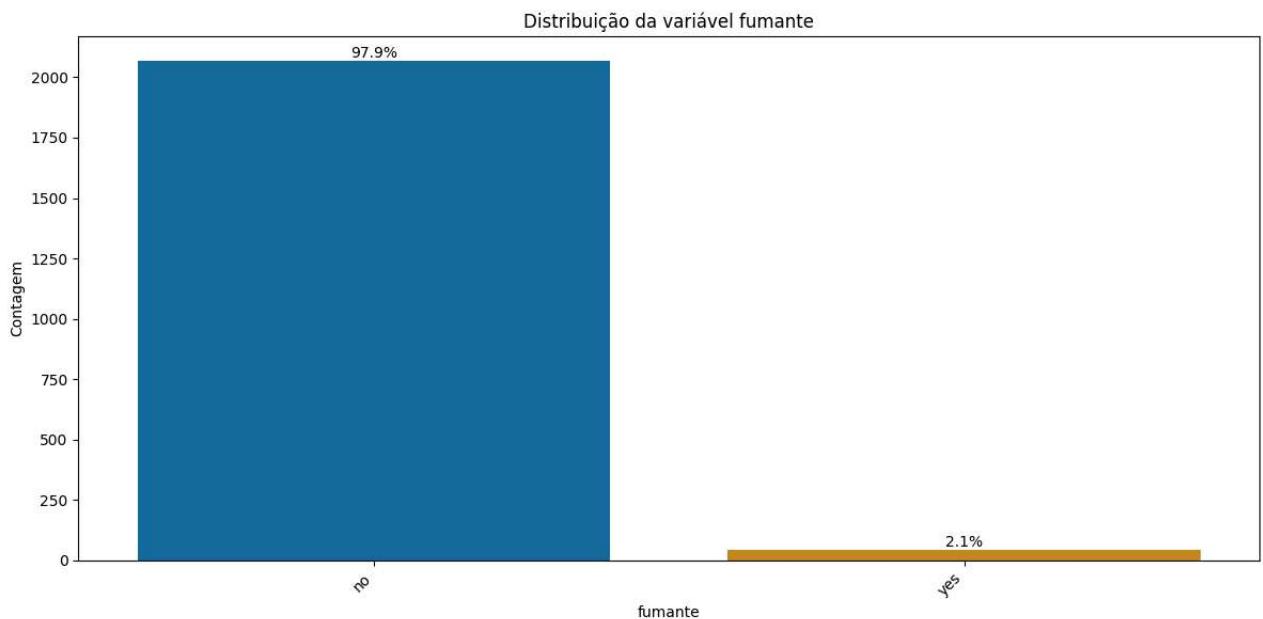
```
tab_freq_classificacao_fumante = criar_dataframe_frequencia(df = df, coluna = 'fumante')
display(tab_freq_classificacao_fumante)
```

	categoria	quantidade	porcentagem	
0	no	2067	97.92	
1	yes	44	2.08	

Próximas etapas: ([Gerar código com tab\\_freq\\_classificacao\\_fumante](#)) ([New interactive sheet](#))

#### Gráfico de barras

```
gera_grafico_barras(df = df , x_col = 'fumante', title = f"Distribuição da variável {dicionario_dados_nivel_obesidade['fumante']}
```



#### Teste de aderência (univariado)

```
frequencias_observadas_fumante = tab_freq_classificacao_fumante['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_fumante)
```

```
--- Interpretação ---
Estatística Qui-Quadrado ( $\chi^2$ ): 1938.6684
Valor-p (p-value): 0.0000
Como p (0.0000) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).
Conclusão: A distribuição NÃO é estatisticamente balanceada.
(np.float64(1938.6684036001895), np.float64(0.0))
```

#### Análise Bi-variada

favc vs classificacao\_peso\_alvo

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'fumante',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_colunas = ordem_classificao_peso_corporal
)
```

--- Análise da Feature: 'fumante' vs. Alvo: 'classificacao\_peso\_corporal' ---

--- 1. Resultados do Teste Qui-Quadrado ---

Estatística Qui-Quadrado ( $\chi^2$ ): 32.1378

P-valor (p-value): 1.535e-05

Graus de Liberdade (dof): 6

Interpretação (p < 0.05): REJEITA H0.

CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

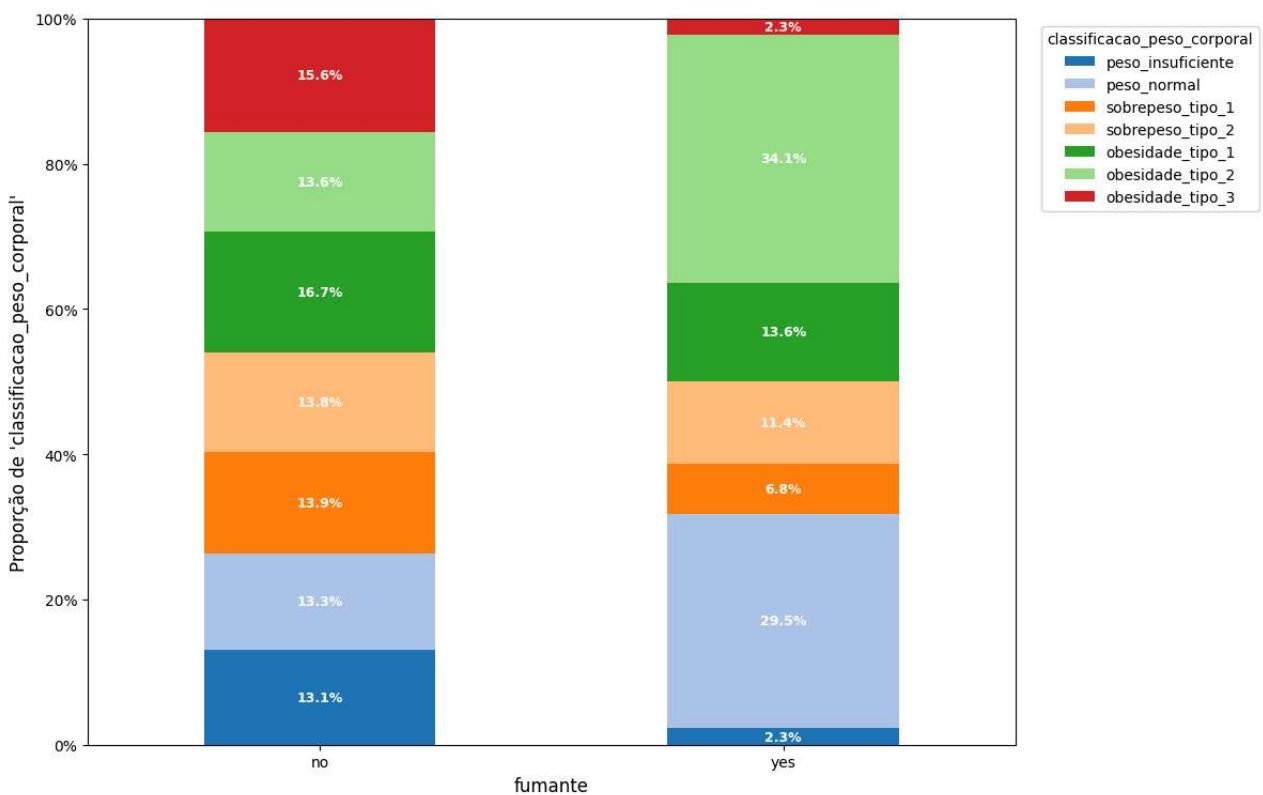
=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---

Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobrepeso\_tipo\_1', 'sobrepeso\_tipo\_2', 'obesi

	fumante	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobrepeso_tipo_1	sobrepeso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
	no		13.11	13.26	13.88	13.79	13.79	16.69	
	yes		2.27	29.55	6.82	11.36	11.36	13.64	

Distribuição de 'classificacao\_peso\_corporal' por 'fumante'



## ▼ Análise variável scc

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['scc']['nome_original']}: {dicionario_dados_nivel_obesidade['scc']['descricao']}
```

Descrição da variável SCC: Voce monitora as calorias que ingere diariamente?

### Conclusão:

A análise univariada mostrou que a amostra é estatisticamente desbalanceada em relação aos grupos que compõe a variável fumante e temos uma amostra pequena do grupo sim. O volume de amostras está em uma zona de atenção devido ao risco de overfitting. No entanto a associação preditiva é significativamente relevante.

### Sugestão para a feature `scc`:

- Manter a feature para o treinamento do modelo preditivo.
- Após o treinamento Verificar a importância da feature. Se a importância dela for pequena para o modelo, retirar a feature do treinamento do modelo é uma boa opção.
- Realizar a validação cruzada para ter certeza se devemos continuar ou não com a feature.
- Caso optem por utilizar a feature é necessário realizar a transformação dos valores que atualmente são do tipo objeto, em inteiro utilizando `label_encoding`.

### Análise univariada

#### Tabela de Freqüência

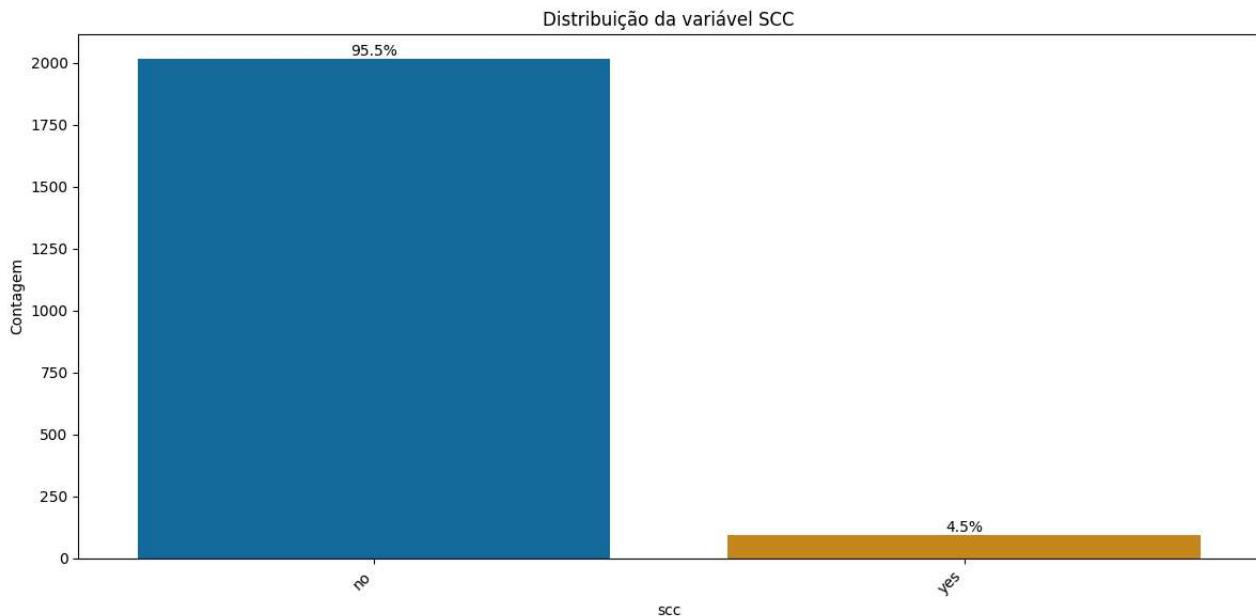
```
tab_freq_classificacao_scc = criar_dataframe_frequencia(df = df, coluna = 'scc')
display(tab_freq_classificacao_scc)
```

	categoria	quantidade	porcentagem	
0	no	2015	95.45	
1	yes	96	4.55	

Próximas etapas: ([Gerar código com tab\\_freq\\_classificacao\\_scc](#)) [New interactive sheet](#)

### Gráfico de barras

```
gera_grafico_barras(df = df , x_col = 'scc', title = f"Distribuição da variável {dicionario_dados_nivel_obesidade['scc']['nome_original']}
```



#### Teste de aderência (univariado)

```
frequencias_observadas_scc = tab_freq_classificacao_scc['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_scc)
```

```
--- Interpretação ---
Estatística Qui-Quadrado ( $\chi^2$ ): 1744.4628
Valor-p (p-value): 0.0000
Como p (0.0000) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).
Conclusão: A distribuição NÃO é estatisticamente balanceada.
(np.float64(1744.462813832307), np.float64(0.0))
```

## Análise Bi-variada

scc vs classificacao\_peso\_alvo

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'scc',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_colunas = ordem_classificao_peso_corporal
)
```

```
--- Análise da Feature: 'scc' vs. Alvo: 'classificacao_peso_corporal' ---
```

--- 1. Resultados do Teste Qui-Quadrado ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 123.0239  
 P-valor (p-value): 3.773e-24  
 Graus de Liberdade (dof): 6

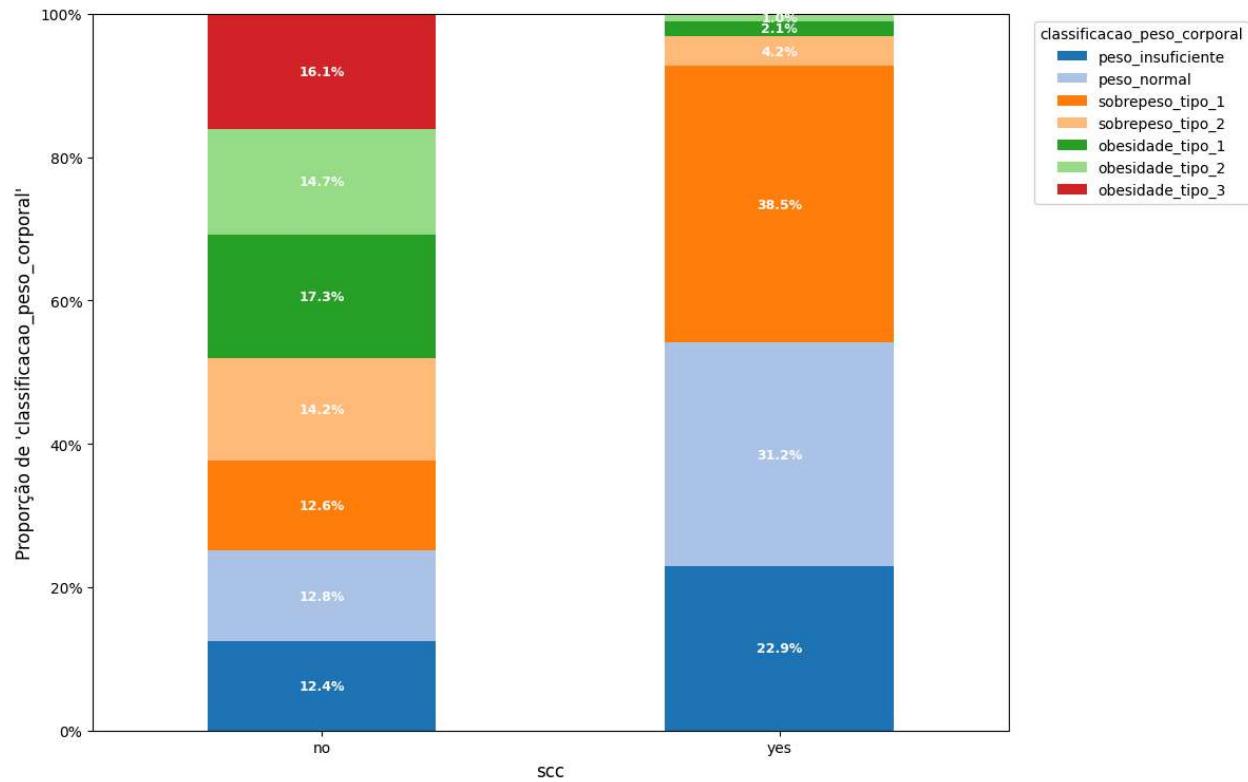
Interpretação (p < 0.05): REJEITA H0.  
 CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---  
 Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobrepeso\_tipo\_1', 'sobrepeso\_tipo\_2', 'obesidade\_tipo\_1', 'obesidade\_tipo\_2', 'obesidade\_tipo\_3']

	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobrepeso_tipo_1	sobrepeso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
scc								
no		12.41	12.75	12.56	14.19	17.32		
yes		22.92	31.25	38.54	4.17	2.08		

Distribuição de 'classificacao\_peso\_corporal' por 'scc'



## ▼ Análise variável mtrans

```
print(f"Descrição da variavel {dicionario_dados_nivel_obesidade['mtrans']['nome_original']}: {dicionario_dados_nivel_obesidade['mtrans']['descricao']}
```

Descrição da variavel MTRANS: Qual meio de transporte voce costuma usar?

### Conclusão:

A análise univariada mostrou que a amostra é estatisticamente desbalanceada em relação aos grupos que compõe a variável mtrans. No entanto as amostras de `transporte_publico` (1580), `carro` (457) possuem uma boa quantidade de registros para realizar o

treinamento do modelo. A quantidade de valores de `bicicleta` (7), `moto` (11) estão em uma zonas de descarte devido a baixa volumetria, enquanto `caminhando` (56) está em uma zona de cuidado.

A análise bivariada mostrou uma associação estatisticamente relevante entre as categorias e a variável alvo. As categorias raras `bicicleta` (7), `moto` (11) e `andando` (56) possuem um padrão parecido com acumulo de registros relacionados a `peso_normal` e `baixa obesidade`. Dessa forma é possível agrupá-las em uma única categoria para treinamento, deixando elas ainda na zona de atenção devido a quantidade de 74 casos, mas com a possibilidade de utilização devido ao potencial preditivo.

#### Sugestão para a feature `mtrans`:

- Transformar as categorias `bicicleta`, `moto` e `caminhando` em uma única categoria .
- Utilizar a técnica `one_hot_encoding` transformando em 2 categorias para evitar a armadilha das Variáveis Dummy.
- Após o treinamento Verificar a importância da feature. Se a importância dela for pequena para o modelo, retirar a feature do treinamento do modelo é uma boa opção.
- Realizar a validação cruzada para ter certeza se devemos continuar ou não com a feature.

### Análise univariada

#### Tabela de Frequência

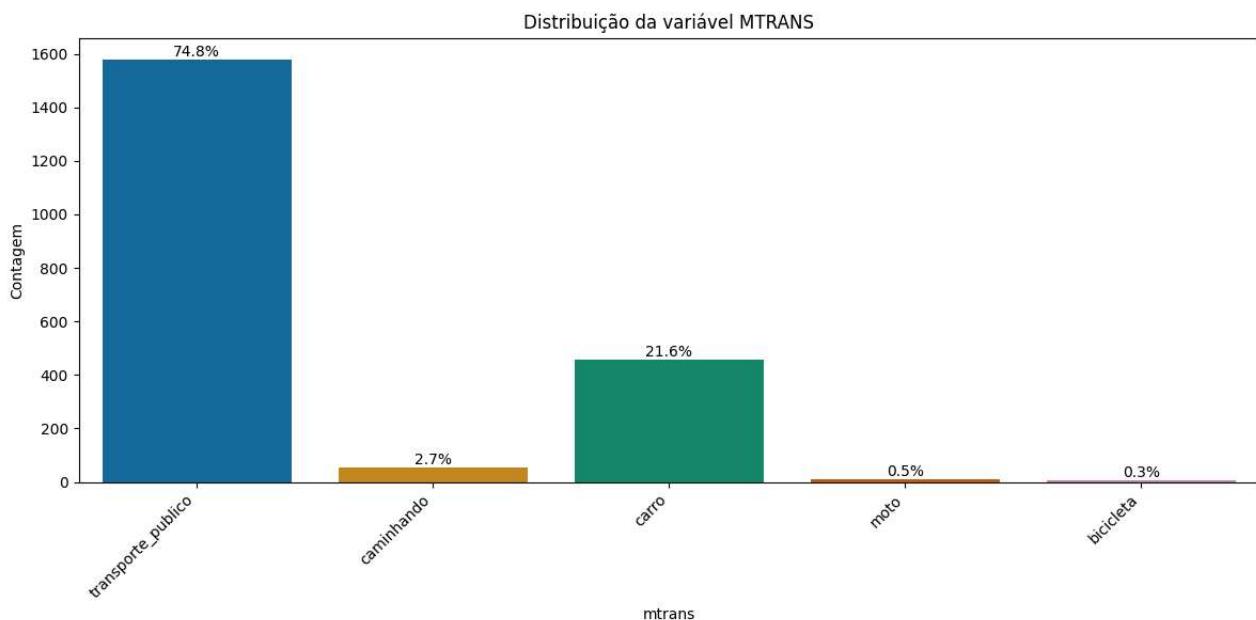
```
tab_freq_classificacao_mtrans = criar_dataframe_frequencia(df = df, coluna = 'mtrans')
display(tab_freq_classificacao_mtrans)
```

	categoria	quantidade	porcentagem	
0	transporte_publico	1580	74.85	
1	carro	457	21.65	
2	caminhando	56	2.65	
3	moto	11	0.52	
4	bicicleta	7	0.33	

Próximas etapas: ([Gerar código com tab\\_freq\\_classificacao\\_mtrans](#)) ([New interactive sheet](#))

#### Gráfico de barras

```
gera_grafico_barras(df = df , x_col = 'mtrans', title = f'Distribuição da variável {dicionario_dados_nivel_obesidade['mtr
```



#### Teste de aderência (univariado)

```
frequencias_observadas_mtrans = tab_freq_classificacao_mtrans['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_mtrans)
```

```
--- Interpretação ---
Estatística Qui-Quadrado ( $\chi^2$ ): 4304.3363
Valor-p (p-value): 0.0000
Como p (0.0000) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).
Conclusão: A distribuição NÃO é estatisticamente balanceada.
(np.float64(4304.3363334912365), np.float64(0.0))
```

## Análise Bi-variada

mtrans vs classificacao\_peso\_alvo

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'mtrans',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_colunas = ordem_classificacao_peso_corporal
)
```

--- Análise da Feature: 'mtrans' vs. Alvo: 'classificacao\_peso\_corporal' ---

--- 1. Resultados do Teste Qui-Quadrado ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 292.5939  
 P-valor (p-value): 5.178e-48  
 Graus de Liberdade (dof): 24

Interpretação (p < 0.05): REJEITA H0.

CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

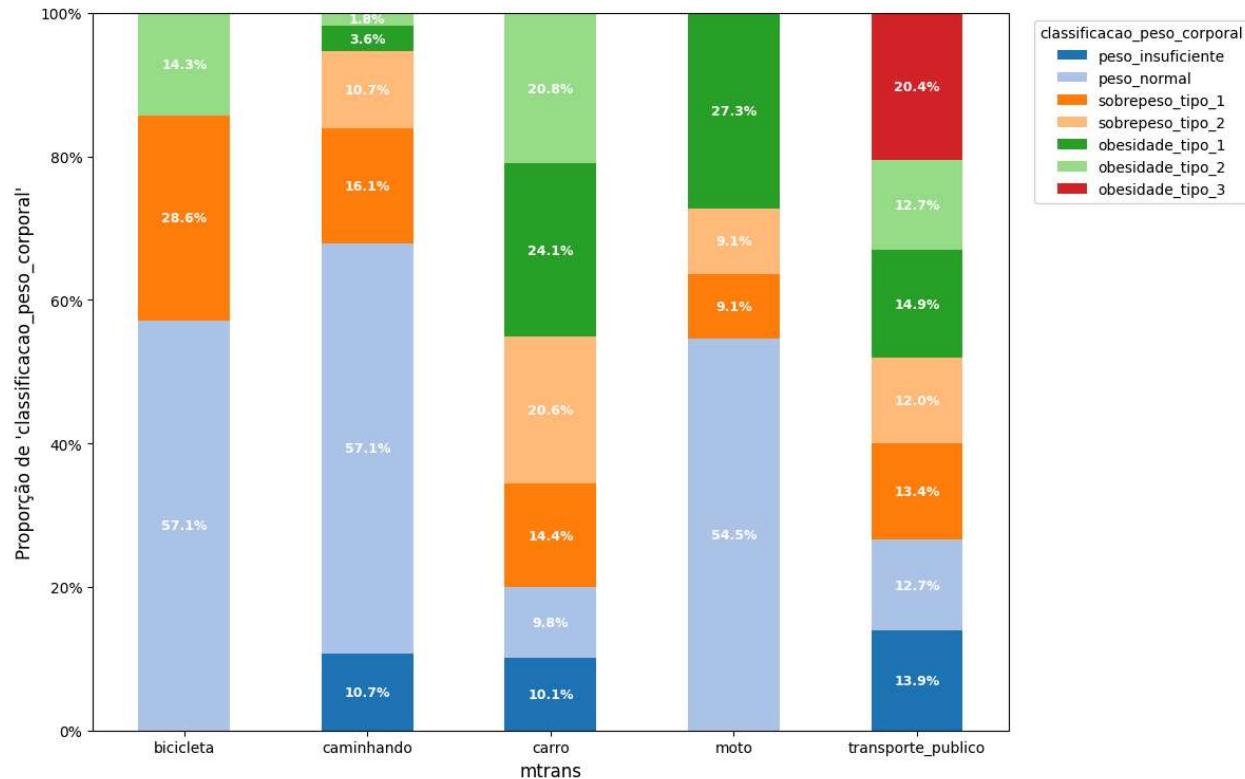
=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---

Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobre peso\_tipo\_1', 'sobre peso\_tipo\_2', 'obesidade\_tipo\_1', 'obesidade\_tipo\_2', 'obesidade\_tipo\_3']

	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobre peso_tipo_1	sobre peso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
mtrans								
bicicleta		0.00	57.14	28.57	0.00	0.00	0.00	0.00
caminhando		10.71	57.14	16.07	10.71	3.57	0.00	0.00
carro		10.07	9.85	14.44	20.57	24.07	0.00	0.00
moto		0.00	54.55	9.09	9.09	27.27	0.00	0.00
transporte_publico		13.92	12.66	13.42	11.96	14.94	0.00	0.00

Distribuição de 'classificacao\_peso\_corporal' por 'mtrans'



## ▼ Análise variáveis categóricas ordinais

Variáveis categóricas ordinais:

[caec], [calc]

## Técnicas utilizadas:

### Analise univariada:

- Tabela de frequênciia
- Gráfico de barras
- Teste de aderência (univariado) utilizando qui-quadrado

### Análise bi-variada:

- Tabela cruzada
- Gráfico de barras emoilhado
- Teste de Independência (Bivariado) utilizando qui-quadrado

## ▼ Análise variável caec

```
print(f"Descrição da variavel {dicionario_dados_nivel_obesidade['caec']['novo_nome']}: {dicionario_dados_nivel_obesidade['caec']}
```

Descrição da variavel caec: Voce come alguma coisa entre as refeicoes?

### Valores únicos da variável

```
print(f'Os valores únicos da variável {dicionario_dados_nivel_obesidade["caec"]["novo_nome"]} são: {df.caec.unique()}'
```

Os valores únicos da variável caec são: ['as\_vezes' 'frequentemente' 'sempre' 'nunca']

### Conclusão:

A variável caec é uma amostra desbalanceada, com `nunca` e `sempre` no limite inferior entre a zona de perigo e descarte de uma variável. A analise bivariada sinalizou um associação estatisticamente significante. Diferente da coluna `mtrans` juntar categorias não é uma opção, pois elas possuem associações bem diferentes com a variavel alvo.

Neste caso devido ao poder preditivo sugiro manter a feature no treinamento utilizando a tecnica de ordinal\_encoder, mas analisaria a relevancia da mesma no treinamento do modelo para verificar se devemos mante-la e assumir o risco de overfitting.

### Sugestão para a feature caec :

- Utilizar a técnica `OrdinalEncoder`.
- Verificar a importância da feature.
- Teste A/B comparando os scores na validação cruzada
- Caso a importância da feature e o teste A/B voltem sem relevância ou indicando overfitting retirar a mesma do treinamento.

### Análise univariada

#### Tabela de Frequênciia

```
tab_freq_classificacao_caec = criar_dataframe_frequencia(df = df, coluna = 'caec', ordem_categorias = ordem_frequencia)
display(tab_freq_classificacao_caec)
```

	categoria	quantidade	porcentagem	
0	nunca	51	2.42	
1	as_vezes	1765	83.61	
2	frequentemente	242	11.46	
3	sempre	53	2.51	

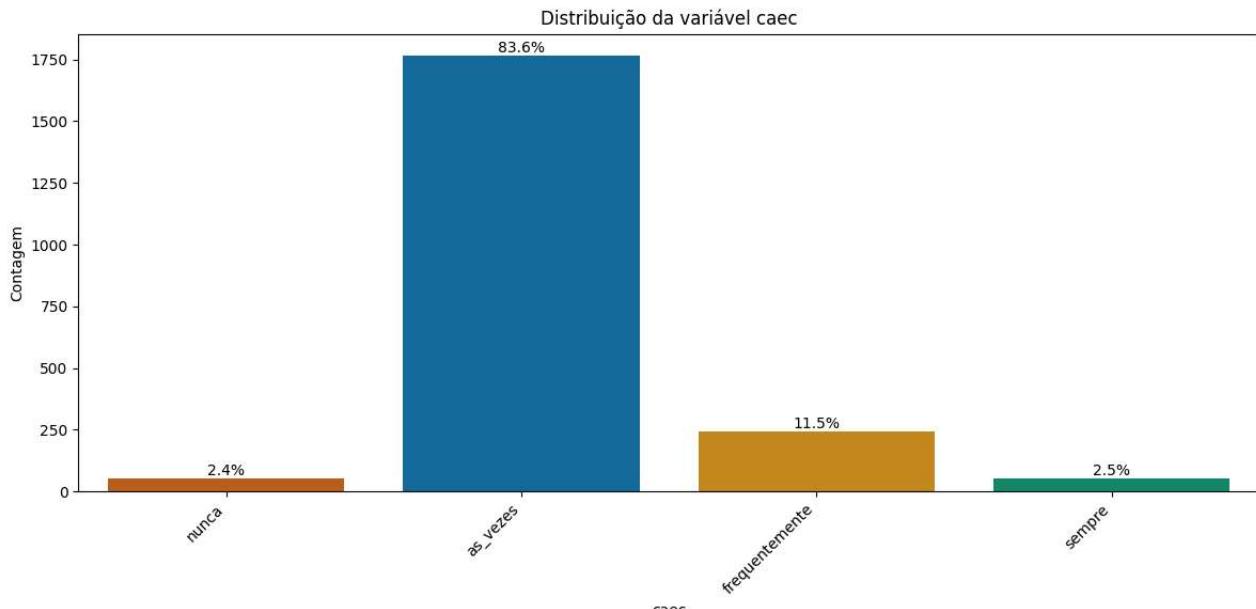
Próximas etapas: ([Gerar código com tab\\_freq\\_classificacao\\_caec](#)) ([New interactive sheet](#))

### Gráfico de barras

```
ordem_frequencia
```

```
['nunca', 'as_vezes', 'frequentemente', 'sempre']
```

```
gera_grafico_barras(df = df , x_col = 'caec', title = f"Distribuição da variável {dicionario_dados_nivel_obesidade['caec']}|
```



### Teste de aderência (univariado)

```
frequencias_observadas_caec = tab_freq_classificacao_caec['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_caec)
```

--- Interpretação ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 3913.0625  
 Valor-p (p-value): 0.0000  
 Como p (0.0000) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).  
 Conclusão: A distribuição NÃO é estatisticamente balanceada.  
 (np.float64(3913.0625), np.float64(0.0))

### Análise Bi-variada

`caec` vs `classificacao_peso_alvo`

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'caec',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_linhas = ordem_frequencia,
    ordem_colunas = ordem_classificacao_peso_corporal
)
```

--- Análise da Feature: 'caec' vs. Alvo: 'classificacao\_peso\_corporal' ---

--- 1. Resultados do Teste Qui-Quadrado ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 802.9773  
 P-valor (p-value): 7.384e-159  
 Graus de Liberdade (dof): 18

Interpretação (p < 0.05): REJEITA H0.

CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

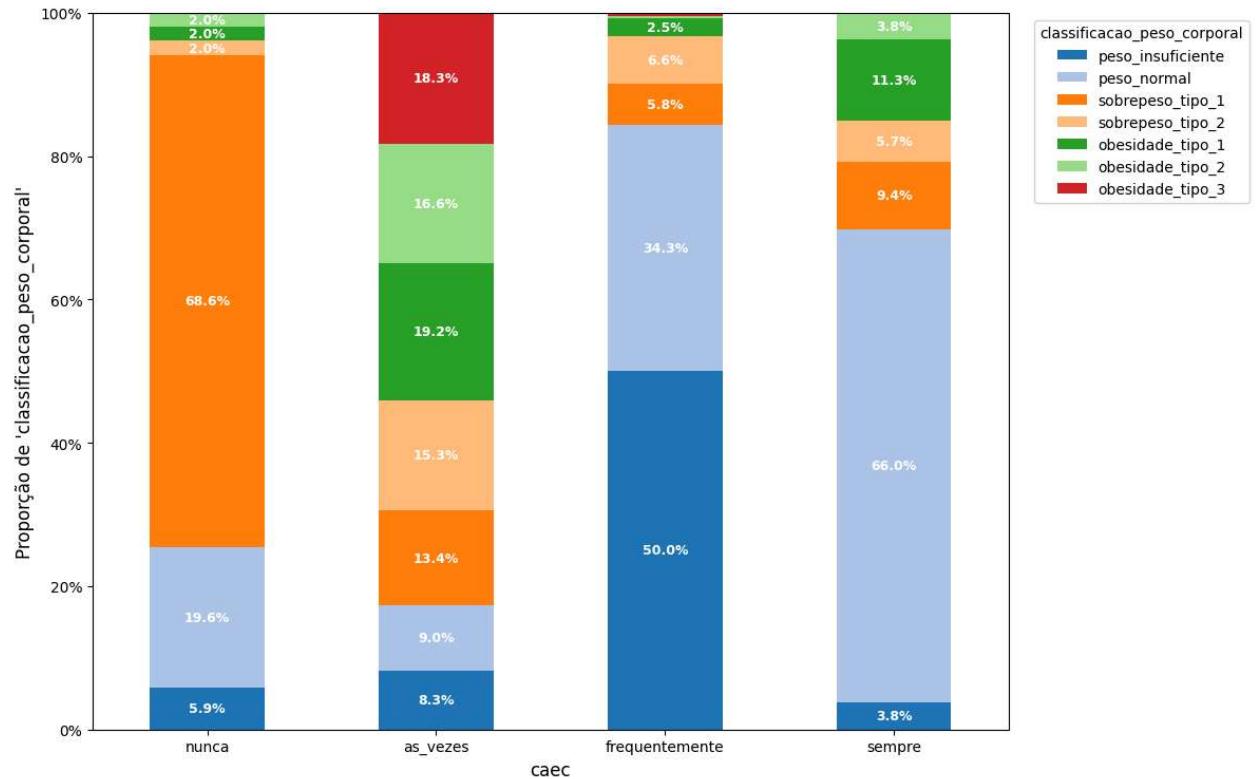
=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---

Ordenação de linhas (eixo X) aplicada: ['nunca', 'as\_vezes', 'frequentemente', 'sempre']  
 Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobrepeso\_tipo\_1', 'sobrepeso\_tipo\_2', 'obesidade\_tipo\_1', 'obesidade\_tipo\_2', 'obesidade\_tipo\_3']

	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobrepeso_tipo_1	sobrepeso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
caec								
nunca		5.88	19.61	68.63	1.96	1.96		
as_vezes		8.27	9.01	13.37	15.30	19.15		
frequentemente		50.00	34.30	5.79	6.61	2.48		
sempre		3.77	66.04	9.43	5.66	11.32		

Distribuição de 'classificacao\_peso\_corporal' por 'caec'



## ▼ Análise variável calc

```
print(f"Descrição da variavel {dicionario_dados_nivel_obesidade['calc']['novo_nome']}: {dicionario_dados_nivel_obesidade['calc']}
```

Descrição da variavel calc: Com que frequencia voce bebe alcool?

## Valores únicos da variável

```
print(f'Os valores únicos da variável {dicionario_dados_nivel_obesidade["calc"]["novo_nome"]} são: {df.calc.unique()}')
Os valores únicos da variável calc são: ['nunca' 'as_vezes' 'frequentemente' 'sempre']
```

### Conclusão :

A variável calc é estatisticamente desbalanceada e possui a categoria 'sempre' com apenas 1 registro, o que a torna rara e inviável para o treinamento.

A categoria 'frequentemente' está na zona de perigo para ser utilizada no treinamento de aprendizado de máquina, com 70 registros.

A análise Bi-variada mostra que existe poder preditivo entre as categorias e a variável alvo. A categoria 'frequentemente', em ordem de grandeza, é mais próxima da categoria 'sempre', por esse motivo 'sugiro juntá-las'. Outra opção seria excluir o registro.

Em qualquer um dos casos é interessante verificar após o treinamento a relevância da feature e da avaliação do teste A/B

### Sugestão para a feature `calc` :

- Manter a variável juntando as categorias frequentemente e sempre.
- Utilizar a técnica `OrdinalEncoder`.
- Verificar a importância da feature.
- Teste A/B comparando os scores na validação cruzada
- Caso a importância da feature e o teste A/B voltem sem relevância, ou indicando overfitting, retirar a mesma do treinamento.

## Análise univariada

### Tabela de Frequência

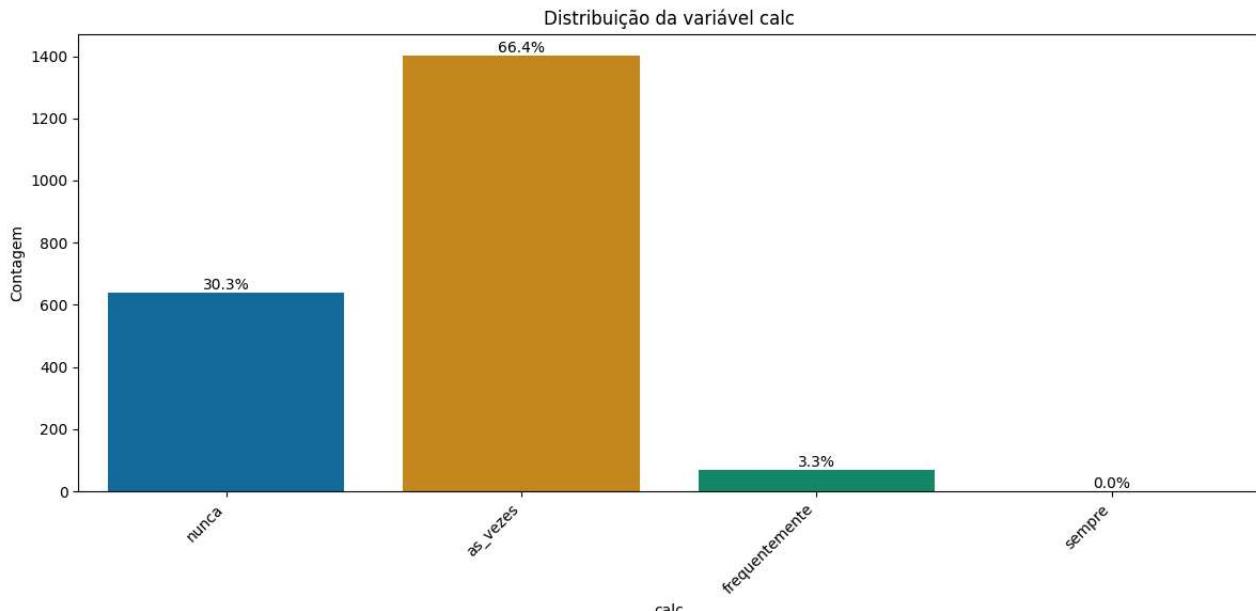
```
tab_freq_classificacao_calc = criar_dataframe_frequencia(df = df, coluna = 'calc', ordem_categorias = ordem_frequencia)
display(tab_freq_classificacao_calc)
```

	categoria	quantidade	porcentagem	
0	nunca	639	30.27	
1	as_vezes	1401	66.37	
2	frequentemente	70	3.32	
3	sempre	1	0.05	

Próximas etapas: ([Gerar código com tab\\_freq\\_classificacao\\_calc](#)) ([New interactive sheet](#))

### Gráfico de barras

```
gera_grafico_barras(df = df , x_col = 'calc', title = f'Distribuição da variável {dicionario_dados_nivel_obesidade["calc"]}' )
```



### Teste de aderência (univariado)

```
frequencias_observadas_calc = tab_freq_classificacao_calc['quantidade']
realizar_teste_qui_quadrado(frequencias_observadas_calc)
```

--- Interpretação ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 2391.1753  
 Valor-p (p-value): 0.0000  
 Como p (0.0000) <= 0.05, rejeitamos a Hipótese Nula ( $H_0$ ).  
 Conclusão: A distribuição NÃO é estatisticamente balanceada.  
 (np.float64(2391.175272382757), np.float64(0.0))

### Análise Bi-variada

`calc` vs `classificacao_peso_alvo`

```
analisar_variavel_categorica_bi_variada(
    df= df,
    coluna_feature = 'calc',
    coluna_alvo= 'classificacao_peso_corporal',
    ordem_linhas = ordem_frequencia,
    ordem_colunas = ordem_classificacao_peso_corporal
)
```

--- Análise da Feature: 'calc' vs. Alvo: 'classificacao\_peso\_corporal' ---

--- 1. Resultados do Teste Qui-Quadrado ---  
 Estatística Qui-Quadrado ( $\chi^2$ ): 338.5775  
 P-valor (p-value): 5.287e-61  
 Graus de Liberdade (dof): 18

Interpretação (p < 0.05): REJEITA H0.

CONCLUSÃO: Existe uma associação ESTATISTICAMENTE SIGNIFICATIVA entre as variáveis.

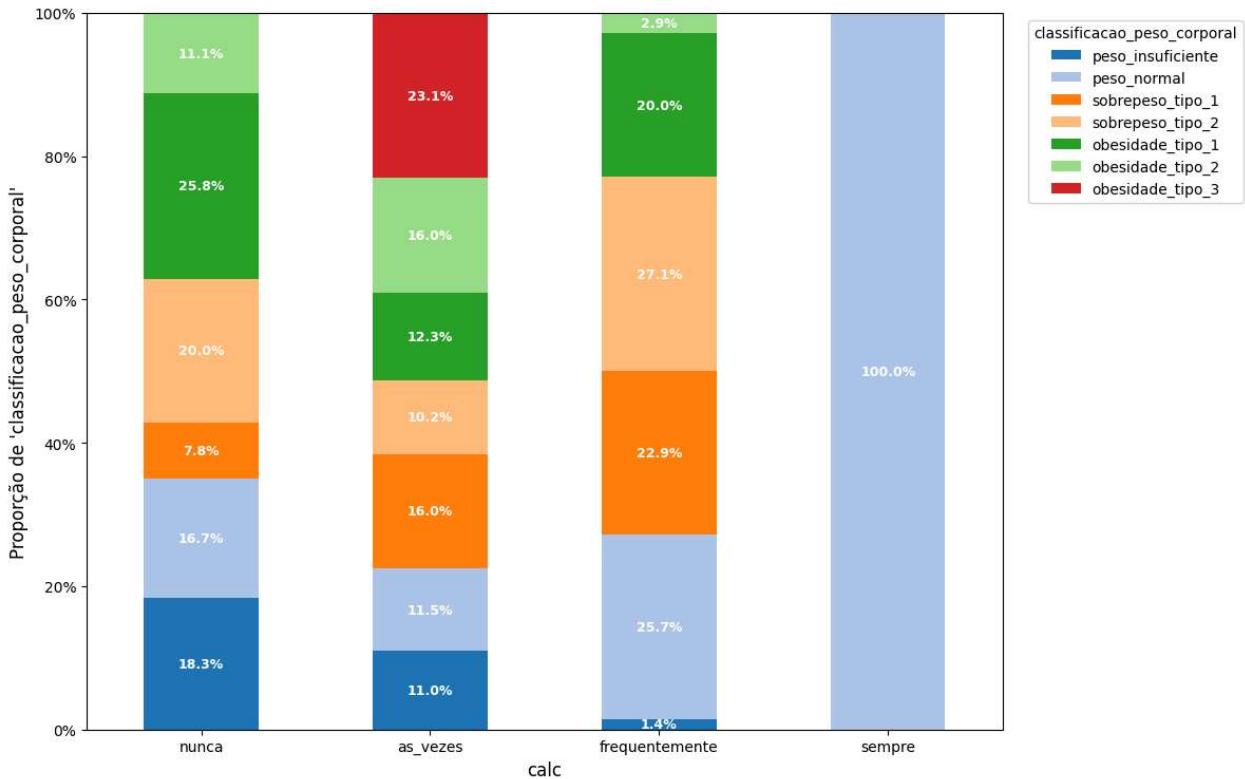
=====

--- 2. Visualização (Gráfico de Barras Empilhado 100%) ---

Ordenação de linhas (eixo X) aplicada: ['nunca', 'as\_vezes', 'frequentemente', 'sempre']  
 Ordenação de colunas (legenda) aplicada: ['peso\_insuficiente', 'peso\_normal', 'sobrepeso\_tipo\_1', 'sobrepeso\_tipo\_2', 'obesidade\_tipo\_1', 'obesidade\_tipo\_2', 'obesidade\_tipo\_3']

	classificacao_peso_corporal	peso_insuficiente	peso_normal	sobrepeso_tipo_1	sobrepeso_tipo_2	obesidade_tipo_1	obesidade_tipo_2	obesidade_tipo_3
calc								
nunca		18.31	16.74	7.82	20.03	25.82		
as_vezes		10.99	11.49	15.99	10.21	12.28		
frequentemente		1.43	25.71	22.86	27.14	20.00		
sempre		0.00	100.00	0.00	0.00	0.00		

Distribuição de 'classificacao\_peso\_corporal' por 'calc'



## ▼ Análise variáveis numéricas

### Veriáveis numéricas

idade, altura, peso, fcvc, ncp, ch20,faf, tue

## Técnicas utilizadas

### Análise Univariada

- Estatística descritiva
- Histograma
- Boxplot

### Análise Bivariada

- Boxplot agrupado
- Teste estatístico de análise de variância (ANOVA)
- Hipótese Nula(H0)

## ▼ Análise variável `idade`

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['idade']['novo_nome']}: {dicionario_dados_nivel_obesidade[
```

Descrição da variável idade: Idade

### Conclusão

Na análise univariada percebemos que a variável idade não possui erros, mas é fortemente assimétrica à direita, com uma concentração de pacientes jovens.

Na análise Bivariada temos diferença nas medianas de idade entre os grupos da variável alvo e o teste estatístico confirma a diferença estatisticamente relevante da distribuição da variável idade.

### Sugestões

- Incluir a variável `idade` no treinamento do modelo
- Devido à assimetria da variável idade deve passar por Feature Scaling (como `StandardScaler` ou `MinMaxScaler`) para normalizar sua distribuição.

### Visualização prévia dos dados

Esse dataset estava pronto quando recebemos para esse estudo. Analisando os dados da variável idade percebemos registros com 6 casas decimais, indicando que algum método estatístico foi utilizado na amostra. Seja para tratamento de dados ausentes ou para uma geração de dados sintéticos.

Para o objetivo do estudo podemos continuar.

```
df.idade
```

	idade
0	21.000000
1	21.000000
2	23.000000
3	27.000000
4	22.000000
...	...
2106	20.976842
2107	21.982942
2108	22.524036
2109	24.361936
2110	23.664709

2111 rows × 1 columns

**dtype:** float64

### Análise Univariada

#### Estatística descritiva

**Análise:**

- Os valores mínimos e máximos da amostra estão dentro do esperado para uma mostra de idade populacional.
- A média é maior que a mediana indicando assimetria

```
df['idade'].describe()
```

idade	
<b>count</b>	2111.000000
<b>mean</b>	24.312600
<b>std</b>	6.345968
<b>min</b>	14.000000
<b>25%</b>	19.947192
<b>50%</b>	22.777890
<b>75%</b>	26.000000
<b>max</b>	61.000000

**dtype:** float64

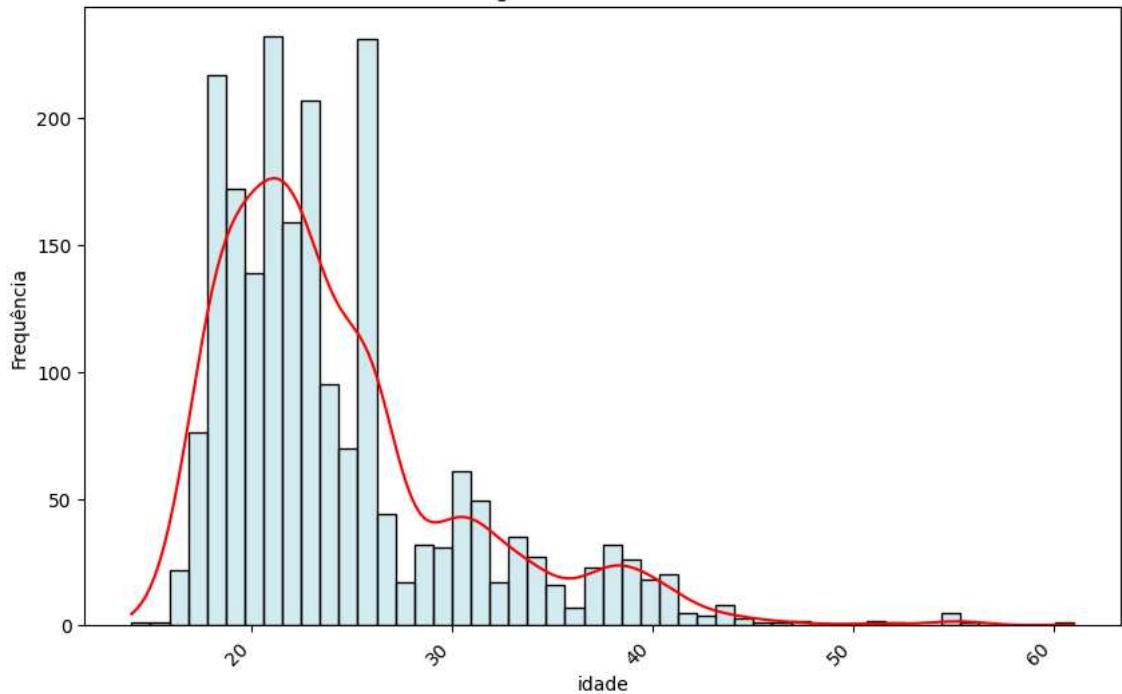
**Histograma**

O histograma revela uma distribuição assimétrica para a direita, confirmando a análise da estatística básica.

Temos uma maior concentração de pacientes jovens.

```
gera_histograma(df = df, coluna = "idade", titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['idade']}")
```

Histograma da variável idade

**Boxplot**

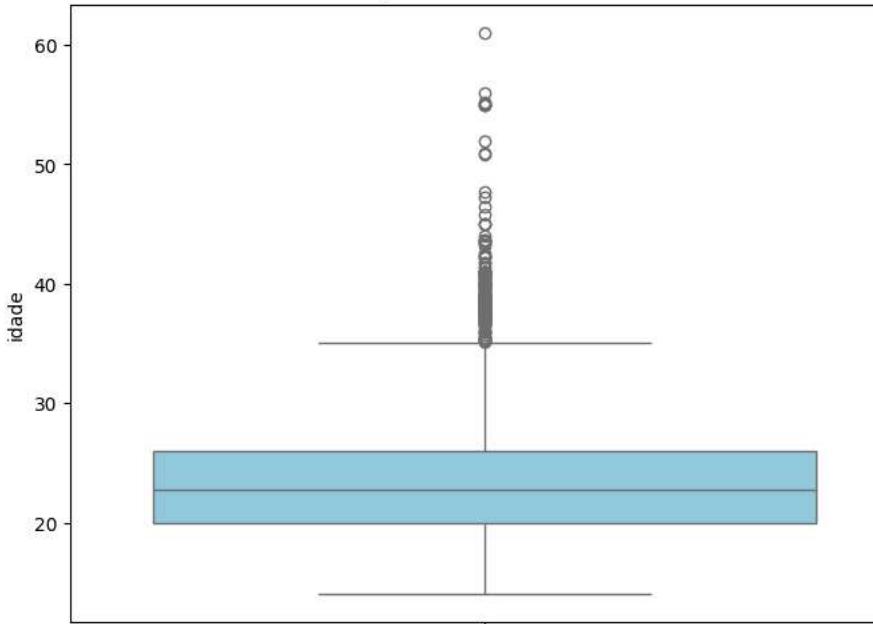
O boxplot confirma a assimetria e mostra que temos outliers a direita com idade superior a 35 anos.

No entanto esses outliers estão dentro de um limite esperado para uma amostra de idade populacional, não sendo necessário o tratamento desses dados.

```
plotar_boxplot_numerica(
    df,
    'idade',
    titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['idade']}['novo_nome']",
    cor = 'skyblue'
```

)

Boxplot da variável idade

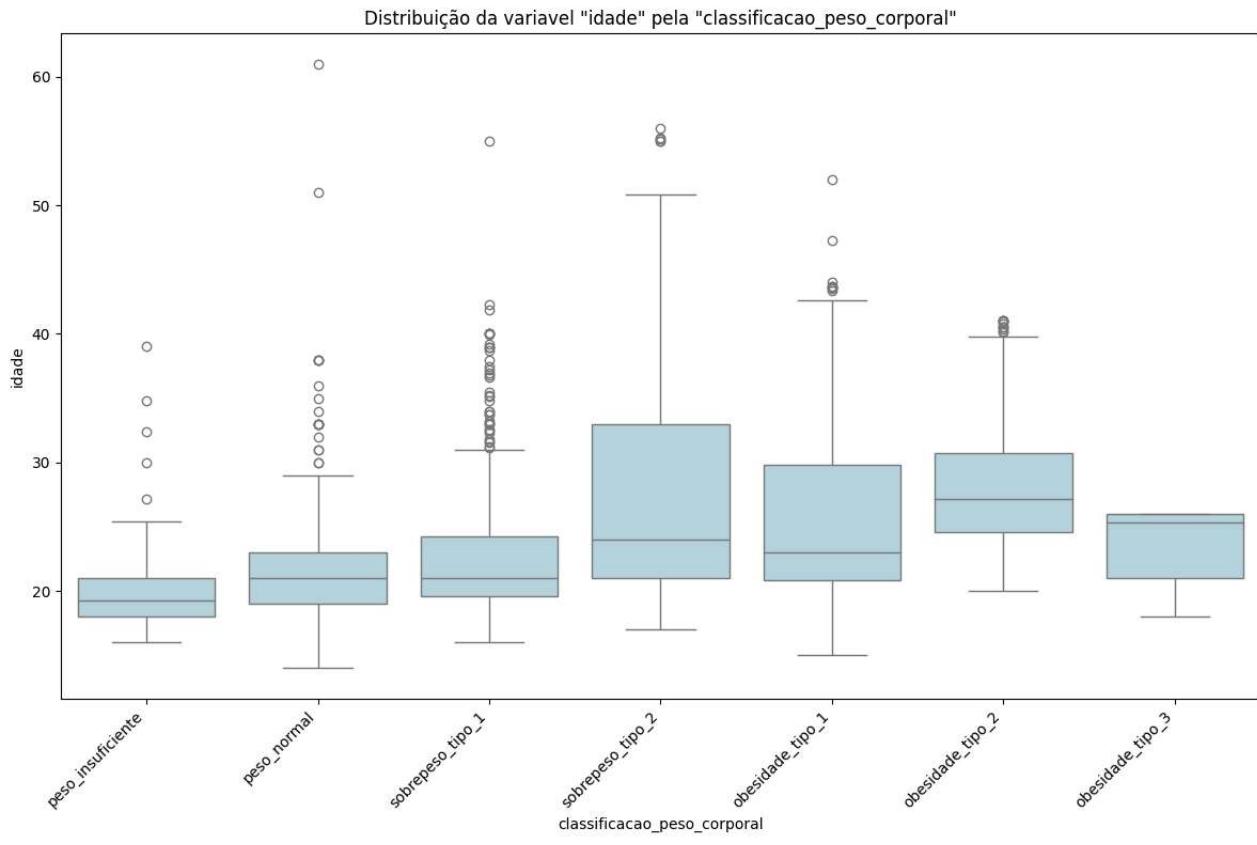


## Análise Bivariada

idade vs classificacao\_peso\_corporal

### Boxplot agrupado

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'idade',
    variavel_alvo = 'classificacao_peso_corporal',
    ordem=ordem_classificacao_peso_corporal,
    cor='lightblue')
```



## Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'idade')

--- Resultados do Teste ANOVA (idade vs classificacao_peso_corporal) ---
Estatística F: 77.9542
P-valor: 3.593e-88
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'idade' entre os grupos de 'classificacao_peso_corporal' (np.float64(77.95415423043558), np.float64(3.592579516699011e-88))
```

## ▼ Análise variável altura

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['altura']['novo_nome']}: {dicionario_dados_nivel_obesidade['altura']}")
```

Descrição da variável altura: Altura em metros

### Conclusão

Na análise univariada percebemos que a variável altura é assimétrica, mas possui uma curva bimodal

Na análise Bivariada temos diferença nas medianas de altura entre os grupos da variável alvo e o teste estatístico confirma a diferença estatisticamente relevante da distribuição da variável idade.

### Sugestões

- Incluir a variável altura no treinamento do modelo
- Com o intuito de padronizar as escalas entre as variáveis, indicamos executar Feature Scaling (como StandardScaler ou MinMaxScaler) na variável.

### Visualização prévia dos dados

Esse dataset estava pronto quando recebemos para esse estudo. Analisando os dados da variável idade percebemos registros com 6 casas decimais, indicando que algum método estatístico foi utilizado na amostra. Seja para tratamento de dados ausentes ou para uma geração de dados sintéticos.

Para o objetivo do estudo podemos continuar.

```
df.altura
altura
0    1.620000
1    1.520000
2    1.800000
3    1.800000
4    1.780000
...
2106   1.710730
2107   1.748584
2108   1.752206
2109   1.739450
2110   1.738836
2111 rows × 1 columns
dtype: float64
```

## Análise Univariada

### Estatística descritiva

#### Análise:

- Os valores mínimos e máximos da amostra estão dentro do esperado para uma amostra de altura.
- A média e mediana são quase idênticas indicando simetria dos dados

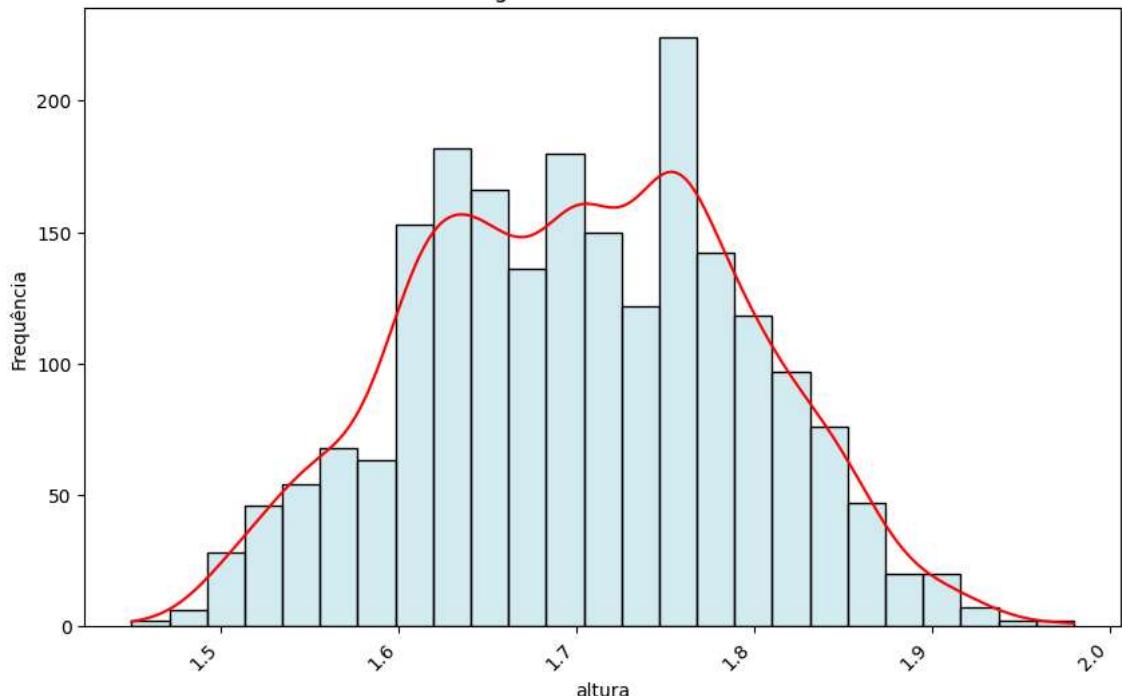
```
df['altura'].describe()
altura
count    2111.000000
mean     1.701677
std      0.093305
min     1.450000
25%     1.630000
50%     1.700499
75%     1.768464
max     1.980000
dtype: float64
```

### Histograma

O histograma revela uma distribuição simétrica, confirmando a análise da estatística básica. No entanto, não possui uma curva de sino normal, mas sim bimodal.

```
gera_histograma(df = df, coluna = "altura", titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['altura']}
```

Histograma da variável altura

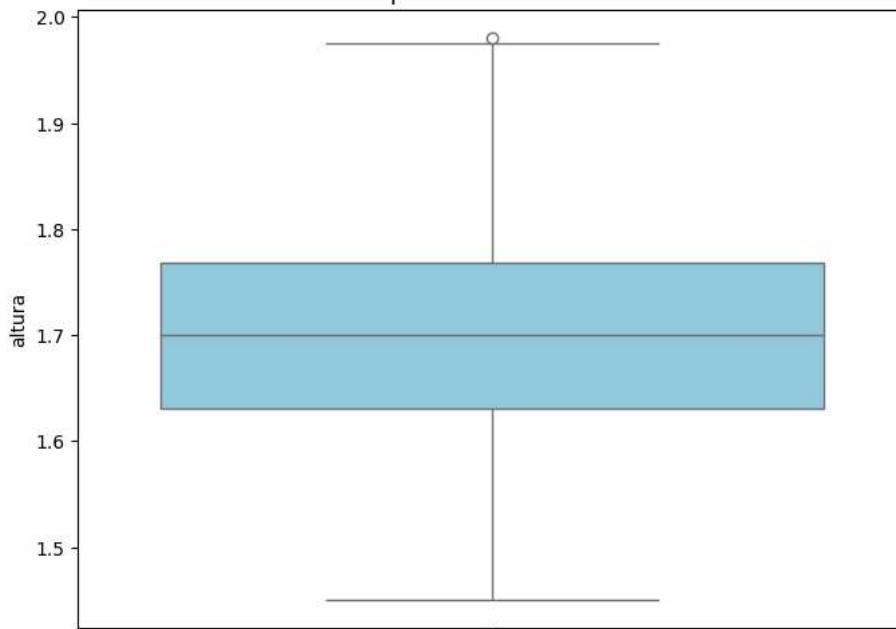


### Boxplot

O boxplot confirma a simetria

```
plotar_boxplot_numerica(
    df,
    'altura',
    titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['altura']['novo_nome']}",
    cor = 'skyblue'
)
```

Boxplot da variável altura

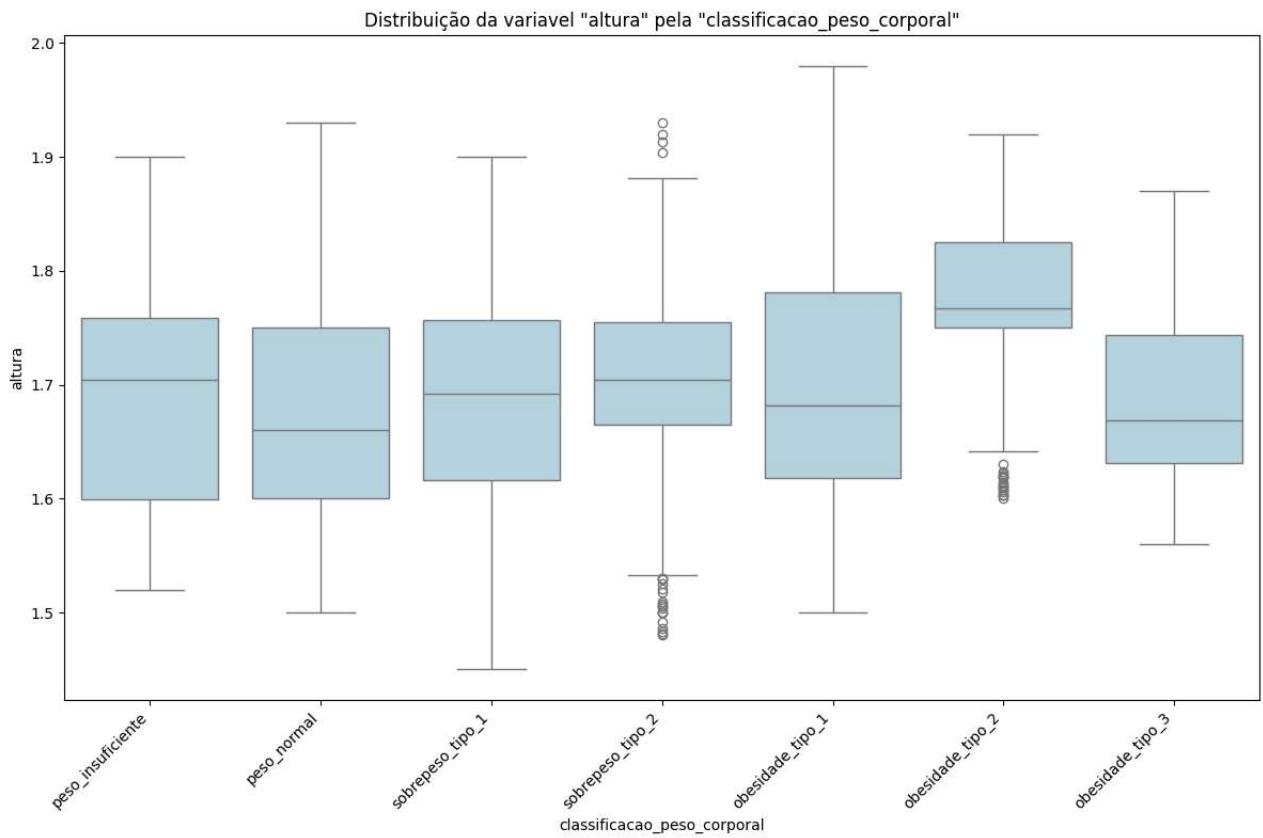


### Análise Bivariada

`altura` vs `classificacao_peso_corporal`

#### Boxplot agrupado

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'altura',
    variavel_alvo = 'classificacao_peso_corporal',
    ordem=ordem_classificacao_peso_corporal,
    cor='lightblue')
```



## Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'altura')

--- Resultados do Teste ANOVA (altura vs classificacao_peso_corporal) ---
Estatística F: 38.4323
P-valor: 1.686e-44
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'altura' entre os grupos de 'classificacao_peso'
(np.float64(38.43231255660025), np.float64(1.6858535844061656e-44))
```

## ▼ Análise variável peso

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['peso']['novo_nome']}: {dicionario_dados_nivel_obesidade['peso']}
```

Descrição da variável peso: Peso em kgs.

### Conclusão

A variável peso é limpa, multimodal e levemente assimétrica à direita.

Na análise Bivariada indica uma forte signifiância estatística. Os valores de Estatística F = 1966.5180, P-valor = 0 indicam um possível vazamento de dados da variável alvo.

Como se trata de um modelo para prever obesidade, comparar a variável alvo com a classificação de IMC que utiliza peso e altura no cálculo foi o caminho escolhido.

Realizamos teste de acerto = 95,2% que confirmou a hipótese de vazamento e confirmamos com a criação de uma crosstab, mapa de calor e a execução de um teste de independência que confirmaram o vazamento.

### Sugestão

- Excluir a variável `peso` do treinamento do modelo. A variável `altura` também deve ser excluída porque está associada ao cálculo do IMC.

### Visualização prévia dos dados

Esse dataset estava pronto quando recebemos para esse estudo. Analisando os dados da variável `idade` percebemos registros com 6 casas decimais, indicando que algum método estatístico foi utilizado na amostra. Seja para tratamento de dados ausentes ou para uma geração de dados sintéticos.

Para o objetivo do estudo podemos continuar.

```
df['peso']

      peso
0    64.000000
1    56.000000
2    77.000000
3    87.000000
4    89.800000
...
2106  131.408528
2107  133.742943
2108  133.689352
2109  133.346641
2110  133.472641
2111 rows × 1 columns

dtype: float64
```

### Análise Univariada

#### Estatística descritiva

##### Análise:

- Os valores mínimos e máximos da amostra estão dentro do esperado para uma amostra de altura.
- A média e mediana são quase idênticas indicando simetria dos dados

```
df['peso'].describe()

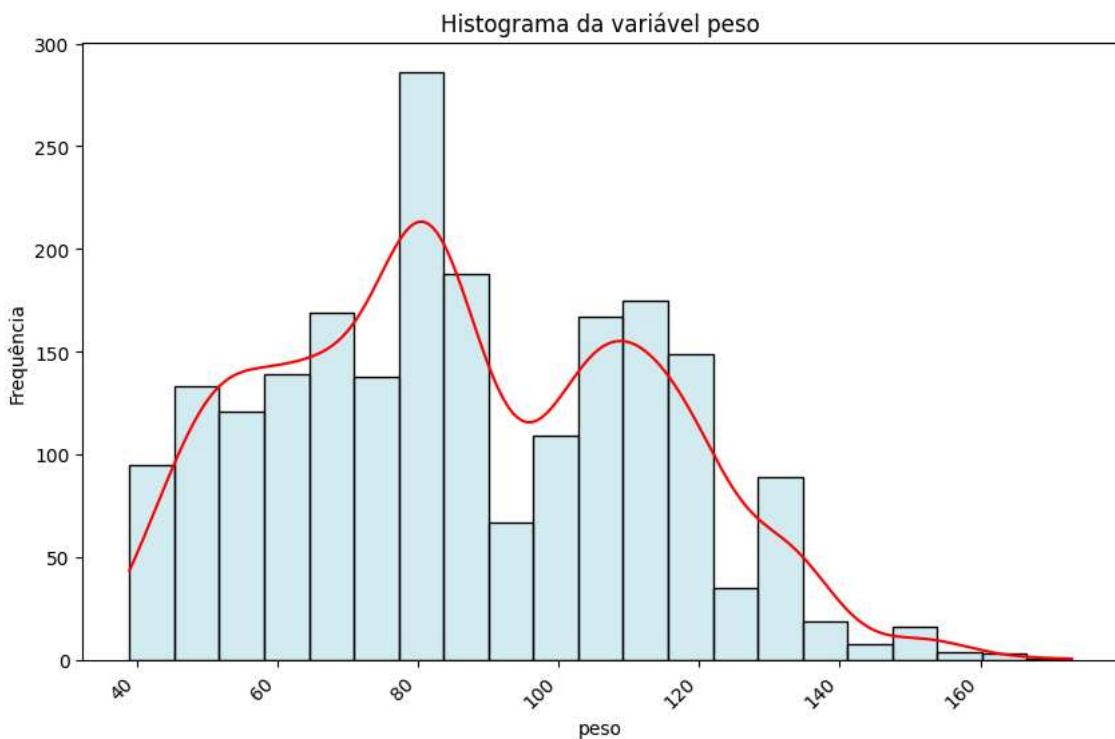
      peso
count  2111.000000
mean   86.586058
std    26.191172
min    39.000000
25%   65.473343
50%   83.000000
75%   107.430682
max   173.000000

dtype: float64
```

### Histograma

O histograma revela uma distribuição simétrica, confirmando a análise da estatística básica. No entanto, não possui uma curva de sino normal, mas sim bimodal.

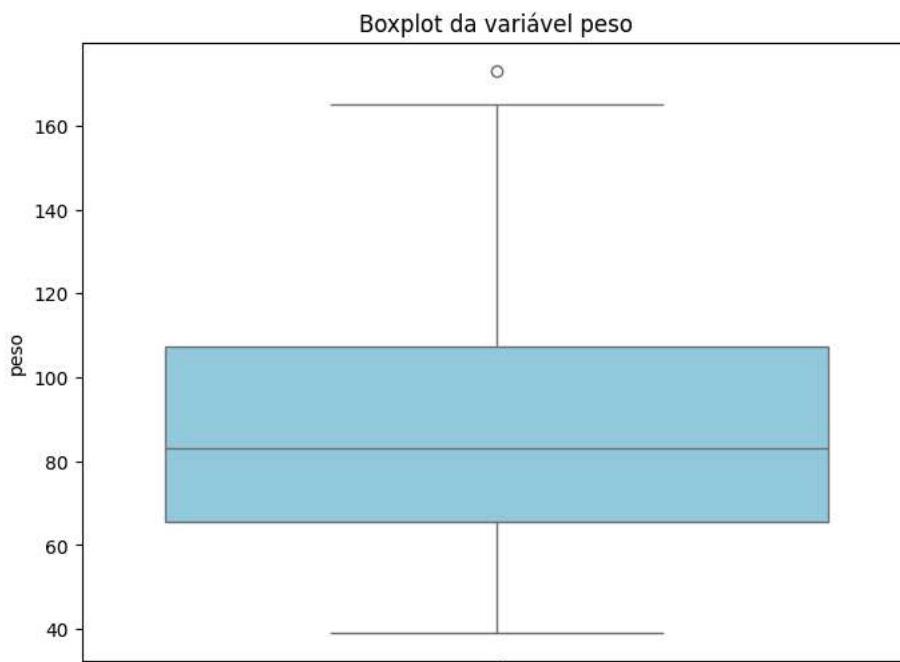
```
gera_histograma(df = df, coluna = "peso", titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['peso'][ 'novo_nome']}")
```



## Boxplot

O boxplot confirma a simetria

```
# Chamar a nova função para plotar o boxplot da variável 'idade'
plotar_boxplot_numerica(
    df,
    'peso',
    titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['peso'][ 'novo_nome']}",
    cor = 'skyblue'
)
```



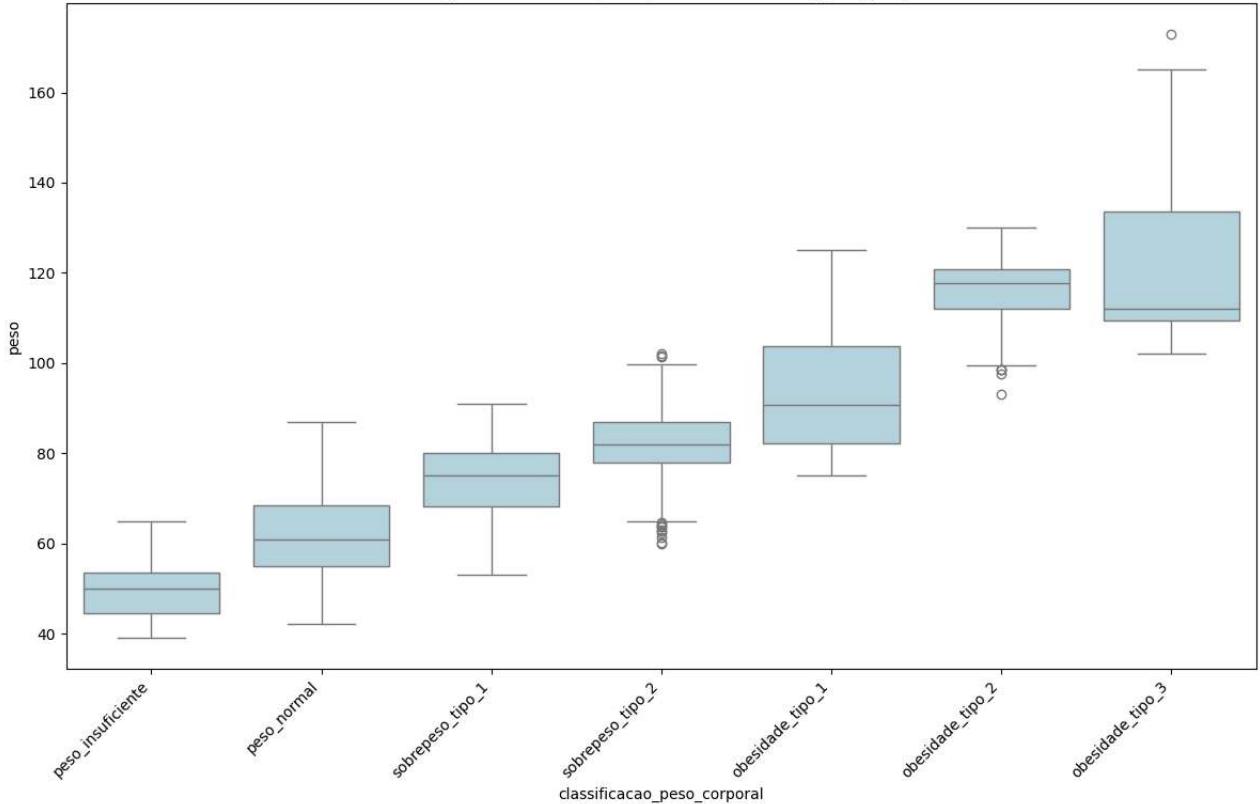
## Análise Bivariada

`peso` vs `classificacao_peso_corporal`

### Boxplot agrupado

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'peso',
    variavel_alvo = 'classificacao_peso_corporal',
    ordem=ordem_classificao_peso_corporal,
    cor='lightblue')
```

Distribuição da variável "peso" pela "classificacao\_peso\_corporal"



### Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'peso')

--- Resultados do Teste ANOVA (peso vs classificacao_peso_corporal) ---
Estatística F: 1966.5180
P-valor: 0
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'peso' entre os grupos de 'classificacao_peso_corporal' (np.float64(1966.518017627475), np.float64(0.0))
```

### Verificando possível vazamento de dados na variável peso

Hipótese: a variável alvo `classificacao_peso_corporal` é determinada pelo IMC , que é calculado da seguinte forma:

$$\text{IMC} = \frac{\text{peso}}{\text{altura}^2}$$

### Variáveis criadas

`imc` e `classificacao_imc`

Nova variável traduzindo os resultados da variável alvo `classificacao_peso` para corresponder a `classificacao_imc`

`sobre peso_tipo_1` e `sobre peso_tipo_2` == `sobre peso`

```
mapa_de_traducao = {
    'peso_insuficiente': 'abaixo_do_peso',
    'peso_normal': 'peso_normal',
    'sobre peso_tipo_1': 'sobre peso',
    'sobre peso_tipo_2': 'sobre peso',
    'obesidade_tipo_1': 'obesidade_grau_1',
    'obesidade_tipo_2': 'obesidade_grau_2',
    'obesidade_tipo_3': 'obesidade_grau_3'
}

df['classificacao_peso_corporal_traduzido'] = df['classificacao_peso_corporal'].map(mapa_de_traducao)

taxa_de_acerto_correta = (df['classificacao_peso_corporal_traduzido'] == df['classificacao_imc']).mean()

print(f'Taxa de correspondência correta: {taxa_de_acerto_correta * 100:.2f}%')

Taxa de correspondência correta: 95.22%
```

Crosstab `classificacao_peso_corporal` vs `classificacao_imc`

```
tabela_comparacao = pd.crosstab(
    pd.Categorical(df['classificacao_peso_corporal'], categories=ordem_classificacao_peso_corporal, ordered=True),
    df['classificacao_imc'],
    rownames=['classificacao_peso_corporal']
)
display(tabela_comparacao)
```

	classificacao_imc	abaixo_do_peso	peso_normal	sobre peso	obesidade_grau_1	obesidade_grau_2	obesidade_grau_3
classificacao_peso_corporal							
<b>peso_insuficiente</b>		268	4	0	0	0	0
<b>peso_normal</b>		2	285	0	0	0	0
<b>sobre peso_tipo_1</b>		0	12	278	0	0	0
<b>sobre peso_tipo_2</b>		0	0	286	4	0	0
<b>obesidade_tipo_1</b>		0	0	2	346	3	0
<b>obesidade_tipo_2</b>		0	0	0	18	279	0
<b>obesidade_tipo_3</b>		0	0	0	0	56	268

Próximas etapas: ([Gerar código com tabela\\_comparacao](#)) ([New interactive sheet](#))

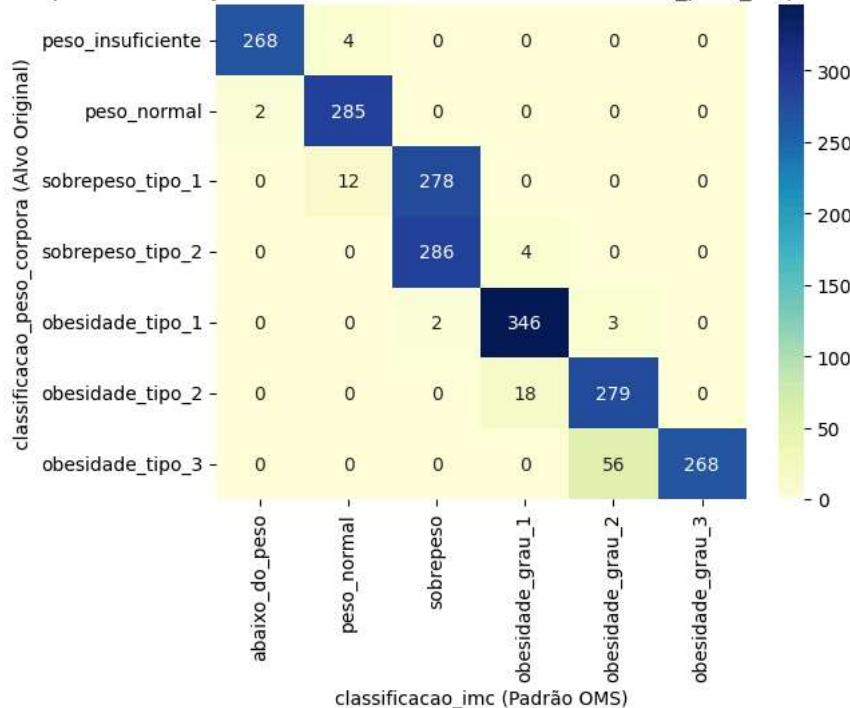
## Mapa de Calor

**tabela:** tabela\_comparacao

```
sns.heatmap(
    tabela_comparacao,
    annot=True,
    fmt='d',
    cmap="YlGnBu"
)

plt.title('Mapa de calor para confirmação de vazamento de dados: classificacao_peso_corporal vs classificacao_imc')
plt.xlabel('classificacao_imc (Padrão OMS)')
plt.ylabel('classificacao_peso_corporal (Alvo Original)')
plt.show()
```

Mapa de calor para confirmação de vazamento de dados: classificacao\_peso\_corporal vs classificacao\_imc



## Testes estatísticos

**Teste Qui-Quadrado de Independência:** Objetivo de determinar se existe uma associação estatisticamente significativa entre duas variáveis categóricas

**Cálculo do V de Cramé:** Métrica para medir a força da associação entre duas variáveis categóricas

```

chi2, p_valor, _, _ = chi2_contingency(tabela_comparacao)

print(f"--- Teste Qui-Quadrado ---")
print(f"Estatística Chi2: {chi2:.2f}")
print(f"P-valor: {p_valor}")
print("CONCLUSÃO: As variáveis são EXTREMAMENTE dependentes.\n")

n = tabela_comparacao.sum().sum()
min_dim = min(tabela_comparacao.shape) - 1

v_cramer = np.sqrt(chi2 / (n * min_dim))

print(f"--- V de Cramer ---")
print(f"Força da Associação (0 a 1): {v_cramer:.4f}")
print("CONCLUSÃO: Um valor ~1.0 confirma uma associação quase perfeita (Leakage).")

--- Teste Qui-Quadrado ---
Estatística Chi2: 9395.13
P-valor: 0.0
CONCLUSÃO: As variáveis são EXTREMAMENTE dependentes.

--- V de Cramer ---
Força da Associação (0 a 1): 0.9435
CONCLUSÃO: Um valor ~1.0 confirma uma associação quase perfeita (Leakage).

```

## Análise

O Teste Qui-Quadrado de Independência mostra que as variáveis são dependentes com um p-valor 0. O teste v-cramer confirma a associação quase perfeita entre as variáveis, sugerindo um vazamento de dados entre as variáveis classificacao\_peso\_corporal e classificacao\_imc.

Isso sugere que a variável alvo classificacao\_peso\_corporal da amostra será fortemente influenciada pelas variáveis peso e altura que são a base do IMC.

## ▼ Análise variável fcvc

```

print(f"Descrição da variavel {dicionario_dados_nivel_obesidade['fcvc']['novo_nome']}: {dicionario_dados_nivel_obesidade['fcvc']}
Descrição da variavel fcvc: Voce costuma comer vegetais nas suas refeicoes?

```

## Conclusão

A variável `fcvc` devido a a sua pergunta (Voce costuma comer vegetais nas suas refeicoes?) indica ser uma variável ordinal. Baseado nos valores mínimos e máximos as categorias provavelmente correspondiam a 1, 2 e 3. Das 2111 linhas, 1285 correspondem a essas categorias e as outras corespondem a valores decimais das mesmas. Provavelmente alguma técnica para preenchimento de valores nulos foi empregada.

O histograma revela uma distribuição assimétrica a esquerda. Ele não bate com a estatística basica muito provavelmente por temos dois picos relacionados aos valores 2 e 3. Percebemos tambem uma barra em destaque no valor 1 reforçando a hipótese sugerida na análise inicial dos daos.

A analise bivariada confirma assimetria a esquerda e revela uma diferença estatisticamente significante em relação a variável alvo

## Sugestões

- Realizar um processo de limpeza do ruído relacionado a variável `fcvc` por meio de arredondamento dos dados.
- Após o arredondamento tratá-la como uma variável ordinal e usar a técnica de `ordinal encoding`. Mapeamento = {1:1, 2:2, 3:3}

## Visualização prévia dos dados

A variável `fcvc` devido a a sua pergunta (Voce costuma comer vegetais nas suas refeicoes?) indica ser uma variável ordinal. Baseado nos valores mínimos e máximos as categorias provavelmente correspondiam a 1, 2 e 3. Das 2111 linhas, 1285 correspondem a essas categorias e as outras corespondem a valores decimais das mesmas. Provavelmente alguma técnica para preenchimento de valores nulos foi empregada.

Por isso vamos seguir com a análise dessa variável a considerando como numérica.

```
df.fcvc.unique()
```

```
[2.232836, 2.571274, 2.49619, 2.151335, 2.01055, 2.927187,
1.412566, 1.289315, 1.624366, 1.528331, 2.037042, 2.191108,
1.431346, 1.84199, 2.499388, 2.230742, 2.240757, 2.996186,
2.649406, 2.525884, 2.736647, 2.684335, 2.425503, 1.469384,
2.906269, 2.808027, 2.636719, 2.996717, 2.880483, 2.885693,
2.913452, 2.919526, 2.724121, 2.801992, 2.748971, 2.680375])
```

```
fcvc_amostra = len(df.fcvc)
fcvc_min = df.fcvc.min()
fcvc_max = df.fcvc.max()
fcvc_total_valores = len(df.fcvc.unique())
print(f'Valor mínimo na variável fcvc: {fcvc_min} ')
print(f'Valor máximo na variável fcvc: {fcvc_max} ')
print(f'Total de valores na variável fcvc: {fcvc_amostra} ')
print(f'Total de valores únicos na variável fcvc: {fcvc_total_valores} ')

Valor mínimo na variável fcvc: 1.0
Valor máximo na variável fcvc: 3.0
Total de valores na variável fcvc: 2111
Total de valores únicos na variável fcvc: 810
```

```
resposta_1 = len(df.query("fcvc == 1"))
resposta_2 = len(df.query("fcvc == 2"))
resposta_3 = len(df.query("fcvc == 3"))
total_inteiros = resposta_1 + resposta_2 + resposta_3
total_decimais = fcvc_amostra - total_inteiros

print(f'Quantidade de linhas na variável fcvc igual a 1: {resposta_1}')
print(f'Quantidade de linhas na variável fcvc igual a 2: {resposta_2}')
print(f'Quantidade de linhas na variável fcvc igual a 3: {resposta_3}')
print(f'Total de linhas com numeros inteiros: {total_inteiros}')
print(f'Total de linhas com numeros decimais: {total_decimais}')

Quantidade de linhas na variável fcvc igual a 1: 33
Quantidade de linhas na variável fcvc igual a 2: 600
Quantidade de linhas na variável fcvc igual a 3: 652
Total de linhas com numeros inteiros: 1285
Total de linhas com numeros decimais: 826
```

## Análise Univariada

### Estatística descritiva

#### Análise:

- Os valores mínimos e máximos da amostra estão dentro do esperado para uma amostra de altura.
- A média e mediana são quase idênticas indicando simetria dos dados

```
df['fcvc'].describe()
```

	fcvc
<b>count</b>	2111.000000
<b>mean</b>	2.419043
<b>std</b>	0.533927
<b>min</b>	1.000000
<b>25%</b>	2.000000
<b>50%</b>	2.385502
<b>75%</b>	3.000000
<b>max</b>	3.000000

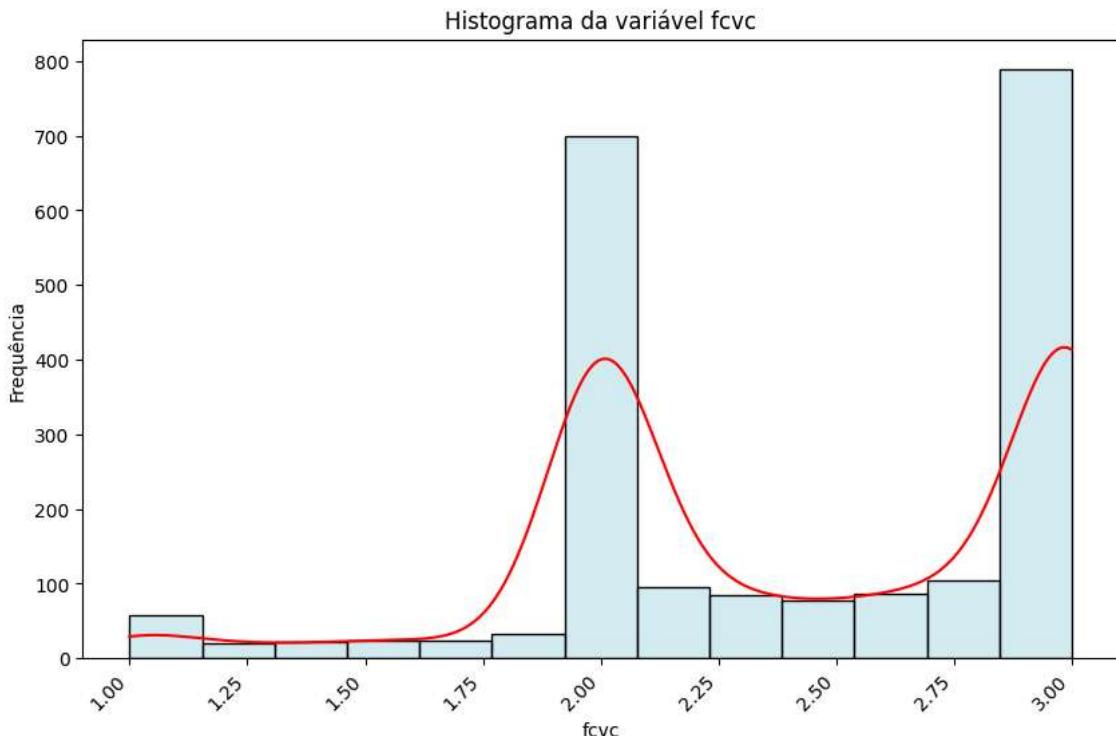
```
dtype: float64
```

### Histograma

O histograma revela uma distribuição assimétrica a esquerda. Ele não bate com a estatística basica muito provavelmente por temos dois picos relacionados aos valores 2 e 3. Percebemos tambem uma barra em destaque no valor 1 reforçando a hipótese sugerida na análise inicial dos daos.

```
gera_histograma(
    df = df,
```

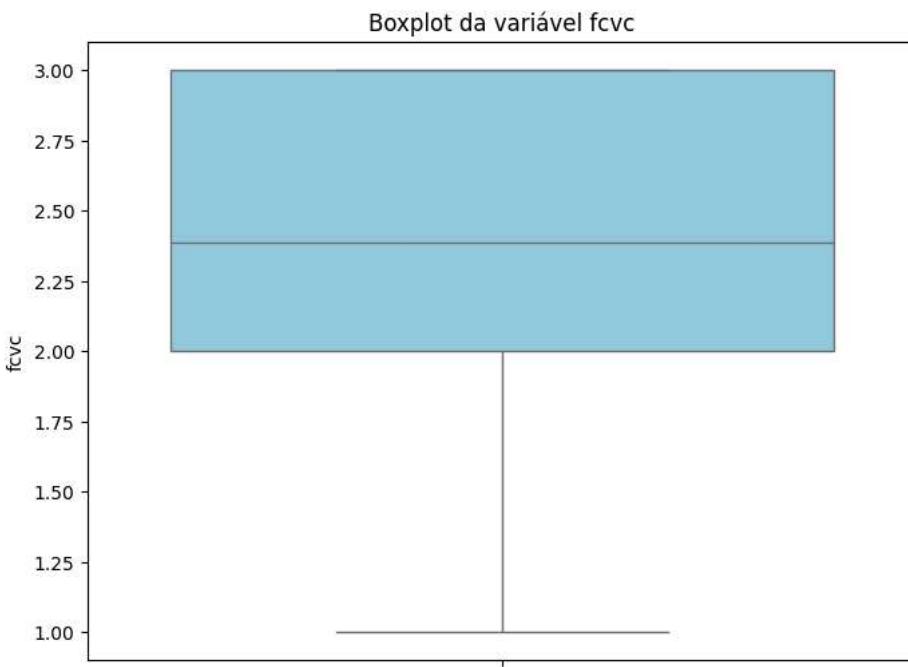
```
coluna = "fcvc",
titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['fcvc']['novo_nome']}",
cor="lightblue",
cor_kde="red"
)
```



## Boxplot

O boxplot confirma a simetria

```
plotar_boxplot_numerica(
    df,
    'fcvc',
    titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['fcvc']['novo_nome']}",
    cor = 'skyblue'
)
```

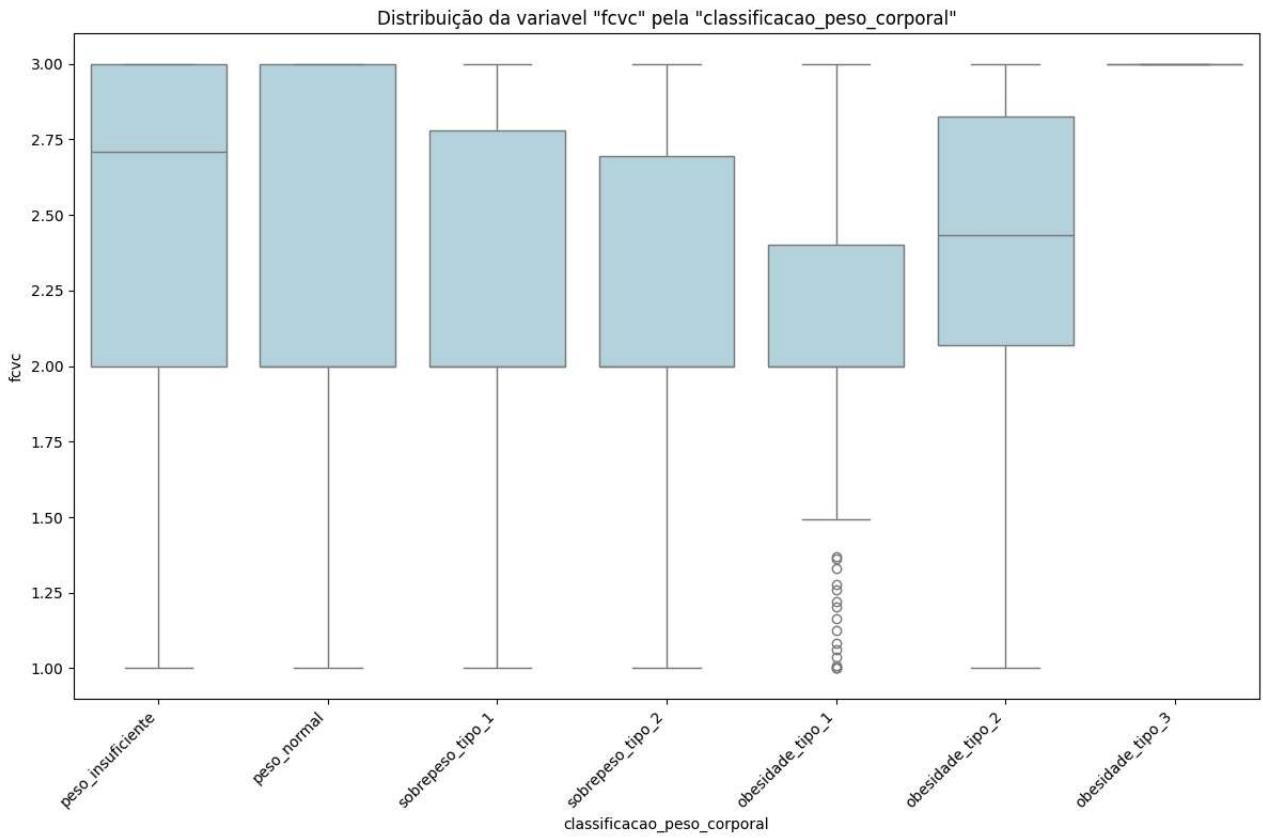


## Análise Bivariada

`fcvc` vs `classificacao_peso_corporal`

### Boxplot agrupado

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'fcvc',
    variavel_alvo = 'classificacao_peso_corporal',
    ordem=ordem_classificacao_peso_corporal,
    cor='lightblue')
```



### Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'fcvc')

--- Resultados do Teste ANOVA (fcvc vs classificacao_peso_corporal) ---
Estatística F: 112.3155
P-valor: 3.732e-123
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'fcvc' entre os grupos de 'classificacao_peso_corporal' (np.float64(112.31546186980427), np.float64(3.73246930865404e-123))
```

### ▼ Análise variável ncp

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['ncp']}['novo_nome']: {dicionario_dados_nivel_obesidade['ncp']}")
```

Descrição da variável ncp: Quantas refeicoes principais voce faz diariamente?

### Conclusão ncp

A variável `ncp` demonstra o mesmo padrão da variável `fcvc` no que tange a ser uma variável originalmente categórica ordinal, mas que sofreu algum pré-processamento para inclusão de valores. Estranho é não ter nenhum valor inteiro == 2, visto que o valor mínimo é igual a 1 e máximo igual a 4, o que sugeriria quatro categorias.

Outro ponto que pensando sobre o prisma de uma variável categórica, temos a categoria == 4 pode ser considerada rara com 69 registros. No entanto, possui uma quantidade suficiente para seguirmos com o treinamento e executar um teste A\B para pesar risco versus benefício.

A análise univariada mostra uma assimetria a esquerda, mas com uma grande concentração de valores no valor 3. Temos também notavelmente dois pequenos picos nos valor 1 e 4 confirmado a análise inicial.

A análise bivariada informa que temos associação estatística relevante entre as variáveis. Temos 2 categorias sem nenhuma associação o que a torna boa para identificar grupos específicos

### Sugestões

- Realizar um processo de limpeza do ruído relacionado a variável `ncp` por meio de arredondamento dos dados.
- Após o arredondamento tratá-la como uma variável ordinal e usar a técnica de `ordinal encoding`. Mapeamento = {1:1, 3:3, 4:4}
- Monitorar risco devido a raridade da categoria 4. Usar as técnicas de importância de feature e Teste A/B pós treinamento.

### Visualização prévia dos dados

```
df.ncp.unique()
```

```
1.835543, 2.853676, 2.337035, 1.631184, 1.005391, 3.250467,
2.948721, 1.80993, 1.94671, 2.9796, 2.994198, 2.579291,
2.449067, 1.532833, 2.961706, 1.152521, 1.729553, 1.346987,
1.977221, 2.443812, 2.491315, 2.843319, 2.157164, 2.601675,
1.171027, 3.362758, 1.672706, 2.714115, 2.521546, 1.338033,
1.081805, 3.087119, 2.909117, 1.466393, 1.471053, 2.983201,
2.783336, 1.863012, 3.053598, 3.165837, 2.741413, 1.058123,
2.997414, 1.672958, 1.680838, 3.070386, 2.608416, 1.599464,
2.815255, 2.395785, 2.844138, 2.716106, 1.402771, 2.865657,
1.836226, 2.884848, 2.375026, 2.9774, 2.693646, 2.832018,
2.270163, 2.646717, 1.193729, 1.477581, 3.209508, 2.756622,
2.658478, 1.971472, 1.818026, 1.820779, 1.724887, 3.129155,
3.856434, 2.753418, 2.116195, 2.049908, 1.044628, 2.98212,
1.009426, 2.667711, 3.000974, 2.698883, 2.598079, 2.837388,
2.946063, 1.873484, 2.473911, 1.79558, 2.623079, 2.174968,
2.676148, 3.394788, 2.137068, 2.937607, 1.313403, 2.113575,
2.092179, 2.974204, 1.001633, 1.2919, 1.139317, 2.029858,
2.977543, 1.476204, 2.57038, 2.879541, 1.458507, 1.105617,
1.854536, 1.08687, 1.25535, 2.870661, 1.482103, 1.81698,
2.582591, 2.164839, 2.141839, 2.877583, 2.95833, 2.119826,
2.992606, 2.050121, 1.923607, 2.27374, 1.418985, 1.320768,
2.964024, 2.933409, 2.962004, 1.271624, 1.00061, 1.068443,
2.952821, 1.851088, 2.119682, 2.970675, 2.966803, 1.508685,
2.209314, 1.114564, 2.036794, 2.988539, 1.630506, 2.977999,
2.733077, 2.427137, 1.867836, 1.548407, 2.937989, 1.237454,
1.169173, 2.391753, 2.475444, 2.478794, 2.070033, 2.282392,
1.049534, 1.66338, 1.672751, 1.734762, 1.92822, 2.475228,
2.093831, 1.231915, 1.374791, 1.326982, 2.900915, 1.317884,
2.977909, 1.355752, 1.015488, 2.986172, 2.993623, 1.001383,
1.001542, 2.590283, 1.773916, 2.989112, 1.496776, 2.994046,
1.000283, 1.000414, 1.135278, 2.463113, 1.782109, 2.376374,
2.976098, 2.968098, 1.792695, 1.116401, 2.984523, 2.978103,
1.578751, 1.874532, 2.735706, 1.014916, 2.622055, 2.806566,
1.355354, 1.478334, 1.130751, 1.099151, 2.358455, 1.706551,
1.24884, 1.250548, 1.202179, 1.194815, 2.880817, 2.918124,
1.073421, 1.416309, 2.18162, 2.152733, 1.046144, 1.974233,
2.880794, 2.765213, 2.967089, 2.37985, 2.655265, 1.941307,
1.709546, 2.883984, 2.935381, 2.961192, 2.973476, 2.392811,
1.293342, 2.930044, 2.129909, 2.831771, 2.595126, 2.52751,
2.120936, 1.468948, 3.990925, 2.834253, 3.734323, 2.049565,
2.272214, 2.218285, 1.418833, 1.109956, 3.914454, 3.169089,
2.419656, 2.175432, 2.806298, 2.987652, 2.842848, 1.134321,
1.924168, 2.701689, 2.992903, 2.658837, 2.80742, 1.834472,
2.123138, 1.265463, 1.340361, 3.989492, 2.650088, 2.89292,
2.938902, 2.986637, 2.701521, 2.014671, 1.971659, 1.311797,
1.262831, 3.770379, 2.87747, 2.175153, 2.13229, 2.993634,
2.999346, 2.976211, 2.842035, 1.09749, 1.91863, 2.510135,
2.65772, 2.604998, 2.567567, 2.434347, 1.845858, 1.627555,
1.569176, 2.371658, 2.378211, 2.301129, 2.454432, 2.915921,
2.992083, 2.849347, 2.954446, 3.755976, 2.902639, 2.372705,
2.100918, 2.272801, 2.692889, 2.280545, 2.675411, 1.487674,
1.02075, 1.240424, 1.154318, 3.219347, 3.656401, 3.220181,
2.850948, 2.187145, 2.415522, 2.138375, 1.077331, 2.888193,
2.574108, 2.740492, 2.956422, 2.749334, 2.993856, 2.993084,
2.989791, 2.920373, 2.911568, 1.134042, 1.120102, 2.040582,
2.015675, 2.711238, 2.695396, 2.996834, 2.996444, 2.894142,
2.683061, 2.806341, 2.791366, 1.703299, 1.685134, 2.269799,
2.142328, 1.437959, 1.343117, 1.213431, 1.089048])
```

```
ncp_amostra = len(df.ncp)
ncp_min = df.ncp.min()
ncp_max = df.ncp.max()
ncp_total_valores = len(df.ncp.unique())
print(f'Valor mínimo na variável ncp: {ncp_min} ')
```

```
print(f'Valor máximo na variável ncp: {ncp_max} ')
print(f'Total de valores na variável ncp: {ncp_amostra} ')
print(f'Total de valores únicos na variável ncp: {ncp_total_valores} ')
```

```
Valor mínimo na variável ncp: 1.0
Valor máximo na variável ncp: 4.0
Total de valores na variável ncp: 2111
Total de valores únicos na variável ncp: 635
```

```
resposta_ncp_1 = len(df.query("ncp == 1"))
resposta_ncp_2 = len(df.query("ncp == 2"))
resposta_ncp_3 = len(df.query("ncp == 3"))
resposta_ncp_4 = len(df.query("ncp == 4"))
total_inteiros_ncp = resposta_ncp_1 + resposta_ncp_2 + resposta_ncp_3 + resposta_ncp_4
total_decimais_ncp = ncp_amostra - total_inteiros_ncp
```

```
print(f'Quantidade de linhas na variável ncp igual a 1: {resposta_ncp_1}')
print(f'Quantidade de linhas na variável ncp igual a 2: {resposta_ncp_2}')
print(f'Quantidade de linhas na variável ncp igual a 3: {resposta_ncp_3}')
print(f'Quantidade de linhas na variável ncp igual a 4: {resposta_ncp_4}')
print(f'Total de linhas com numeros inteiros: {total_inteiros_ncp}')
print(f'Total de linhas com numeros decimais: {total_decimais_ncp}')
```

```
Quantidade de linhas na variável ncp igual a 1: 199
Quantidade de linhas na variável ncp igual a 2: 0
Quantidade de linhas na variável ncp igual a 3: 1203
Quantidade de linhas na variável ncp igual a 4: 69
Total de linhas com numeros inteiros: 1471
Total de linhas com numeros decimais: 640
```

## Análise Univariada

### Estatística descritiva

#### Análise:

- Os valores mínimos e máximos da amostra estão dentro do esperado para uma amostra de altura.
- A média e mediana são quase idênticas indicando simetria dos dados

```
df['ncp'].describe()
```

	ncp
<b>count</b>	2111.000000
<b>mean</b>	2.685628
<b>std</b>	0.778039
<b>min</b>	1.000000
<b>25%</b>	2.658738
<b>50%</b>	3.000000
<b>75%</b>	3.000000
<b>max</b>	4.000000

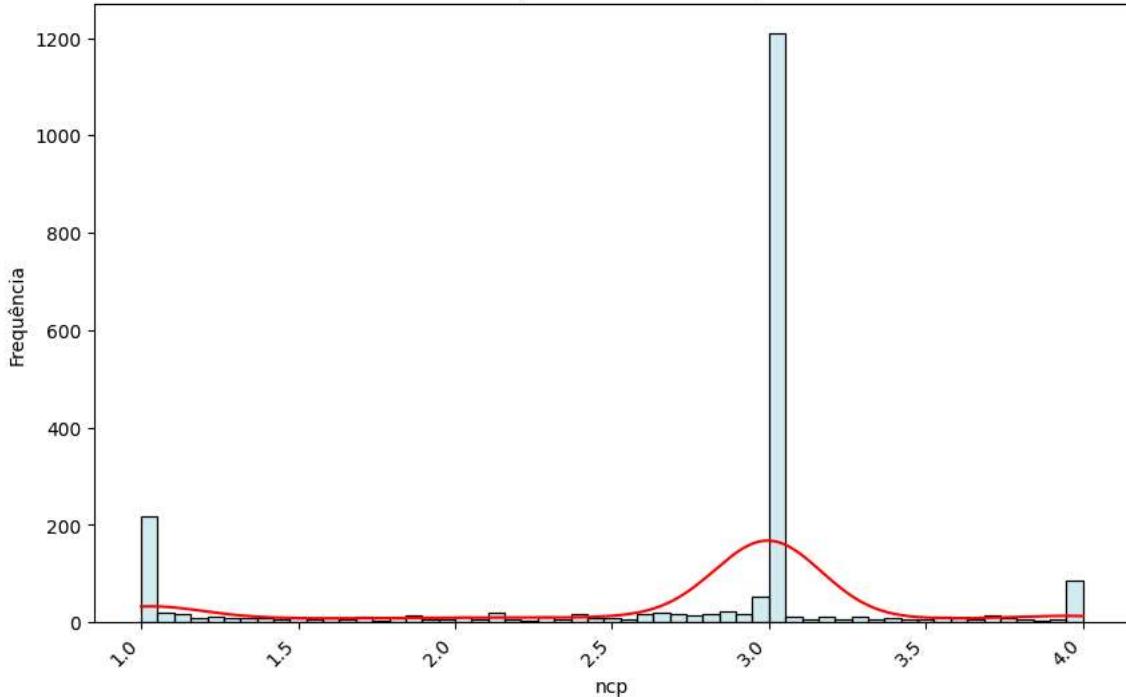
```
dtype: float64
```

### Histograma

O histograma revela uma distribuição assimétrica a esquerda. Ele não bate com a estatística basica muito provavelmente por temos dois picos relacionados aos valores 2 e 3. Percebemos tambem uma barra em destaque no valor 1 reforçando a hipótese sugerida na análise inicial dos daos.

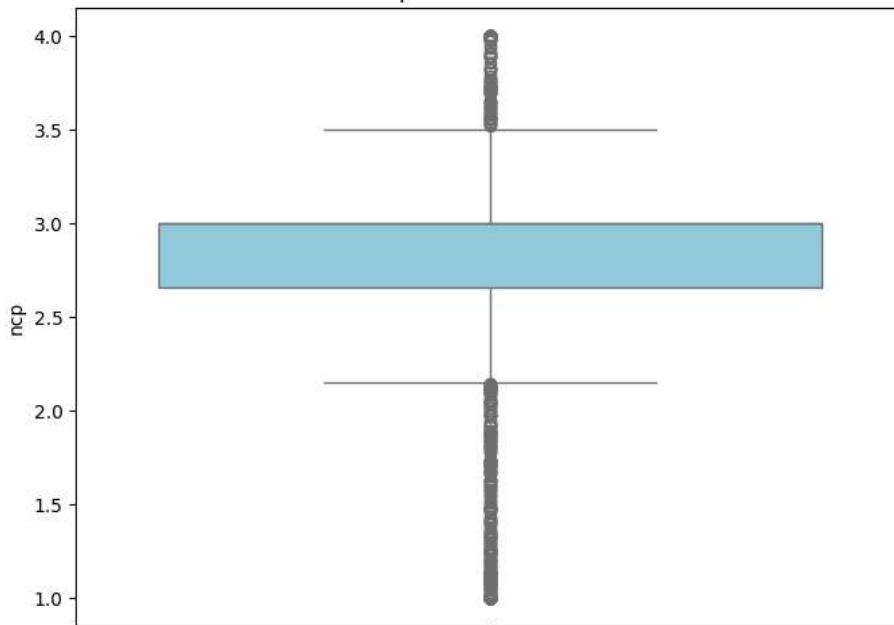
```
gera_histograma(
    df = df,
    coluna = "ncp",
    titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['ncp']['novo_nome']}",
    cor="lightblue",
    cor_kde="red"
)
```

Histograma da variável ncp

**Boxplot**

```
plotar_boxplot_numerica(
    df,
    'ncp',
    titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['fcvc']['novo_nome']}",
    cor = 'skyblue'
)
```

Boxplot da variável fcvc

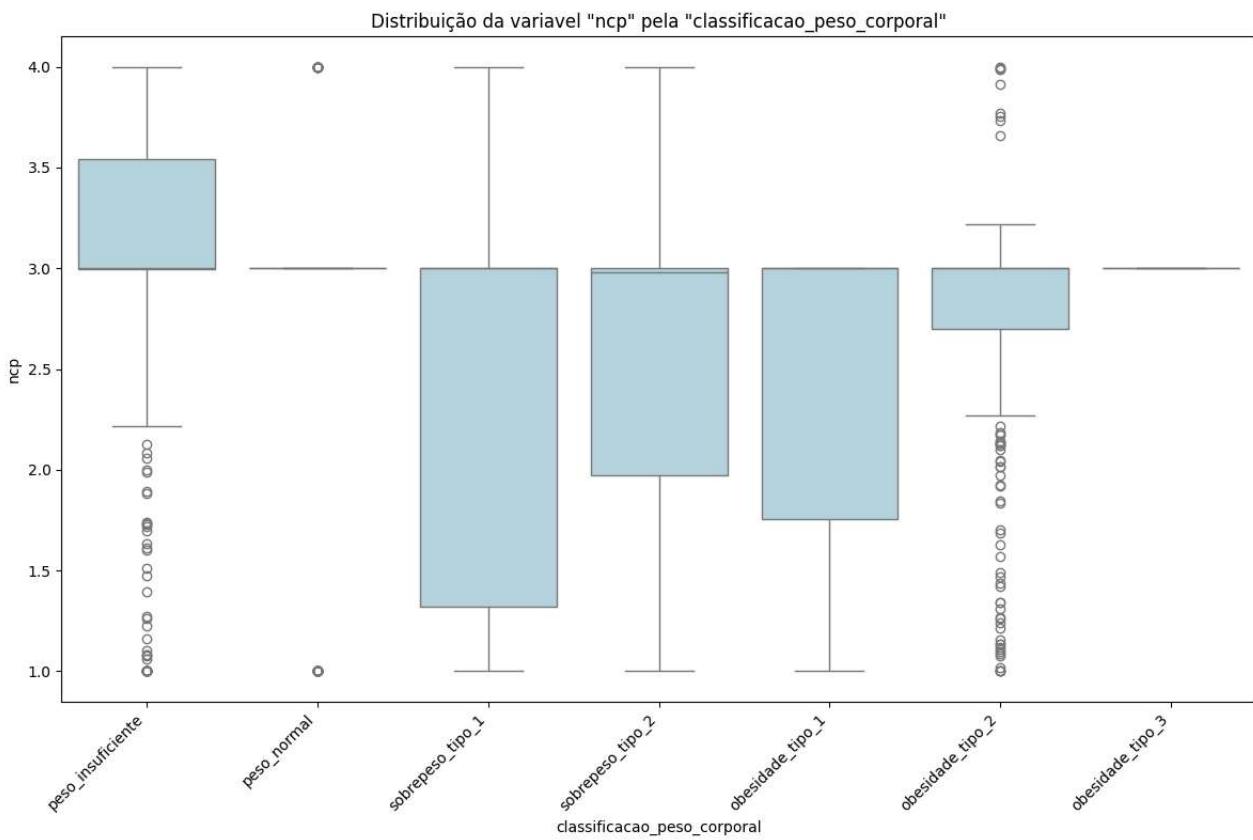
**Análise Bivariada**

`ncp` vs `classificacao_peso_corporal`

**Boxplot agrupado**

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'ncp',
    variavel_alvo = 'classificacao_peso_corporal',
```

```
ordem=ordem_classificacao_peso_corporal,
cor='lightblue')
```



## Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'ncp')

--- Resultados do Teste ANOVA (ncp vs classificacao_peso_corporal) ---
Estatística F: 26.8117
P-valor: 6.259e-31
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'ncp' entre os grupos de 'classificacao_peso_cc
(np.float64(26.81166184274833), np.float64(6.258631620911331e-31))
```

## ▼ Análise variável ch20

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['ch20']['novo_nome']}: {dicionario_dados_nivel_obesidade['c
Descrição da variável ch20: Quanta agua voce bebe diariamente?
```

### Conclusão

A variável `ch20` devido a sua pergunta (Quanta agua voce bebe diariamente?) pode indicar tanto uma variável contínua se prensarmos em quantidade de água em litros, quanto categórica se pensar em pouco, normal e muito. Na análise incial verificamos os valor mínimo como 1 e máximo como 3 e o comportamento identificado em outras variáveis que sugere um preenchimento de dados nulos.

O histograma revela três picos relacionados aos valores inteiros com maior quantidade no valor 2 o que atesta o dado apresentado na estatística descritiva de média e mediana idênticas. Aqui temos uma distribuição simétrica que corrobora a hipótese inicial que se trata de uma variável ordinal com 3 categorias.

A análise bivariada informa que temos associação estatística relevante entre as variáveis.

### Sugestões

- Realizar um processo de limpeza do ruído relacionado a variável `ch20` por meio de arredondamento dos dados.
- Após o arredondamento tratá-la como uma variável ordinal e usar a técnica de `ordinal encoding`. Mapeamento = {1:1, 2:2, 3:3}

### Visualização prévia dos dados

A variável `ch20` devido a sua pergunta (Quanta agua voce bebe diariamente?) pode indicar tanto uma variável contínua se prensarmos em quantidade de água em litros, quanto categórica se pensar em pouco, normal e muito. Na análise incial verificamos os valor minimo como 1 e maximo como 3 e o comportamento identificado em outras variáveis que sugere um preenchimento de dados nulos.

Minha hipótese é que essa variável está mais para uma variável categorica ordinal do que variável contínua.

```
df.ch20.unique()
array([2.0, 3.0, 1.0, ..., 2.054193, 2.852339, 2.863513])
```

```
ch20_amostra = len(df.ch20)
ch20_min = df.ch20.min()
ch20_max = df.ch20.max()
ch20_total_valores = len(df.ch20.unique())
print(f'Valor mínimo na variável ch20: {ch20_min} ')
print(f'Valor máximo na variável ch20: {ch20_max} ')
print(f'Total de valores na variável ch20: {ch20_amostra} ')
print(f'Total de valores únicos na variável ch20: {ch20_total_valores} ')
```

```
Valor mínimo na variável ch20: 1.0
Valor máximo na variável ch20: 3.0
Total de valores na variável ch20: 2111
Total de valores únicos na variável ch20: 1268
```

```
resposta_ch20_1 = len(df.query("ch20 == 1"))
resposta_ch20_2 = len(df.query("ch20 == 2"))
resposta_ch20_3 = len(df.query("ch20 == 3"))
total_inteiros_ch20 = resposta_ch20_1 + resposta_ch20_2 + resposta_ch20_3
total_decimais_ch20 = ch20_amostra - total_inteiros_ch20
```

```
print(f'Quantidade de linhas na variável ch20 igual a 1: {resposta_ch20_1}')
print(f'Quantidade de linhas na variável ch20 igual a 2: {resposta_ch20_2}')
print(f'Quantidade de linhas na variável ch20 igual a 3: {resposta_ch20_3}')
print(f'Total de linhas com numeros inteiros: {total_inteiros_ch20}')
print(f'Total de linhas com numeros decimais: {total_decimais_ch20}')
```

```
Quantidade de linhas na variável ch20 igual a 1: 211
Quantidade de linhas na variável ch20 igual a 2: 448
Quantidade de linhas na variável ch20 igual a 3: 162
Total de linhas com numeros inteiros: 821
Total de linhas com numeros decimais: 1290
```

### Análise Univariada

#### Estatística descritiva

##### Análise:

- Os valores mínimos e máximos da amostra estão dentro do esperado para uma amostra de altura.
- A média e mediana são quase idênticas indicando simetria dos dados

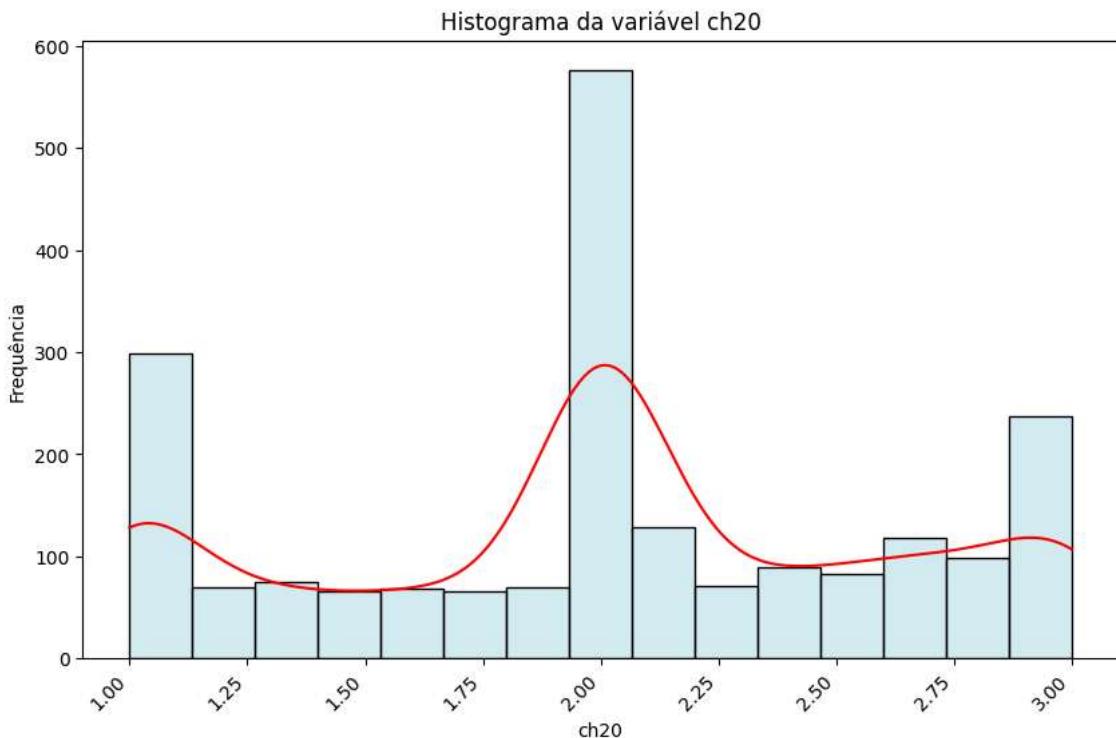
```
df['ch20'].describe()
```

```
ch20
count    2111.000000
mean     2.008011
std      0.612953
min      1.000000
25%     1.584812
50%     2.000000
75%     2.477420
max      3.000000
dtype: float64
```

## Histograma

O histograma revela três picos relacionados aos valores inteiros com maior quantidade no valor 2 o que atesta o dado apresentado na estatística descritiva de média e mediana idênticas. Aqui temos uma distribuição simétrica que corrobora a hipótese inicial que se trata de uma variável ordinal com 3 categorias.

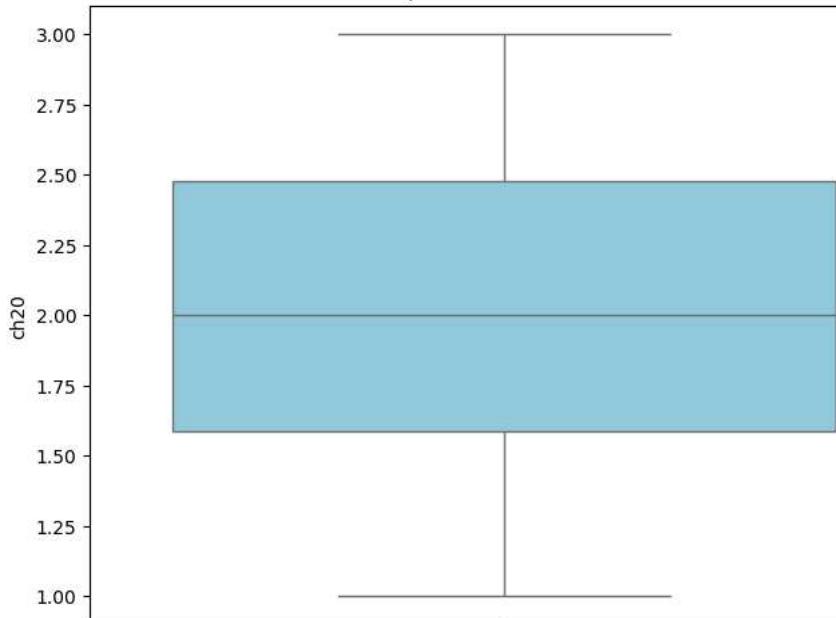
```
gera_histograma(
  df = df,
  coluna = "ch20",
  titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['ch20']['novo_nome']}",
  cor="lightblue",
  cor_kde="red"
)
```



## Boxplot

```
plotar_boxplot_numerica(
  df,
  'ch20',
  titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['ch20']['novo_nome']}",
  cor = 'skyblue'
)
```

Boxplot da variável ch20

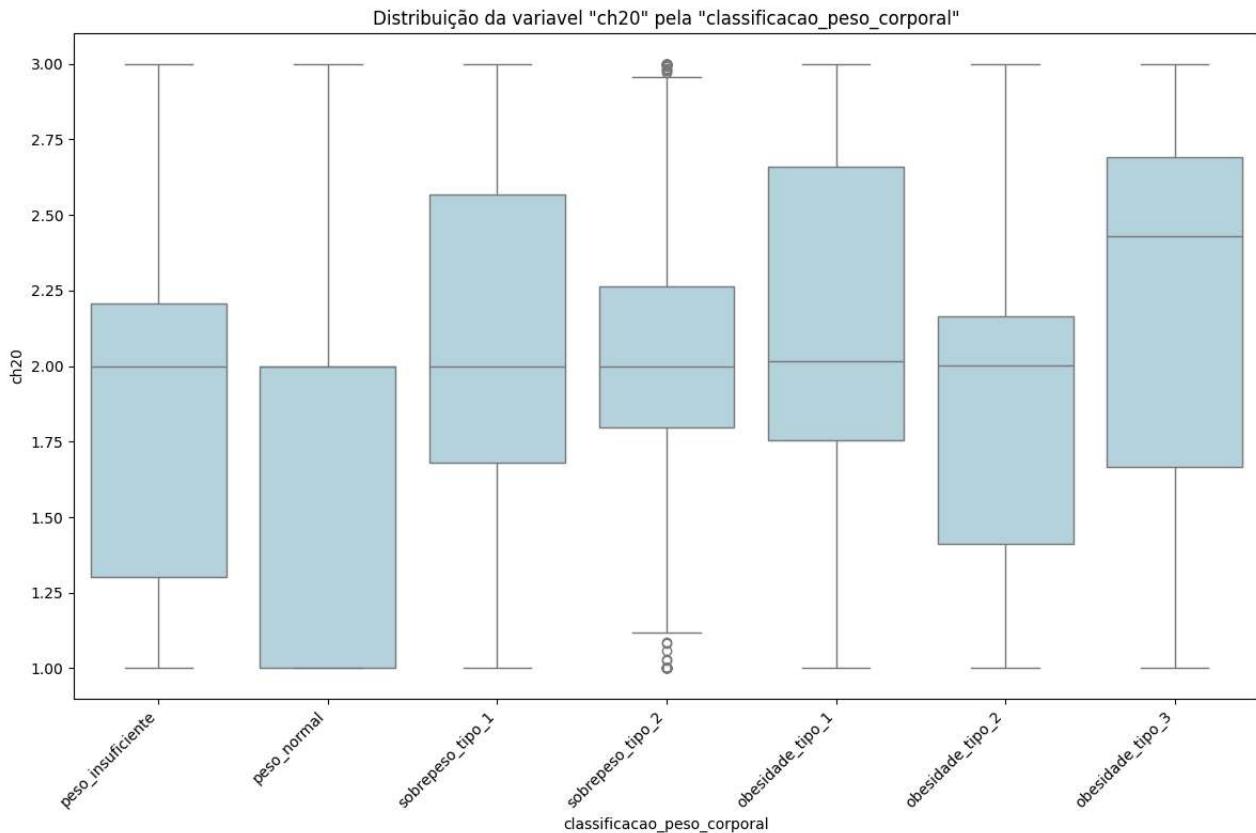


## Análise Bivariada

ch20 vs classificacao\_peso\_corporal

### Boxplot agrupado

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'ch20',
    variavel_alvo = 'classificacao_peso_corporal',
    ordem=ordem_classificao_peso_corporal,
    cor='lightblue')
```



## Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'ch20')

--- Resultados do Teste ANOVA (ch20 vs classificacao_peso_corporal) ---
Estatística F: 16.1711
P-valor: 2.837e-18
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'ch20' entre os grupos de 'classificacao_peso_corporal' (np.float64(16.17142194378753), np.float64(2.8373240173998843e-18))
```

## ▼ Análise variável `faf`

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['faf']['novo_nome']}: {dicionario_dados_nivel_obesidade['faf']}
```

Descrição da variável `faf`: Com que frequencia voce pratica atividade fisica?

### Conclusão

A variável `faf` devido a sua pergunta (Quanta agua voce bebe diariamente?) pode indicar tanto uma variável contínua se prensarmos em quantidade de água em litros, quanto categórica ordinal com o padrão `['nunca', 'as vezes', 'frequentemente', 'sempre']`.

Os valores mínimos de 0 e maximo 4 corroboram as hipótese e também foi identificado o mesmo comportamento de outras variáveis que sugere um preenchimento de dados nulos.

Presumindo que se trata de uma variável categorica ordinal podemos considerar a categoria 3 como rara (75 registros)

O histograma revela 4 picos relacionados aos valores inteiros o que é mais um indício sobre a natureza da variável apresentada anteriormente. Aqui temos uma distribuição assimétrica a direita.

A análise bivariada informa uma associação estatística relevante entre a variável alvo e a variável `faf`. No entanto podemos perceber que essa relevância está fortemente associada a uma maior prática de atividade física e insuficiência de peso e uma menor prática a obesidade 3

## Sugestões

- Realizar um processo de limpeza do ruído relacionado a variável `faf` por meio de arredondamento dos dados.
- Após o arredondamento tratá-la como uma variável ordinal e usar a técnica de `ordinal encoding`. Mapeamento = {0:0, 1:1, 2:2, 3:3}
- Após o treinamento verificar a importância da feature e realizar o teste A/B para saber se vale o risco de manter essa variável no treinamento por conta da raridade da categoria 3.

## Visualização prévia dos dados

A variável `faf` devido a sua pergunta (Com que frequencia voce pratica atividade fisica?) indica ser uma variável categórica com padrão. Na análise inicial verificamos os valor mínimo como 1 e máximo como 3 e o comportamento identificado em outras variáveis que sugere um preenchimento de dados nulos.

Minha hipótese é que essa variável está mais para uma variável categorica ordinal do que variável contínua.

```
df.faf.unique()
array([0.      , 3.      , 2.      , ..., 1.414209, 1.139107, 1.026452])

faf_amostra = len(df.faf)
faf_min = df.faf.min()
faf_max = df.faf.max()
faf_total_valores = len(df.faf.unique())
print(f'Valor mínimo na variável faf: {faf_min} ')
print(f'Valor máximo na variável faf: {faf_max} ')
print(f'Total de valores na variável faf: {faf_amostra} ')
print(f'Total de valores únicos na variável faf: {faf_total_valores} ')

Valor mínimo na variável faf: 0.0
Valor máximo na variável faf: 3.0
Total de valores na variável faf: 2111
Total de valores únicos na variável faf: 1190

resposta_faf_0 = len(df.query("faf == 0"))
resposta_faf_1 = len(df.query("faf == 1"))
resposta_faf_2 = len(df.query("faf == 2"))
resposta_faf_3 = len(df.query("faf == 3"))
total_inteiros_faf = resposta_faf_0 + resposta_faf_1 + resposta_faf_2 + resposta_faf_3
total_decimais_faf = faf_amostra - total_inteiros_faf

print(f'Quantidade de linhas na variável faf igual a 0: {resposta_faf_0}')
print(f'Quantidade de linhas na variável faf igual a 1: {resposta_faf_1}')
print(f'Quantidade de linhas na variável faf igual a 2: {resposta_faf_2}')
print(f'Quantidade de linhas na variável faf igual a 3: {resposta_faf_3}')
print(f'Total de linhas com numeros inteiros: {total_inteiros_faf}')
print(f'Total de linhas com numeros decimais: {total_decimais_faf}')

Quantidade de linhas na variável faf igual a 0: 411
Quantidade de linhas na variável faf igual a 1: 234
Quantidade de linhas na variável faf igual a 2: 183
Quantidade de linhas na variável faf igual a 3: 75
Total de linhas com numeros inteiros: 903
Total de linhas com numeros decimais: 1208
```

## Análise Univariada

### Estatística descritiva

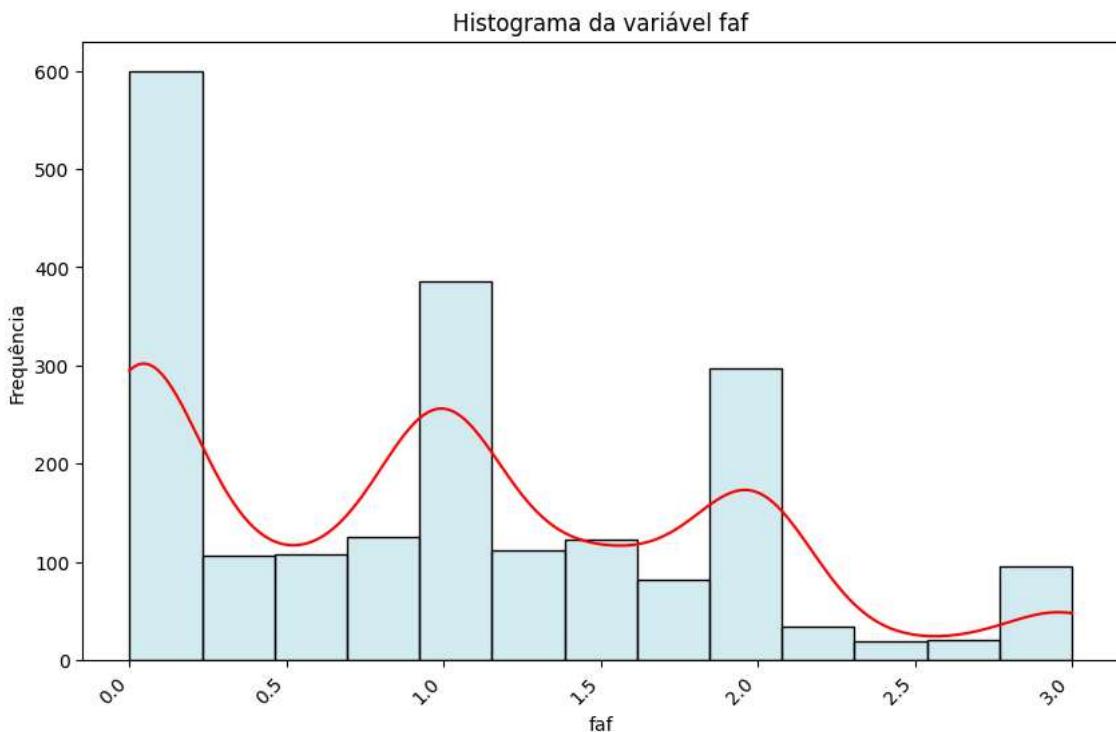
```
df['faf'].describe()
```

```
faf
count    2111.000000
mean     1.010298
std      0.850592
min      0.000000
25%     0.124505
50%     1.000000
75%     1.666678
max      3.000000

dtype: float64
```

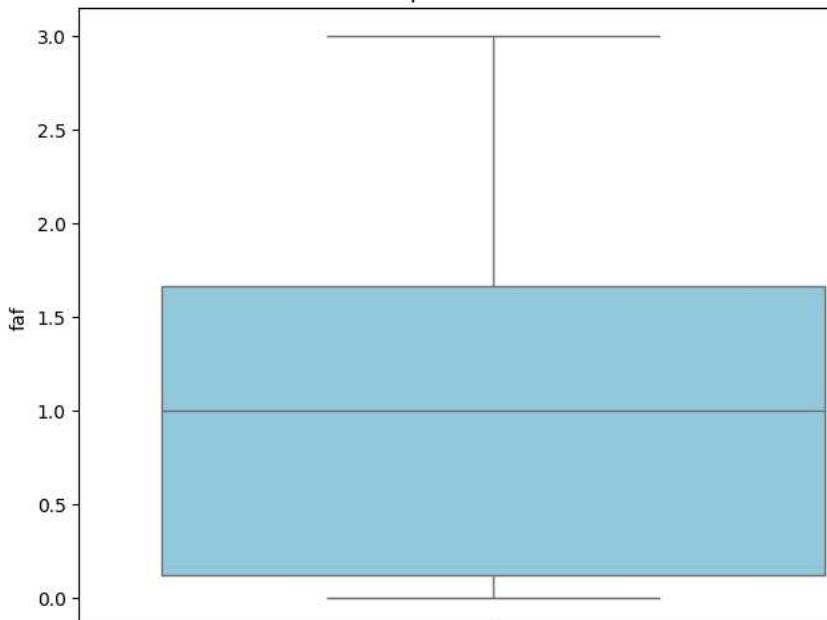
### Histograma

```
gera_histograma(
    df = df,
    coluna = "faf",
    titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['faf']['novo_nome']}",
    cor="lightblue",
    cor_kde="red"
)
```



### Boxplot

```
plotar_boxplot_numerica(
    df,
    'faf',
    titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['faf']['novo_nome']}",
    cor = 'skyblue'
)
```

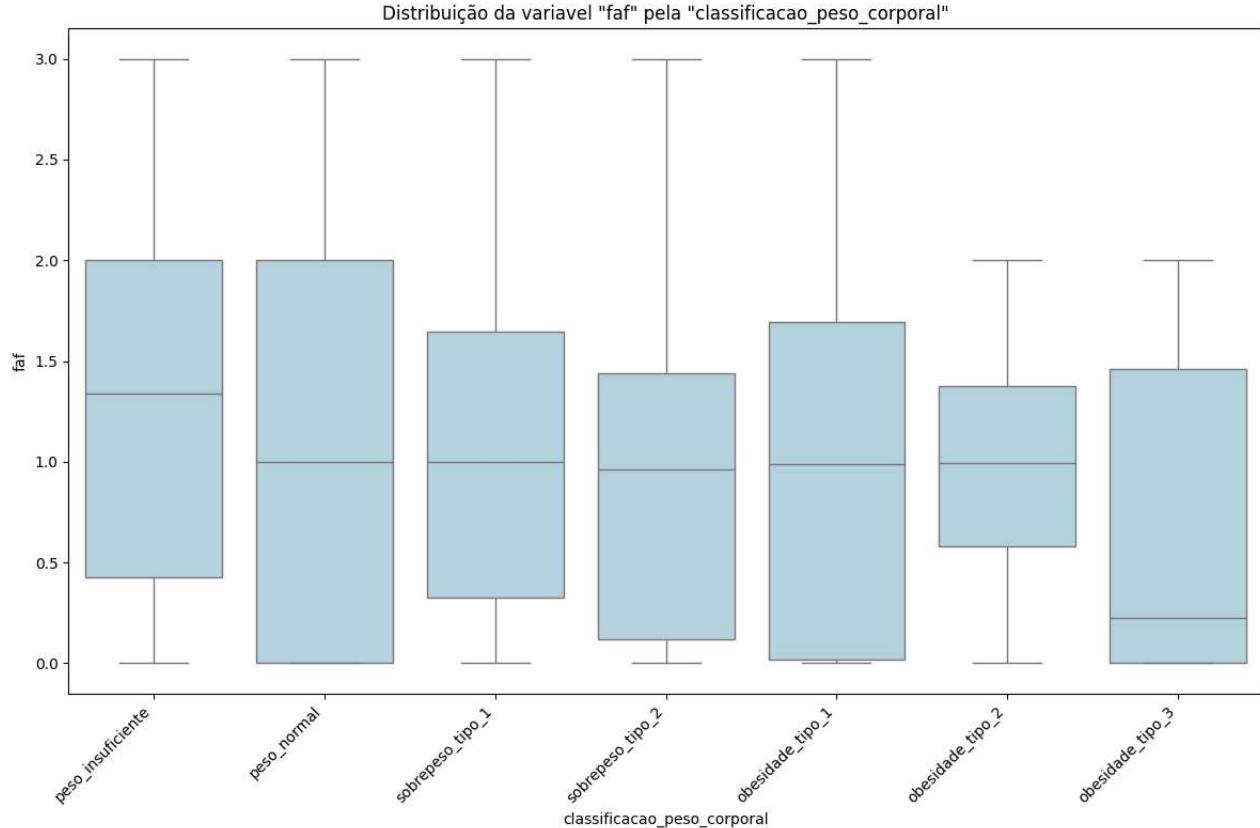
Boxplot da variável `faf`

## Análise Bivariada

`faf` vs `classificacao_peso_corporal`

### Boxplot agrupado

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'faf',
    variavel_alvo = 'classificacao_peso_corporal',
    ordem=ordem_classificao_peso_corporal,
    cor='lightblue')
```



## Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'faf')

--- Resultados do Teste ANOVA (faf vs classificacao_peso_corporal) ---
Estatística F: 17.4842
P-valor: 7.653e-20
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'faf' entre os grupos de 'classificacao_peso_corporal'.
```

## ▼ Análise variável tue

```
print(f"Descrição da variável {dicionario_dados_nivel_obesidade['tue']['novo_nome']}: {dicionario_dados_nivel_obesidade['tue']}
```

Descrição da variável tue: Quanto tempo voce usa dispositivos tecnologicos como celular,

### Conclusão

A variável `tue` devido a sua pergunta (Quanto tempo voce usa dispositivos tecnologicos como celular?) pode indicar tanto uma variável numérica continua que os valores representem o tempo de uso como uma variável categórica que represente três categorias de uso visto que temos valor mínimo 0 e máximo 2. Foi observado um volume de resposta com os valores inteiros 0 (557), 1(292) e 2(109). A feature apresenta um comportamento parecido com o identificado em outras variáveis, sugerindo um preenchimento de dados nulos.

O histograma revela 3 picos relacionados aos valores inteiros o que é mais um indício sobre a natureza da variável apresentada anteriormente. Aqui temos uma distribuição assimétrica a direita.

A análise bivariada informa uma associação estatística relevante entre a variável alvo e a variável `tue`.

### Sugestões

- Realizar um processo de limpeza do ruído relacionado a variável `tue` por meio de arredondamento dos dados.
- Após o arredondamento tratará-la como uma variável ordinal e usar a técnica de `ordinal encoding`. Mapeamento = {0:0, 1:1, 2:2}

## Visualização prévia dos dados

A variável `tue` devido a sua pergunta (Quanto tempo você usa dispositivos tecnológicos como celular?) pode indicar tanto uma variável numérica contínua que os valores representem o tempo de uso como uma variável categórica que represente três categorias de uso visto que temos valor mínimo 0 e máximo 2. Foi observado um volume de resposta com os valores inteiros 0 (557), 1(292) e 2(109). A feature apresenta um comportamento parecido com o identificado em outras variáveis, sugerindo um preenchimento de dados nulos.

```
df.tue.unique()
array([1.0, 0.0, 2.0, ..., 0.646288, 0.586035, 0.714137])
```

```
tue_amostra = len(df.tue)
tue_min = df.tue.min()
tue_max = df.tue.max()
tue_total_valores = len(df.tue.unique())
print(f'Valor mínimo na variável tue: {tue_min}')
print(f'Valor máximo na variável tue: {tue_max}')
print(f'Total de valores na variável faf: {tue_amostra}')
print(f'Total de valores únicos na variável faf: {tue_total_valores}')
```

```
Valor mínimo na variável tue: 0.0
Valor máximo na variável tue: 2.0
Total de valores na variável faf: 2111
Total de valores únicos na variável faf: 1129
```

```
resposta_tue_0 = len(df.query("tue == 0"))
resposta_tue_1 = len(df.query("tue == 1"))
resposta_tue_2 = len(df.query("tue == 2"))
total_inteiros_tue = resposta_tue_0 + resposta_tue_1 + resposta_tue_2
total_decimais_tue = tue_amostra - total_inteiros_tue

print(f'Quantidade de linhas na variável tue igual a 0: {resposta_tue_0}')
print(f'Quantidade de linhas na variável tue igual a 1: {resposta_tue_1}')
print(f'Quantidade de linhas na variável tue igual a 2: {resposta_tue_2}')
print(f'Total de linhas com números inteiros: {total_inteiros_tue}')
print(f'Total de linhas com números decimais: {total_decimais_tue}')
```

```
Quantidade de linhas na variável tue igual a 0: 557
Quantidade de linhas na variável tue igual a 1: 292
Quantidade de linhas na variável tue igual a 2: 109
Total de linhas com números inteiros: 958
Total de linhas com números decimais: 1153
```

## Análise Univariada

### Estatística descritiva

```
df['tue'].describe()
```

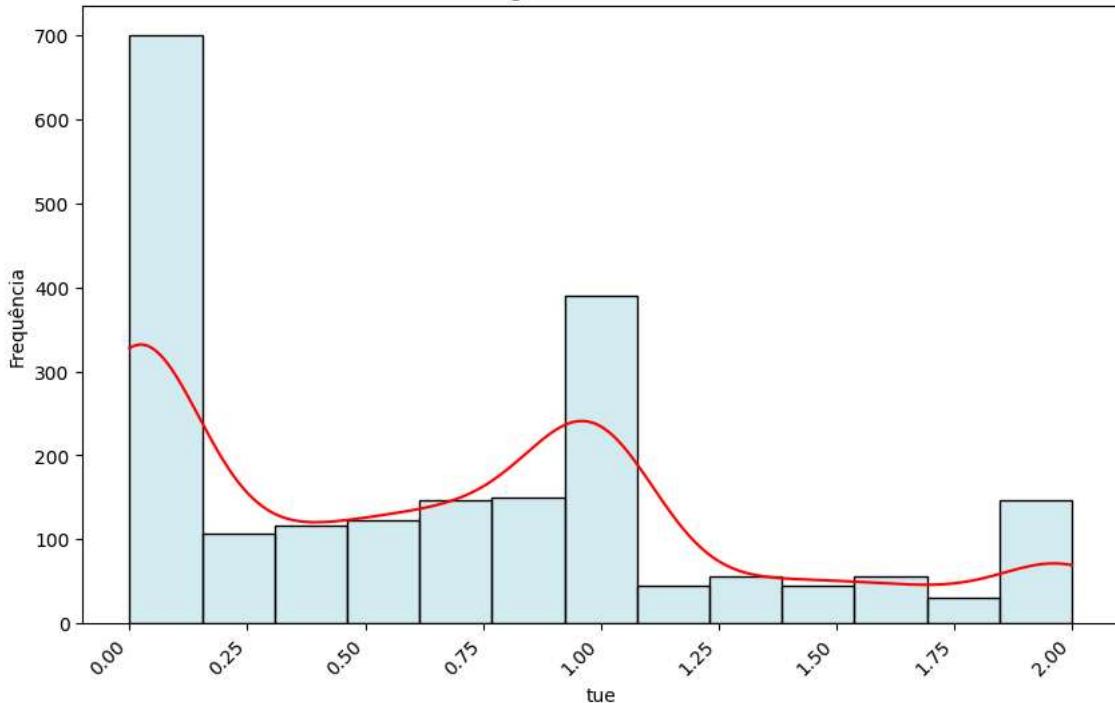
	tue
<b>count</b>	2111.000000
<b>mean</b>	0.657866
<b>std</b>	0.608927
<b>min</b>	0.000000
<b>25%</b>	0.000000
<b>50%</b>	0.625350
<b>75%</b>	1.000000
<b>max</b>	2.000000

```
dtype: float64
```

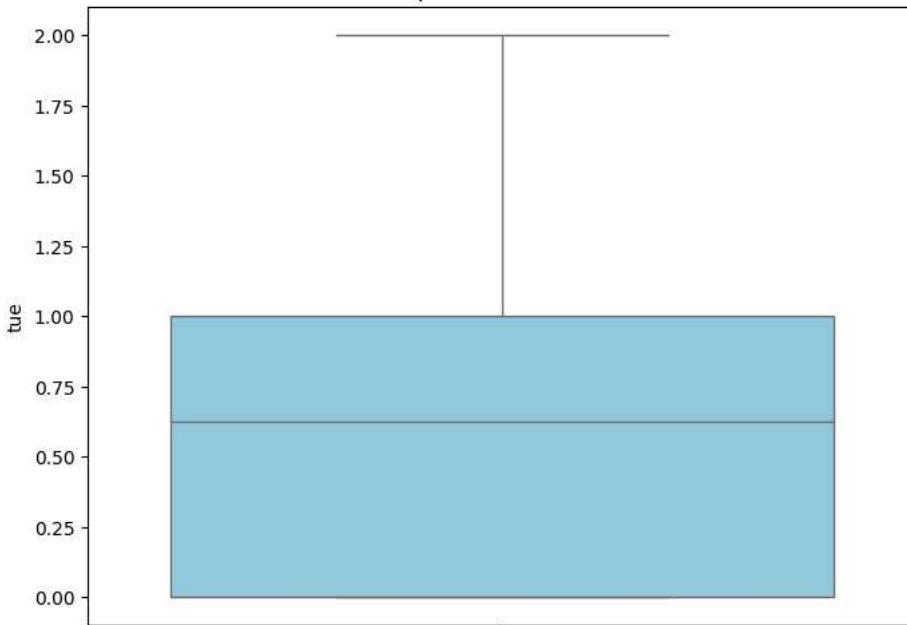
## Histograma

```
gera_histograma(
    df = df,
    coluna = "tue",
    titulo = f"Histograma da variável {dicionario_dados_nivel_obesidade['faf']['novo_nome']}")
```

```
cor="lightblue",
cor_kde="red"
)
```

Histograma da variável *faf***Boxplot**

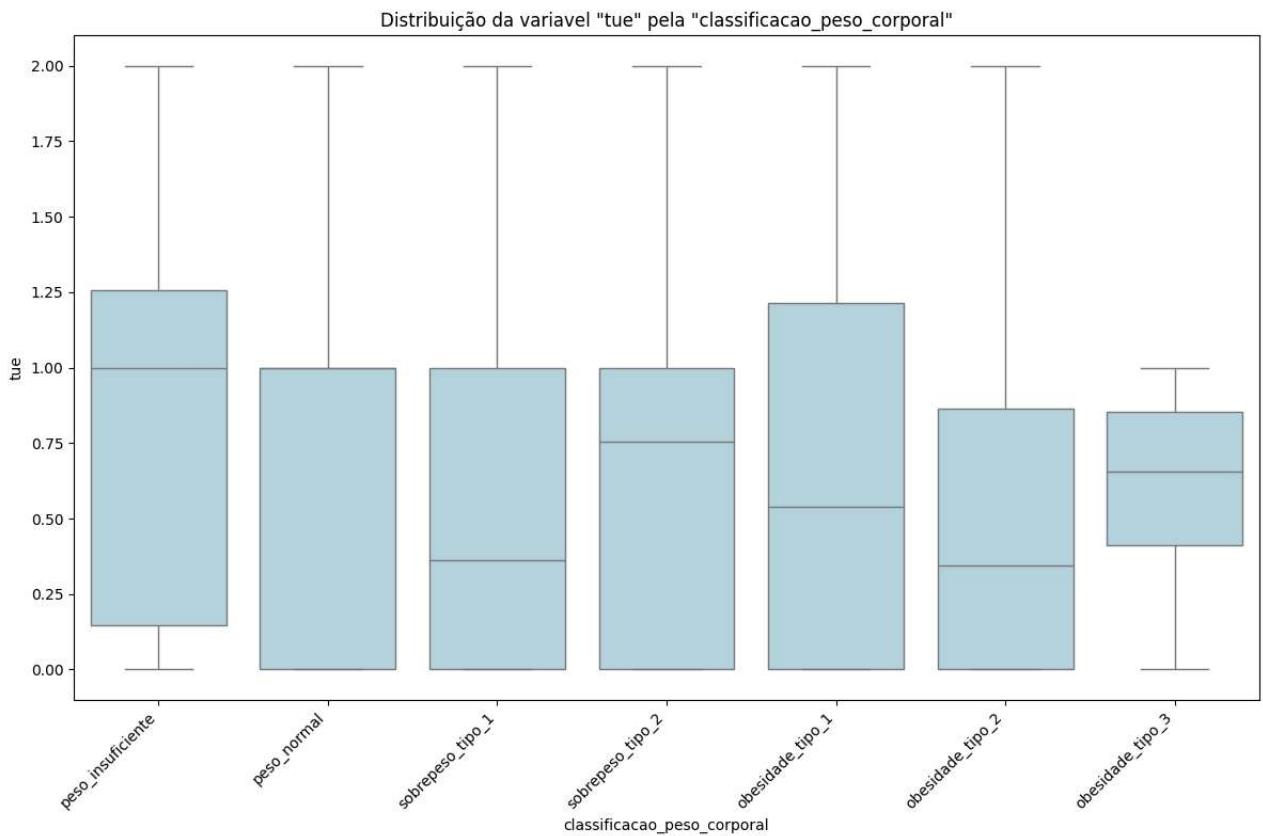
```
plotar_boxplot_numerica(
    df,
    'tue',
    titulo= f"Boxplot da variável {dicionario_dados_nivel_obesidade['faf'][ 'novo_nome']}",
    cor = 'skyblue'
)
```

Boxplot da variável *faf***Análise Bivariada**

`tue` vs `classificacao_peso_corporal`

**Boxplot agrupado**

```
plotar_boxplot_agrupado_horizontal(
    df = df,
    variavel = 'tue',
    variavel_alvo = 'classificacao_peso_corporal',
    ordem=ordem_classificacao_peso_corporal,
    cor='lightblue')
```



## Teste ANOVA

```
teste_anova(df = df, agrupa_col = 'classificacao_peso_corporal', nome_col = 'tue')

--- Resultados do Teste ANOVA (tue vs classificacao_peso_corporal) ---
Estatística F: 7.8767
P-valor: 2.069e-08
Interpretação (p < 0.05): REJEITA H0.
CONCLUSÃO: Existe uma diferença ESTATISTICAMENTE SIGNIFICATIVA nas médias de 'tue' entre os grupos de 'classificacao_peso_corporal'.
```

## Engenharia de Features

### Cópia do dataframe original

```
df_processado = df.copy()
```

### Limpeza de variáveis

### Arredondamento de dados sintéticos e conversão de float para int.

```
[fcvc, ncp, ch20, faf, tue]
```

```
df_processado['fcvc_arredondada'] = df_processado['fcvc'].round(0).astype(int)
df_processado['ncp_arredondada'] = df_processado['ncp'].round(0).astype(int)
df_processado['ch20_arredondada'] = df_processado['ch20'].round(0).astype(int)
df_processado['faf_arredondada'] = df_processado['faf'].round(0).astype(int)
df_processado['tue_arredondada'] = df_processado['tue'].round(0).astype(int)

print("Colunas convertidas e arredondadas:")
display(df_processado[['fcvc_arredondada', 'ncp_arredondada', 'ch20_arredondada', 'faf_arredondada', 'tue_arredondada']].head())
```

Colunas convertidas e arredondadas:

	<code>fcvc_arredondada</code>	<code>ncp_arredondada</code>	<code>ch20_arredondada</code>	<code>faf_arredondada</code>	<code>tue_arredondada</code>	
0	2	3	2	0	1	
1	3	3	3	3	0	
2	2	3	2	2	1	
3	3	3	2	2	0	
4	2	1	2	0	0	

```
df_processado.ncp_arredondada.value_counts()
```

	count
<code>ncp_arredondada</code>	
3	1470
1	316
2	176
4	149

`dtype: int64`

## ▼ Agrupamento de categorias raras

`mtrans`

Ação: Agrupar os valores `moto`, `bicicleta` e `caminhando` em uma nova categoria chamada `outros`.

Motivo: Devido à raridade e padrão bivariado similar

`calc`

Ação: Agrupar os valores `frequente` e `sempre` em uma nova categoria `frequentemente_sempre`

Motivo: Devido a raridade e proximidade por ordem de grandeza.

## ▼ mtrans

```
mapa_mtrans_agrupado = {
    'transporte_publico': 'transporte_publico',
    'carro': 'carro',
    'caminhando': 'outros',
    'moto': 'outros',
    'bicicleta': 'outros'
}

df_processado['mtrans_mod'] = df_processado['mtrans'].map(mapa_mtrans_agrupado)

print("Totais valores originais variável 'mtrans':")
display(df_processado['mtrans'].value_counts())

print("\nTotais valores modificados variável 'mtrans_mod':")
display(df_processado['mtrans_mod'].value_counts())

if dicionario_dados_nivel_obesidade is not None:
    atualizar_dicionario_novas_colunas(dicionario_dados_nivel_obesidade, df_processado)
    descricao_mtrans_mod = "Variável 'mtrans' com categorias raras (caminhando, moto, bicicleta) agrupadas em 'outros'."
    atualizar_campo_dicionario(dicionario_dados_nivel_obesidade, 'mtrans_mod', 'descricao', descricao_mtrans_mod)
    transformacao_info = {
        'tipo': 'agrupamento_categorias',
        'mapa_aplicado': mapa_mtrans_agrupado
    }
    atualizar_campo_dicionario(dicionario_dados_nivel_obesidade, 'mtrans_mod', 'transformacao', transformacao_info)
```