UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Ciência e Tecnologia de Sorocaba (ICTS) - Campus de Sorocaba

LUCAS YUTA AMEDA
LUIZ ADALBERTO BARBOZA JÚNIOR

RELATÓRIO DO TRABALHO DE ESTRUTURA DE DADOS

Sorocaba
2025

RESUMO

O presente relatório faz uma análise das diferentes estruturas de dados baseado no conjunto de dados ou *dataset* escolhido pela dupla cursando a matéria de Estrutura de Dados na turma A, ministrado pelo professor Doutor Leopoldo André Dutra Lusquino Filho, sendo este um trabalho da matéria. O trabalho tem como objetivo analisar o desempenho das estruturas de dados, também escolhidas pela dupla, e descobrir o impacto causado por essa estrutura, juntamente com algumas restrições. Além disso, também é objetivo obter um modelo preditivo para prever possíveis resultados, utilizando programação para gerar as informações necessárias para a análise mencionada, além de utilizar o conhecimento obtido em sala de aula, tanto teórico como prático. A escolha do *dataset* foi feita no site do Kaggle, um repositório com vários *dataset*, no qual o tema escolhido foi de ataques cardíacos. Sendo assim, o ambiente de trabalho utilizado foi o Visual Code Studios com a extensão de Python para gerar os gráficos e contas matemáticas envolvendo questões de estatística e configurar estruturas com ajuda de bibliotecas que facilitam esse processo.

Palavras-chave: *dataset*; dados; estrutura; desempenho.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o desenvolvimento contínuo de novas tecnologias, a estruturação de dados se tornou uma prática bastante importante na atualidade. Sendo assim, com sua vasta aplicação e pelo conhecimento gerado pela matéria de Estrutura de Dados, elaborou-se um trabalho de comparação de estruturas de dados e um modelo preditivo junto.

O conjunto de dados escolhido foi o de ataques cardíacos, adquirido no repositório Kaggle, em que analisou-se a implementação de estruturas de lista encadeada, tabela hash, árvores AVL, Bloom Filter, Cuckoo Hashing e Árvore K-D para inserção, busca e remoção de itens em alguma variável, bem como o uso da memória.

Para isso, criou-se as estruturas em uma pasta dedicada e importou-as em um notebook (Jupyter) para testá-las de forma sequencial e controlada, utilizando Python como linguagem para esse trabalho.

1.1 Escolha do Python

Para a realização de estruturas de dados geralmente é usado a linguagem C++, pois ela permite um controle muito grande sobre a memória utilizada o que gera alto desempenho dos programas, além de ter bibliotecas especificamente para estrutura de dados muito robustas como ‘std::vector’, ‘std::stack’ e ‘std::map’. Porém este trabalho foi realizado em Python, já que como em C++, o Python tem bibliotecas muito robustas para fazer análises e estruturas de dados como: ‘matplotib’, ‘scikit-learn’ e ‘networkx’, ademais existem muitos tutoriais de como usar Python para está finalidade por ser uma linguagem mais recente e de maior nível que C++ por isso é mais usado para aprendizagem em comparação a C++, e por ser de mais alto nível, ela também é mais ágil para prototipar e testar códigos aumentando muito a produtividade.

1.2 Objetivo

Os objetivos do trabalho são analisar o comportamento das estruturas de dados no *dataset* escolhido, realizando benchmarks e aplicando restrições de

funcionamento e criar um modelo preditivo e definir se há risco alto ou baixo de um ataque cardíaco.

1.3 Divisão de Tarefas

Lucas Yuta Ameda: Implementou as estruturas de dados clássicas (Lista Encadeada, Tabela Hash e Árvore AVL),otimizou a Lista Encadeada e fez os testes de (tempo, memória e latência) para comparar com o não otimizado, fez os benchmarks de todas as estruturas de dados (Latência, memória, estabilidade e tempo), modelagem dos modelos preditivos (DecisionTree, Random Forest, Gradient Boosting, SMOTE) junto com avaliação de features, e escrita da revisão teórica e metodologia das estruturas de dados.

Luiz Adalberto Barboza Júnior: realizou o tratamento de dados no dataset, implementou as estruturas de dados recentes (Cuckoo Hashing, KD - Tree e Bloom Filter), aplicou as 5 restrições (Compactação, Carga computacional, Latência de rede, Dados com anomalias e Encriptação nas operações), visualização gráfica das estruturas(Gráficos de estabilidade, otimizada X original), e realização dos benchmarks dos modelos preditivos

2 REVISÃO TEÓRICA

O desenvolvimento do trabalho demanda uma certa gama de conhecimentos sobre algumas teorias e elementos fundamentais para o entendimento do problema ou da solução. Isto é, o funcionamento das estruturas de dados.

2.1 Estruturas de dados clássicas

A primeira estrutura implementada foi a lista encadeada pela sua simplicidade e por ser uma das primeiras a ser estudada. Essa estrutura utiliza de nós e ponteiros para guardar valores em cada nó, ou seja, diferentemente de um array, os dados não precisam estar um do lado do outro, facilitando operações. Ela pode ser utilizada em filas e pilhas.

A tabela hash, uma das primeiras estruturas básicas, mas eficiente, utiliza chaves e valores para guardar os dados. Esse método permite operações constantes, permitindo alta velocidade e eficiência, porém seu maior problema é a colisão de chaves e o método de tratamento dessas colisões. Utilizado em banco de dados e dicionários.

A árvore AVL, por sua vez, parte de uma árvore de busca binária auto-balanceada que utiliza de rotações para seu fator de balanceamento e deixar que a diferença entre as ramificações não seja maior que 1, permitindo uma organização e simetria nas operações, o que no caso de não balancear pode agir como uma lista, garantindo seu desempenho eficiente.

2.2 Estruturas de dados recentes

A primeira estrutura recente implementada no trabalho foi o Bloom Filter. Ela funciona com base no teste para verificar se faz parte de um grupo objetivo, permitindo o uso de pouca memória. No entanto, ela pode gerar resultados falsos positivos ou negativos. Ela foi escolhida para análise do impacto estatístico.

O Cuckoo Hashing é uma técnica de tratamento das colisões mencionadas nas tabelas hash. Ela utiliza de duas funções para resolver as colisões de modo que uma função calcula o valor da chave e quando há colisão, remove o valor antigo e substitui-o fazendo com que a outra função calcule um outro lugar. Isso faz com que a eficiência da estrutura seja constante independentemente da quantidade de dados

e a inserção um pouco menos eficiente. Escolhida como técnica avançada para as colisões e garantir alta performance.

O K-D Tree utiliza como base uma árvore binária, alterando a dimensão para dividir o espaço. Isto é, em um caso de duas dimensões (k = 2), a árvore alterna de x e y a cada nível da árvore, sendo o nó nesse nível, um valor médio que segue a ordem de: menor para esquerda (ou igual) e maior (ou igual) para direita, e guardando as coordenadas (x,y) conforme essa estrutura. Ela é utilizada para uma busca de proximidade. Escolhida para observar a eficiência ao dimensionar os dados.

3. TRABALHO

Nesta seção será comentado metodologias para o desenvolvimento do trabalho em Python (versão 3.13.3).

3.1 Dataset

O dataset de ataques cardíacos foi selecionado pelo interesse em fazer predições na área da saúde, garantindo modelos eficientes. Esse conjunto de dados possui 8763 amostras (linhas) e 26 variáveis (features), sendo elas: ID do Paciente, Idade, Sexo, Colesterol, Pressão Arterial, Frequência Cardíaca, Diabetes, Histórico Familiar, Tabagismo, Obesidade, Consumo de Álcool, Horas de Exercício por Semana, Dieta, Problemas Cardíacos Anteriores, Uso de Medicamentos, Nível de Estresse, Horas Sedentárias por Dia, Renda, IMC, Triglicerídeos, Dias de Atividade Física por Semana, Horas de Sono por Dia, País, Continente, Hemisfério, Risco de Infarto. Antes de começar qualquer atividade ao redor deste *dataset*, foi feito seu tratamento.

3.2 Estruturas de dados

A princípio, as estruturas de dados foram feitas em lugares separados e destinados apenas para elas em arquivos python (.py). Para implementá-las, utilizou-se de arquivos Jupyter (.ipynb), fornecendo células para executar separadamente e em etapas cada código. Para elas, não foi utilizado qualquer biblioteca já existente para essas estruturas, e sim um código do zero com ajuda da

inteligência artificial e materiais de aula (traduzidas da linguagem C). Nisso, utilizou-se também de classes de definição de nós e a própria classe da estrutura em si, em que permite a execução das funções definidas para as operações necessárias

As operações mencionadas que foram analisadas são: tempo de inserção, busca, remoção, uso de memória, taxa de colisão da tabela hash, tempo médio de acesso, escalabilidade.

Outra funcionalidade proposta no trabalho é a de otimização de estruturas de dados. No caso, a estrutura escolhida foi a lista encadeada, que revelou altos valores para os tempos de remoção, busca e inserção, em específico, o que levou a uma expectativa de possível otimização dessa estrutura. Então, primeiramente, a lista encadeada percorre apenas a partir de uma extremidade, o que afeta diretamente em sua eficiência no geral e, para melhorar isso, foi proposto a leitura também a partir do outro lado da lista, reduzindo drasticamente esses valores.

Além disso, realizou-se restrições para observar o impacto deles em diferentes cenários e estruturas. Utilizou-se das restrições: compactação de memória (R3), aumentando a carga computacional com número excessivo de operações por segundo (R9), rede de baixa velocidade (R11), situação de sensores defeituosos na leitura de dados com 10% de valores anômalos e uma carga adicional de encriptação (R23).

Por fim, criou-se um modelo preditivo para solucionar a tarefa do sistema: predizer ataques cardíacos, dividindo em 80% treino e 20% teste.

4 RESULTADOS

Primeiramente, a visualização dos dados do *dataset* revelou que o conjunto não possuía nenhum valor faltante e as mudanças para uma melhor análise foram a exclusão da coluna ID e a separação da variável Blood Pressure em pressão sistólica e diastólica. Então, resultou os seguintes gráficos e tabela:

Figura 1 - Distribuição das variáveis numéricas

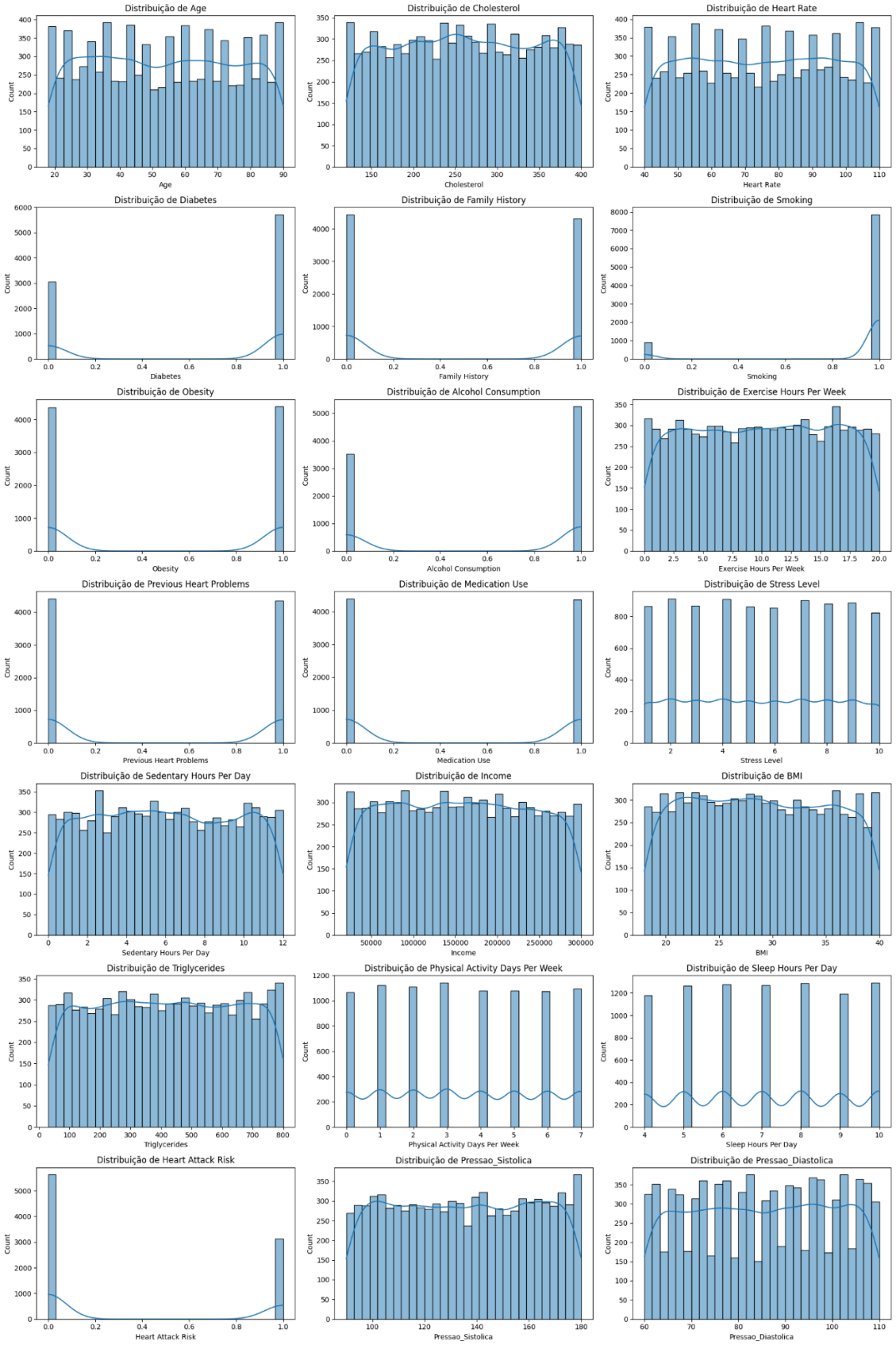


Figura 2 - Distribuição das variáveis categóricas

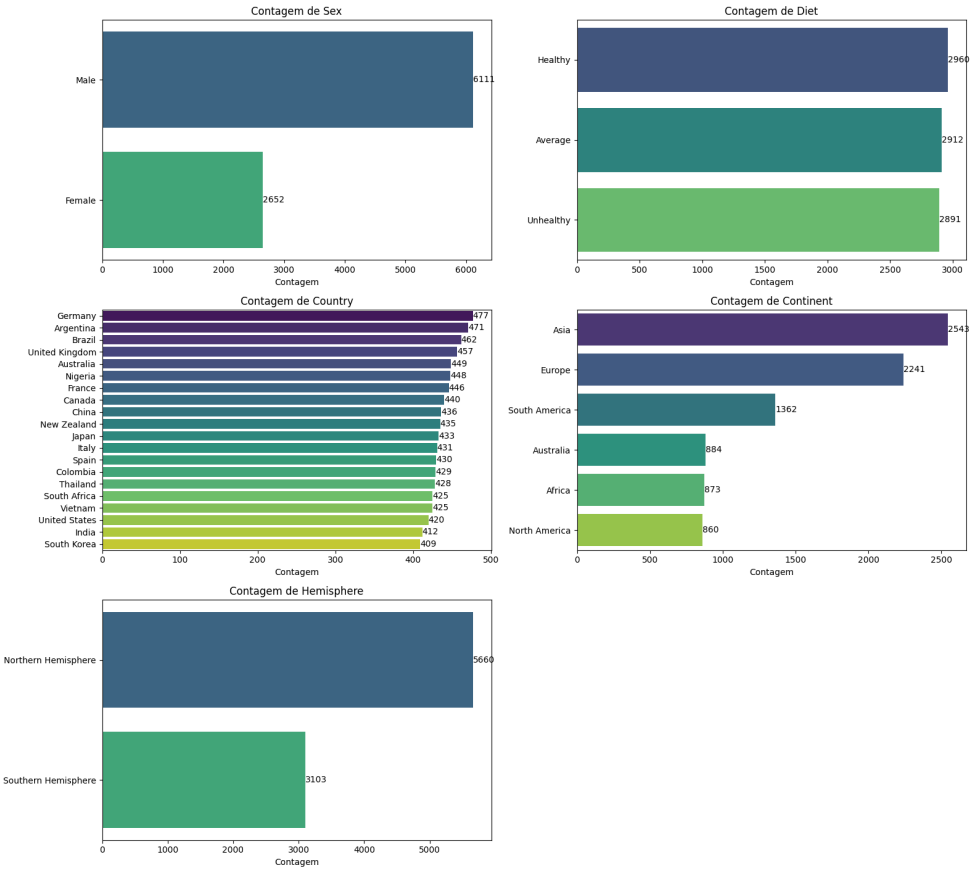
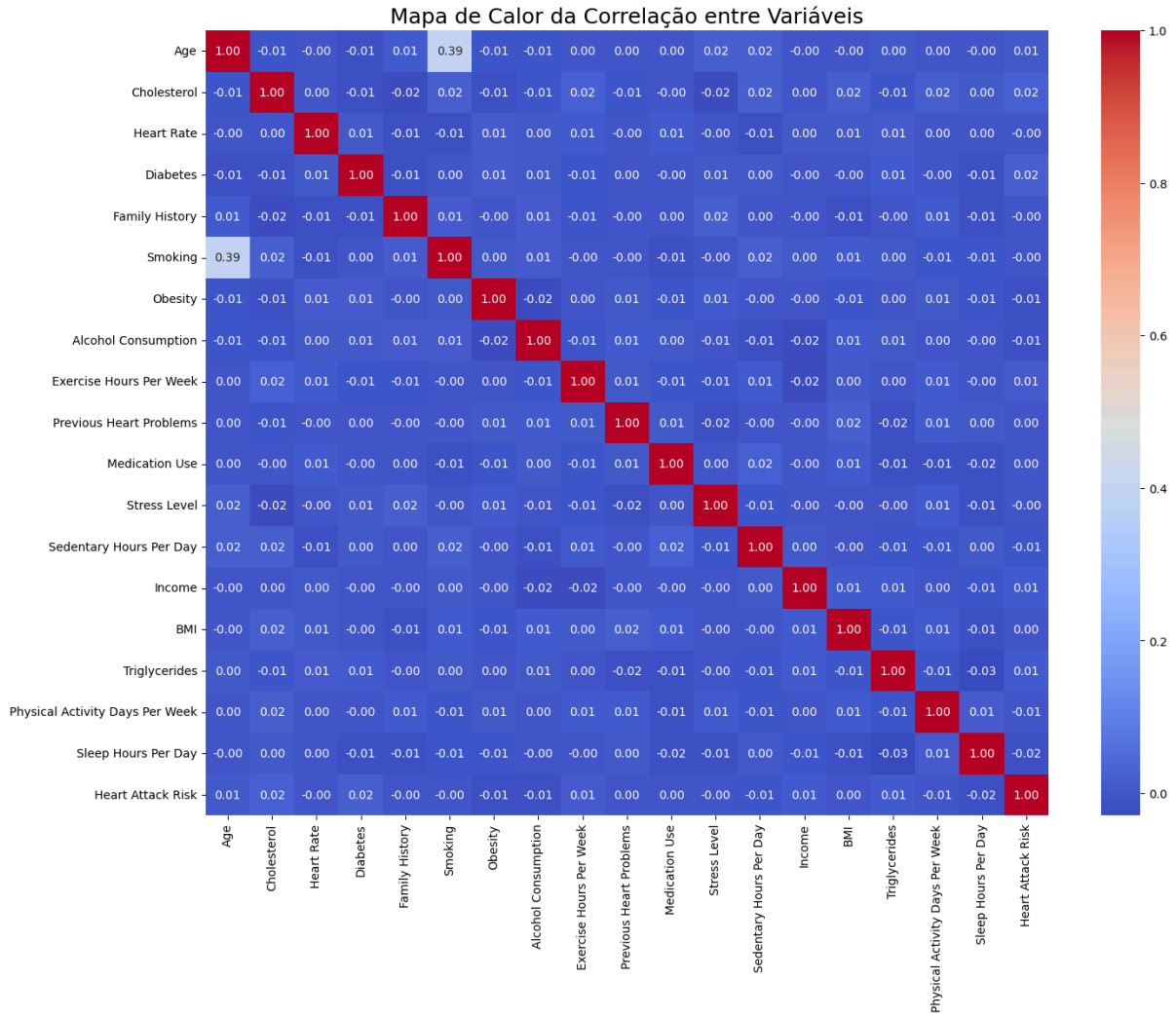


Tabela 1 - Estatística descritiva dos dados

```
================================================
   ESTATÍSTICAS DESCRITIVAS (COLUNAS NUMÉRICAS)
================================================
```

	count	mean	std	min	25%	50%	75%	max
Age	8763.0	53.707977	21.249509	18.000000	35.000000	54.000000	72.000000	90.000000
Cholesterol	8763.0	259.877211	80.863276	120.000000	192.000000	259.000000	330.000000	400.000000
Heart Rate	8763.0	75.021682	20.550948	40.000000	57.000000	75.000000	93.000000	110.000000
Diabetes	8763.0	0.652288	0.476271	0.000000	0.000000	1.000000	1.000000	1.000000
Family History	8763.0	0.492982	0.499979	0.000000	0.000000	0.000000	1.000000	1.000000
Smoking	8763.0	0.896839	0.304186	0.000000	1.000000	1.000000	1.000000	1.000000
Obesity	8763.0	0.501426	0.500026	0.000000	0.000000	1.000000	1.000000	1.000000
Alcohol Consumption	8763.0	0.598083	0.490313	0.000000	0.000000	1.000000	1.000000	1.000000
Exercise Hours Per Week	8763.0	10.014284	5.783745	0.002442	4.981579	10.069559	15.050018	19.998709
Previous Heart Problems	8763.0	0.495835	0.500011	0.000000	0.000000	0.000000	1.000000	1.000000
Medication Use	8763.0	0.498345	0.500026	0.000000	0.000000	0.000000	1.000000	1.000000
Stress Level	8763.0	5.469702	2.859622	1.000000	3.000000	5.000000	8.000000	10.000000
Sedentary Hours Per Day	8763.0	5.993690	3.466359	0.001263	2.998794	5.933622	9.019124	11.999313
Income	8763.0	158263.181901	80575.190806	20062.000000	88310.000000	157866.000000	227749.000000	299954.000000
BMI	8763.0	28.891446	6.319181	18.002337	23.422985	28.768999	34.324594	39.997211
Triglycerides	8763.0	417.677051	223.748137	30.000000	225.500000	417.000000	612.000000	800.000000
Physical Activity Days Per Week	8763.0	3.489672	2.282687	0.000000	2.000000	3.000000	5.000000	7.000000
Sleep Hours Per Day	8763.0	7.023508	1.988473	4.000000	5.000000	7.000000	9.000000	10.000000
Heart Attack Risk	8763.0	0.358211	0.479502	0.000000	0.000000	0.000000	1.000000	1.000000

Figura 3 - Mapa de calor da matriz de correlação



Depois de analisar o *dataset* e criar as devidas estruturas de dados escolhidas, fez-se seu benchmarking com o tempo de inserção, busca, remoção, uso de memória, taxa de colisão da tabela hash, tempo médio de acesso, escalabilidade com 7500 amostras.

Tabela 2 - Benchmark de tempo (s)

Estrutura	Tempo de Inserção (s)	Tempo de Busca (s)	Tempo de Remoção (s)	Tempo Médio de Acesso (s)
Lista Encadeada (Original)	0.673613	0.000028	0.000023	0.000005
Lista Encadeada	0.004529	0.000013	0.000016	0.000005
Tabela Hash	0.006940	0.000005	0.000004	0.000001
Árvore AVL	0.058573	0.000008	0.000021	0.000001
Cuckoo Hashing	0.035641	0.000010	0.000004	0.000006
Bloom Filter	0.173309	0.000037	0.000052	0.000006
KD-Tree	0.026651	0.000061	0.000014	0.000020

Tabela 3 - Benchmark de memória (bytes)

Estrutura	Uso de Memória (bytes)
Lista Encadeada (Original)	360,000
Lista Encadeada	360,000
Tabela Hash	1,621,376
Árvore AVL	359,952
Cuckoo Hashing	120,112
Bloom Filter	1,200,056
KD-Tree	360,000

Tabela 4 - Benchmark das colisões de hash e predição do Bloom Filter

Estrutura	Taxa de Colisão	Rehashes	Taxa Falsos Positivos
Tabela Hash	0.00%		
Cuckoo Hashing		0.0	
Bloom Filter			0.00%

Figura 4 - Latência média em inserção, busca e remoção

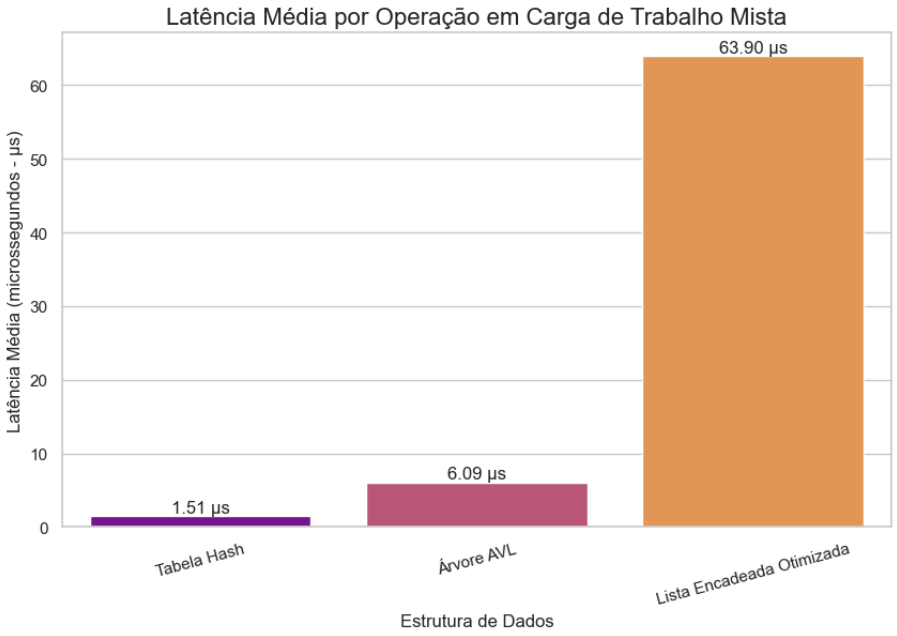


Tabela 5 - Resultado da escalabilidade em tabela

	N	LL_Insert	HashTable_Insert	AVL_Insert	Cuckoo_Insert	Bloom_Insert	KDTree_Build	LL_Search	HashTable_Search	AVL_Search	Cuckoo_Search	Bloom_Search	KDTree_Search
0	500	0.000240	0.000629	0.001960	0.005339	0.006940	0.003187	0.001253	0.000200	0.000265	0.001150	0.006666	0.006660
1	1000	0.000453	0.000579	0.004165	0.003071	0.013793	0.004903	0.002689	0.000337	0.000545	0.002685	0.016208	0.011761
2	2500	0.000688	0.001016	0.017650	0.013991	0.035583	0.008690	0.011804	0.000824	0.001317	0.010405	0.045607	0.038256
3	5000	0.002668	0.003284	0.034099	0.022898	0.079125	0.025639	0.022357	0.002674	0.003935	0.018944	0.090975	0.088144
4	7500	0.002361	0.005547	0.056743	0.037122	0.115364	0.032152	0.022327	0.001915	0.005415	0.023978	0.123081	0.125367

Figura 5 - Gráfico da escalabilidade em estruturas clássicas de inserção e busca

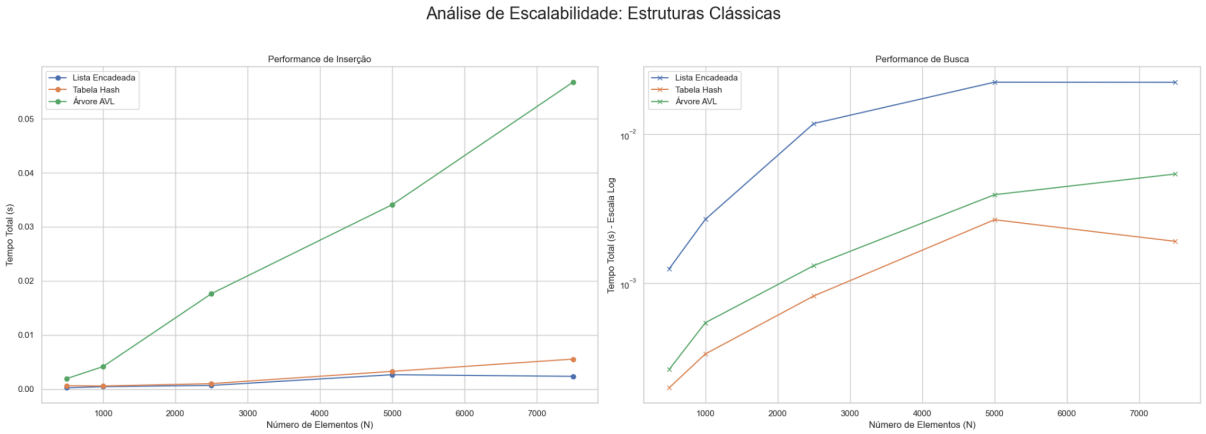
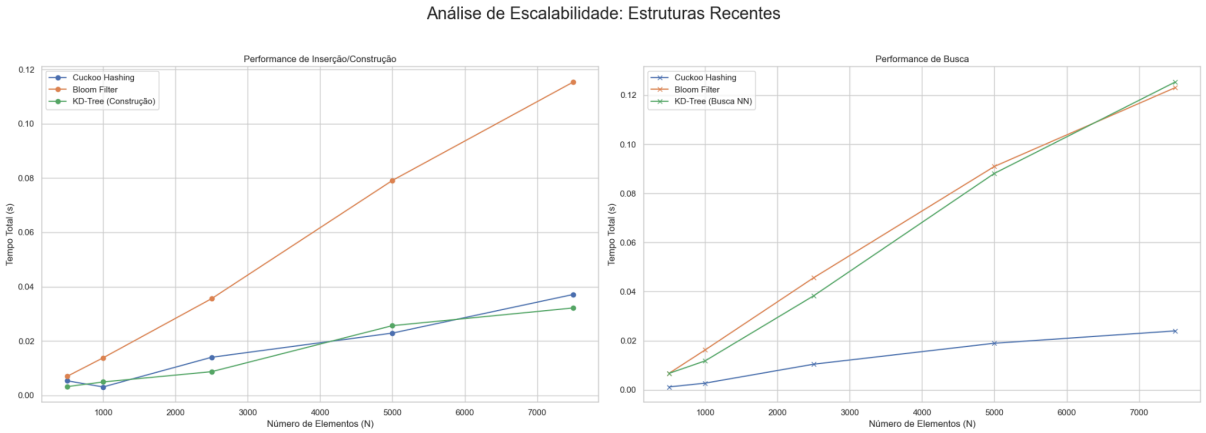


Figura 6 - Gráfico da escalabilidade em estruturas recentes de inserção e busca



Depois, foi feito uma otimização da lista encadeada resultando nos seguintes gráficos em uma amostra de 10000:

Tabela 6 - Benchmark de tempo (s)

	Tempo Original (s)	Tempo Otimizado (s)	Ganho (x mais rápido)
Inserção (Total)	1.235836	0.009419	131.212293
Busca (1 item)	0.000665	0.001393	0.477240
Remoção (1 item)	0.001431	0.000870	1.645050

Figura 7 - Gráfico do benchmarking de tempo (s)

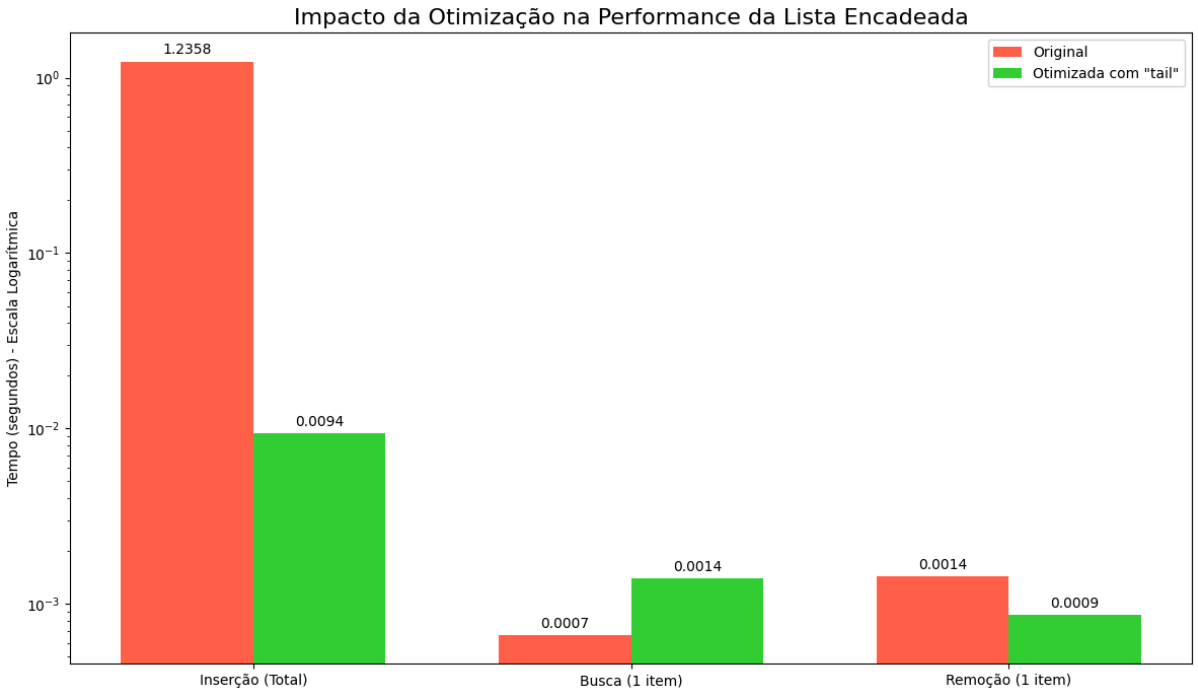


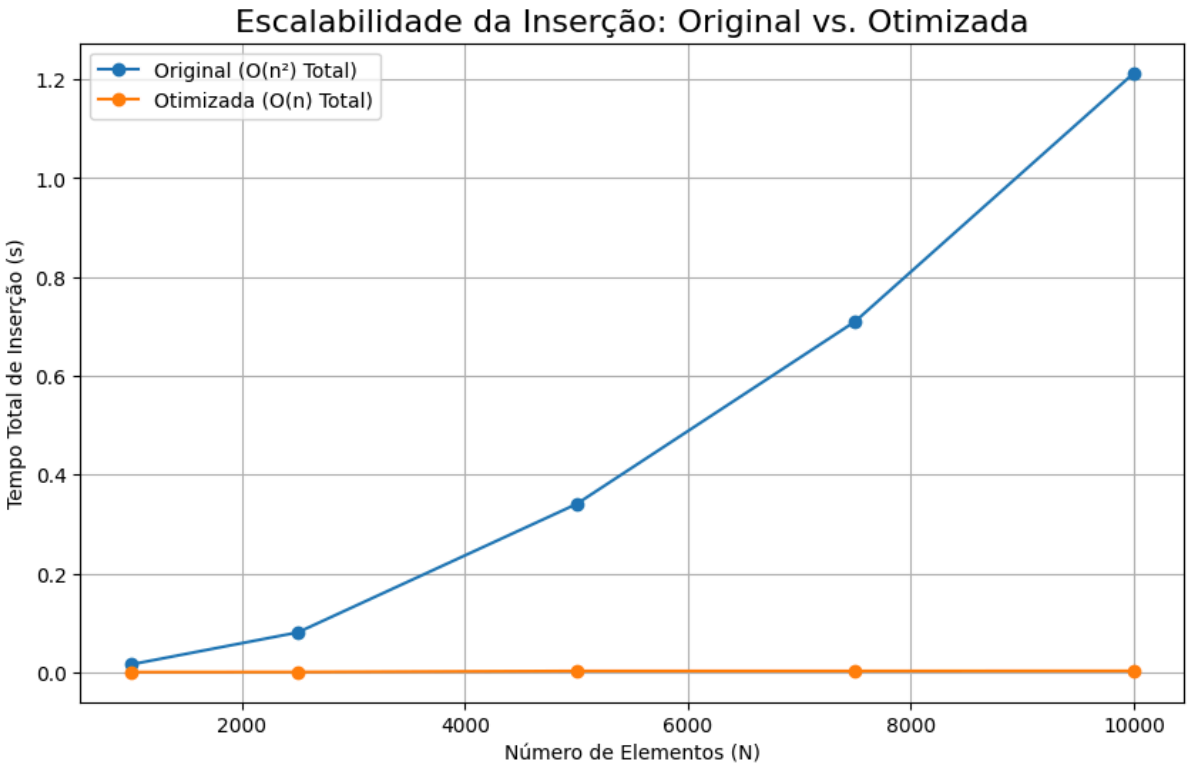
Tabela 7 - Comparação de uso de memória entre a original e otimizada (bytes)

```
--- Resultados do Uso de Memória ---
Memória da Lista Original:   480,048 bytes
Memória da Lista Otimizada:  480,048 bytes
Diferença:                   0 bytes a mais na versão otimizada
```

Tabela 8 - Comparação da latência média entre a original e otimizada (µs)

```
--- Comparativo de Latência Média (Carga Mista) ---
Latência Média (Original): 102.24 µs por operação
Latência Média (Otimizada): 61.21 µs por operação
```

Figura 8 - Gráfico comparativo da escalabilidade entre a original e otimizada



Feito isso, aplicou-se as restrições, obtendo os seguintes resultados:

Tabela 9 - Comparação entre a memória utilizada original e a compactada (R8)

```
Uso de memória original: 4.64 MB
Uso de memória compactado: 3.64 MB
Redução de memória: 21.45%
```

Figura 9 - Tipos de dados do dataset original (R8)

```
Tipos de dados antes:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Patient ID                      8763 non-null   object
 1   Age                             8763 non-null   int64
 2   Sex                             8763 non-null   object
 3   Cholesterol                     8763 non-null   int64
 4   Blood Pressure                  8763 non-null   object
 5   Heart Rate                      8763 non-null   int64
 6   Diabetes                        8763 non-null   int64
 7   Family History                  8763 non-null   int64
 8   Smoking                         8763 non-null   int64
 9   Obesity                         8763 non-null   int64
 10  Alcohol Consumption             8763 non-null   int64
 11  Exercise Hours Per Week         8763 non-null   float64
 12  Diet                            8763 non-null   object
 13  Previous Heart Problems         8763 non-null   int64
 14  Medication Use                  8763 non-null   int64
 15  Stress Level                    8763 non-null   int64
 16  Sedentary Hours Per Day         8763 non-null   float64
 17  Income                          8763 non-null   int64
 18  BMI                             8763 non-null   float64
 19  Triglycerides                   8763 non-null   int64
 20  Physical Activity Days Per Week 8763 non-null   int64
 21  Sleep Hours Per Day             8763 non-null   int64
 22  Country                         8763 non-null   object
 23  Continent                       8763 non-null   object
 24  Hemisphere                      8763 non-null   object
 25  Heart Attack Risk               8763 non-null   int64
dtypes: float64(3), int64(16), object(7)
memory usage: 1.7+ MB
None
```

Figura 10 - Tipos de dados do dataset compactado (R8)

```
Tipos de dados depois:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Patient ID                      8763 non-null   object
 1   Age                             8763 non-null   int8
 2   Sex                             8763 non-null   object
 3   Cholesterol                     8763 non-null   int16
 4   Blood Pressure                  8763 non-null   object
 5   Heart Rate                      8763 non-null   int8
 6   Diabetes                        8763 non-null   int8
 7   Family History                  8763 non-null   int8
 8   Smoking                         8763 non-null   int8
 9   Obesity                         8763 non-null   int8
 10  Alcohol Consumption             8763 non-null   int8
 11  Exercise Hours Per Week         8763 non-null   float32
 12  Diet                            8763 non-null   object
 13  Previous Heart Problems         8763 non-null   int8
 14  Medication Use                  8763 non-null   int8
 15  Stress Level                    8763 non-null   int8
 16  Sedentary Hours Per Day         8763 non-null   float32
 17  Income                          8763 non-null   float32
 18  BMI                             8763 non-null   float32
 19  Triglycerides                   8763 non-null   int16
 20  Physical Activity Days Per Week 8763 non-null   int8
 21  Sleep Hours Per Day             8763 non-null   int8
 22  Country                         8763 non-null   object
 23  Continent                       8763 non-null   object
 24  Hemisphere                      8763 non-null   object
 25  Heart Attack Risk               8763 non-null   int8
dtypes: float32(4), int16(2), int8(13), object(7)
memory usage: 761.8+ KB
None
```

Tabela 10 - Comparação da inserção sem e com carga extra na AVL (R9)

```
Executando 5 rodadas de cada teste...
Rodada 1/5 concluída...
Rodada 2/5 concluída...
Rodada 3/5 concluída...
Rodada 4/5 concluída...
Rodada 5/5 concluída...

--- Resultados Médios ---
Tempo médio de inserção normal: 0.0211 segundos
Tempo médio com carga extra:   0.0247 segundos

A carga extra aumentou o tempo médio de execução em: 17.04%
```

Tabela 11 - Comparação da rede devagar (R11)

```
--- Iniciando Teste de Latência (R11) Comparativo ---

Testando: Lista Encadeada
  Tempo para 2000 buscas (sem latência): 0.0083 segundos
  Tempo com latência: 1.2049 segundos

Testando: Árvore AVL
  Tempo para 2000 buscas (sem latência): 0.0012 segundos
  Tempo com latência: 1.2082 segundos

Testando: Tabela Hash
  Tempo para 2000 buscas (sem latência): 0.0013 segundos
  Tempo com latência: 1.3395 segundos
```

Tabela 12 - Comparação de dados sem e com anomalias (R18)

```
Acurácia do modelo em dados limpos: 64.18%
Acurácia do modelo em dados corrompidos: 64.18%
A performance do modelo degradou em: 0.00%
```

Tabela 13 - Comparação do efeito da encriptação (R23)

```
--- Testando Inserção Normal ---
Tempo de inserção normal: 0.0044 segundos

--- Testando Inserção com 'Encriptação' ---
Tempo de inserção com 'encriptação': 0.0127 segundos

--- Conclusão ---
A sobrecarga da 'encriptação' foi de 0.0082 segundos.
Isso representa um aumento de 185.38% no tempo total de inserção.
```

A partir disso, foram feitos modelos de predição de ataque cardíaco, até obter um resultado satisfatório.

Tabela 14 - Modelo 1 de predição com árvore de decisão sem balanceamento

```
Acurácia do modelo: 63.95%
Acurácia significa a porcentagem de vezes que o modelo acertou a previsão.

--- Matriz de Confusão ---
[[1117    8]
 [ 624    4]]

Interpretação da Matriz:
Verdadeiros Negativos (previu 'não' e era 'não'): 1117
Falsos Positivos (previu 'sim' e era 'não'): 8
Falsos Negativos (previu 'não' e era 'sim'): 624  <-- Erro potencialmente perigoso!
Verdadeiros Positivos (previu 'sim' e era 'sim'): 4

--- Relatório de Classificação ---
              precision    recall  f1-score   support

           0       0.64      0.99      0.78      1125
           1       0.33      0.01      0.01       628

    accuracy                           0.64      1753
   macro avg       0.49      0.50      0.40      1753
weighted avg       0.53      0.64      0.50      1753
```

Tabela 15 - Modelo 2 de predição com árvore de decisão com balanceamento (melhor modelo)

```
Acurácia do modelo: 53.05%
Acurácia significa a porcentagem de vezes que o modelo acertou a previsão.

--- Matriz de Confusão ---
[[658 467]
 [356 272]]

Interpretação da Matriz:
Verdadeiros Negativos (previu 'não' e era 'não'): 658
Falsos Positivos (previu 'sim' e era 'não'): 467
Falsos Negativos (previu 'não' e era 'sim'): 356  <-- Erro potencialmente perigoso!
Verdadeiros Positivos (previu 'sim' e era 'sim'): 272

--- Relatório de Classificação ---
              precision    recall  f1-score   support

           0       0.65      0.58      0.62      1125
           1       0.37      0.43      0.40       628

    accuracy                           0.53      1753
   macro avg       0.51      0.51      0.51      1753
weighted avg       0.55      0.53      0.54      1753
```

Tabela 16 - Modelo 3 de predição com Random Forest

```
Acurácia do modelo: 52.54%
Acurácia significa a porcentagem de vezes que o modelo acertou a previsão.

--- Matriz de Confusão ---
[[676 449]
 [383 245]]

Interpretação da Matriz:
Verdadeiros Negativos (previu 'não' e era 'não'): 676
Falsos Positivos (previu 'sim' e era 'não'): 449
Falsos Negativos (previu 'não' e era 'sim'): 383  <-- Erro potencialmente perigoso!
Verdadeiros Positivos (previu 'sim' e era 'sim'): 245

--- Relatório de Classificação ---
              precision    recall  f1-score   support

           0       0.64      0.60      0.62      1125
           1       0.35      0.39      0.37       628

    accuracy                           0.53      1753
   macro avg       0.50      0.50      0.49      1753
weighted avg       0.54      0.53      0.53      1753
```

Tabela 17 - Modelo 4 de predição Smote e Gradient Boosting

```
--- Avaliação do Gradient Boosting com SMOTE ---
Acurácia: 62.64%

Matriz de Confusão:
[[1077   48]
 [ 607   21]]

Relatório de Classificação:
              precision    recall  f1-score   support

           0       0.64      0.96      0.77      1125
           1       0.30      0.03      0.06       628

    accuracy                           0.63      1753
   macro avg       0.47      0.50      0.41      1753
weighted avg       0.52      0.63      0.51      1753
```

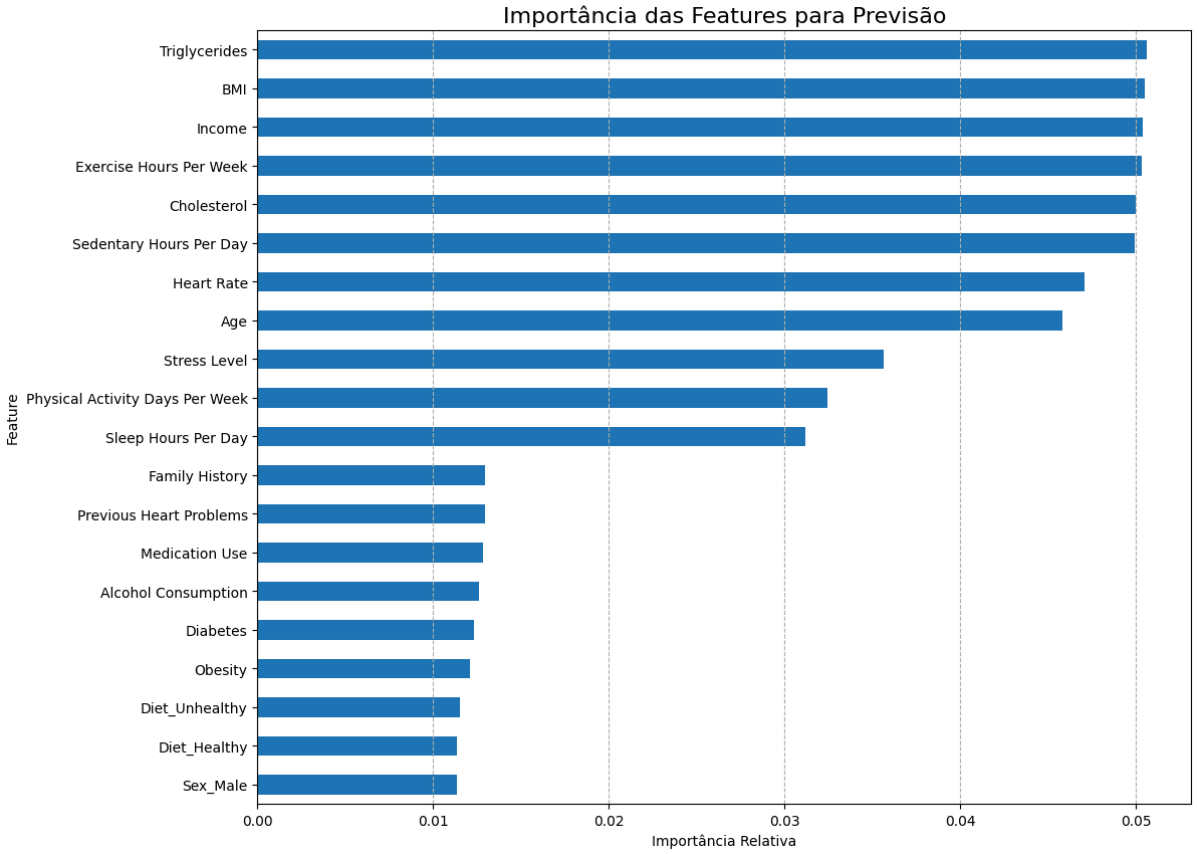
Tabela 18 - Modelo 5 de predição Random Forest e Grid Search

```
--- Avaliação do Melhor Modelo Encontrado ---
              precision    recall  f1-score   support

           0       0.64      0.60      0.62      1125
           1       0.35      0.39      0.37       628

    accuracy                           0.53      1753
   macro avg       0.50      0.50      0.50      1753
weighted avg       0.54      0.53      0.53      1753
```

Figura 11 - Importância das features



5 CONCLUSÃO

A visualização dos dados não revelou estatísticas faltantes para a sua análise. O tratamento de dados favoreceu a manipulação deles na estrutura. A correlação revelou uma correlação extremamente baixa, tornando a predição muito difícil.

A estrutura com melhor desempenho, originalmente, foi a tabela hash, destacando-se pela sua velocidade de executar as operações necessárias. Uma estrutura surpreendente foi a lista encadeada otimizada, que exibiu resultados impressionantes da sua evolução. Comparando por cada operação, na sequência de melhor e pior, para a inserção: lista encadeada otimizada e lista encadeada; para a busca: tabela hash e KD-Tree; para a remoção: tabela hash ou cuckoo hashing e bloom filter; para tempo médio de acesso: tabela hash ou AVL e KD-Tree; para o uso de memória: cuckoo hashing e tabela hash. Para as colisões, rehashing e falso-positivos na tabela hash, cuckoo hashing e bloom filter, respectivamente, foram 0 devido ao pequeno volume de amostras utilizados na sua implementação e o grande número de chaves.

Ademais teimosa latência e a estabilidade das estruturas, referente a latência a tabela hash se mostrou a melhorem tempo médio, em operações de busca e exclusão, lista encadeada otimizada mostrou uma grande redução em comparação a lista encadeada simples especialmente em buscas e inserções, já para a estabilidade as estruturas AVL e Cuckoo Hashing demonstram manter a performance em comparação às demais quando estão lidando com muitos dado, às estruturas mais simples como listas encadeadas demonstram um crescimento linear com o aumento dos dados a serem lidados, os benchmarks também reforçam que estruturas balanceadas ou com tratamento avançado de colisões como a Cuckoo Hash tem tempos menores quando estão lidando com muitos dados.

Nas aplicações das restrições observou-se que uma compactação da memória reduz a utilização dela em 21,45% ao diminuir o tamanho do tipo de dado, como é possível ver da figura 9 e 10 que alguns inteiros (int) mudam de 64 para 16 ou 8. A inserção com carga extra, faz com que o processador faça processamentos desnecessários, mas que mesmo assim o tempo diminui, isso ocorre na verdade porque é ativado no próprio processador, o modo turbo, que otimiza esse tipo de processamento e tenha um tempo consistente ao longo das operações requisitadas,

isso também explica o fato de ser realizado mais de uma vez e observar suas inconsistência na utilização ou não do turbo. Para a velocidade de rede lenta, observa-se um aumento considerável no tempo de busca em todas as estruturas. Para a próxima restrição, o modelo resistiu bem à 10% de anomalias nos dados, o que não alterou o resultado final. A encriptação, por sua vez, revelou um aumento considerável de 185,38% de aumento no tempo da inserção

Com base, na figura 11, observa-se que há uma dificuldade em criar um modelo confiável para essa análise, pois a importância de cada variável em relação ao risco de ataque cardíaco é extremamente baixa, menos de 5%. O melhor modelo resultou numa precisão de 53,05% e um retorno positivo de alto risco de ataques cardíacos de 43%, isto é, ele previu certo o alto risco. Com isso, é possível ver como as variáveis realmente não tem muita influência sobre a variável alvo gerando essa baixa porcentagem de acertos da predição de ataques cardíacos, isto vem de uma limitação do dataset que ainda não tem dados o suficiente para poder gerar modelos de previsão mais confiáveis.

REFERÊNCIAS

KAGGLE, Heart Attack Risk Prediction Dataset. Disponível em: https://www.kaggle.com/datasets/iamsouravbanerjee/heart-attack-prediction-dataset. Acesso em: 12 mai. 2025.

SEABORN. Seaborn: Statistical data visualization. [S. l.]: PyData, 2024. Disponível em: https://seaborn.pydata.org/. Acesso em: 21 mar. 2025.

MATPLOTLIB. Matplotlib: Visualization with Python. [S. l.]: Matplotlib Development Team, 2024. Disponível em: https://matplotlib.org/. Acesso em: 21 mar. 2025.

PANDAS. Pandas Documentation. [S. l.]: pandas-dev, 2024. Disponível em: https://pandas.pydata.org/. Acesso em: 21 mar. 2025.

NUMPY. NumPy Documentation. [S. l.]: NumPy Developers, 2024. Disponível em: https://numpy.org/doc/. Acesso em: 24 mar. 2025.

SCIPY. SciPy: Scientific Computing Tools for Python. [S. l.]: SciPy Developers, 2024. Disponível em: https://docs.scipy.org/doc/scipy/. Acesso em: 24 mar. 2025.

SCIKIT-LEARN. Scikit-learn: Machine Learning in Python. [S. l.]: Scikit-learn Developers, 2024. Disponível em: https://scikit-learn.org/stable/. Acesso em: 30 mar. 2025.

GEEKS FOR GEEKS. Bloom Filters. GeeksforGeeks, 2024. Disponível em: https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation. Acesso em: 7 jun. 2025.

GEEKS FOR GEEKS. Cuckoo Hashing. GeeksforGeeks, 2024. Disponível em: https://www.geeksforgeeks.org/cuckoo-hashing-in-python/. Acesso em: 7 jun. 2025.

GEEKS FOR GEEKS. KD-Tree. GeeksforGeeks, 2024. Disponível em: https://www.geeksforgeeks.org/ball-tree-and-kd-tree-algorithms/. Acesso em: 7 jun. 2025.

APÊNDICE A - Colunas do conjunto de dados

Coluna: Patient ID
Valores únicos: ['BMW7812' 'CZE1114' 'BNI9906' ... 'XKA5925' 'EPE6801' 'ZWN9666']

Coluna: Age
Valores únicos: [67 21 84 66 54 90 20 43 73 71 77 60 88 69 38 50 45 36 48 40 79 63 27 25
 86 42 52 29 30 47 44 33 51 70 85 31 56 24 74 72 55 26 53 46 57 22 35 39
 80 65 83 82 28 19 75 18 34 37 89 32 49 23 59 62 64 61 76 41 87 81 58 78
 68]

Coluna: Sex
Valores únicos: ['Male' 'Female']

Coluna: Cholesterol
Valores únicos: [208 389 324 383 318 297 358 220 145 248 373 374 228 259 122 379 166 303
 340 294 359 202 133 159 271 273 328 154 135 197 321 375 360 263 201 347
 129 229 251 121 190 185 279 336 192 180 203 368 222 243 218 120 285 377
 369 311 139 266 153 339 329 333 398 124 183 163 362 390 200 396 255 209
 247 250 227 246 223 330 195 194 178 155 240 237 216 276 224 326 198 301
 314 304 334 213 254 230 316 277 388 206 384 205 261 308 338 382 291 168
 171 378 253 245 226 281 123 173 231 234 268 306 186 293 161 380 239 149
 320 219 335 265 126 307 270 225 193 148 296 136 364 353 252 232 387 299
 357 214 370 345 351 344 152 150 131 272 302 337 170 356 274 188 125 138
 376 181 184 275 394 128 217 399 283 289 284 327 262 212 350 385 162 141
 361 244 295 287 144 354 363 352 140 196 172 319 325 331 392 147 187 346
 286 151 300 165 343 366 317 386 158 157 242 241 365 257 348 175 298 269
 267 397 310 341 204 127 290 280 132 322 179 199 143 312 288 395 189 156
 238 381 391 355 210 400 260 235 167 256 249 207 130 134 137 305 236 315
 292 323 146 258 332 372 142 309 177 367 371 211 282 342 264 176 160 233

313 164 349 221 191 174 393 278 215 169 182]

Coluna: Blood Pressure
Valores únicos: ['158/88' '165/93' '174/99' ... '137/94' '94/76' '119/67']

Coluna: Heart Rate
Valores únicos: [72 98 73 93 48 84 107 68 55 97 70 85 102 40 56 104 71
69
 66 81 52 105 96 74 49 45 50 46 44 106 83 86 65 101 51 43
 79 90 94 78 92 54 109 61 64 82 110 42 63 41 100 76 75 58
 53 60 77 47 59 57 87 67 88 99 80 95 108 89 62 103 91]

Coluna: Diabetes
Valores únicos: [0 1]

Coluna: Family History
Valores únicos: [0 1]

Coluna: Smoking
Valores únicos: [1 0]

Coluna: Obesity
Valores únicos: [0 1]

Coluna: Alcohol Consumption
Valores únicos: [0 1]

Coluna: Exercise Hours Per Week
Valores únicos: [4.16818884 1.81324162 2.07835299 ... 3.14843791 3.78994983
 18.08174797]

Coluna: Diet
Valores únicos: ['Average' 'Unhealthy' 'Healthy']

Coluna: Previous Heart Problems
Valores únicos: [0 1]

Coluna: Medication Use
Valores únicos: [0 1]

Coluna: Stress Level
Valores únicos: [9 1 6 2 7 4 5 8 10 3]

Coluna: Sedentary Hours Per Day
Valores únicos: [6.61500145 4.96345884 9.46342584 ... 2.37521373 0.02910426 9.00523438]

Coluna: Income
Valores únicos: [261404 285768 235282 ... 36998 209943 247338]

Coluna: BMI
Valores únicos: [31.25123273 27.19497335 28.17657068 ... 35.40614616 27.29402009
 32.91415086]

Coluna: Triglycerides
Valores únicos: [286 235 587 378 231 795 284 370 790 232 469 523 590 506 635 773 68 402 517 247 747 360 358 526 605 667 316 551 482 718 297 661 558 209 586 743 411 785 697 519 595 452 158 679 675 792 584 366 741 474 92 410 398 493 614 682 106 216 408 628 481 67 82 305 164 211 511 766 547 327 367 681 131 42 692 664 543 689 569 458 683 779 136 643 653 55 275 314 760 404 576 690 648 385 255 468 784 509 205 109 530 654 331 485 250 113 377 180 229 602 285 471 554 344 416 445 709 426 528 388 441 306 749 347 341 451 356 336 455 223 262 239 555 363 489 788 121 553 617 174 167 563 665 65 657 237 141 767 292 214 221 447 634 460 711 97 267 695 717 383 332 449 701 524 549 31 276 744 128 52 394 54 739 407 751 436 473 218 129 579 492 696 202 197 521 325 35 123 694 434 248 348 750 431 714 649 668 401 610 244 691 88 532 777 420 350 652 413 754 753 457 122 312 778 676 775 183 601 317 592 191 83 32 453

423 234 650 565 798 769 412 63 198 93 764 737 94 298 288 735 190 281 146
574 359 155 719 466 273 515 187 544 103 132 118 115 85 38 117 362 133 498
645 339 787 733 663 291 502 78 81 257 624 91 374 270 797 446 464 450 722
556 184 428 796 656 134 196 623 522 376 730 463 99 593 47 148 302 57 280
389 629 294 186 700 774 181 375 467 603 616 380 495 698 318 207 780 51 84
425 310 126 56 472 669 688 655 39 333 501 479 540 433 179 490 204 644 525
546 486 320 319 58 591 165 732 195 478 461 631 301 50 315 194 199 160 149
527 406 161 125 200 277 308 69 427 236 77 500 269 79 168 575 606 355 636 64
251 245 228 287 800 483 791 260 604 536 559 124 254 159 73 542 390 755 60
61 491 40 437 215 440 379 789 266 505 243 783 403 637 156 729 438 507 725
562 324 87 253 626 541 364 456 30 182 621 494 776 442 429 684 219 70 98 166
95 135 646 337 226 710 608 208 724 704 512 206 224 622 598 465 119 293 630
386 513 45 578 261 217 715 282 391 580 192 399 249 396 278 448 782 419 503
220 49 304 157 150 545 627 582 178 263 33 299 303 66 763 256 139 651 756
372 345 48 46 421 43 771 210 781 41 508 353 566 726 736 326 759 477 369 188
104 329 309 384 599 415 770 571 552 145 632 373 71 550 583 322 475 357 673
454 757 201 100 274 258 613 233 330 731 761 296 573 335 716 642 142 674 572
638 222 752 740 397 594 705 381 615 539 242 499 435 680 535 238 283 89 589
666 678 76 176 620 75 721 143 723 570 44 203 259 677 734 662 707 745 487
577 443 120 111 365 116 538 162 742 212 581 313 36 400 619 609 252 706 264
290 138 300 346 712 34 387 140 154 758 462 672 713 86 414 699 529 382 432
368 193 72 537 560 189 342 531 311 241 685 497 640 321 480 144 585 171 727
660 799 600 597 213 708 151 265 618 658 746 307 53 514 611 153 352 225 567
702 520 417 102 607 548 647 476 762 147 424 459 409 74 510 37 323 240 175
786 80 439 504 772 670 59 334 703 392 90 496 422 279 343 671 794 163 328
625 272 227 152 105 693 96 484 568 633 659 230 112 793 101 172 110 612 185
289 418 533 686 641 169 349 173 516 62 557 596 728 371 738 444 561 114 765
338 588 246 295 564 488 177 687 395 518 127 639 137 354 271 107 340 534 768
130 720 405 430 268 108 748 351 393 361 170 470]

Coluna: Physical Activity Days Per Week
Valores únicos: [0 1 4 3 5 6 7 2]

Coluna: Sleep Hours Per Day

Valores únicos: [6 7 4 5 10 8 9]

Coluna: Country
Valores únicos: ['Argentina' 'Canada' 'France' 'Thailand' 'Germany' 'Japan' 'Brazil'
 'South Africa' 'United States' 'Vietnam' 'China' 'Italy' 'Spain' 'India'
 'Nigeria' 'New Zealand' 'South Korea' 'Australia' 'Colombia'
 'United Kingdom']

Coluna: Continent
Valores únicos: ['South America' 'North America' 'Europe' 'Asia' 'Africa' 'Australia']

Coluna: Hemisphere
Valores únicos: ['Southern Hemisphere' 'Northern Hemisphere']

Coluna: Heart Attack Risk
Valores únicos: [0 1]

Coluna: Pressao_Sistolica
Valores únicos: [158 165 174 163 91 172 102 131 144 160 107 101 169 112 114
173 120 180 130 175 161 140 148 113 99 178 111 103 94 127 134 115 124 104
149 106 168 159 100 118 152 151 109 92 179 132 177 110 116 139 164 122 95
155 108 105 146 166 117 145 170 125 135 137 121 126 142 128 143 96 153 156
162 98 150 133 157 154 123 171 90 176 167 141 138 119 97 93 147 129 136]

Coluna: Pressao_Diastolica
Valores únicos: [88 93 99 100 86 73 68 105 70 69 71 72 81 75 74 98 101
84 60 109 90 95 76 78 77 91 85 107 92 96 108 63 106 79 80 83 61 65 97
94 66 110 104 67 89 102 82 62 64 87 103]

APÊNDICE B - Prompts utilizados

‘De um exemplo de como importar um dataset em python’
‘De um exemplo de como tratar um dataset em python’
‘De um exemplo de como plotar um dataset em python’

‘De um exemplo de como fazer inscrição para listas encadeadas’
‘De um exemplo de como fazer exclusão para listas encadeadas’
‘De um exemplo de como fazer busca para listas encadeadas’

‘De um exemplo de como fazer inscrição para árvores balanceadas’
‘De um exemplo de como fazer exclusão para árvores balanceadas’
‘De um exemplo de como fazer busca para árvores balanceadas’

‘De um exemplo de como fazer inscrição para tabela hash’
‘De um exemplo de como fazer exclusão para tabela hash’
‘De um exemplo de como fazer busca para tabela hash’

‘De um exemplo de como fazer inscrição para kd-tree’
‘De um exemplo de como fazer exclusão para kd-tree’
‘De um exemplo de como fazer busca para kd-tree’

‘De um exemplo de como fazer inscrição para bloom filter’
‘De um exemplo de como fazer exclusão para bloom filter’
‘De um exemplo de como fazer busca para bloom filter’

‘De um exemplo de como fazer inscrição para cuckoo hashing’
‘De um exemplo de como fazer exclusão para cuckoo hashing’
‘De um exemplo de como fazer busca para cuckoo hashing’

‘De um exemplo de como fazer uma lista encadeada otimizada’

‘De um exemplo de como fazer um modelo e predição com árvore de decisão sem balanceamento’
‘De um exemplo de como fazer um modelo e predição com árvore de decisão com balanceamento’
‘De um exemplo de como fazer um modelo e predição com random forest’

‘De um exemplo de como fazer um modelo e predição com smote e gradient boosting’
‘De um exemplo de como fazer um modelo e predição com random forest e grid search’

‘Como fazer benchmark de tempo e memória’
‘Como fazer benchmark de colisão de hash’
‘Como fazer benchmark de colisão de bloom filter’
‘Como medir a latência de inserção busca e exclusão’

‘Faça o gráfico de distribuição das features numéricas’
‘Faça o gráfico de distribuição das features categóricas’
‘Faça a matriz de correlação das features’

‘Faça um exemplo de gráfico de estabilidade em na estrutura lista encadeada para inserção busca’

‘Faça um exemplo de gráfico para comparar uma estrutura original e uma otimizada’

‘De um exemplo de como comparar o tempo de inserção sem e com carga extra na avl ’

‘De um exemplo de como comparar o tempo de busca sem e com Latência em hashtable’

‘De um exemplo de como comparar os dados com e sem anomalia’