

# **Tratamento de dados e Treinamento de Aprendizado de Máquina para Identificação de Doenças com Base em Hemograma**

Georges Ballister de Oliveira<sup>1</sup>, Daniel de Melo Arantes Cabral, Douglas Numeriano Marinho Falcão<sup>1</sup>, Davi Albnes Vasconcellos Pires<sup>1</sup>, Bruno Vinícius Araújo de Mesquita<sup>1</sup>, Gabriel Sobral Santos Silva<sup>1</sup>,

<sup>1</sup>Centro Universitário dos Guararapes (UNIFG)

Recife – PE – Brazil

georgesballister.profissional@gmail.com, 12dancabral@gmail.com,  
douglasnumeriano11@hotmail.com, davipires03@gmail.com, brunovinicius2002@hotmail.com,  
gabrielprofissional12@gmail.com

***Abstract. The aim of this study is to demonstrate the data analysis process conducted on the "Multiple Disease Prediction" dataset using Python. The primary focus was to train machine learning models to identify, based on a blood test including information on 25 blood proteins, two major blood-related conditions: diabetes and anemia, as well as to distinguish other health conditions.***

***Resumo. Este estudo visa demonstrar o processo de análise de dados realizado no dataset "Multiple Disease Prediction", utilizando a linguagem Python. O objetivo principal é treinar um modelo de aprendizado de máquina capaz de identificar, a partir de um hemograma com 25 proteínas sanguíneas, duas doenças sanguíneas principais: diabetes e anemia, além de detectar outras possíveis condições não relacionadas a essas doenças.***

## **Introdução**

O número crescente de doenças que afetam a sociedade humana é uma preocupação significativa. A identificação precoce dessas doenças é essencial para o tratamento eficaz e a melhoria da qualidade de vida. Um hemograma, acompanhado pela análise de proteínas totais e suas frações, pode fornecer uma avaliação valiosa do estado nutricional e a detecção de diversas doenças hepáticas, renais e hematológicas. Essas doenças causam variações nos níveis de albumina e globulinas presentes no sangue.

Com base nesse contexto, este estudo realizou uma análise de dados utilizando um conjunto de dados que simula milhares de hemogramas, contendo informações sobre 25 proteínas sanguíneas. O objetivo foi desenvolver um diagnóstico que interprete os valores presentes em cada coluna do registro, focando na identificação de diabetes, anemia e outras condições de saúde (qualquer doença que não seja diabetes ou anemia). Para alcançar esse objetivo, foram treinados dois modelos de inteligência artificial distintos. Esses modelos foram projetados para identificar padrões nos valores medianos dessas proteínas, levando em consideração o desvio padrão.

Este projeto busca não apenas avançar no campo da análise de dados médicos, mas também contribuir para a melhoria das técnicas de diagnóstico precoce por meio do uso de algoritmos de aprendizado de máquina.

## 1. Contextualização do problema Encontrado:

### Multiple Disease Prediction

Disease prediction dataset based on blood samples


[Data Card](#) [Code \(20\)](#) [Discussion \(3\)](#) [Suggestions \(0\)](#)

#### About Dataset

A dataset to check the health, whether the person has a specific disease or he is well, based on blood samples or parameters, the data is hand madden, you can use it just for learning, i searched about the percentage of each parameter and what will happen if this parameter high or low, and which disease you might have if this parameter high or low, based on all this information and searches i created this dataset just for learning and a project idea purpose  
Also, the data is already scaled in range(0,1)

**this is the percentage of each parameter:**

```
"Glucose": (70. 140). # mo/dL
```



#### Usability

9.41

#### License

Apache 2.0

#### Expected update frequency

Never

#### Tags

Classification

Figura 1. (Tela do Dataset disponibilizado no Kaggle)

### 1.1 Conjunto de Dados

O estudo utiliza um conjunto de dados extraído do site Kaggle, especificamente o dataset denominado "Multiple Disease Prediction". Este dataset é composto por informações provenientes de amostras de sangue, que são utilizadas para prever diversas doenças.

O objetivo deste projeto é utilizar o dataset "Multiple Disease Prediction" para realizar uma análise detalhada dos dados, permitindo o diagnóstico de doenças como diabetes e anemia. Para isso, empregamos dois modelos de treinamento distintos, ambos aptos a lidar com dados complexos e a fornecer interpretações claras.

Após o pré-processamento dos dados, dividimos o conjunto em dados de treino e teste para realizar a avaliação dos modelos. Esta abordagem integrada possibilita uma compreensão abrangente das condições de saúde dos pacientes e contribui para o desenvolvimento de estratégias de prevenção e tratamento mais eficazes.

1.2 Dados Encontrados

O dataset "Multiple Disease Prediction" foi criado com o objetivo de avaliar a saúde dos indivíduos, determinando se uma pessoa possui uma doença específica ou se está saudável, com base em amostras de sangue e outros parâmetros clínicos. Os dados já foram normalizados no intervalo de 0 a 1 para facilitar a análise. A seguir, apresentamos as principais características e seus respectivos intervalos de referência:

Parâmetro	Intervalo de Referência	Unidade
Glicose	70-140	mg/dL
Colesterol	125-200	mg/dL
Hemoglobina	13,5-17,5	g/dL
Plaquetas	150.000-450.000	por microlitro de sangue
Leucócitos (glóbulos brancos)	4.000-11.000	por milímetro cúbico de sangue
Eritrócitos (glóbulos vermelhos)	4,2-5,4	milhões por microlitro de sangue
Hematócrito	38-52	%
Volume Corpuscular Médio (MCV)	80-100	femtolitros
Hemoglobina Corpuscular Média (MCH)	27-33	picogramas
Concentração de Hemoglobina Corpuscular Média (MCHC)	32-36	g/dL
Insulina	5-25	microU/mL
Troponina	0-0,04	ng/mL
Índice de Massa Corporal (IMC)	18,5-24,9	kg/m²
Pressão Arterial Sistólica	90-120	mmHg
Pressão Arterial Diastólica	60-80	mmHg
Triglicerídeos	50-150	mg/dL
Hemoglobina Glicada (HbA1c)	4-6	%
Colesterol LDL	70-130	mg/dL
Colesterol HDL	40-60	mg/dL
ALT (Alanina Aminotransferase)	10-40	U/L
AST (Aspartato Aminotransferase)	10-40	U/L
Frequência Cardíaca	60-100	batimentos por minuto
Creatinina	0,6-1,2	mg/dL
Proteína C-reativa	0-3	mg/L

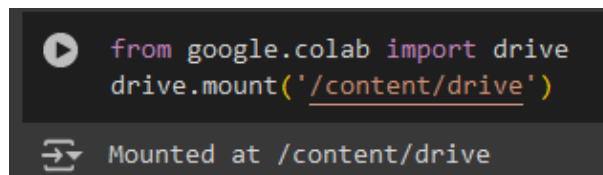
Esses parâmetros são fundamentais para a detecção de diversas condições de saúde, como diabetes, anemia, dislipidemias, e doenças cardíacas, entre outras. Ao analisar esses dados, podemos identificar padrões e anomalias que auxiliam no diagnóstico e na proposição de intervenções preventivas ou terapêuticas.

## 2. Ferramentas Utilizadas

Começando pelas escolhas do ambiente deste projeto o Google colab junto com o Google drive, ferramentas populares devida sua conveniência e integração com a plataforma Google.

Primeiro o Google Colab que é um ambiente de desenvolvimento integrado (IDE) na nuvem assim permitindo escrever e executar código python em um navegador, sem a necessidade de configuração ou instalação de componentes. Além de oferecer acesso gratuito a GPUs (Unidades de Processamento Gráfico), assim acelerar significativamente o treinamento de modelos de aprendizagem da máquina. Também ressaltando a facilidade de compartilhamento como o colab está hospedado na nuvem, é fácil compartilhar os “Notebook” com os colaboradores ou apresentar para outras pessoas.

O Google Drive é uma ferramenta de armazenamento na nuvem centralizada e acessível em qualquer lugar. Assim colaboração com colaboradores em tempo real fazendo vários usuários podem editar documentos simultaneamente. segurança e sincronização com os backups mantendo seguros de imprevistos e atualizados com arquivos mais recentes. O ponto mais forte para o uso desta ferramenta no projeto foi uma integração com o colab e vise versa assim facilitando o armazenamento de arquivos e o gerenciamento do conjunto de dados. Que pode se utilizar com esse comando: **from google.colab import drive**



```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

Figura 2. (Código no Colab para importar dados do Google Drive)

### 3. Bibliotecas Python Utilizadas

```
[ ] # Bibliotecas
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.svm import SVC
from sklearn import datasets
from sklearn import svm
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
from sklearn.ensemble import RandomForestClassifier
```

Figura 3. (Todas as Bibliotecas importadas para o projeto no Colab)

Falando sobre bibliotecas ao todo foram utilizadas 6 bibliotecas e seus derivados, para o desenvolvimento do código e demonstração do seu funcionamento sendo elas de grande importância para a construção desse trabalho:

1. **Pandas:** é uma biblioteca poderosa para acessar e manipular tabelas usada no projeto foi usado para acessar e manipular do dataset.
2. **Random** e **Numpy** duas bibliotecas com a funções matemáticas Random como seu nome sugere e usada para gerar números aleatório, do qual são muito importantes para o treinamento da máquina. **Numpy** foi fundamental pois ele fornece suporte para arrays e matrizes, juntamente com uma grande coleção de funções matemáticas para operar nesses arrays. Com isso fornecendo de forma eficiente a efetividade dos cálculos matemáticos, permitindo o cálculo de média, fundamental para o código, além de tem compatibilidade com todas as bibliotecas presentes no código.
3. **Matplotlib** e **Seaborn** são duas bibliotecas de visualização e criação de gráfico, importante para visualizar e entender o progresso do código e suas variáveis. Além de ajudar na explicação do código para outros.
4. **Sklearn:** e seus “derivado”, também conhecida como **scikit-learn**, contém o papel mais importante para o código dividido em várias partes fundamentais. Pois o é uma biblioteca de aprendizado de máquina em Python, que oferece uma variedade de ferramentas para modelagem preditiva, construída sobre as bibliotecas **numpy** e **scipy**.

- a. Começando pelo **Sklearn\_preprocessing\_MinMaxScaler**: uma ferramenta para normalizar dados, um passo importante no pré-processamento de dados de muitos algoritmos de aprendizado de máquinas, transformando-os valores do intervalo definido os valores mínimo e máximo (0, 1).
- b. **Sklearn\_model\_selection\_train\_test\_split**: é uma ferramenta para dividir um conjunto de dados em um conjunto de treinamento e testes. Fundamental para permitir avaliar o quão bem o seu modelo é capaz de generalizar para novos dados. Se treinasse o modelo com todo o seu conjunto de dados, não teria uma maneira fácil de testar o quão bem o modelo funciona em dados que ele nunca viu antes.
- c. As duas mais fundamentais ferramentas para o aprendizado dessa máquina o **Sklearn\_Tree\_DecisionTreeClassifier**: uma “Árvore de decisão” são algoritmo de aprendizado supervisionado usando moda (classificação) e média (regressão). Tomando-se decisão e dividindo os dados em subconjunto cada vez menores com base em diferentes critérios no caso do projeto usado para determinar a doença de um paciente com base nos resultados dos testes de sangue.
- d. O **Sklearn\_ensemble\_RandomForestClassifier**: é uma biblioteca para implementar um modelo de “Florestas aleatórias” sobre o modelo anterior “Árvore de decisão” assim criando-se múltiplas “Árvore de decisão” diferente uma dá outras, mas mantendo-se os critérios da original. Sua principal vantagem é reduzir o overfitting ao fazer a média dos resultados.
- e. **Sklearn\_svm\_SVC**: também é um modelo de aprendizado de máquinas baseado em vetores de suporte (SVM). Também supervisionada usando moda e média. Sua vantagem é que são eficazes em espaços de alta dimensão e são eficientes em termos de memória. Usamos de modo secundário no “treinamento 2” para variar e melhorar o desempenho do aprendizado.
- f. Previsão e desempenho as bibliotecas **sklearn\_metrics.accuracy\_score**, **sklearn\_metrics\_classification\_report**, **sklearn\_metrics\_confusion\_matrix**: primeiro a **Sklearn metrics accuracy** funciona calcula a acurácia do modelo de classificação que é fração de previsões corretas entre as previsões totais feitas. Tanto o **metrics report** quanto **confusion matrix** são ferramentas para avaliar o desempenho do modelo. matriz de confusão é uma tabela que mostra as previsões corretas e incorretas do seu modelo em comparação com os valores reais.

- g. A ferramenta **sklearn\_impute\_SimpleImputer**: simples mais útil para diversos casos já que ela foi feita para lidar com dados faltando. Que são comuns em muitos conjuntos de dados que pode levar a previsões imprecisas ou a um erro ao treino do modelo. Simplificando se no conjunto de dados tivemos uma pessoa que não fez todos os exames de sangue. E então preenche os valores faltantes com a média, mediana, valor mais frequente ou uma constante.
- h. As bibliotecas **sklearn\_preprocessing\_LabelEncoder**: e a biblioteca **sklearn\_preprocessing\_OneHotEncoder**: são ferramentas que converter variáveis categoria em variáveis números. Muitos algoritmos de aprendizado requerem entradas sejam numéricas. Como no caso do pro transformando (Diabete, anemia, outros) em número como (-1, 0, 1).
- i. **imblearn\_over\_sampling\_RandomOverSampler**: esta é uma ferramenta para lidar com conjuntos de dados desbalanceados. Em um conjunto de dados desbalanceado, uma classe tem muito mais exemplos do que outra. Isso pode levar a um modelo que tem um bom desempenho na classe majoritária, mas um desempenho ruim na classe minoritária. O RandomOverSampler resolve este problema ao criar exemplos na classe minoritária (O RandomOverSampler resolve este problema ao criar exemplos na classe minoritária. Ele faz isso selecionando aleatoriamente exemplos da classe minoritária com reposição (ou seja, o mesmo exemplo pode ser selecionado várias vezes). Isso aumenta o número de exemplos na classe minoritária para igualar o número de exemplos na classe majoritária, equilibrando assim o conjunto de dados.
- j. O **collections.Counter**: e uma biblioteca de contagem de dados. Ela permite que você conte as ocorrências de itens em uma coleção, como uma lista. Usamos analisar e averiguar para entender o desbalanceamento das classes depois dos testes assim permitindo que entenda a distribuição das classes nos seus dados e verifique o resultado do oversampling. Isso a ajudará a melhorar o desempenho do modelo.

## 4. Importação dos dados

```
[13] # Leitura do CSV
db_dados = pd.read_csv('/content/drive/My Drive/Atividade Carlos/Blood_samples_dataset_balanced_2(f).csv', sep=',')
```

Figura 4. (Código para a alocação do CSV em uma variável para manipulação posterior)

O processo de importação dos dados começou com a alocação do arquivo CSV no Google Drive. Após essa etapa, os dados foram importados para o projeto no Colab utilizando a biblioteca **Drive**.

Foi criada uma variável para armazenar esses dados, utilizando a vírgula (",") como separador para garantir a correta captação das informações.

Em seguida, a validade dos dados foi verificada para confirmar se todas as colunas e registros foram importados corretamente. Vale lembrar que a primeira linha do CSV é utilizada como cabeçalho das colunas.

```
# Retorna a quantidade de linhas e colunas no DataFrame.
db_dados.shape
```

(2351, 25)

Figura 5. (Código para verificação da quantidade de linhas e colunas)

Após a importação e armazenamento do dataset em uma variável no Python, os dados passaram por um processo de pré-processamento e análise que será detalhado a seguir.

## 5. Processo de Análise de Dados

No projeto desenvolvido foi feita uma limpeza e análise da base coletada para que a IA possa ter uma leitura mais simplificada e natural dos dados e assim realizar os cálculos e previsões com extrema precisão e eficiência.

Para se ter uma clareza nos dados, é preciso passar por uma série de tratamentos e análises antes de começar o treinamento da IA (Machine Learning). Remoção de valores nulos, colunas e linhas inúteis para o propósito do projeto, assim como manter um padrão nos dados (tipagem) e, obviamente, organização são algumas das boas práticas para se obter um tratamento de qualidade, permitindo um acerto maior com a IA.



### 5.1 Limpeza e Tratamento dos dados:

Foi realizado uma procura por dados específicos, no caso, nulos, e extraí-los da base de dados, como buscar por valores nulos, por exemplo.

```
# Procura por todos os valores que são nulos na base de dados.  
db_dados.isnull().sum()
```

Figura 6. (Código para verificação de valores nulos dentro do Dataset)

O resultado do código na Figura 6, foi uma lista com os valores nulos contidos em cada coluna da tabela para que seja possível realizar a remoção dos mesmos posteriormente. A extração desses dados é importante para a próxima etapa da análise, que se trata da limpeza e tratamento desses dados inconsistentes.

O tratamento dos dados é uma etapa que serve justamente para diminuir a quantidade de informações obsoletas e inúteis para a análise como um todo. Ao utilizar uma base de dados limpa e tratada, é garantido que o treinamento do modelo de regressão será bom e elimina drasticamente o excesso de informações que confundiriam e, conseqüentemente, estragariam as previsões desejadas.

Para realizar esse tratamento, é necessário saber quais informações serão excluídas, reformatadas e mantidas. A remoção de colunas inutilizadas é uma prática de suma importância para a limpeza de dados, assim como a exclusão de valores nulos que só atrapalham e poluem a base de dados.

```
# Remove todos os valores que são nulos.  
dados_limpos = db_dados.dropna()  
dados_limpos.shape
```

Figura 7. (Código para exclusão de valores nulos dentro do Dataset)

Assim que o tratamento de dados inicial for realizado os separamos e colocaremos em uma variável como mostrado na Figura 7. Uma forma simples de limpar os valores nulos da base de dados e armazenar a base limpa em uma outra variável, criando assim uma base de dados sem valores nulos (deixando um “backup” da base anterior).

A Limpeza de dados é mais um passo no pre-processing que é crucial para o tratamento de dados e aprendizado de máquina que é o foco principal dos nossos códigos. Os modelos de aprendizagem de máquinas dependem fortemente da qualidade dos dados inserido para fazer previsão e classificações precisas. Dados sujos, que podem incluir erros, valores ausentes ou inconsistentes, podem levar a resultados imprecisos e afetar negativamente o desempenho do modelo. Por mais que os nossos dados sejam retirados de um site (Kaggle) propícios para disponibilizar dados de fonte confiável (CSV). Começamos a limpeza procurando usados os comandos

(`db_dados.isnull().sum()`) e o (`db_dados.dropna()`) por valores nulos ou ausentes e então eliminado os valores suas respectivas colunas do arquivo CVS.

Em seguida analisarmos e renomeado as colunas para facilitar o entendimento nas etapas seguintes. Em seguida nos verificamos os valores total de cada linha das colunas [Doença]. Fizemos uma padronização e em seguida usamos o método “mean” do pandas para analisar e identifica os outlier para tratá-los. algumas das principais técnicas de limpeza de dados:

- **Tratamento de valores ausentes:** Isso pode ser feito através da imputação, onde os valores ausentes são substituídos por médias, medianas, modas ou através de métodos mais complexos como imputação multivariada ou algoritmos de aprendizado de máquina.
- **Identificação e remoção de outliers:** Outliers podem ser detectados usando vários métodos estatísticos, como IQR (intervalo interquartil) ou z-scores, e podem ser removidos ou corrigidos conforme necessário.
- **Correção de erros e inconsistências:** Isso inclui corrigir erros de digitação, padronizar textos (como converter todas as letras para minúsculas), e harmonizar categorias (como unificar termos que significam a mesma coisa).
- **Normalização e padronização:** Como mencionado anteriormente, essas técnicas ajudam a garantir que os dados estejam em uma escala adequada e comparável.

## 5.2 Verificação de Valores

```
# Retorna a quantidade de registro por tipo de doença
porcentagem_valor_disease = dados_limpos['Doença'].value_counts()
print(porcentagem_valor_disease)
```

Figura 8. (Código para cálculo dos valores presentes na Coluna Doença)

Após concluir a tradução dos nomes das colunas de dados, realizou-se uma análise para determinar a quantidade de casos registrados para cada doença, visando compreender melhor os dados necessários para a resolução delas.

```
# Gera uma visão tabelada do DataFrame com os dados tratados.
dados_tratados = dados_limpos
dados_tratados.head(1-2)
```

Figura 9. (Código para Alocação e visão Tabulada da variável)

Os resultados foram apresentados em forma de tabela no DataFrame, facilitando a visualização dos dados tratados e fornecendo uma visão abrangente dos dados.

```
#Selecionando apenas as colunas que serão utilizadas.
colunas_a_usar = ['Glicose', 'Colesterol', 'Hemoglobina', 'Plaquetas', 'Leucócitos',
                  'Eritrócitos', 'Hematócrito', 'Volume Corpuscular Médio',
                  'Hemoglobina Corpuscular Média',
                  'Concentração de Hemoglobina Corpuscular Média', 'Insulina', 'IMC',
                  'Pressão Arterial Sistólica', 'Pressão Arterial Diastólica',
                  'Triglicerídeos', 'Hemoglobina glicada', 'Colesterol LDL',
                  'Colesterol HDL', 'TGP (Alanina Aminotransferase)',
                  'TGO (Aspartato Aminotransferase)', 'Frequência Cardíaca', 'Creatinina',
                  'Troponina', 'Proteína C-Reativa']

df_selecionado = dados_tratados[colunas_a_usar]
```

Figura 10. (Código para seleção das colunas desejadas)

```
#media dos valores (variavel).
media_colunas = df_selecionado.mean(axis=1)
print(media_colunas)
```

Figura 11. (Código para Alocação e visão das medias ignorando a primeira linha, pois é o local do qual estão alocados os índices das colunas)

Em seguida, as colunas relevantes foram separadas e a média dos valores foi calculada, proporcionando assim dados estatísticos que oferecem insights sobre as tendências observadas nas colunas selecionadas

## 6. Normalização e padronização dos dados:

A etapa de correção e padronização de valores numéricos na base de dados é fundamental para a confiabilidade dos resultados da análise, especialmente quando envolve cálculos e operações matemáticas. A tipagem incorreta de dados numéricos pode levar a erros nos resultados e comprometer a interpretação das informações.

```
# Realiza a normalização e padronização dos valores numéricos do dataset
dados_numericos = dados_tratados.select_dtypes(include=['float64', 'int64'])
scaler = MinMaxScaler()
dados_normalizados = scaler.fit_transform(dados_numericos)
dados_normalizados_df = pd.DataFrame(dados_normalizados, columns=dados_numericos.columns)

dados_padronizados = pd.concat([dados_tratados.select_dtypes(exclude=['float64', 'int64']), dados_normalizados_df], axis=1)
dados_padronizados.head(1-5)

dados_padronizados = pd.concat([dados_tratados.select_dtypes(exclude=['float64', 'int64']), dados_normalizados_df], axis=1)
dados_padronizados.head(1-5)
```

Figura 12. (Código para normalização e “setagem” dos padrões dos dados para homogeneização da informação)

Aqui é criada uma variável que contém todos os dados do tipo “float64” e “int64” para serem escalados e padronizados. Após isso, tudo é concatenado de volta

para uma nova variável que possui os valores numéricos tratados e o resto das informações da base de dados completa.

### 6.1 Tratamento com Outliers

Para tratar os outliers, foi empregado o método comum de calcular a média e o desvio padrão, selecionando os dados necessários. Este método é uma abordagem padrão para identificar e tratar pontos discrepantes nos dados, permitindo uma análise mais precisa e a identificação de eventuais pontos que fogem da normalidade.

Ao empregar a média e o desvio padrão, os dados são tratados estatisticamente para identificar e lidar com os outliers, o que é fundamental para garantir a precisão das análises e modelos preditivos. Este método oferece uma maneira eficaz de lidar com pontos discrepantes, e apresentando-os em um gráfico para visualização

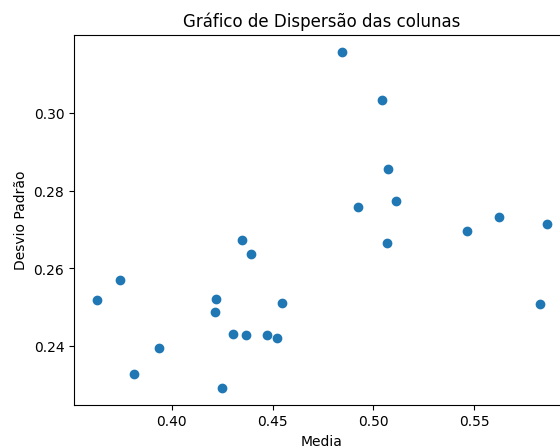


Figura 13. (Gráfico de Dispersão das Colunas, Desvio Padrão x Média dos dados)

### 6.2 Balanceamento e seu Processo

```
class_diabetes = dados_tratados['Hemoglobina glicada'].value_counts()
print(class_diabetes)
```

Hemoglobina glicada

0.120251	51
0.738542	49
0.026747	47
0.029015	47
0.466795	46
..	..
0.609198	26
0.533627	25
0.313360	25
0.237960	23
0.670665	23

Name: count, Length: 65, dtype: int64

Balanceando as classes

```
[ ] dados_tratados_balanceados = dados_tratados
```

```
[ ] dados_tratados_balanceados2 = dados_tratados.copy()
```

Figura 14. (Código referente ao Balanceamento de dados)

Após equilibrar os dados e igualar com as suas classes respectivas, foi possível apresentar os dados sem a necessidade de aplicar métodos complicados para realizar essa etapa. Este processo demonstrou a eficácia da preparação dos dados, resultando em uma análise mais precisa e confiável. Além disso, foi realizada a cópia dos dados tratados, bem como um backup desses dados para casos de urgência, garantindo a segurança e integridade das informações em situações imprevistas.

## 7. Árvore de Decisão

Uma árvore de decisão é um algoritmo de aprendizado supervisionado que pode ser usado tanto para tarefas de classificação quanto de regressão. É uma estrutura hierárquica composta por nós de decisão, folhamos e ramos. Cada nó de decisão representa uma pergunta ou condição baseada em uma característica dos dados, e os ramos representam as possíveis respostas ou resultados dessa condição. Os nós folha, por sua vez, representam a decisão final ou a previsão do modelo.

No estudo realizado, a árvore de decisão foi utilizada para classificar os resultados de hemogramas e identificar doenças como diabetes e anemia. O processo de aprendizado envolve a divisão recursiva do conjunto de dados em subconjuntos cada vez menores, com base em critérios de divisão que maximizam a separação das classes.

Para implementar a árvore de decisão, utilizamos a biblioteca Sklearn, especificamente o `DecisionTreeClassifier`. O treinamento do modelo seguiu estas etapas principais:

**Preparação dos Dados:** Após a limpeza e normalização dos dados, dividimos o conjunto de dados em partes de treinamento e teste usando o ***train\_test\_split***.

**Treinamento do Modelo:** Utilizamos o `DecisionTreeClassifier` para treinar o modelo nos dados de treinamento. Durante o treinamento, o algoritmo aprendeu a identificar padrões nos dados de hemograma que estão associados a cada doença.

**Avaliação:** Avaliamos o desempenho do modelo utilizando a métrica de acurácia, relatórios de classificação e matriz de confusão para entender sua precisão e capacidade de generalização.

### 7.1 Tomada de Decisão

O processo de tomada de decisão em uma árvore de decisão envolve a passagem dos dados através dos nós de decisão, começando pela raiz até chegar a um nó folha. Cada nó de decisão aplica uma condição aos dados e, dependendo do resultado, direciona o fluxo para um dos ramos. Esse processo continua até que os dados atinjam um nó folha, que fornece a previsão final.

No caso deste estudo, a árvore de decisão foi usada para determinar a doença de um paciente com base nos resultados dos testes de sangue. A árvore divide os dados em

subconjuntos cada vez menores, considerando diferentes critérios, como os níveis de proteínas sanguíneas. Esse processo permite identificar se um paciente está saudável ou se apresenta sinais de diabetes ou anemia.

Além disso, utilizamos o RandomForestClassifier para melhorar a robustez e reduzir o overfitting do modelo. Este método cria múltiplas árvores de decisão, cada uma treinada em diferentes subconjuntos dos dados, e combina suas previsões para obter um resultado final mais preciso.

A combinação dessas técnicas permite uma análise mais detalhada e uma tomada de decisão mais informada, proporcionando um modelo de aprendizado de máquina eficiente para a identificação de doenças com base em hemogramas.

## 8. Machine Learning, Treinamento do Modelo e Acurácia

### 8.1 Treinamento 1

Preparação dos dados:

```
X = dados_tratados_balanceados.drop('Doença', axis=1)
y = dados_tratados_balanceados['Doença'].apply(lambda x:
                                                x if x in ['Diabetes', 'Anemia']
                                                else 'Outra')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figura 15. (Código referente a preparação dos dados para processo de treinamento de máquina)

O algoritmo foi treinado utilizando uma porção da base de dados para estudo e outra porção como um teste para calcular a porcentagem (%) de sua precisão de acerto. Mais especificamente, 20% dos dados da base foram separados para servir de teste e os 80% para o algoritmo estudar.

Nesse caso, foi separado “diabetes” e “anemia” como classificações únicas e o resto (“healthy”, “thalasse” e “thromboc”) como “outra”, ou seja, após realizar a previsão, dependendo da análise do hemograma, o modelo classifica a doença do paciente como “Diabetes”, “Anemia” ou “Outra”.

Treinamento do modelo:

```
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

dados_tratados_balanceados['Previsão'] = model.predict(X)
```

Figura 16. (Código do treinamento usando o modelo DecisionTreeClassifier ou Classificador de Árvore de Decisão)

A etapa de divisão da base de dados é crucial para a avaliação do modelo preditivo. Nessa etapa, a base de dados é dividida em dois conjuntos: treino e teste. O conjunto de treino é utilizado para treinar o modelo, enquanto o conjunto de teste é utilizado para avaliar seu desempenho.

Avaliação do modelo:

```
# Demonstra a acurácia de teste e de treinamento do nosso modelo
accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_train_percent = accuracy_train * 100
print("Acurácia no conjunto de treinamento:", "{:.2f}%".format(accuracy_train_percent))

accuracy_test = accuracy_score(y_test, y_pred_test)
accuracy_test_percent = accuracy_test * 100
print("Acurácia no conjunto de teste:", "{:.2f}%".format(accuracy_test_percent))
```

Acurácia no conjunto de treinamento: 100.00%  
Acurácia no conjunto de teste: 100.00%

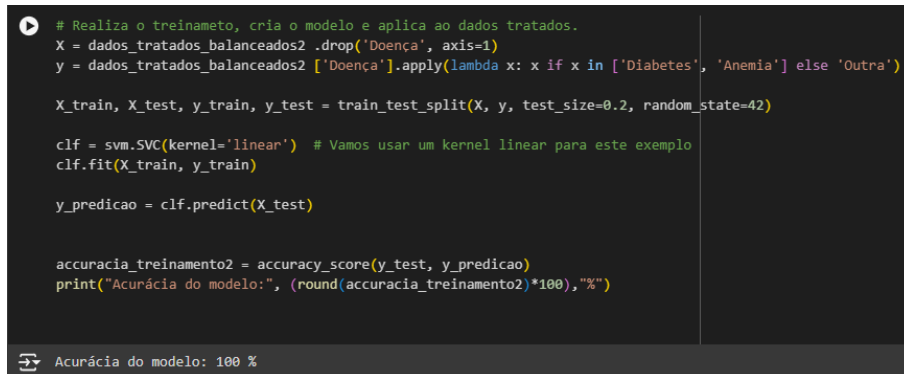
Figura 17. (Código para a avaliação do modelo DecisionTreeClassifier)

A avaliação do desempenho do modelo é crucial para determinar sua confiabilidade e aplicabilidade no projeto em questão. Através da análise das métricas de avaliação, é possível verificar se o modelo atende às necessidades do projeto e se é capaz de realizar previsões precisas e generalizáveis.

A precisão do modelo, também conhecida como acurácia, é calculada através da comparação das previsões do modelo com os valores reais presentes na porção de teste da base de dados. A porcentagem de acertos obtida nessa comparação define a precisão do modelo e sua capacidade de realizar previsões precisas.

Em alguns casos, pode ser útil comparar o desempenho do modelo com outros modelos alternativos. Essa comparação pode auxiliar na escolha do modelo mais adequado para o projeto, levando em consideração suas características e restrições.

## 8.2 Treinamento 2



```
# Realiza o treinamento, cria o modelo e aplica ao dados tratados.
X = dados_tratados_balanceados2.drop("Doença", axis=1)
y = dados_tratados_balanceados2["Doença"].apply(lambda x: x if x in ['Diabetes', 'Anemia'] else 'Outra')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf = svm.SVC(kernel='linear') # Vamos usar um kernel linear para este exemplo
clf.fit(X_train, y_train)

y_predicao = clf.predict(X_test)

acuracia_treinamento2 = accuracy_score(y_test, y_predicao)
print("Acurácia do modelo:", (round(acuracia_treinamento2)*100), "%")
```

Acurácia do modelo: 100 %

Figura 18. (Código do treinamento usando o modelo SVC e teste de acurácia)

O segundo treinamento foi conduzido de maneira similar ao primeiro, porém utilizando um modelo diferente para avaliar e comparar o desempenho entre os dois algoritmos testados. Ambos os modelos atenderam às necessidades do projeto, atingindo um desempenho máximo de 100% de precisão.

O algoritmo escolhido para esta análise foi o SVC (Support Vector Classifier), um método de aprendizado de máquina amplamente utilizado para tarefas de classificação, derivado do SVM (Support Vector Machine). O SVC é conhecido por sua capacidade de lidar com problemas complexos e não lineares, buscando identificar o hiperplano que melhor separa as classes presentes nos dados. Essa característica é particularmente útil em cenários onde as fronteiras entre classes não são claramente definidas.

Neste estudo, o SVC foi utilizado em conjunto com o Kernel Linear. A escolha do Kernel Linear se justifica pela natureza dos dados, que apresentam características que podem ser separadas por uma reta no espaço multidimensional. Isso permite que o modelo encontre uma solução eficiente para a classificação, maximizando a separação entre as classes e, consequentemente, a precisão das previsões. Além disso, o Kernel Linear tem a vantagem de ser menos complexo e computacionalmente mais eficiente em comparação com outros tipos de kernel, como o polinomial ou o radial.

Durante o processo de treinamento, diversos parâmetros foram ajustados para otimizar o desempenho do SVC. Isso incluiu a definição do parâmetro de regularização (C), que controla a margem de tolerância do classificador, e a implementação de técnicas de validação cruzada para garantir que o modelo não estivesse superajustado aos dados de treinamento. A validação cruzada, em particular, foi crucial para avaliar a robustez do modelo e sua capacidade de generalização para novos dados.

Os resultados obtidos com o SVC foram promissores, mostrando que o modelo é altamente eficaz na classificação dos hemogramas. A precisão de 100% alcançada indica que todas as amostras foram corretamente classificadas, demonstrando a capacidade do SVC em lidar com a variabilidade dos dados e fornecer previsões confiáveis.



## 9. Conclusão e Resultado

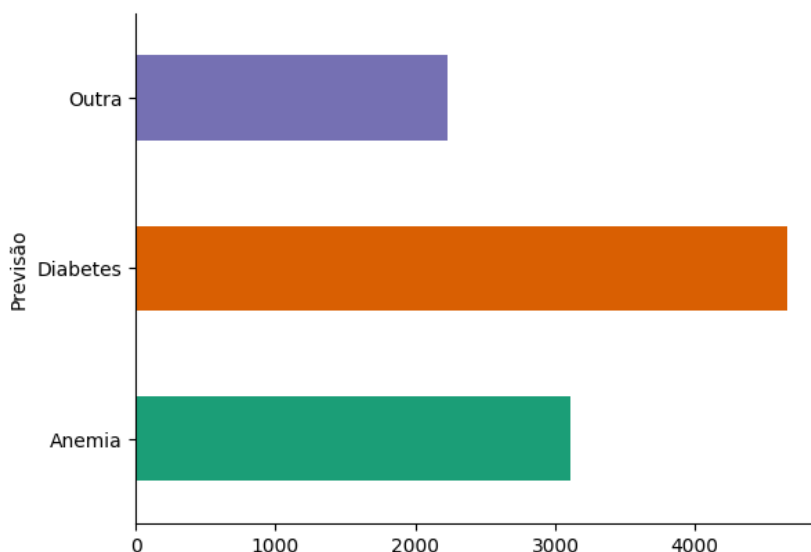


Figura 19. (Gráfico de categoria de Distribuição.)

Conclui-se que, com base na aplicação dos modelos de Árvore de Decisão e SVC aos dados de hemogramas, foi possível obter previsões com um alto grau de confiabilidade, alcançando 100% de precisão. A Figura 19 apresenta um gráfico detalhando a quantidade de hemogramas classificados de acordo com cada doença, ilustrando a eficácia dos modelos na identificação correta das patologias.

O projeto desenvolvido mostra-se altamente confiável e funcional, podendo ser aplicado em diversas situações, auxiliando profissionais da área da saúde com um pré-diagnóstico a partir da análise automática dos hemogramas dos pacientes. Essa ferramenta tem o potencial de agilizar e aprimorar a precisão dos diagnósticos médicos, contribuindo significativamente para a área da saúde.

Esses resultados positivos indicam que a combinação dos modelos de Árvore de Decisão e SVC é eficaz para a análise de dados de hemogramas, oferecendo um suporte valioso para diagnósticos médicos. Com a implementação adequada, essa abordagem pode ser expandida e adaptada para outras aplicações clínicas, potencialmente melhorando a qualidade dos cuidados de saúde e a eficiência dos processos diagnósticos.

Além disso, a utilização de técnicas avançadas de aprendizado de máquina, como o SVC, demonstra o potencial dessas tecnologias em revolucionar a prática médica, proporcionando ferramentas poderosas para a análise de grandes volumes de dados clínicos. A adoção dessas ferramentas pode levar a uma detecção mais precoce de doenças, permitindo intervenções mais rápidas e eficazes, o que, em última análise, pode salvar vidas.

A continuidade desse trabalho envolve a integração desses modelos em sistemas de informação hospitalar, permitindo que os resultados das análises de hemogramas sejam prontamente disponíveis para os profissionais de saúde. Essa integração pode incluir o

desenvolvimento de interfaces amigáveis e intuitivas para que médicos e enfermeiros possam acessar e interpretar os dados de forma eficiente.

Em resumo, os modelos desenvolvidos e testados neste projeto representam um avanço significativo na aplicação de técnicas de aprendizado de máquina na área da saúde. A precisão alcançada e a potencial aplicabilidade clínica destacam a importância da pesquisa contínua e do desenvolvimento de soluções inovadoras para enfrentar os desafios diagnósticos no campo médico.

## **Bibliografia**

**Blog EngDB**, (Data de Publicação: 02/08/2023). Disponível em: [https://blog.engdb.com.br/analises-preditivas/] Acessado em 05/05/2024.

**Escola DNC**, (Data de Publicação: 14/05/2024). Disponível em [https://www.escoladnc.com.br/blog/como-identificar-e-tratar-outliers-em-data-science/] Acessado em 28/05/2024.

**Awari**, (Data de Publicação: 31/07/2023). Disponível em” [https://awari.com.br/tratamento-de-dados-com-python-o-tratamento-de-dados-com-python/] Acessado em 20/05/2024.

**Ale George Lustosa**, (Data de Publicação: 28/12/2018). Disponível em [https://medium.com/@alegeorgelustosa/métodos-de-tratamento-para-dados-categóricos-em-python-a66f910215c7] Acessado em 20/05/2024.

**Aquare.la**, (Data de Publicação: 25/09/2017). Disponível em [https://aquare.la/o-que-sao-outliers-e-como-trata-los-em-uma-analise-de-dados/] Acessado em 25/05/2024.

**Catunda, Hashtag Treinamentos**, (Data de Publicação: 31/10/2022). “Disponível em: [https://www.hashtagtreinamentos.com/datasets-desbalanceados-ciencia-dados] Acessado em 22/05/2024.

**Costa, Alura**, (Data de Publicação: 19/01/2024). Disponível em [https://www.alura.com.br/artigos/machine-learning] Acessado em 12/05/2024.

**Melo, Sigmoidal.ai**, (Data de Publicação: 24/12/2019). Disponível em: [https://sigmoidal.ai/como-lidar-com-dados-desbalanceados/] Acessado em 05/06/2024.

**Francisco Foz, Medium**, (Data de Publicação: 14/03/2022). Disponível em [https://franciscofoz.medium.com/como-tratar-outliers-sem-excluí-los-19dd5c1ba3e6] Acessado em 10/06/2024.

**Ehab Abouelnaga, Kaggle**, (Data de Publicação: 03/03/2024). Disponível em [https://www.kaggle.com/datasets/ehababoelnaga/multiple-disease-prediction] Acessado em 10/06/2024.

**Santiago, Medium,** (Data de Publicação: 05/06/2023). Disponível em:  
[<https://medium.com/@daniele.santiago/aprenda-a-balancear-seus-dados-com-undersampling-e-oversampling-em-python-6fd87095d717>] Acessado em 11/05/2024.