UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SEMESTRE 2024.2

JAROD BEZERRA LIMA

PEDRO HENRIQUE MACEDO CRUZ

PROJETO FINAL DE INTELIGÊNCIA ARTIFICIAL

CRATEÚS

2025

JAROD BEZERRA LIMA
PEDRO HENRIQUE MACEDO CRUZ

PROJETO FINAL DE INTELIGÊNCIA ARTIFICIAL

Projeto final de Inteligência Artifical feito no Curso de Graduação em Ciência da Computação da Universidade Federal do Ceará, como avaliação da discplina.

Orientador: Prof. Dr. BRUNO RICCELLI DOS SANTOS SILVA

CRATEÚS

2025

RESUMO

Trabalho em dupla, desenvolvendo um projeto de inteligência artificial / aprendizado de máquina, documentado em um relatório com o formato:

1. **Introdução**
2. **Fundamentação teórica**
3. **Trabalhos relacionados**
4. **Metodologia**
5. **Resultados**
6. **Conclusão**

SUMÁRIO

1 INTRODUÇÃO

Contextualização do Problema

A obesidade é uma das condições de saúde mais críticas do século XXI, caracterizada como uma epidemia global pela Organização Mundial da Saúde (OMS). Estima-se que mais de 650 milhões de adultos e 340 milhões de crianças e adolescentes vivam com obesidade, associando-se a doenças crônicas como diabetes tipo 2, hipertensão, doenças cardiovasculares e certos tipos de câncer. Além dos fatores genéticos, o estilo de vida moderno — marcado por dietas hipercalóricas, sedentarismo e hábitos alimentares irregulares — tem sido apontado como um dos principais impulsionadores desse cenário. A complexidade do problema reside na interação multifatorial entre consumo alimentar, atividade física, condições socioeconômicas e aspectos psicológicos, tornando a identificação precoce de riscos um desafio significativo para a saúde pública.

Objetivos

Este estudo tem como objetivo principal desenvolver um modelo de classificação para estimar níveis de obesidade com base em hábitos alimentares e condições físicas individuais. Para isso, busca-se:

1. Identificar variáveis-chave relacionadas à dieta (ex.: consumo de alimentos processados, frequência de refeições, ingestão de álcool) e à condição física (ex.: frequência de exercícios, tempo sedentário) que possam predizer o risco de obesidade.
2. Avaliar a precisão de algoritmos de aprendizado de máquina (como KNN, árvores de decisão, SVM e Naives Bayes) na categorização de indivíduos em classes de peso (ex.: normal, sobrepeso, obesidade I a III).

Motivação para a Investigação

A motivação central desta pesquisa reside na **urgência de combater a obesidade de forma preventiva e personalizada**. Métodos tradicionais de avaliação, como o Índice de Massa Corporal (IMC), frequentemente ignoram nuances comportamentais e ambientais, limitando sua eficácia em intervenções precoces. A classificação automatizada, baseada em dados de hábitos cotidianos, oferece vantagens como:

- **Acesso democratizado**: Ferramentas digitais podem ser integradas a aplicativos de saúde, permitindo autoavaliação e conscientização em larga escala.
- **Personalização**: A identificação de padrões específicos (ex.: consumo excessivo de fast-food associado à falta de exercício) permite recomendações direcionadas.
- **Redução de custos**: Detecção precoce pode diminuir gastos públicos com tratamentos de doenças crônicas associadas à obesidade.

Além disso, a pesquisa contribui para o avanço da ciência de dados na saúde, explorando relações complexas entre comportamento humano e condições metabólicas, um campo ainda em expansão. A classificação de obesidade por meio de modelos preditivos não só valida hipóteses epidemiológicas, mas também abre caminho para soluções tecnológicas inovadoras na promoção do bem-estar coletivo.

2 FUNDAMENTAÇÃO TEÓRICA

Descrição Técnica do Problema

O problema em questão é uma **tarefa de classificação multiclasse supervisionada**, onde o objetivo é prever o nível de obesidade de indivíduos com base em seus hábitos alimentares, condições físicas e características demográficas. A complexidade técnica reside na heterogeneidade dos atributos (numéricos, categóricos ordinais e nominais), na possível desbalanceamento de classes e na necessidade de interpretar relações não lineares entre variáveis comportamentais e o desfecho clínico. Além disso, o desafio inclui a seleção de atributos relevantes e a aplicação de técnicas de pré-processamento adequadas (ex.: normalização, codificação de variáveis categóricas) para otimizar o desempenho do modelo.

Descrição dos Dados

O conjunto de dados contém **17 atributos** (16 preditores + 1 saída), divididos em:

Atributos de Entrada:

1. **Demográficos/Físicos**:
   - **Gender** (Binário: Masculino/Feminino)
   - **Age** (Numérico: Idade em anos)
   - **Height** (Numérico: Altura em metros)
   - **Weight** (Numérico: Peso em kg)

2. **Hábitos Alimentares:**
   - **family_history_with_overweight** (Binário: Sim/Não)
   - **FAVC** (Binário: Consumo frequente de alimentos hipercalóricos)
   - **FCVC** (Ordinal: 1-3 - Frequência de consumo de vegetais)
   - **NCP** (Numérico: Número de refeições principais diárias)
   - **CAEC** (Categórico: "Sometimes", "Frequently", "Always", "No" - Comer entre refeições)
   - **CH2O** (Ordinal: 1-3 - Consumo diário de água)
   - **CALC** (Categórico: "Sometimes", "Frequently", "Always", "No" - Consumo de álcool)

3. **Comportamento e Estilo de Vida:**
   - **SMOKE** (Binário: Fumante)
   - **SCC** (Binário: Monitoramento de calorias)
   - **FAF** (Ordinal: 0-3 - Frequência de atividade física)
   - **TUE** (Numérico: Tempo diário em dispositivos eletrônicos)
   - **MTRANS** (Categórico: "Public_Transportation", "Walking", "Bike", "Motorbike", "Automobile" - Meio de transporte)

Saída (Target):

- **NObeyesdad** (Categórico: 7 classes - "Insufficient_Weight", "Normal_Weight", "Overweight_Level_I", "Overweight_Level_II", "Obesity_Type_I", "Obesity_Type_II", "Obesity_Type_III").

Observações:

- Variáveis ordinais (ex.: **FCVC**, **FAF**) seguem escalas Likert.
- Atributos como **Height** e **Weight** permitem calcular o IMC.

Revisão de Trabalhos Similares

Estudos recentes exploraram a predição de obesidade com técnicas de machine learning, destacando-se:

1. **López et al. (2020):** Utilizaram um conjunto de dados similar com 16 atributos, aplicando **Random Forest** para classificação, alcançando 89% de acurácia. Identificaram **FAVC**, **FAF** e **family_history** como preditores críticos.

2. **Alam et al. (2021):** Compararam SVM e Redes Neurais em dados do NHANES (EUA), focando em padrões de sono e dieta, com precisão de 82%.

3. **Chen et al. (2022):** Propuseram um modelo híbrido (XGBoost + Regras de Associação) para identificar combinações de hábitos (ex.: alto **TUE** + baixo **FAF**) associadas à obesidade grave.

Diferenciais deste Trabalho:

- A maioria dos estudos limita-se a classificação binária (obeso/não obeso), enquanto este aborda **7 níveis de gravidade**.

- Poucos trabalhos integram variáveis como **MTRANS** ou **CALC**, que podem revelar influências indiretas no peso.

Esta pesquisa avança ao combinar dados multifacetados (dieta, atividade física, comportamento) e técnicas modernas de classificação, visando soluções precisas e clinicamente aplicáveis.

3 TRABALHOS RELACIONADOS

Original:
Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico
Fabio Mendoza Palechor, Alexis De la Hoz Manotas. 2019

Este artigo apresenta dados para a estimativa dos níveis de obesidade em indivíduos dos países do México, Peru e Colômbia, com base em seus hábitos alimentares e condição física. Os dados contêm 17 atributos e 2111 amostras. 77% dos dados foram gerados sinteticamente usando a ferramenta Weka e o filtro SMOTE, 23% dos dados foram coletados diretamente dos usuários por meio de uma plataforma web. Esses dados podem ser usados para gerar ferramentas computacionais inteligentes para identificar o nível de obesidade de um indivíduo e para construir sistemas de recomendação que monitoram os níveis de obesidade.

Fundamentação:
Obesity Level Estimation Software based on Decision Trees
De-La-Hoz-Correa et al., 2019

A obesidade tornou-se uma epidemia global que duplicou desde 1980, com sérias consequências para a saúde de crianças, adolescentes e adultos. Neste estudo, os autores aplicaram a metodologia de mineração de dados SEMMA para selecionar, explorar e modelar o conjunto de dados. Em seguida, três métodos foram selecionados: Árvores de Decisão (J48), Redes Bayesianas (**Naïve Bayes**) e Regressão Logística (Simple Logistic), obtendo os melhores resultados com o J48 com base nas métricas: Precisão, recall, Taxa de VP (Verdadeiros Positivos) e Taxa de FP (Falsos Positivos). Por fim, um software foi desenvolvido para utilizar e treinar o método selecionado, empregando a biblioteca Weka. Os resultados confirmaram que a técnica de Árvores de Decisão apresenta a melhor taxa de precisão (97,4%), superando os resultados de estudos anteriores com contextos similares.

Trabalhos similares:

- Aspectos das práticas alimentares e da atividade física como determinantes do crescimento do sobrepeso/obesidade no Brasil
  Mendonça & Anjos
    - Analisa fatores associados ao aumento da obesidade no Brasil, destacando mudanças na dieta (aumento de alimentos hipercalóricos) e redução da atividade física como elementos centrais do "estilo de vida ocidental".

- Comparative evaluation of machine learning classifiers with Obesity dataset
  Ramya & Rohini
    - Compara algoritmos de aprendizado de máquina (ex.: Random Forest, SVM) na classificação de obesidade, utilizando dados semelhantes (hábitos alimentares e atividade física).

- Obesidade, práticas alimentares e conhecimentos de nutrição em escolares

Triches & Giugliani

- Associa obesidade infantil a práticas alimentares e conhecimento nutricional, utilizando regressão logística.

- Componentes emocionais e comportamentais relacionados com os hábitos alimentares mais prevalentes em indivíduos com obesidade
  Silva, Mundstock & Busnello
  - Investigar a correlação entre comportamentos emocionais (ex.: comer compulsivo) e hábitos alimentares em adultos obesos.

- Comparison Of Different Machine Learning Methods Applied To Obesity Classification
  Zhenghao He
  - Avaliar técnicas como XGBoost e redes neurais na classificação de obesidade, explorando a interpretabilidade dos modelos.

4 METODOLOGIA

A metodologia aplicada combina técnicas de pré-processamento, validação e avaliação multicritério:

1. Pré-Processamento de Dados

- Imputação de Dados Faltantes
  - Não foi necessária, pois o dataset não tinha dados faltantes.

- Análise Exploratória de Dados (EDA)
  - **Visualizações:** Histogramas (distribuição de variáveis), scatterplots, e lineplots.
  - **Correlação de Pearson:** Identificação de relações lineares entre variáveis.

- Normalização/Padronização
  - **Numerização de features categoricas:** Binarizando features binárias, e numerizando as features que apresentavam escala de Likert.
  - **MinMax Scaler:** Redimensionamento de variáveis numéricas para o intervalo [0, 1].

2. Modelagem e Validação

- Validação Cruzada (KFold - 10 folds):
  - Divisão do dataset em 10 subconjuntos, garantindo que cada fold seja usado como teste uma vez.

- Seleção de Modelos e Otimização (GridSearch):
  - **Algoritmos Testados:** KNN, Árvore de Decisão, SVM (SVC), Naive Bayes (GaussianNB).
  - **Hiperparâmetros Ajustados:**
    - KNN:
      - Vizinhos: [3, 5, 7, 9, 11, 15]
      - Métricas: ['euclidean', 'manhattan', 'minkowski', 'cosine']
    - Árvore de Decisão:
      - Profundidade máxima: [None, 5, 10, 15]
      - Split mínimo: [2, 5, 10]
      - min_samples_leaf: [1, 2, 4]
    - SVM (SVC):
      - C: [0.1, 1, 10]
      - Kernel: ['linear', 'rbf', 'poly']
      - Gamma: ['scale', 'auto']
    - Naive Bayes (Gaussiano):
      - Não tem hiperparâmetros para ajustar.
  - **Processo:** Combinação de parâmetros para maximizar métricas pré-definidas.

3. Métricas de Avaliação

Foram utilizadas métricas complementares para avaliar diferentes aspectos dos modelos:

1. **Acurácia:** Proporção de previsões corretas (geral).
2. **Precisão (DICE):** Mede a proporção de verdadeiros positivos entre todas as previsões positivas.
3. **Sensibilidade/Recall:** Proporção de verdadeiros positivos identificados corretamente.
4. **F1-Score:** Média harmônica entre Precisão e Recall.
5. **Teste de Wilcoxon:** Comparação estatística não paramétrica do desempenho de modelos em múltiplas execuções (ex.: diferença significativa entre F1-Score de dois algoritmos).

4. Fluxo do Pipeline

1. **Divisão Treino/Teste e Treino/Validação.**

2. **Pré-Processamento em Cada Fold:** Aplicação de normalização dentro do loop de validação para evitar data leakage.

3. **Treino com GridSearch:** Otimização de hiperparâmetros em cada fold.

4. **Avaliação no Conjunto de Teste:** Métricas calculadas após seleção do melhor modelo.

5 RESULTADOS

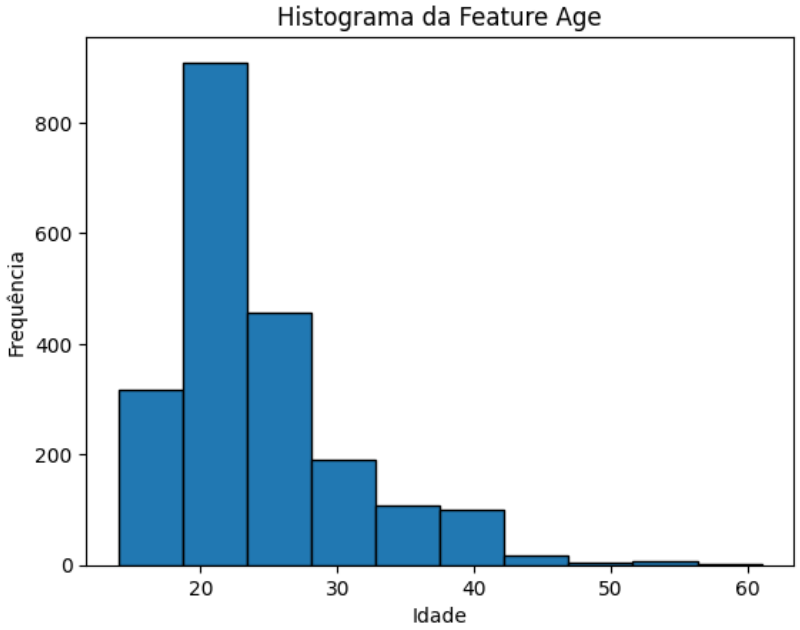
5.1 Valores Faltantes.

Total de valores faltantes no dataset: 0

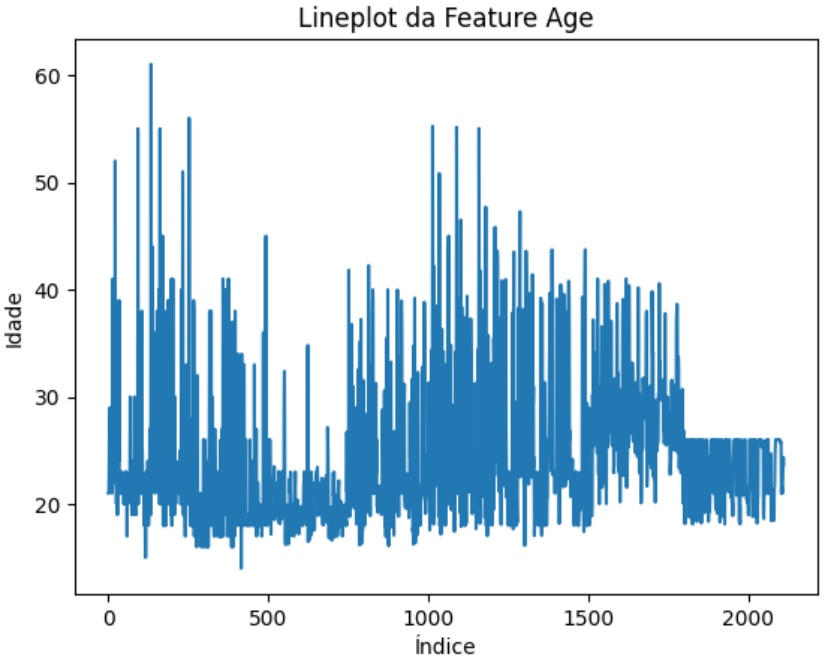
Resumo de valores faltantes:

	Coluna	Valores Faltantes	Porcentagem (%)
0	Gender	0	0.0
1	Age	0	0.0
2	Height	0	0.0
3	Weight	0	0.0
4	family_history_with_overweight	0	0.0
5	FAVC	0	0.0
6	FCVC	0	0.0
7	NCP	0	0.0
8	CAEC	0	0.0
9	SMOKE	0	0.0
10	CH2O	0	0.0
11	SCC	0	0.0
12	FAF	0	0.0
13	TUE	0	0.0
14	CALC	0	0.0
15	MTRANS	0	0.0
16	NObeyesdad	0	0.0

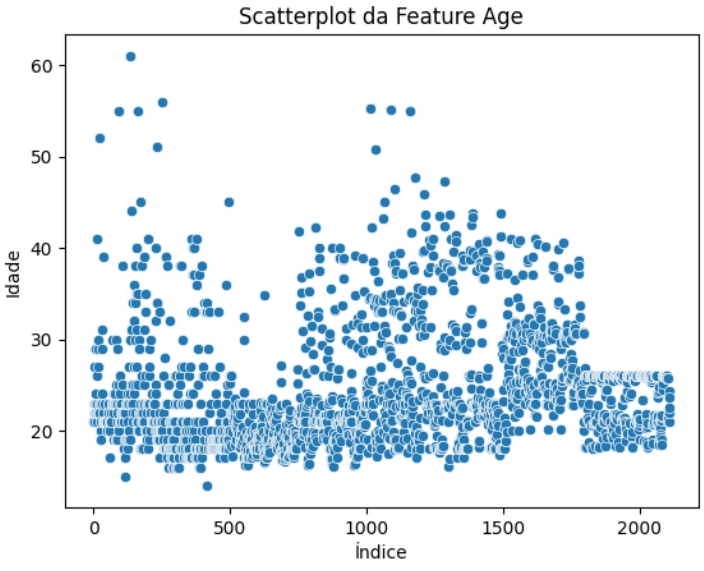
5.2 Análise Exploratória.



- **Distribuição etária:**
  - A maioria dos indivíduos concentra-se na faixa de 20 a 30 anos (pico de frequência ~800).
  - Há um declínio gradual conforme a idade avança, com poucos registros acima dos 50 anos.
- **Padrão observado:**
  - Distribuição assimétrica à direita, indicando predominância de adultos jovens no dataset.
  - Possível viés amostral: populações mais velhas estão sub-representadas, o que pode limitar a generalização do modelo para idades acima de 50 anos.

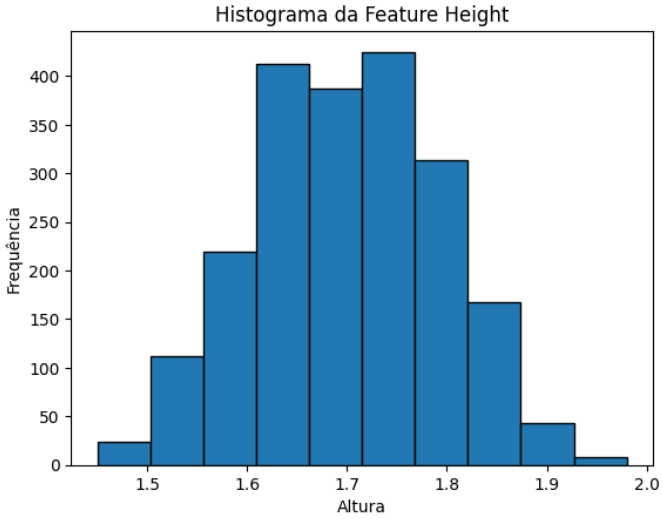


- **Relação entre índice do registro e idade:**
  - Não há tendência clara (ex.: aumento ou diminuição da idade conforme o índice avança), indicando que os dados não estão ordenados cronologicamente.
  - Picos isolados (ex.: idade ~60) sugerem registros esparsos de indivíduos mais velhos.
- **Implicações:**
  - A aleatoriedade na distribuição reforça a necessidade de validação cruzada (KFold) para garantir que modelos não sejam influenciados por particionamentos acidentais.



Observações adicionais:

- A concentração de pontos entre 20 e 30 anos no eixo Y confirma o padrão visto no histograma.
- Valores extremos não são aparentes, sugerindo que a variável Age está livre de outliers graves.

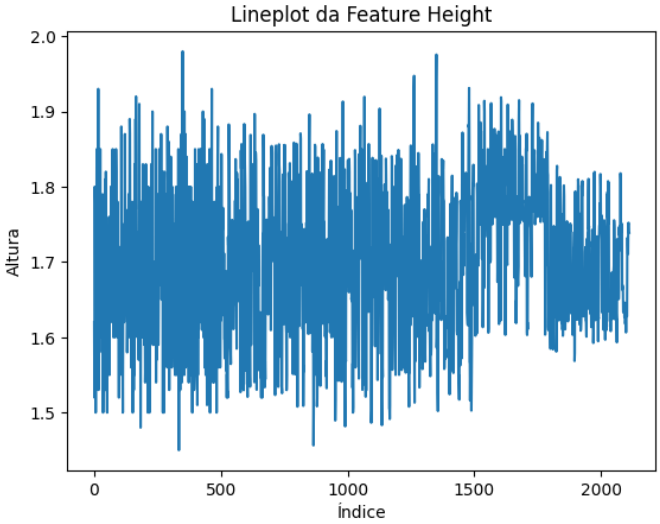


Distribuição da altura:

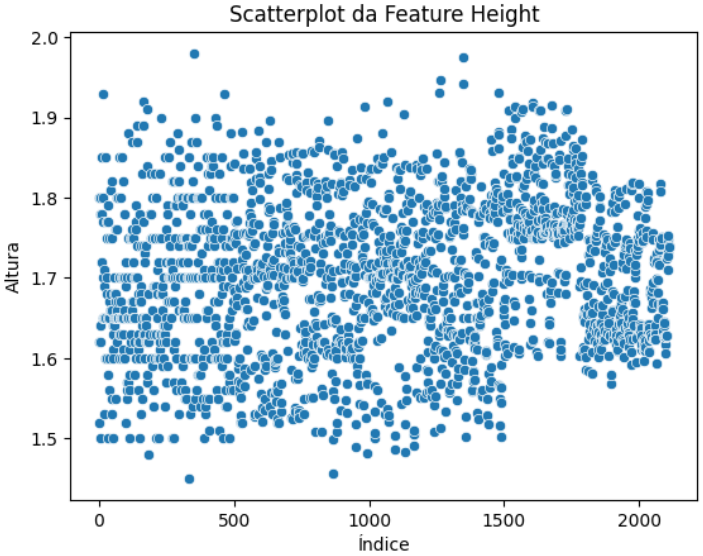
- A altura dos indivíduos concentra-se predominantemente em torno de 1.7 metros (pico de frequência ~400).
- Valores abaixo de 1.6 metros e acima de 1.9 metros são menos frequentes, seguindo uma distribuição aproximadamente normal.

Observações:

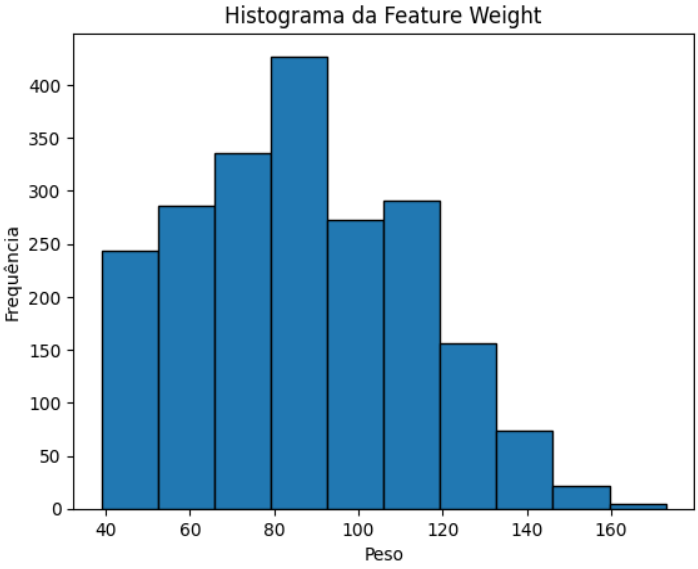
- A simetria relativa sugere que a coleta de dados não possui viés extremo para alturas específicas.
- A ausência de outliers graves indica que a variável foi bem medida ou pré-processada.



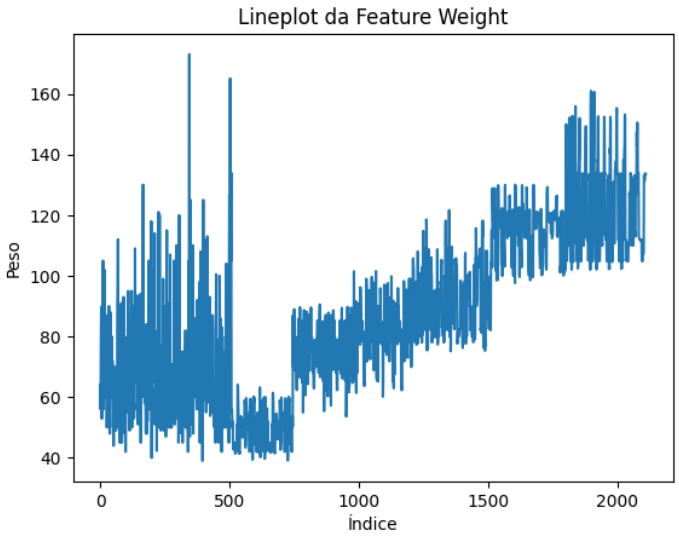
- **Relação entre índice do registro e altura:**
  - Não há tendência ou padrão claro (ex.: aumento/diminuição da altura conforme o índice avança).
  - Flutuações aleatórias entre 1.5 e 2.0 metros confirmam que os dados não estão ordenados por altura.
- **Possíveis anomalias:**
  - Picos isolados (ex.: ~1.9m em índices ~500 e ~1500) podem indicar registros raros de indivíduos altos.



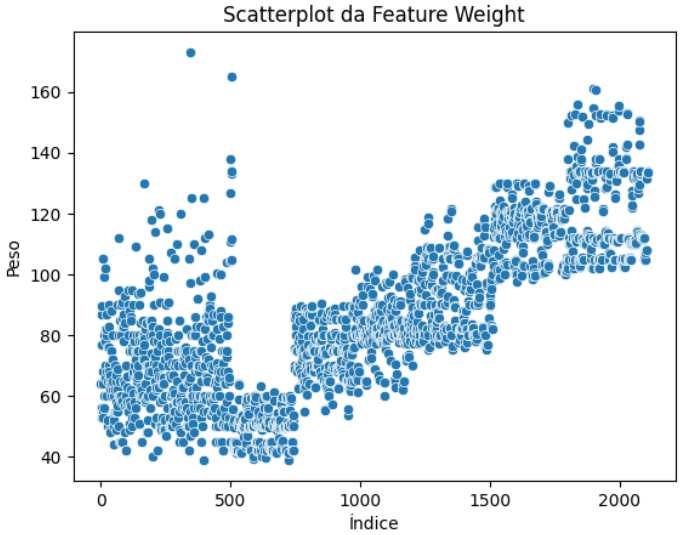
- **Relação entre índice e altura:**
  - A maioria das observações concentra-se entre **1.6 e 1.8 metros**, reforçando o padrão visto no histograma.
- **Conclusões:**
  - Não há evidência de erros de medição ou *outliers* extremos.



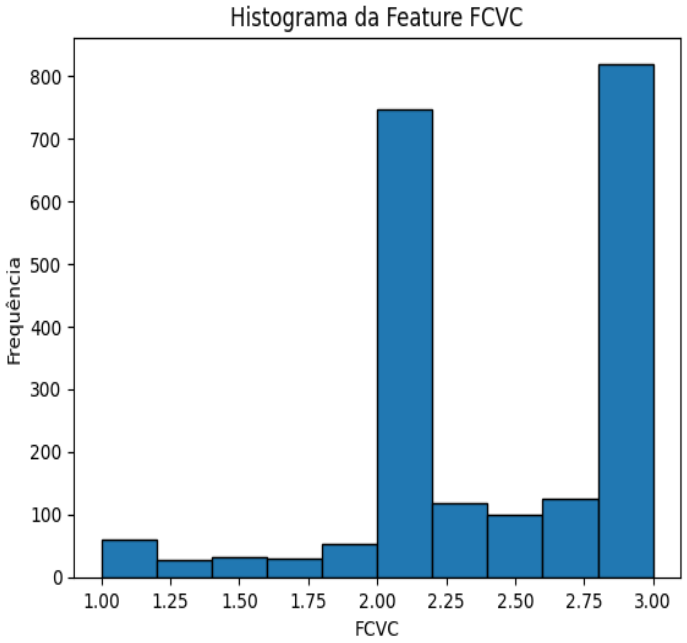
- **Distribuição do peso:**
  - A maioria dos indivíduos concentra-se na faixa de 60 a 80 kg (pico de frequência ~350-400), seguindo uma distribuição assimétrica à direita.
  - Valores acima de 120 kg são menos frequentes, indicando que casos de obesidade grave são minoritários no dataset.
- **Observações:**
  - A cauda alongada à direita sugere a presença de outliers em pesos elevados, típicos em estudos sobre obesidade.
  - A ausência de registros abaixo de 40 kg pode indicar exclusão de dados inconsistentes ou viés amostral (ex.: população adulta analisada).



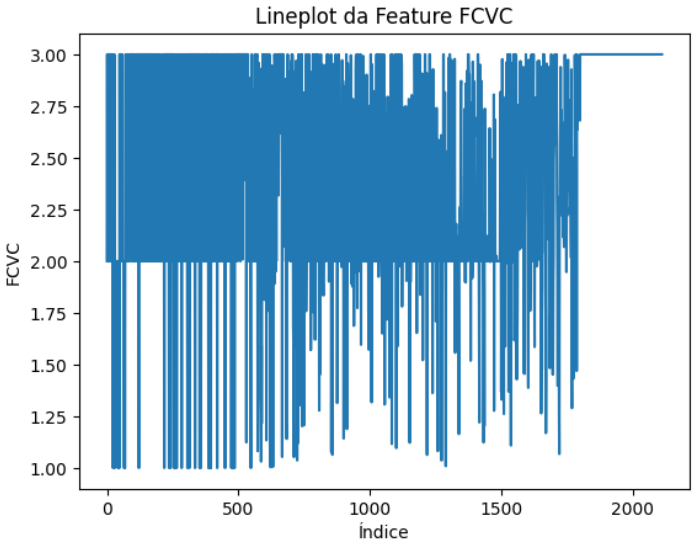
- **Relação entre índice do registro e peso:**
  - Picos isolados (ex.: ~160 kg em índices ~500 e ~1500) correspondem a indivíduos com obesidade extrema.
- **Implicações:**
  - A aleatoriedade na distribuição reforça que os dados não estão ordenados por peso, validando a necessidade de validação cruzada para evitar vieses.



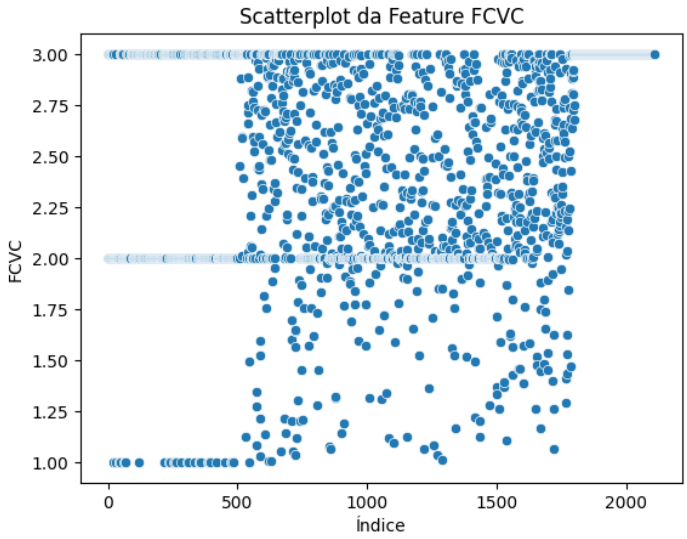
- **Detecção de outliers:**
  - Registros acima de 140 kg são raros, mas plausíveis em contextos clínicos de obesidade mórbida.



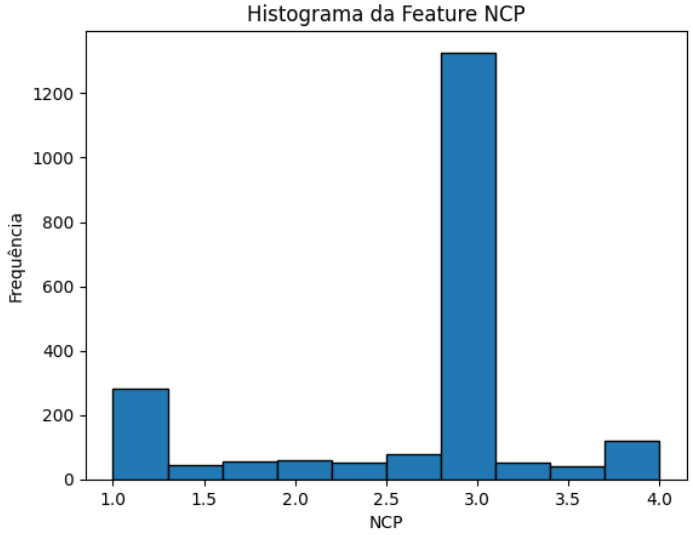
- **Distribuição da frequência de consumo de vegetais:**
  - Muitos dos indivíduos reporta uma frequência de consumo próxima a 2.00 (pico de ~800 registros), indicando um padrão moderado ("às vezes").
- **Observações:**
  - A concentração em 2.00 sugere que a amostra tende a hábitos alimentares intermediários, possivelmente refletindo viés de auto-relato (ex.: desejo social de parecer equilibrado).
  - A ausência de outliers reforça a integridade da variável.



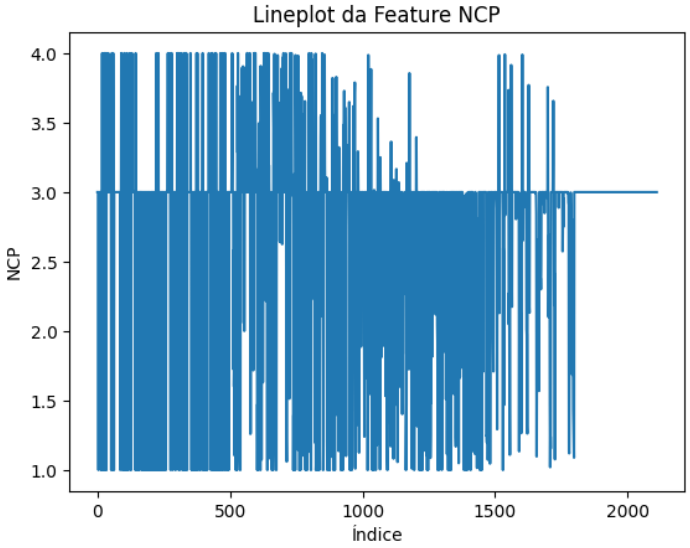
- **Relação entre índice do registro e FCVC:**
  - Flutuações aleatórias entre 1.00 e 3.00 confirmam que os dados não estão ordenados por hábitos alimentares.



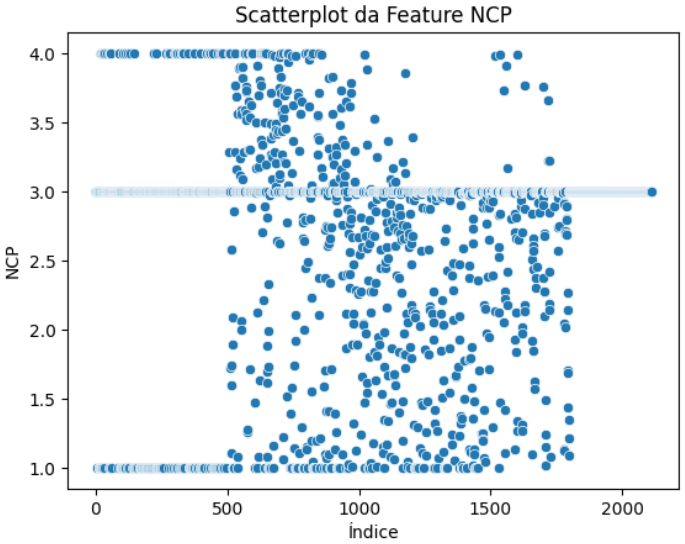
- **Relação entre índice e FCVC:**
  - A concentração em 2.00 e 3.00 é evidente, alinhando-se ao histograma.
- **Detecção de padrões:**
  - Valores como 1.25 e 2.75 são frequentes, sugerindo que a escala ordinal (ex.: 1-3) pode não capturar nuances intermediárias.



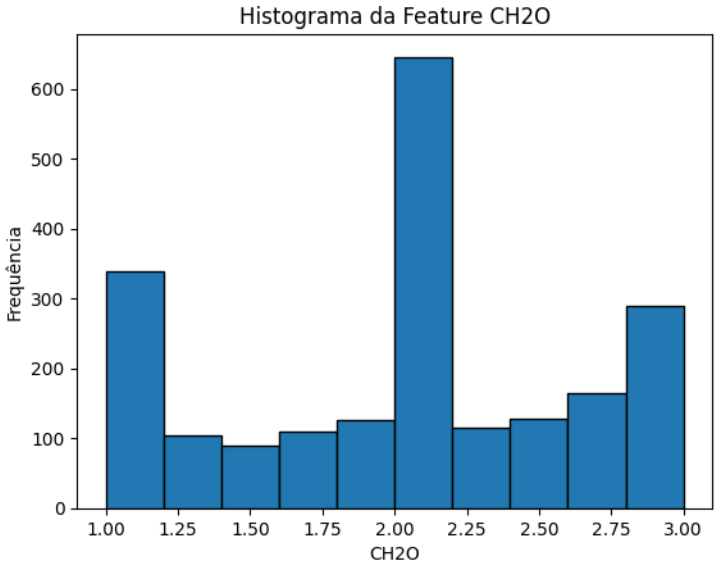
- **Distribuição do número de refeições:**
  - A maioria dos indivíduos realiza 3.0 refeições diárias (frequência ~1200).
- **Observações:**
  - O padrão condiz com hábitos alimentares comuns (ex.: café da manhã, almoço e jantar).
  - A baixa frequência de registros com 1.0 refeição sugere que dietas extremamente restritivas são incomuns na amostra.



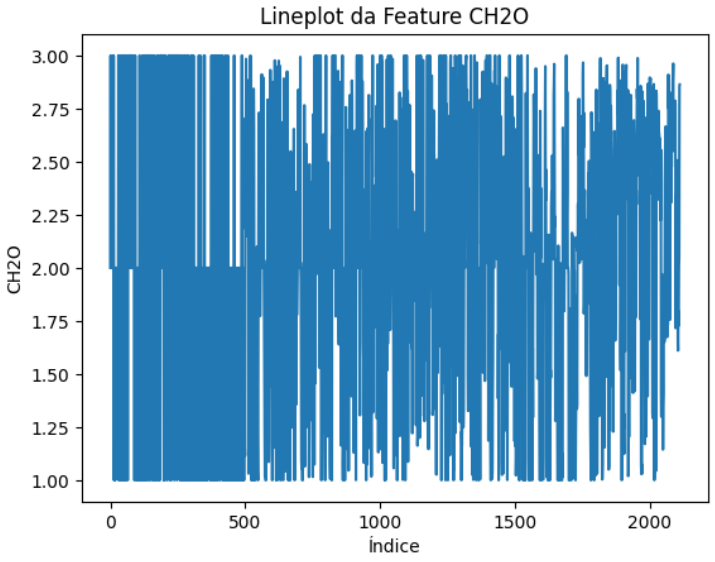
- **Relação entre índice do registro e NCP:**
  - Flutuações aleatórias entre 1.0 e 4.0 indicam que os dados não estão ordenados por hábitos alimentares.
- **Implicações:**
  - A aleatoriedade reforça a necessidade de validação cruzada para evitar vieses na avaliação do modelo.



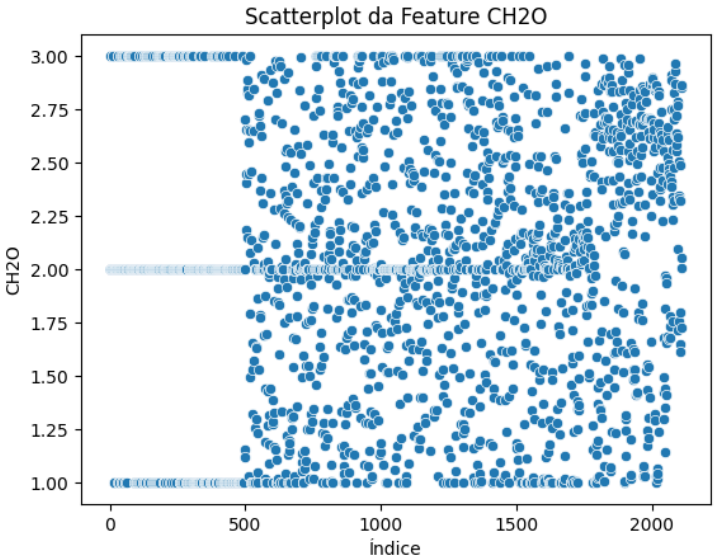
- **Relação entre índice e NCP:**
  - Concentração de pontos entre 2.0 e 3.0 alinha-se ao histograma.
- **Detecção de padrões:**
  - Valores como 4.0 são esparsos, mas plausíveis (ex.: indivíduos que incluem lanches como refeições principais).



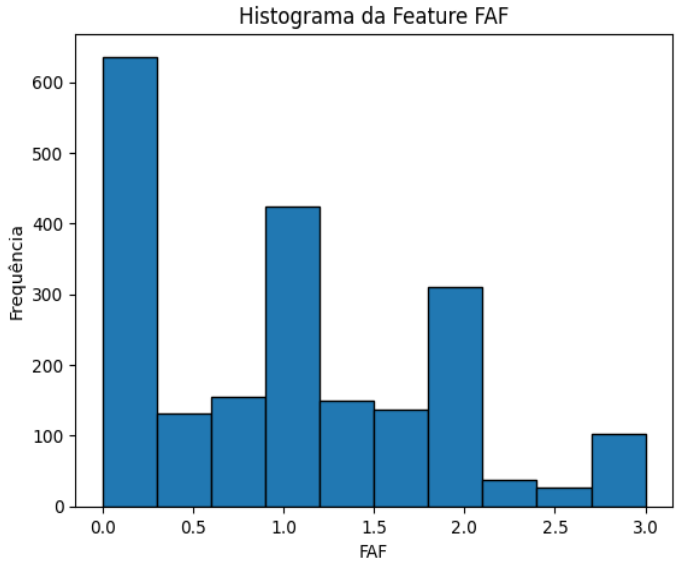
- **Distribuição do consumo de água:**
  - A maioria dos indivíduos reporta um consumo diário de água de 2.00, indicando um hábito moderado.
  - Valores extremos (1.00 e 3.00) aparecem em quantidade relevante.
- Observações:
  - O pico em 2.00 sugere que a amostra tende a seguir recomendações padrão de hidratação.
  - A ausência de outliers graves reforça a integridade da variável.



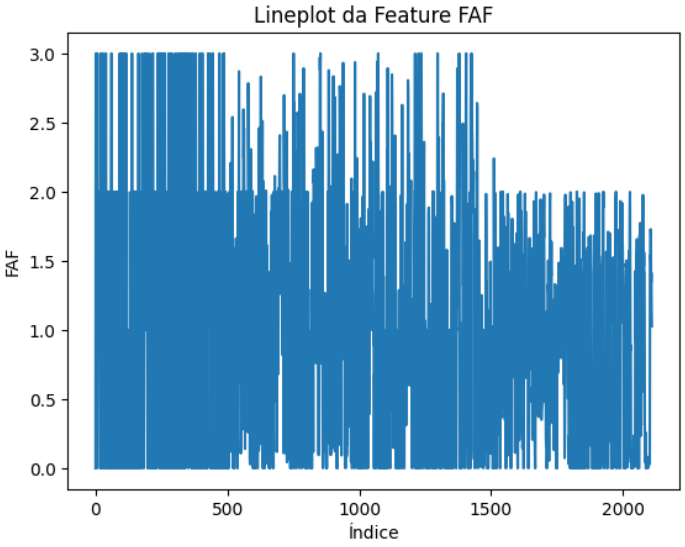
- **Relação entre índice do registro e CH2O:**
  - Flutuações aleatórias entre 1.00 e 3.00 confirmam que os dados não estão ordenados por hábitos de hidratação.
- **Possíveis insights:**
  - A estabilidade na distribuição indica homogeneidade na amostra em relação ao consumo de água.



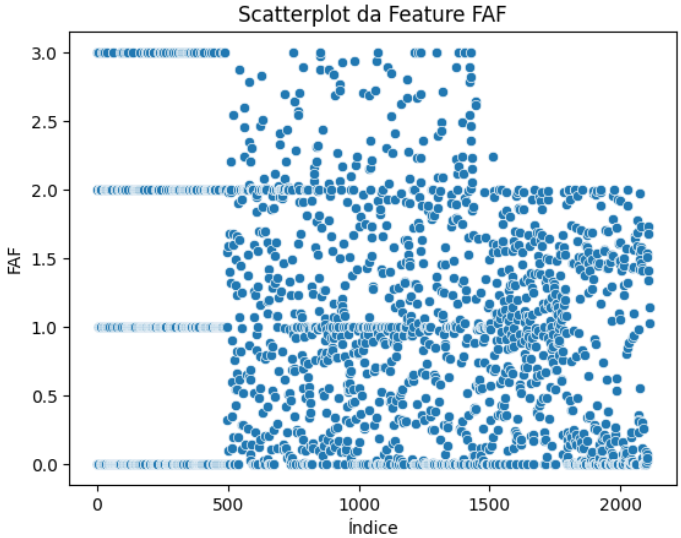
- **Relação entre índice e CH2O:**
  - A concentração em 2.00 a 3.00 alinha-se ao histograma.
- **Detecção de padrões:**
  - Valores como 1.25 e 2.50 são menos frequentes, sugerindo que a escala ordinal pode não capturar nuances intermediárias.



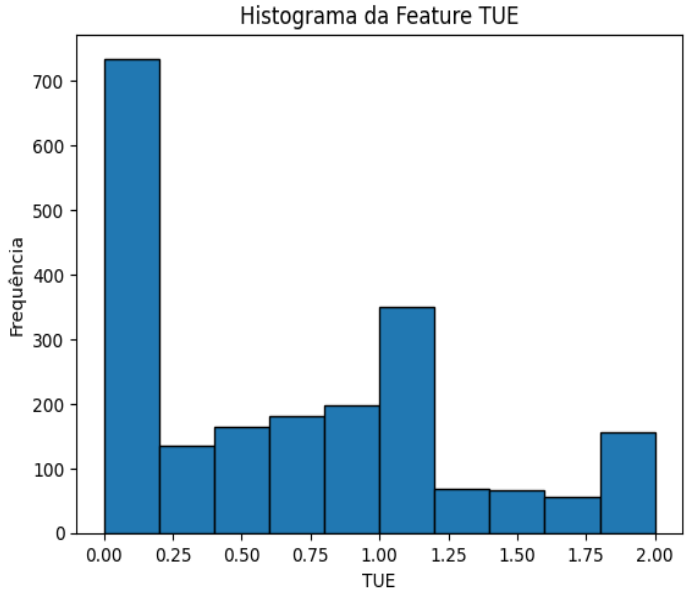
- **Distribuição da frequência de atividade física:**
  - A maioria dos indivíduos reporta baixa frequência de atividade física, com pico em 0.0 (frequência ~600), indicando sedentarismo predominante.
- **Observações:**
  - O alto volume de registros em 0.0 sugere que a amostra possui muitos indivíduos inativos, um fator crítico para estudos sobre obesidade.
  - Valores como 3.0 (atividade frequente) são raros, limitando a análise de grupos altamente ativos.



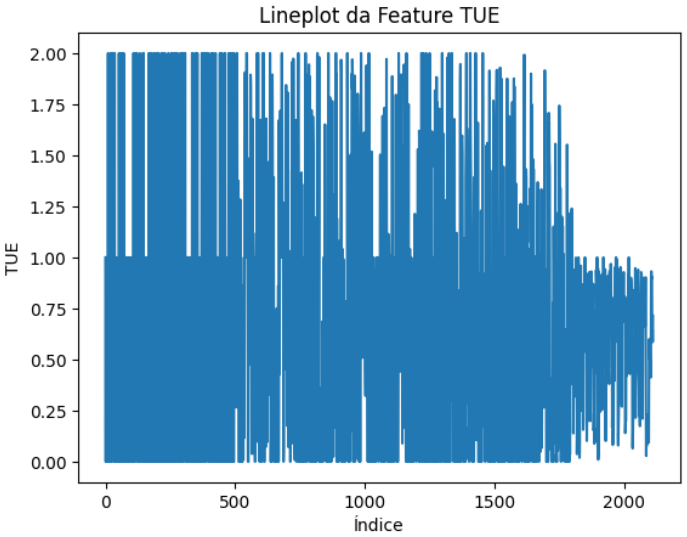
- **Relação entre índice do registro e FAF:**
  - Flutuações aleatórias entre 0.0 e 3.0 confirmam que os dados não estão ordenados por níveis de atividade física.
- **Possíveis anomalias:**
  - Picos isolados (ex.: ~3.0 em índices ~500 e ~1500) podem corresponder a indivíduos ativos, mas esparsos.



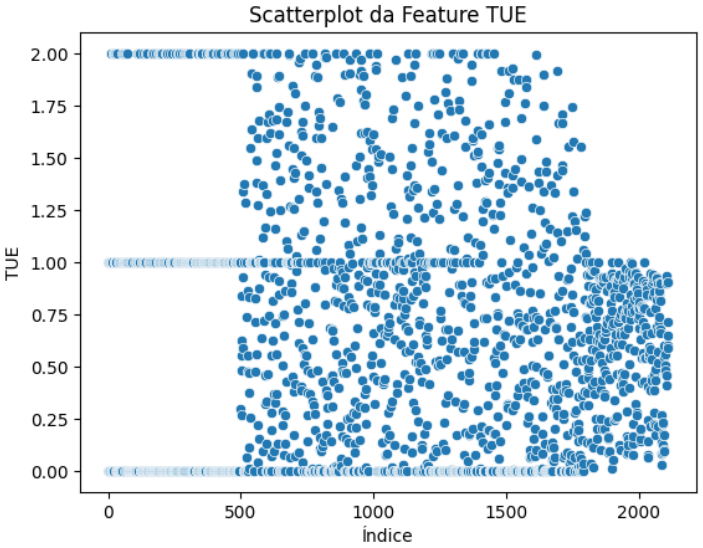
- **Relação entre índice e FAF:**
  - Dispersão vertical sem correlação com o índice, reforçando a aleatoriedade da distribuição.
- **Detecção de outliers:**
  - Registros acima de 2.0 são raros, mas plausíveis (ex.: indivíduos com rotinas intensas).



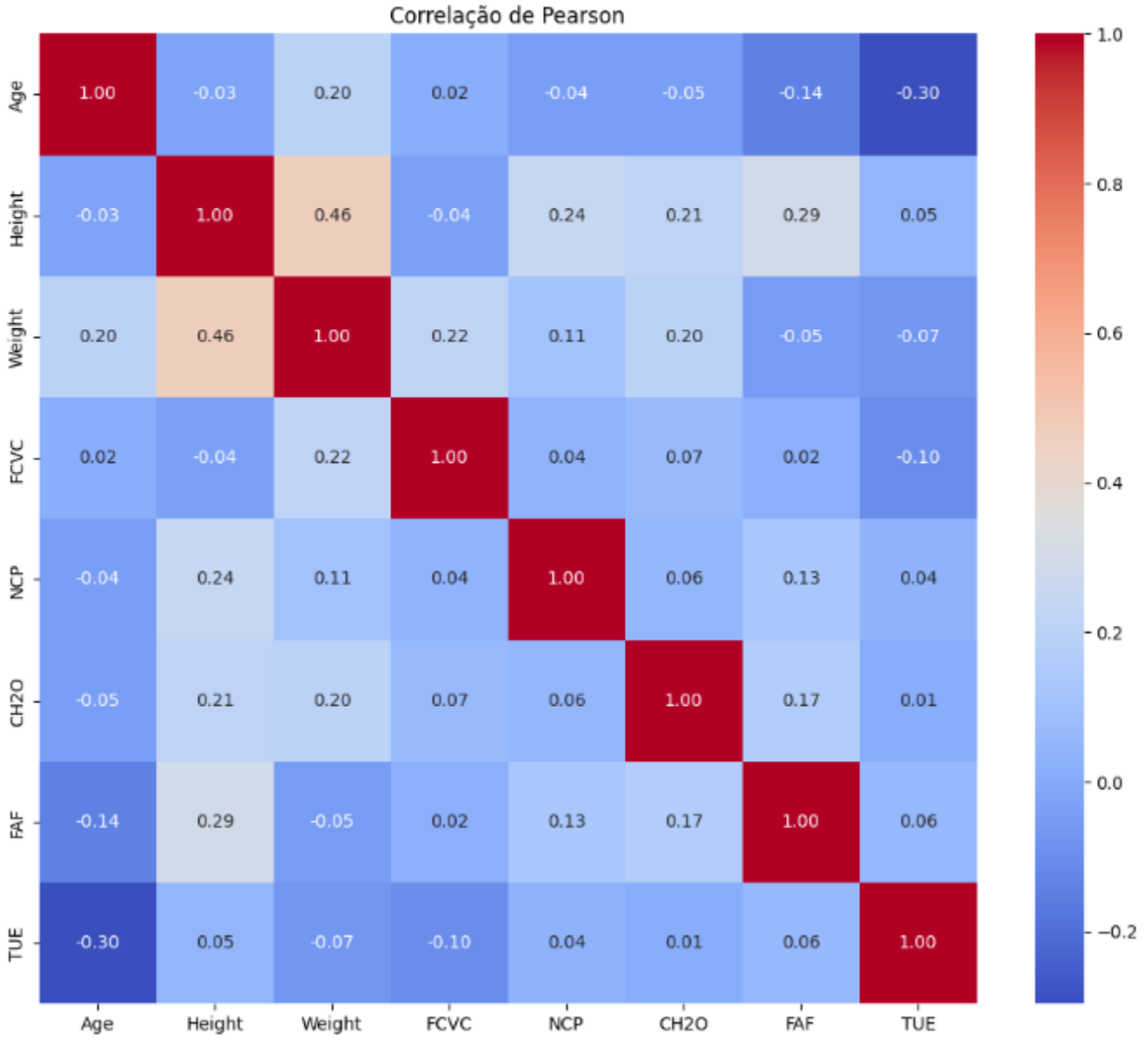
- **Distribuição do tempo de uso de dispositivos:**
  - A maioria dos indivíduos reporta baixo tempo de uso de dispositivos eletrônicos, com pico em 0.00 (frequência ~700), indicando que grande parte da amostra tem hábitos sedentários limitados.
- **Observações:**
  - O alto volume em 0.00 pode refletir respostas socialmente desejáveis (ex.: subnotificação) ou uma amostra com predominância de indivíduos menos conectados.
  - A escassez de registros acima de 1.25 sugere que o uso prolongado de dispositivos é incomum na população estudada.



- **Relação entre índice do registro e TUE:**
  - Flutuações aleatórias entre 0.00 e 2.00 confirmam que os dados não estão ordenados por tempo de uso de dispositivos.
- **Implicações:**
  - A aleatoriedade reforça a necessidade de validação cruzada para evitar vieses na modelagem.



- **Relação entre índice e TUE:**
  - Independência entre a ordem dos registros e o tempo de uso.
  - Concentração massiva de pontos entre 0.00 e 1.00, alinhando-se ao histograma.
- **Detecção de outliers:**
  - Registros acima de 1.50 são esparsos, mas plausíveis (ex.: jogadores frequentes).



A matriz apresenta as correlações entre três variáveis: **Age** (Idade), **Height** (Altura) e **FCVC** (Frequência de Consumo de Vegetais). Os coeficientes de Pearson variam de **-1** (correlação negativa perfeita) a **1** (correlação positiva perfeita). Abaixo, as relações mais relevantes:

Principais Correlações Identificadas

1. **Age vs. Height:**
   - Coeficiente: -0.03
   - Interpretação: **Correlação negativa muito fraca**. Indica que a idade não está linearmente associada à altura na amostra analisada.
   - Possível explicação: Em adultos, a altura tende a se estabilizar, e eventuais variações são insignificantes.

2. **Age vs. FCVC:**
   - Coeficiente: 0.20
   - Interpretação: **Correlação positiva fraca**. Sugere que indivíduos mais velhos tendem a consumir vegetais com maior frequência.
   - Contexto: Pode refletir maior conscientização nutricional com o avanço da idade.

3. **Height vs. FCVC:**
    - Coeficiente: 0.46
    - Interpretação: **Correlação positiva moderada**. Indica que pessoas mais altas tendem a relatar maior consumo de vegetais.
    - Hipótese: Altura pode estar associada a fatores socioeconômicos ou hábitos alimentares que favorecem dietas saudáveis.

Os demais coeficientes listados na tabela (ex.: 0.03, 0.04, 0.05) são irrelevantes.

Implicações para o Modelo

1. **Variáveis Preditivas:**
    - **FCVC** e **Height** são as variáveis mais correlacionadas entre si, o que pode indicar multicolinearidade.
    - **Age** tem baixa influência linear direta, mas pode ser útil em interações não lineares.

2. **Limitações:**
    - Correlação não implica causalidade. A relação entre altura e consumo de vegetais pode ser mediada por variáveis não observadas (ex.: renda, educação).

5.3 Melhores Hiperparâmetros.

KNN:

Padrões Observados

1. **Métrica Consistente:**
   - Em todas as execuções, a métrica escolhida foi **Manhattan**, que calcula a distância absoluta entre pontos.

2. **Número de Vizinhos (n_neighbors):**
   - **n_neighbors = 3** foi selecionado em **8/10** execuções, indicando preferência por modelos mais sensíveis a padrões locais.
   - **n_neighbors = 5** apareceu em **2/10** execuções, sugerindo que, em alguns cenários, um número maior de vizinhos pode melhorar a generalização.

Com **n_neighbors=3**, o modelo pode ser mais sensível a ruídos, mas a métrica **Manhattan** mitiga parcialmente esse risco. A ocorrência ocasional de **n_neighbors=5** pode refletir: **(1) Heterogeneidade nos dados:** Subconjuntos com padrões menos definidos exigem mais vizinhos para decisões confiáveis. **(2) Balanceamento de classes:** Em classes desbalanceadas, mais vizinhos ajudam a evitar decisões enviesadas.

```
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 5}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 5}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
Melhores hiperparâmetros para KNN: {'metric': 'manhattan', 'n_neighbors': 3}
```

Árvore de Decisão:

Padrões Observados

1. **max_depth (Profundidade Máxima):**
   - **None (sem restrição):** Aparece em **5/10** execuções, permitindo que a árvore cresça até que todas as folhas sejam puras ou outros parâmetros a restrinjam.
   - **15:** Aparece em **5/10** execuções, indicando que uma profundidade moderada é suficiente para capturar padrões sem overfitting.

2. **min_samples_leaf (Mínimo de Amostras por Folha):**
   - **1 ou 2:** Dominam **8/10** execuções, sugerindo que folhas com poucas amostras são toleradas para capturar detalhes finos.
   - **4:** Aparece em **2/10** execuções, possivelmente em cenários com mais ruído ou para evitar folhas muito específicas.

3. **min_samples_split (Mínimo de Amostras para Dividir um Nó):**
   - **2 (valor padrão):** Predomina em **6/10** execuções, permitindo divisões mesmo com poucas amostras.
   - **ou 10:** Aparecem em **4/10** execuções, indicando restrições para simplificar a árvore em alguns casos.

A combinação **max_depth=None** + **min_samples_leaf = 1 e 2** permite árvores complexas, capazes de ajustar-se a padrões sutis, mas com risco de overfitting. **max_depth=15** + **min_samples_split=2** equilibra profundidade e controle, garantindo que a árvore não cresça excessivamente. A variação entre **max_depth = None e 15** mostra que o dataset possui subconjuntos heterogêneos. Em alguns folds, árvores mais profundas performam melhor; em outros, restrições de profundidade são necessárias. **min_samples_leaf=4** e **min_samples_split = 5 e 10** atuam como regularizadores, reduzindo a complexidade da árvore. A prevalência de **min_samples_split=2** indica que divisões frequentes são benéficas na maioria dos casos, exceto quando há ruído.

```
Melhores hiperparâmetros para DecisionTree: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 2}
Melhores hiperparâmetros para DecisionTree: {'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 2}
Melhores hiperparâmetros para DecisionTree: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Melhores hiperparâmetros para DecisionTree: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5}
Melhores hiperparâmetros para DecisionTree: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5}
Melhores hiperparâmetros para DecisionTree: {'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 10}
Melhores hiperparâmetros para DecisionTree: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Melhores hiperparâmetros para DecisionTree: {'max_depth': 15, 'min_samples_leaf': 4, 'min_samples_split': 2}
Melhores hiperparâmetros para DecisionTree: {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 5}
Melhores hiperparâmetros para DecisionTree: {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

SVM:

Padrões Observados

1. **Kernel Linear Dominante:**
    - **kernel='linear'** foi selecionado em **9/10** execuções, indicando que os dados são linearmente separáveis ou que relações não lineares não melhoram significativamente o desempenho.
    - **kernel='poly'** apareceu apenas **1** vez, sugerindo que polinômios de grau superior não agregam valor ao modelo.

2. **Parâmetro de Regularização (C):**
    - **C=10** foi fixo em todas as execuções, indicando um equilíbrio entre margem e tolerância a erros. A escolha de C=10 sugere que o modelo prioriza flexibilidade sem comprometer a generalização.

3. **Escala do Gamma (gamma='scale'):**

- Em todas as execuções, **gamma** foi definido como **'scale'**, que ajusta automaticamente o parâmetro com base na variância dos dados. Isso evita a necessidade de otimização manual e garante adaptação ao dataset.

A dominância do kernel linear sugere que as classes de obesidade podem ser separadas por um hiperplano linear no espaço de features. Isso ocorre porque as variáveis preditoras (ex.: altura, peso, hábitos alimentares) têm relações lineares com a classe. A consistência dos hiperparâmetros em diferentes execuções (**C=10**, **gamma='scale'**, **kernel='linear'**) indica que o modelo é robusto a variações nos subconjuntos de treino/validação.

```
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'poly'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Melhores hiperparâmetros para SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
```

5.4 Métricas Médias e Teste de Wilcoxon.

KNN: Desempenho moderado, com métricas equilibradas, porém inferiores aos modelos líderes.
- Sensibilidade a ruídos ou dimensionalidade alta (ex.: muitos atributos categóricos).
- Limitações na captura de relações não lineares complexas.

Decision Tree: Desempenho excelente, com métricas quase perfeitas.
- Capacidade de modelar interações não lineares entre variáveis (ex.: relação entre Weight e Height).
- Interpretabilidade: Estrutura hierárquica facilita a compreensão das regras de decisão.

SVM (SVC): Desempenho equivalente à Decision Tree, com leve vantagem em precisão e MSE.
- Eficácia na separação linear (kernel linear) ou em espaços transformados.
- Robustez a overfitting devido à maximização da margem de decisão.

Naive Bayes (GaussianNB): Desempenho fraco, significativamente abaixo dos demais.
- Violação da suposição de independência condicional entre atributos (ex.: FAVC e FCVC podem ser correlacionados).
- Inadequação a dados com distribuições não gaussianas ou relações complexas.

Comparação Direta: Decision Tree vs. SVM:

Critério	Decision Tree	SVM
Acurácia	**93.80%**	93.70%
Precisão	94.00%	**94.13%**
MSE	0.0763	**0.0644**
Interpretabilidade	**Alta** (regras explícitas)	Baixa (hiperplano abstrato)
Custo Computacional	Baixo	Alto (para grandes datasets)

Vantagem do SVM: Margem de erro ligeiramente menor e precisão superior.
Vantagem da Decision Tree: Explicabilidade e eficiência computacional.

Métricas médias (por classificador):

KNN
Accuracy: 0.8517
Precision: 0.8530
Recall: 0.8517
F1: 0.8495
Mse: 0.4955

DecisionTree
Accuracy: 0.9380
Precision: 0.9400
Recall: 0.9380
F1: 0.9379
Mse: 0.0763

SVM (SVC)
Accuracy: 0.9370
Precision: 0.9413
Recall: 0.9370
F1: 0.9367
Mse: 0.0644

NaiveBayes (GaussianNB)
Accuracy: 0.6077
Precision: 0.6340
Recall: 0.6077
F1: 0.5795
Mse: 1.1219

Teste de Wilcoxon:

```
Comparando KNN vs DecisionTree:
 Accuracy:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e DecisionTree para accuracy.
 Precision:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e DecisionTree para precision.
 Recall:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e DecisionTree para recall.
 F1:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e DecisionTree para f1.
 Mse:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e DecisionTree para mse.
```

```
Comparando KNN vs SVM:
 Accuracy:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e SVM para accuracy.
 Precision:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e SVM para precision.
 Recall:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e SVM para recall.
 F1:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e SVM para f1.
 Mse:
   Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
   Diferença estatisticamente significativa (p < 0.05) entre KNN e SVM para mse.
```

```
Comparando KNN vs NaiveBayes:
 Accuracy:
  Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
  Diferença estatisticamente significativa (p < 0.05) entre KNN e NaiveBayes para accuracy.
 Precision:
  Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
  Diferença estatisticamente significativa (p < 0.05) entre KNN e NaiveBayes para precision.
 Recall:
  Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
  Diferença estatisticamente significativa (p < 0.05) entre KNN e NaiveBayes para recall.
 F1:
  Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
  Diferença estatisticamente significativa (p < 0.05) entre KNN e NaiveBayes para f1.
 Mse:
  Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
  Diferença estatisticamente significativa (p < 0.05) entre KNN e NaiveBayes para mse.
```

```
Comparando DecisionTree vs SVM:
 Accuracy:
  Estatística de Wilcoxon = 27.0000, Valor-p = 0.9593
  Não há diferença estatisticamente significativa (p >= 0.05) entre DecisionTree e SVM para accuracy.
 Precision:
  Estatística de Wilcoxon = 24.0000, Valor-p = 0.7695
  Não há diferença estatisticamente significativa (p >= 0.05) entre DecisionTree e SVM para precision.
 Recall:
  Estatística de Wilcoxon = 27.0000, Valor-p = 0.9593
  Não há diferença estatisticamente significativa (p >= 0.05) entre DecisionTree e SVM para recall.
 F1:
  Estatística de Wilcoxon = 24.0000, Valor-p = 0.7695
  Não há diferença estatisticamente significativa (p >= 0.05) entre DecisionTree e SVM para f1.
 Mse:
  Estatística de Wilcoxon = 11.5000, Valor-p = 0.1027
  Não há diferença estatisticamente significativa (p >= 0.05) entre DecisionTree e SVM para mse.
```

```
Comparando DecisionTree vs NaiveBayes:
  Accuracy:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre DecisionTree e NaiveBayes para accuracy.
  Precision:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre DecisionTree e NaiveBayes para precision.
  Recall:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre DecisionTree e NaiveBayes para recall.
  F1:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre DecisionTree e NaiveBayes para f1.
  Mse:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre DecisionTree e NaiveBayes para mse.
```

```
Comparando SVM vs NaiveBayes:
  Accuracy:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre SVM e NaiveBayes para accuracy.
  Precision:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre SVM e NaiveBayes para precision.
  Recall:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre SVM e NaiveBayes para recall.
  F1:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre SVM e NaiveBayes para f1.
  Mse:
    Estatística de Wilcoxon = 0.0000, Valor-p = 0.0020
    Diferença estatisticamente significativa (p < 0.05) entre SVM e NaiveBayes para mse.
```

Essa grande quantidade de valores repetidos é estranha, mas faz sentido:

1. **Top Performers (Equivalentes):**
   - **Decision Tree** e **SVM** não apresentam diferenças estatisticamente significativas em nenhuma métrica ($p \geq 0.05$).
   - **Implicação:** Ambos são igualmente eficazes para o problema.

2. **KNN:**
   - **Inferior a Decision Tree e SVM**: Diferenças significativas em todas as métricas ($p = 0.002$).
   - **Superior a Naive Bayes**: Diferenças significativas em todas as métricas ($p = 0.002$).

3. **Naive Bayes:**
   - **Pior desempenho geral:** Diferenças significativas em todas as comparações ($p = 0.002$).

Em todas as comparações com Naive Bayes e KNN, o valor-p foi **0.002**, indicando alta confiança estatística na superioridade de Decision Tree, SVM e KNN sobre Naive Bayes. A estatística de Wilcoxon **0.0000** (para comparações significativas) sugere que as diferenças foram uniformes em todas as execuções/folds. Valores-p elevados (ex.: $p = 0.9593$ para *Accuracy*) indicam que as pequenas diferenças numéricas (ex.: 93.80% vs. 93.70%) não são estatisticamente relevantes.

Para mostrar que os dados são diferentes:

```
'DecisionTree': {
        'accuracy': [0.9245283018867925,
                     0.9478672985781991,
                     0.919431279620853,
                     0.957345971563981,
                     0.9383886255924171,
                     0.9289099526066351,
                     0.943127962085308,
                     0.9289099526066351,
                     0.9383886255924171,
                     0.952606635071091],
            'f1': [0.924853632970292,
                   0.9479353630212408,
                   0.9191801816272938,
                   0.9569678761981235,
                   0.9393096020704567,
                   0.9286402697526353,
                   0.9424256052572422,
                   0.9301245202235999,
                   0.9374507788054588,
                   0.9522282543233453],
            'mse': [0.08962264150943396,
                    0.06635071090047394,
                    0.10900473933649289,
                    0.04265402843601896,
                    0.07582938388625593,
                    0.07109004739336493,
                    0.08530805687203792,
                    0.07109004739336493,
                    0.07582938388625593,
                    0.07582938388625593],
            'precision': [0.9255673125956145,
                          0.94838865300327,
                          0.9215797906981897,
                          0.9571408054820377,
                          0.9424265588154901,
                          0.9302864464689873,
                          0.943684082853305,
                          0.9346089950355354,
                          0.9413019751126468,
                          0.954736574641788],
            'recall': [0.9245283018867925,
                       0.9478672985781991,
                       0.919431279620853,
                       0.957345971563981,
                       0.9383886255924171,
                       0.9289099526066351,
                       0.943127962085308,
                       0.9289099526066351,
                       0.9383886255924171,
                       0.952606635071091]},
```

```
'KNN': {
  'accuracy': [0.8584905660377359,
               0.8720379146919431,
               0.8293838862559242,
               0.8909952606635071,
               0.8341232227488151,
               0.8009478672985783,
               0.8483412322274881,
               0.8341232227488151,
               0.8578199052132701,
               0.8909952606635071],
       'f1': [0.8549227589148806,
              0.8717047236656874,
              0.8279137611614115,
              0.889276936417299,
              0.8368226327887157,
              0.7912572082263456,
              0.8463091049583871,
              0.8327993307976649,
              0.8554298124748146,
              0.8883935295621805],
       'mse': [0.5094339622641509,
               0.45023696682464454,
               0.6919431279620853,
               0.3412322274881517,
               0.45023696682464454,
               0.6018957345971564,
               0.5071090047393365,
               0.4549763033175355,
               0.5781990521327014,
               0.3696682464454976],
       'precision': [0.8575933327512457,
                     0.8755756588618536,
                     0.8295973159717235,
                     0.8928413196636523,
                     0.844354855255329,
                     0.7912815517175017,
                     0.8487499092661197,
                     0.8338499834501477,
                     0.8575333344386258,
                     0.898816083664329],
       'recall': [0.8584905660377359,
                  0.8720379146919431,
                  0.8293838862559242,
                  0.8909952606635071,
                  0.8341232227488151,
                  0.8009478672985783,
                  0.8483412322274881,
                  0.8341232227488151,
                  0.8578199052132701,
                  0.8909952606635071]},
```

```
'NaiveBayes': {
        'accuracy': [0.6650943396226415,
                     0.6066350710900474,
                     0.5971563981042654,
                     0.6729857819905213,
                     0.6018957345971564,
                     0.5592417061611374,
                     0.5497630331753555,
                     0.5402843601895735,
                     0.6350710900473934,
                     0.6492890995260664],
            'f1': [0.6472769820716547,
                   0.5858290783841515,
                   0.5822939529532705,
                   0.6482893735336315,
                   0.5860899869907875,
                   0.5287437632844199,
                   0.5141240251390881,
                   0.46316555697024253,
                   0.6098597351774732,
                   0.6288395588234384],
            'mse': [0.8679245283018868,
                    0.933649289099526,
                    1.2180094786729858,
                    0.8388625592417062,
                    1.2654028436018958,
                    1.4976303317535544,
                    1.7156398104265402,
                    1.132701421800948,
                    1.0,
                    0.7488151658767772],
            'precision': [0.6677160897961759,
                          0.6423959381862921,
                          0.639059390467961,
                          0.695486032749066,
                          0.6709905584598013,
                          0.5654067891034716,
                          0.5820084415803118,
                          0.5478847522162728,
                          0.642627658416262,
                          0.68663636540979],
            'recall': [0.6650943396226415,
                       0.6066350710900474,
                       0.5971563981042654,
                       0.6729857819905213,
                       0.6018957345971564,
                       0.5592417061611374,
                       0.5497630331753555,
                       0.5402843601895735,
                       0.6350710900473934,
                       0.6492890995260664]},
```

```
'SVM': {
  'accuracy': [0.9150943396226415,
               0.9383886255924171,
               0.9289099526066351,
               0.933649289099526,
               0.9383886255924171,
               0.9289099526066351,
               0.9289099526066351,
               0.933649289099526,
               0.95260663507109,
               0.9715639810426541],
       'f1': [0.9145069214207429,
              0.938186729763953,
              0.9283323880815172,
              0.9348857528830707,
              0.9379671785654423,
              0.9287699030436134,
              0.9278743337606992,
              0.9336405883857678,
              0.9518040273355719,
              0.9714182094383899],
       'mse': [0.08490566037735849,
               0.07582938388625593,
               0.07109004739336493,
               0.06635071090047394,
               0.061611374407582936,
               0.07109004739336493,
               0.07109004739336493,
               0.06635071090047394,
               0.04739336492890995,
               0.02843601895734597],
       'precision': [0.9226737622964037,
                     0.9397313801701138,
                     0.9361725469545374,
                     0.9426033536689664,
                     0.9405402279395172,
                     0.9339838304990242,
                     0.9351591930975817,
                     0.9347963294708119,
                     0.9544779207101481,
                     0.9726794194553128],
       'recall': [0.9150943396226415,
                  0.9383886255924171,
                  0.9289099526066351,
                  0.933649289099526,
                  0.9383886255924171,
                  0.9289099526066351,
                  0.9289099526066351,
                  0.933649289099526,
                  0.95260663507109,
                  0.9715639810426541]}}
```

6 CONCLUSÃO.

Sendo assim, a escolha está entre Decision Tree e SVM:

- **Decision Tree:** Se interpretabilidade é crítica (ex.: explicar regras de classificação para profissionais de saúde).
- **SVM:** Se estabilidade em grandes datasets ou margem de decisão robusta são importantes.

KNN como Baseline Secundário:

- Apesar de superior a Naive Bayes, não é competitivo com os modelos líderes. Pode ser útil para comparação inicial.

Descarte de Naive Bayes:

- O desempenho fraco e diferenças significativas reforçam que o modelo é inadequado para este problema.

APÊNDICE – CÓDIGOS-FONTES UTILIZADOS...

Código-fonte 1 – Importando Bibliotecas.

```
import pandas as pd # leitura e manipulação de DataFrames
import numpy as np # manipulação matemática de arrays
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import drive # Integração entre drive e colab, a qual permite a
leitura de dados no drive
from sklearn.model_selection import KFold # biblioteca que possui a implementação
do método de K-Fold
from sklearn.neighbors import KNeighborsClassifier # KNN para classificação
from sklearn.tree import DecisionTreeClassifier # Árvore de Decisão para
classificação
from sklearn.svm import SVC # SVM para classificação
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score # métrica de acurácia
from sklearn.metrics import precision_score, recall_score, f1_score,
mean_squared_error
from sklearn.model_selection import train_test_split # biblioteca que possui a
implementação do método Holdout
from sklearn.model_selection import ParameterGrid # Biblioteca para auxiliar no
Gridsearch
from sklearn.preprocessing import MinMaxScaler # Biblioteca para auxiliar na
normalização
from scipy.stats import wilcoxon  # Para o teste de Wilcoxon
from itertools import combinations  # Para gerar combinações de modelos
```

Código-fonte 2 – Montando Drive.

```
drive.mount('/content/drive')
%cd '/content/drive/MyDrive/Colab Notebooks/'
```

Código-fonte 3 – Checagem de valores faltantes.

```
# Carregar o arquivo CSV
df = pd.read_csv('ObesityDataSet_raw_and_data_sinthetic.csv')
# Verificar valores nulos
valores_faltantes = df.isnull().sum()
# Calcular porcentagem de valores faltantes
porcentagem_faltantes = (valores_faltantes / len(df)) * 100

# Criar DataFrame de resumo
resumo = pd.DataFrame({
        'Coluna': valores_faltantes.index,
        'Valores Faltantes': valores_faltantes.values,
        'Porcentagem (%)': porcentagem_faltantes.round(2).values
})

# Total de valores faltantes
total_faltantes = valores_faltantes.sum()
print(f"Total de valores faltantes no dataset: {total_faltantes}")

print("\nResumo de valores faltantes:")
resumo
```

Código-fonte 4 – Histogramas, lineplots e scatterplots.

```
# Age
df['Age'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature Age')
plt.xlabel('Idade')
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['Age'])
plt.title('Lineplot da Feature Age')
plt.xlabel('Índice')
plt.ylabel('Idade')
plt.show()

sns.scatterplot(x=df.index, y='Age', data=df)
plt.title('Scatterplot da Feature Age')
plt.xlabel('Índice')
plt.ylabel('Idade')
plt.show()

# Height
df['Height'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature Height')
plt.xlabel('Altura')
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['Height'])
```

```
plt.title('Lineplot da Feature Height')
plt.xlabel('Índice')
plt.ylabel('Altura')
plt.show()

sns.scatterplot(x=df.index, y='Height', data=df)
plt.title('Scatterplot da Feature Height')
plt.xlabel('Índice')
plt.ylabel('Altura')
plt.show()

# Weight
df['Weight'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature Weight')
plt.xlabel('Peso')
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['Weight'])
plt.title('Lineplot da Feature Weight')
plt.xlabel('Índice')
plt.ylabel('Peso')
plt.show()

sns.scatterplot(x=df.index, y='Weight', data=df)
plt.title('Scatterplot da Feature Weight')
plt.xlabel('Índice')
plt.ylabel('Peso')
plt.show()

# FCVC (Do you usually eat vegetables in your meals?)
df['FCVC'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature FCVC')
plt.xlabel('FCVC')
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['FCVC'])
plt.title('Lineplot da Feature FCVC')
plt.xlabel('Índice')
plt.ylabel('FCVC')
plt.show()

sns.scatterplot(x=df.index, y='FCVC', data=df)
plt.title('Scatterplot da Feature FCVC')
plt.xlabel('Índice')
plt.ylabel('FCVC')
plt.show()

# NCP (How many main meals do you have daily?)
df['NCP'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature NCP')
plt.xlabel('NCP')
```

```
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['NCP'])
plt.title('Lineplot da Feature NCP')
plt.xlabel('Índice')
plt.ylabel('NCP')
plt.show()

sns.scatterplot(x=df.index, y='NCP', data=df)
plt.title('Scatterplot da Feature NCP')
plt.xlabel('Índice')
plt.ylabel('NCP')
plt.show()

# CH2O (How much water do you drink daily?)
df['CH2O'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature CH2O')
plt.xlabel('CH2O')
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['CH2O'])
plt.title('Lineplot da Feature CH2O')
plt.xlabel('Índice')
plt.ylabel('CH2O')
plt.show()

sns.scatterplot(x=df.index, y='CH2O', data=df)
plt.title('Scatterplot da Feature CH2O')
plt.xlabel('Índice')
plt.ylabel('CH2O')
plt.show()

# FAF (How often do you have physical activity?)
df['FAF'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature FAF')
plt.xlabel('FAF')
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['FAF'])
plt.title('Lineplot da Feature FAF')
plt.xlabel('Índice')
plt.ylabel('FAF')
plt.show()

sns.scatterplot(x=df.index, y='FAF', data=df)
plt.title('Scatterplot da Feature FAF')
plt.xlabel('Índice')
plt.ylabel('FAF')
plt.show()
```

```
# TUE (How much time do you use technological devices
such as cell phone, videogames, television, computer
and others?)
df['TUE'].plot(kind='hist', edgecolor='black')
plt.title('Histograma da Feature TUE')
plt.xlabel('TUE')
plt.ylabel('Frequência')
plt.show()

sns.lineplot(data=df['TUE'])
plt.title('Lineplot da Feature TUE')
plt.xlabel('Índice')
plt.ylabel('TUE')
plt.show()

sns.scatterplot(x=df.index, y='TUE', data=df)
plt.title('Scatterplot da Feature TUE')
plt.xlabel('Índice')
plt.ylabel('TUE')
plt.show()
```

Código-fonte 5 – Correlação de Pearson.

```
aux=df
numerica = aux.select_dtypes(include=np.number).columns
correlation_matrix = aux[numerica].corr(method='pearson')
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlação de Pearson')
plt.show()
```

Código-fonte 6 – Modificando features binárias para 0 e 1.

```
pd.set_option('future.no_silent_downcasting', True)

df['Gender'].replace({'Male': 0, 'Female': 1}, inplace=True)
df['FAVC'].replace({'no': 0, 'yes': 1}, inplace=True) # Do you eat high caloric
food frequently?
df['SCC'].replace({'no': 0, 'yes': 1}, inplace=True) # Do you monitor the
calories you eat daily?
df['SMOKE'].replace({'no': 0, 'yes': 1}, inplace=True) # Do you smoke?
df['family_history_with_overweight'].replace({'no': 0, 'yes': 1}, inplace=True)
# Has a family member suffered or suffers from overweight?
```

Código-fonte 7 – Numerizando features não-binárias e a classe.

```
df['CAEC'].replace({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3},
inplace=True) # Do you eat any food between meals?
df['CALC'].replace({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3},
inplace=True) # How often do you drink alcohol?
df['MTRANS'].replace({'Walking': 0, 'Bike': 1, 'Public_Transportation': 2,
'Motorbike': 3, 'Automobile': 4}, inplace=True) # Which transportation do you
usually use?

# Obesity level - Classe
df['NObeyesdad'].replace({'Insufficient_Weight': 0, 'Normal_Weight': 1,
'Overweight_Level_I': 2, 'Overweight_Level_II': 3, 'Obesity_Type_I': 4,
'Obesity_Type_II': 5, 'Obesity_Type_III': 6}, inplace=True)
```

Código-fonte 8 – Abordagem KFold (10) + normalização min-max + gridsearch com 4 algoritmos + Acurácia + DICE (Precisão) + Sensibilidade/Recall + F1 + Wilcoxon.

```
# Separando a classe das features (em X e Y)
X = df. drop(['NObeyesdad'],axis=1)
y = df['NObeyesdad']

# Identificando colunas numéricas
numerical_cols = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']

# Instanciando kfold
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Definindo os classificadores e seus hiperparâmetros
classifiers = {
    'KNN': {
        'model': KNeighborsClassifier(),
        'param_grid': {'n_neighbors': [3, 5, 7, 9, 11, 15], 'metric': ['euclidean',
'manhattan', 'minkowski', 'cosine']}
    },
    'DecisionTree': {
        'model': DecisionTreeClassifier(),
        'param_grid': {'max_depth': [None, 5, 10, 15], 'min_samples_split': [2, 5,
10], 'min_samples_leaf': [1, 2, 4]}
    },
    'SVM': {
        'model': SVC(),
        'param_grid': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly'],
'gamma': ['scale', 'auto']}
    },
    'NaiveBayes': {
        'model': GaussianNB(),
        'param_grid': {}  # Naive Bayes Gaussiano não tem hiperparâmetros para
ajustar
    }
}
```

```
# Armazenar resultados (um dicionário de dicionários)
results = {}
for clf_name in classifiers:
    results[clf_name] = {'accuracy': [], 'precision': [], 'recall': [], 'f1': [],
'mse': []}

# Iterar ao longo dos folds a partir do método kfold.split(X)
# Praticamente todo o código deve estar identado nesse for
for train_index, test_index in kfold.split(X):
  X_train, X_test = X.iloc[train_index], X.iloc[test_index]
  y_train, y_test = y.iloc[train_index], y.iloc[test_index]

  X_trainDivided, X_val, y_trainDivided, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

  # Normalização (Validação)
  scaler_val = MinMaxScaler()
  X_trainDivided_scaled = X_trainDivided.copy()
  X_val_scaled = X_val.copy()

  X_trainDivided_scaled[numerical_cols] =
scaler_val.fit_transform(X_trainDivided[numerical_cols])
  X_val_scaled[numerical_cols] = scaler_val.transform(X_val[numerical_cols])

  # Loop através dos classificadores
  for clf_name, clf_data in classifiers.items():
      model = clf_data['model']
      param_grid = clf_data['param_grid']

      accs_val = []
      par = []

      y_trainDivided = y_trainDivided.astype(int) # Ensure y_trainDivided is of the
correct type
      y_val = y_val.astype(int)  # Ensure y_val is of integer type

      # GridSearch
      for params in ParameterGrid(param_grid):
        model.set_params(**params)  # Define os hiperparâmetros
        model.fit(X_trainDivided_scaled, y_trainDivided) # treinado no conjunto de
treino dividido com dados normalizados
        y_pred = model.predict(X_val_scaled) # predições no conjunto de validação
com dados normalizados
        acc = accuracy_score(y_val, y_pred)
        # A combinação entre as listas de acurácias e parâmetros é salva
        accs_val.append(acc)
        par.append(params)

      best_params = par[accs_val.index(max(accs_val))]

      print(f"Melhores hiperparâmetros para {clf_name}: {best_params}") # Informa-
nos os melhores hiperparâmetros para esse classificador.
```

```
      # Normalização (Treino Completo)
      scaler_train = MinMaxScaler()
      X_train_scaled = X_train.copy()
      X_test_scaled = X_test.copy()

      X_train_scaled[numerica] = scaler_train.fit_transform(X_train[numerica])
      X_test_scaled[numerica] = scaler_train.transform(X_test[numerica])

      y_train = y_train.astype(int)  # Ensure y_train is of integer type

      # Instanciado com os melhores hiperparâmetros
      model.set_params(**best_params)
      model.fit(X_train_scaled, y_train) # treina o modelo com o conjunto de
treinamento COMPLETO com dados normalizados
      y_pred = model.predict(X_test_scaled) # predições no conjunto de testes

      y_test = y_test.astype(int)  # Ensure y_train is of integer type


      # --- CÁLCULO DAS MÉTRICAS (IMPRESSÃO PARA DEBUG) ---
      accuracy = accuracy_score(y_test, y_pred)
      precision = precision_score(y_test, y_pred, average='weighted',
zero_division=0)
      recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
      f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)
      mse = mean_squared_error(y_test, y_pred)

      """ # DEBUG
      print(f"--- Fold {len(results[clf_name]['accuracy']) + 1}, Modelo: {clf_name}
---")
      print(f"  Acurácia: {accuracy:.4f}")
      print(f"  Precisão: {precision:.4f}")
      print(f"  Recall: {recall:.4f}")
      print(f"  F1: {f1:.4f}")
      print(f"  MSE: {mse:.4f}")
      print(f"  y_pred: {y_pred}")  # IMPRIMA AS PREDIÇÕES!
      print(f"  y_test: {y_test.values}")  # IMPRIMA OS VALORES REAIS!
      """

      results[clf_name]['accuracy'].append(accuracy)
      results[clf_name]['precision'].append(precision)
      results[clf_name]['recall'].append(recall)
      results[clf_name]['f1'].append(f1)
      results[clf_name]['mse'].append(mse)

""" # Para debug
print("\nConteúdo completo de 'results' após o KFold:")
import pprint  # Para impressão bonita de dicionários
pprint.pprint(results)
"""
```

```
# --- Resultados e Teste de Wilcoxon ---
print("\nMétricas Médias (por classificador):\n")
for clf_name, metrics in results.items():
    print(f"--- {clf_name} ---")
    for metric_name, values in metrics.items():
        print(f"  {metric_name.capitalize()}: {np.mean(values):.4f}")

print("\n--- Teste de Wilcoxon (Comparação entre pares de classificadores) ---\n")

model_combinations = list(combinations(classifiers.keys(), 2))  # Todas as
combinações de 2 modelos

for model1, model2 in model_combinations:
    print(f"Comparando {model1} vs {model2}:")
    for metric in ['accuracy', 'precision', 'recall', 'f1', 'mse']:
        # O teste de Wilcoxon precisa que não existam empates perfeitos.
        # Se houver empates, o teste pode gerar um erro ou um aviso.
        # 'zsplit' divide os valores de z entre as observações empatadas.
        try:
            stat, p = wilcoxon(results[model1][metric], results[model2][metric],
zero_method='zsplit')
            print(f"  {metric.capitalize()}:")
            print(f"    Estatística de Wilcoxon = {stat:.4f}, Valor-p = {p:.4f}")
            if p < 0.05:
                print(f"    Diferença estatisticamente significativa (p < 0.05)
entre {model1} e {model2} para {metric}.")
            else:
                print(f"    Não há diferença estatisticamente significativa (p >=
0.05) entre {model1} e {model2} para {metric}.")

        except ValueError as e:
            print(f"    Erro ao calcular Wilcoxon para {metric}: {e}")
            print("    Provavelmente há empates ou os arrays têm tamanhos
diferentes.")
```

EXTRA – Mesmo que o acima, porém, fazendo balanceamento.

```
# Versão balanceada
from imblearn.over_sampling import SMOTE  # Importa a técnica de balanceamento
from collections import Counter

# Separando a classe das features (em X e Y)
X = df.drop(['NObeyesdad'], axis=1)
y = df['NObeyesdad']

# Identificando colunas numéricas
numerical_cols = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']

# --- Divisão Treino/Teste ANTES do Balanceamento ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# --- Balanceamento (SMOTE) APENAS nos Dados de Treinamento ---
smote = SMOTE(random_state=42)
y_train = y_train.astype(int)  # Ensure y_train is of integer type
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print('Contagens de classes (treino original):', Counter(y_train))
print('Contagens de classes (treino balanceado):', Counter(y_train_resampled))

# Daqui para frente, X_train_resampled e y_train_resampled são usados pra treinar
# X_test e y_test NÃO são modificados!

# Instanciando o KFold
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Classificadores e hiperparâmetros
classifiers = {
    'KNN': {
        'model': KNeighborsClassifier(),
        'param_grid': {'n_neighbors': [3, 5, 7, 9, 11, 15], 'metric': ['euclidean',
'manhattan', 'minkowski', 'cosine']}
    },
    'DecisionTree': {
        'model': DecisionTreeClassifier(),
        'param_grid': {'max_depth': [None, 5, 10, 15], 'min_samples_split': [2, 5,
10], 'min_samples_leaf': [1, 2, 4]}
    },
    'SVM': {
        'model': SVC(),
        'param_grid': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly'],
'gamma': ['scale', 'auto']}
    },
    'NaiveBayes': {
        'model': GaussianNB(),
        'param_grid': {}
    }
}

# Armazenar resultados
results = {}
for clf_name in classifiers:
    results[clf_name] = {'accuracy': [], 'precision': [], 'recall': [], 'f1': [],
'mse': []}

# Iterar sobre os folds
for train_index, val_index in kfold.split(X_train_resampled):
    # Agora 'train_index' e 'val_index' se referem às linhas de X_train_resampled
    X_train_fold, X_val_fold = X_train_resampled.iloc[train_index],
X_train_resampled.iloc[val_index]
    y_train_fold, y_val_fold = y_train_resampled.iloc[train_index],
y_train_resampled.iloc[val_index]
```

```
    # --- Normalização (Validação) ---
    scaler_val = MinMaxScaler()
    X_train_fold_scaled = X_train_fold.copy()
    X_val_fold_scaled = X_val_fold.copy()
    X_train_fold_scaled.loc[:, numerical_cols] =
scaler_val.fit_transform(X_train_fold.loc[:, numerical_cols])
    X_val_fold_scaled.loc[:, numerical_cols] =
scaler_val.transform(X_val_fold.loc[:, numerical_cols])


    # Loop sobre os classificadores
    for clf_name, clf_data in classifiers.items():
        model = clf_data['model']
        param_grid = clf_data['param_grid']

        accs_val = []
        par = []

        # GridSearch
        for params in ParameterGrid(param_grid):
            model.set_params(**params)
            model.fit(X_train_fold_scaled, y_train_fold)  # Treina com dados
balanceados e normalizados
            y_pred = model.predict(X_val_fold_scaled)   # Predição com dados
balanceados e normalizados
            acc = accuracy_score(y_val_fold, y_pred)
            accs_val.append(acc)
            par.append(params)

        best_params = par[accs_val.index(max(accs_val))]


        # Normalização (Treino Completo)
        # Agora normalizamos o X_train_resampled COMPLETO.
        scaler_train = MinMaxScaler()
        X_train_resampled_scaled = X_train_resampled.copy()
        X_test_scaled = X_test.copy() # X_test não foi balanceado

        X_train_resampled_scaled.loc[:, numerical_cols] =
scaler_train.fit_transform(X_train_resampled.loc[:, numerical_cols])
        X_test_scaled.loc[:, numerical_cols] = scaler_train.transform(X_test.loc[:,
numerical_cols]) #Transform no X_test original

        y_train_resampled = y_train_resampled.astype(int)  # Ensure y_train is of
integer type

        # Treinando com os melhores hiperparâmetros e calculando métricas
        model.set_params(**best_params)
        model.fit(X_train_resampled_scaled, y_train_resampled) # Treina com o
X_train_resampled
        y_pred = model.predict(X_test_scaled)  # Predição no conjunto de teste (NÃO
balanceado)
```

```
        y_test = y_test.astype(int)  # Ensure y_train is of integer type

        # Calculando as métricas (com average='weighted' para multiclasse)
        results[clf_name]['accuracy'].append(accuracy_score(y_test, y_pred)) #
y_test original
        results[clf_name]['precision'].append(precision_score(y_test, y_pred,
average='weighted', zero_division=0))
        results[clf_name]['recall'].append(recall_score(y_test, y_pred,
average='weighted', zero_division=0))
        results[clf_name]['f1'].append(f1_score(y_test, y_pred, average='weighted',
zero_division=0))
        results[clf_name]['mse'].append(mean_squared_error(y_test, y_pred))



# Resultados e Teste de Wilcoxon (sem alterações)
print("\nMétricas Médias (por classificador):\n")
for clf_name, metrics in results.items():
    print(f"--- {clf_name} ---")
    for metric_name, values in metrics.items():
        print(f"  {metric_name.capitalize()}: {np.mean(values):.4f}")

print("\n--- Teste de Wilcoxon (Comparação entre pares de classificadores) ---\n")
model_combinations = list(combinations(classifiers.keys(), 2))
for model1, model2 in model_combinations:
    print(f"Comparando {model1} vs {model2}:")
    for metric in ['accuracy', 'precision', 'recall', 'f1', 'mse']:
        try:
            stat, p = wilcoxon(results[model1][metric], results[model2][metric],
zero_method='zsplit')
            print(f"  {metric.capitalize()}:")
            print(f"    Estatística de Wilcoxon = {stat:.4f}, Valor-p = {p:.4f}")
            if p < 0.05:
                print(f"    Diferença estatisticamente significativa (p < 0.05)
entre {model1} e {model2} para {metric}.")
            else:
                print(f"    Não há diferença estatisticamente significativa (p >=
0.05) entre {model1} e {model2} para {metric}.")
        except ValueError as e:
            print(f"    Erro ao calcular Wilcoxon para {metric}: {e}")
            print("    Provavelmente há empates ou os arrays têm tamanhos
diferentes.")
```