

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

**Rodrigo Barbosa Leite**

**APLICAÇÃO DE ALGORITMOS DE *MACHINE LEARNING* PARA DEFINIÇÃO DE  
RETORNO MÉDICO EM PACIENTES PORTADORES DE DIABETES**

Belo Horizonte

2023

**Rodrigo Barbosa Leite**

**APLICAÇÃO DE ALGORITMOS DE *MACHINE LEARNING* PARA DEFINIÇÃO DE  
RETORNO MÉDICO EM PACIENTES PORTADORES DE DIABETES**

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

Belo Horizonte

2023

## SUMÁRIO

|  |           |
|--|-----------|
| <b>1. Introdução.....</b>                                | <b>4</b>  |
| <b>1.1. Contextualização .....</b>                       | <b>4</b>  |
| <b>1.2. O problema proposto.....</b>                     | <b>4</b>  |
| <b>2. Coleta dos dados .....</b>                         | <b>5</b>  |
| <b>3. Processamento/Tratamento de Dados .....</b>        | <b>10</b> |
| <b>4. Análise e Exploração dos Dados .....</b>           | <b>24</b> |
| <b>5. Criação de Modelos de Machine Learning .....</b>   | <b>31</b> |
| <b>6. Interpretação dos Resultados e Conclusão .....</b> | <b>36</b> |
| <b>7. Links.....</b>                                     | <b>37</b> |
| <b>REFERÊNCIAS.....</b>                                  | <b>38</b> |
| <b>APÊNDICE.....</b>                                     | <b>39</b> |

## 1. Introdução

### 1.1. Contextualização

A sociedade tecnológica atual atualiza-se em patamares cada vez mais rápidos, dando-se na área de dados foco de grandes mudanças sociais e empresariais. O termo *Big Data* já se encontra compreendido e integrado em soluções de problemas, quais forem, em corporações de várias vertentes. Dessa maneira um volume alto de profissionais e novas demandas circunscrevem e se adaptam as novas problemáticas desse ramo tecnológico, proporcionando soluções e avanços, principalmente em aplicações reais da sociedade (“O que é Big Data?”, [1]).

Isso também se dá na área médica, que, como qualquer outra área, tem seus problemas respectivos e que devem ser sanados. Visto ao avanços de inteligência artificial, sistemas capazes de trabalhar com grande volume de dados, hospitais e ou centro de saúde, podem se beneficiar das aplicabilidades do “universo *Big Data*”.

Prevenções de doenças por imagens, predições de futuras patologias, necessidades cirúrgicas ou integração em grupos de riscos, finalidades quais tendo disponibilização adequada dos dados, podem ser previstas por algoritmos de *Machine Learning* (“O que é Machine Learning?”, [2]). A capacidade de conectar a área da saúde com intervenções de soluções de algoritmos de aprendizagem de máquina, impacta imensamente pacientes e profissionais, proporcionando resultados mais rápidos e precisos.

### 1.2. O problema proposto

Para a empregar os conceitos de ML foi escolhida uma base sobre retorno médico de pacientes com diabetes. Os valores que estão nessa base possibilitam a aplicação do conceito de *Machine Learning* em seu aspecto geral, tendo valores característicos, que posteriormente serão explicados. Essa base histórica, do ano de 1998 a 2008, de hospital do Estados Unidos da América (EUA), compreende dados sobre a saúde do paciente, consumo de determinados remédios, frequência de exames laboratoriais e informações como região, gênero, entre outros.

Isso tudo tem a finalidade de indicar se o paciente teve ou não retorno ao hospital, podendo ser o retorno superior ou inferior a 30 dias. Ademais toda a configuração do *dataset* permite a aplicabilidade no conceito dentro de ML que chamamos de classificação e de otimização (“Algoritmos de Classificação: o que são e como funcionam”, [3]).

## 2. Coleta de Dados

Nessa seção você deve deixar claro onde obteve os dados, o formato e estrutura dos *datasets*, o relacionamento entre os dados, entre outros. Caso os dados tenham sido obtidos na internet, informe a data e o link em que os dados foram obtidos. Sugere-se que você crie uma tabela com a descrição de cada campo/coluna de cada *dataset* conforme o exemplo a seguir:

Para o tratamento dos dados e aplicação do algoritmo foi coletado o seguinte *dataset*: *Diabetes -US hospitals for years 1999-2008*. O mesmo foi coletado no repositório da *UCI Machine Learning Repository* (“Diabetes 130-US hospitals for years 1999-2008 Data Set”, [4]). A seguir no Quadro 1 está a explicação dos valores contidos na bases de dados.

Quadro 1. Descrição *dataset*

| Nome da coluna | Descrição   | Tipo       |
|----------------|---|------------|
| Encounter ID   | Identificador único de um encontro                                | Numérico   |
| Patient number | Identificador único de um paciente                                | Numérico   |
| Race           | Valores: caucasiano, asiático, afro-americano, hispânico e outros | Categórico |
| Gender         | Valores: masculino, feminino e desconhecido/inválido              | Categórico |
| Age            | Agrupados em intervalos de 10 anos: [0, 10), [10, 20), . . . ,    | Categórico |

|                      |  |            |
|----------------------|--|------------|
|                      | [90, 100)  |            |
| Weight               | Peso em libras   | Numérico   |
| Admission type       | Identificador inteiro correspondente a 9 valores distintos, por exemplo, emergência, urgência, eletivo, recém-nascido e não disponível             | Categórico |
| Discharge dispositon | Identificador inteiro correspondente a 29 valores distintos, por exemplo, descarregados para casa, expirou e não está disponível                   | Categórico |
| Admission source     | Identificador inteiro correspondente a 21 valores distintos, por exemplo, encaminhamento médico, sala de emergência e transferência de um hospital | Categórico |
| Time in hospital     | Número inteiro de dias entre a admissão e a alta   | Numérico   |
| Payer code           | Identificador inteiro correspondente a 23 valores distintos, por exemplo, Blue Cross\Blue Shield, Medicare e autopagamento                         | Categórico |
| Medical speciality   | valores, por exemplo, cardiologia, medicina interna, família\clínica geral   | Categórico |

|                             |   |            |
|-----------------------------|---|------------|
|                             | e<br>cirurgião  |            |
| Number of lab procedures    | Número de exames laboratoriais realizados durante o encontro                                      | Numérico   |
| Number of procedures        | Número de procedimentos (exceto exames laboratoriais) realizados durante o encontro               | Numérico   |
| Number of medications       | Número de nomes genéricos distintos administrados durante o encontro                              | Numérico   |
| Number of outpatient visits | Número de consultas ambulatoriais do paciente no ano anterior ao encontro                         | Numérico   |
| Number of emergency visits  | Número de visitas de emergência do paciente no ano anterior ao encontro                           | Numérico   |
| Number of inpatient visits  | Número de visitas de internação do paciente no ano anterior ao encontro                           | Numérico   |
| Diagnosis 1                 | O diagnóstico primário (codificado como os três primeiros dígitos do CID9); 848 valores distintos | Categórico |
| Diagnosis 2                 | Diagnóstico secundário (codificado como os três primeiros dígitos do CID9); 923 valores distintos | Categórico |
| Diagnosis 3                 | Diagnóstico secundário adicional (codificado como   | Categórico |

|                           |  |            |
|---------------------------|--|------------|
|                           | os três primeiros dígitos do CID9); 954 distinto valores   |            |
| Number of diagnoses       | Número de diagnósticos inseridos no sistema  | Numérico   |
| Glucose serum test result | Indica o intervalo do resultado ou se o teste não foi realizado. Valores: ">200," ">300," "normal" e "nenhum" se não for medido  | Categórico |
| A1c test result           | Indica o intervalo do resultado ou se o teste não foi realizado. Valores: ">8" se o resultado foi superior a 8%, ">7" se o resultado foi superior a 7% mas inferior a 8%, "normal" se o resultado for inferior a 7% e "nenhum" se não for medido | Categórico |
| Change of medications     | Indica se houve alteração nos medicamentos para diabéticos (posologia ou genéricos) nome). Valores: "mudar" e "sem mudança"  | Categórico |
| Diabetes medications      | Indica se foi prescrito algum medicamento para diabéticos. Valores: "sim" e  | Categórico |



|                             |  |            |
|-----------------------------|--|------------|
|                             | “não”  |            |
| 24 features for medications | <p>Para os nomes genéricos: metformina, repaglinida, nateglinida, clorpropamida, glimepirida, acetohexamida, glipizida, gliburida, tolbutamida, pioglitazona, rosiglitazona, acarbose, miglitol, troglitazona, tolazamida, examida, sitagliptina, insulina, gliburida-metformina, glipizida-metformina, glimepirida-pioglitazona, metformina-rosiglitazona e metformina-pioglitazona, o recurso indica se</p> <p>o medicamento foi prescrito ou houve alteração na posologia. Valores: “up” se a dosagem foi aumentada durante o encontro, “down” se a dosagem foi diminuída, “steady” se a dosagem não mudou e “não” se o medicamento não foi prescrito</p> | Categórico |
| Readmitted                  | <p>Dias para reinternação do paciente. Valores: “&lt;30” se o paciente foi readmitido em menos de 30 dias, “&gt;30” se o paciente</p>  | Categórico |

|  |   |  |
|--|---|--|
|  | foi reinternado em mais de 30 dias e “Não” para nenhum registro de readmissão |  |
|  |   |  |

Fonte: Strack, et al., 2014 [5]

### 3. Processamento/Tratamento de Dados

A etapa de processamento e tratamento dos dados foi realizada em linguagem Python (versão 3.8.8) , utilizando a IDE do Jupyter Notebook (versão 6.3.0) (“Jupyter Project Documentation”, [5]) . Ademais todo o tratamento necessitou das seguintes bibliotecas, utilizadas no Python:

- *Pandas*: possibilita a leitura, manipulação e tratamento dos dados em sua estrutura;
- *Numpy*: para trabalhar com *arrays* e ter disponibilidade de tratamento numérico
- *Requests*: para trabalhar com requisições *API*;
- *Io*: para lidar com entradas e saídas de dados;
- *Zipfile*: para trabalhar com arquivos compactados.

Imagem 1. Bibliotecas importadas para tratamento

```
import pandas as pd
import requests
from zipfile import ZipFile
from io import BytesIO
import numpy as np
pd.set_option('display.max_columns', None)
```

Fonte: Própria autor

Após a importação das bibliotecas necessário se dá coleta do *dataframe*

Imagem 2. Coleta dos dados

```
response = requests.get("https://archive.ics.uci.edu/ml/machine-learning-databases/00296/dataset_diabetes.zip")
files = ZipFile(BytesIO(response.content))
df = pd.read_csv(files.open("dataset_diabetes/diabetic_data.csv"))
```

Fonte: própria autor

Na imagem acima usamos o método *get* da biblioteca *requests*, para acessar o *endpoint* onde está o arquivo compactado do *dataframe*, colocando-o numa variável chamada *response*. Após criamos a variável *files*, onde colocamos a entrada do arquivo *.zip*, pela funcionalidade da *BytesIO*. Por fim criamos a variável *df* onde utilizamos o *pandas* para ler o csv *diabetes\_data.csv*.

Com o comando *pd.set\_option('display.max\_columns', None)* consigo visualizar dentro da *IDE* toda a extensão das colunas. A seguir executo o comando para imprimir o *df* para visualização.

Imagem 3. *Dataframe*

df

|        | encounter_id | patient_nbr | race            | gender | age     | weight | admission_type_id | discharge_disposition_id | admission_source_id | time_in_hospital | pa  |
|--------|--------------|-------------|-----------------|--------|---------|--------|-------------------|--------------------------|---------------------|------------------|-----|
| 0      | 2278392      | 8222157     | Caucasian       | Female | [0-10)  | ?      | 6                 | 25                       | 1                   | 1                |     |
| 1      | 149190       | 55629189    | Caucasian       | Female | [10-20) | ?      | 1                 | 1                        | 7                   | 3                |     |
| 2      | 64410        | 86047875    | AfricanAmerican | Female | [20-30) | ?      | 1                 | 1                        | 7                   | 2                |     |
| 3      | 500364       | 82442376    | Caucasian       | Male   | [30-40) | ?      | 1                 | 1                        | 7                   | 2                |     |
| 4      | 16680        | 42519267    | Caucasian       | Male   | [40-50) | ?      | 1                 | 1                        | 7                   | 1                |     |
| ...    | ...          | ...         | ...             | ...    | ...     | ...    | ...               | ...                      | ...                 | ...              | ... |
| 101761 | 443847548    | 100162476   | AfricanAmerican | Male   | [70-80) | ?      | 1                 | 3                        | 7                   | 3                |     |
| 101762 | 443847782    | 74694222    | AfricanAmerican | Female | [80-90) | ?      | 1                 | 4                        | 5                   | 5                |     |
| 101763 | 443854148    | 41088789    | Caucasian       | Male   | [70-80) | ?      | 1                 | 1                        | 7                   | 1                |     |
| 101764 | 443857166    | 31693671    | Caucasian       | Female | [80-90) | ?      | 2                 | 3                        | 7                   | 10               |     |
| 101765 | 443867222    | 175429310   | Caucasian       | Male   | [70-80) | ?      | 1                 | 1                        | 7                   | 6                |     |

101766 rows × 50 columns

Fonte: própria autor

Ademais conseguimos obter informações mais precisas sobre os dados com o comando *info*. Podemos analisar que possui 50 colunas com 101.766 instâncias, sendo que o tipo das colunas se divide entre *int64* e *object*.

Imagem 4. Informações *dataframe*

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 50 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   encounter_id                          101766 non-null int64
1   patient_nbr                           101766 non-null int64
2   race                                  101766 non-null object
3   gender                                101766 non-null object
4   age                                   101766 non-null object
5   weight                                101766 non-null object
6   admission_type_id                     101766 non-null int64
7   discharge_disposition_id              101766 non-null int64
8   admission_source_id                   101766 non-null int64
9   time_in_hospital                      101766 non-null int64
10  payer_code                             101766 non-null object
11  medical_specialty                     101766 non-null object
12  num_lab_procedures                    101766 non-null int64
13  num_procedures                        101766 non-null int64
14  num_medications                       101766 non-null int64
15  number_outpatient                     101766 non-null int64
16  number_emergency                      101766 non-null int64
17  number_inpatient                      101766 non-null int64
18  diag_1                                101766 non-null object
19  diag_2                                101766 non-null object
20  diag_3                                101766 non-null object
21  number_diagnoses                      101766 non-null int64
22  max_glu_serum                         101766 non-null object
23  A1Cresult                             101766 non-null object
24  metformin                             101766 non-null object
25  repaglinide                           101766 non-null object
26  nateglinide                           101766 non-null object
27  chlorpropamide                        101766 non-null object
28  glimepiride                           101766 non-null object
29  acetohexamide                        101766 non-null object
30  glipizide                             101766 non-null object
31  glyburide                             101766 non-null object
32  tolbutamide                           101766 non-null object
33  pioglitazone                          101766 non-null object
34  rosiglitazone                         101766 non-null object
35  acarbose                              101766 non-null object
36  miglitol                              101766 non-null object
37  troglitazone                          101766 non-null object
38  tolazamide                            101766 non-null object
39  examide                               101766 non-null object
40  citoglipton                           101766 non-null object
41  insulin                               101766 non-null object
42  glyburide-metformin                   101766 non-null object
43  glipizide-metformin                   101766 non-null object
44  glimepiride-pioglitazone              101766 non-null object
45  metformin-rosiglitazone               101766 non-null object
46  metformin-pioglitazone                101766 non-null object
47  change                                101766 non-null object
48  diabetesMed                           101766 non-null object
49  readmitted                            101766 non-null object
dtypes: int64(13), object(37)
memory usage: 38.8+ MB
```

Fonte: própria autor

A seguir é necessário verificar os valores únicos das colunas categóricas e numéricas, para verificar a ocorrência de valores *missing*. Para isso precisa-se selecionar as colunas numéricas das categóricas, para melhor visualização dentro do laço *for*. Tal laço aplica a função *unique* que possibilita a visualização dos valores únicos de cada coluna.

Imagem 5. Código para gerar valores únicos

```
#variavel para pegar as colunas do df em forma de lista
cols = df.columns

#variavel para selecionar colunas numéricas do df
num_cols = df._get_numeric_data().columns

#variavel que diminui as colunas numéricas das colunas, obtendo-se as colunas categóricas
cat_col = list(set(cols) - set(num_cols))

#laço for para valores únicos categóricos
for coluna in cat_col:
    print(coluna + str(df[coluna].unique()) + '\n')

#laço for para valores únicos numéricos
for col in num_cols:
    print( str(type(col)) + col + str(df[col].unique()) + '\n')
```

Fonte: própria autor

Imagem 6. Valores únicos colunas categóricas

```
: for coluna in cat_col:
    print(coluna + str(df[coluna].unique()) + '\n')

acarbose['No' 'Steady' 'Up' 'Down']

metformin-rosiglitazone['No' 'Steady']

tolbutamide['No' 'Steady']

metformin-pioglitazone['No' 'Steady']

glipizide['No' 'Steady' 'Up' 'Down']

weight['?' '[75-100]' '[50-75]' '[0-25]' '[100-125]' '[25-50]' '[125-150]'
'[175-200]' '[150-175]' '>200']

troglitazone['No' 'Steady']

A1Cresult['None' '>7' '>8' 'Norm']

nateglinide['No' 'Steady' 'Down' 'Up']
```

Fonte: própria autor

Imagem 7. Valores únicos colunas numéricas

```

: for col in num_cols:
    print( str(type(col)) + col + str(df[col].unique()) + '\n')

<class 'str'>encounter_id[ 88792836  88986678  89032962 ... 437959022 439577312 439606454]

<class 'str'>patient_nbr[100654011  58682736  69250302  62022042  30950811  58763808  63813420
84387969 110949741  49167621  56356434  109527102  78098634  21850101
99090900  79327116  83177253  106419474  83232054  108730161  49469211
77730093 114086232  84746214  20875761  38644884  3749778  63002484
107551395 101722149 110721573  99154062  63872937 110306691  85456377
75016944  5347143  70932573  87758757  16352748 108564255  58151448
113098887  80369244  97780491  50173128 113262084  67316697 109737909
25685226 100627389 104373990  61658370  91441701  39852603  87254991
74547189  70100586  2554614  87701625  2728863 105968844  46422936
72874890  80348409  77525172  88485201  22855590 100179639  86806809
103359609 16853211  60125787 100592829  75753909 16622676 107613405
85160916  41828310 113203098  75511719  47738574  63453312  52100514
53485947  38785977 114508206  65129058  94630329  64493568 102548961
3771018  68278050  66399966  62785548  98244774  88116966 113452965
5303844  78806592  87737328  28219392  62105535  1497051 108595224
105565014 72537417  60014520 101705697  37413621  37247364  74804031
91503747  59484456  75963663  59722938  69795441  79927353 100912059

```

Fonte: própria autor

Com isso é possível observar que existem valores como “?” e “Unknown/Invalid”. A fim de deixar a base mais correta possível e para fins didáticos de aplicação do algoritmo de KNN exclui-se as linhas onde estão tais valores, não substituindo por outros, evitando-se a atribuição de pesos errôneos ao *df*.

Imagem 7. Valore excluídos

```

#valores a serem excluídos
drop_values = ["?", "Unknown/Invalid"]

#lógica para excluir linhas que possuem valores na lista drop_values
df = df[~df.isin(drop_values).any(axis=1)]

```

Fonte: própria autor

Após a exclusão dos valores acima é necessário executar novamente o laço for para ratificar os valores únicos. Logo visualizamos que o *dataframe* passa a ter ainda 50 colunas e a 1043 linhas.

Imagem 8. *Shape df*

```
df.shape
(1043, 50)
```

Fonte: própria autor

Ademais para fins de otimização exclui-se as colunas que não agregam valores informacionais e pesos para o resultado. São as seguintes colunas: *payer\_code*, *medical\_speciality*, *encouter\_id*, *admission\_type\_id*, *discharge\_disposition\_id* e *patient\_nbr*. Usa-se o comando *drop*, que *deleta as colunas do df*.

Imagem 9. Colunas excluídas

```
colunas_excluidas = ['payer_code'
, 'medical_specialty'
, 'encounter_id'
, 'admission_type_id'
, 'discharge_disposition_id'
, 'patient_nbr']

df = df.drop(colunas_excluidas, axis = 1)
```

Fonte: própria autor

Para facilitar o processamento do algoritmo atribuímos valores numéricos para as variáveis categóricas. Dessa maneira cada categoria dentro de uma coluna (ou variável) recebeu um peso inteiro e aleatório específico para cada ocorrência sua, isso realizado através da função *map*. Veja o quadro a seguir.

Quadro 2. Atribuição de valores as variáveis categóricas

| Nome da coluna | Atribuição dos valores  |
|----------------|---|
| Race           | <ul style="list-style-type: none"> <li>• Caucasian = 0</li> <li>• AfricanAmerican = 1</li> <li>• Asian = 2</li> </ul> |

|             |  |
|-------------|--|
|             | <ul style="list-style-type: none"> <li>• Hispanic = 3</li> <li>• Other = 4</li> </ul>  |
| Gender      | <ul style="list-style-type: none"> <li>• Male = 0</li> <li>• Female = 1</li> </ul>   |
| Age         | <ul style="list-style-type: none"> <li>• [0-10) = 0</li> <li>• [10-20) = 1</li> <li>• [20-30) = 2</li> <li>• [30-40) = 3</li> <li>• [40-50) = 4</li> <li>• [50-60) = 5</li> <li>• [60-70) = 6</li> <li>• [70-80) = 7</li> <li>• [80-90) = 8</li> <li>• [90-100) = 9</li> </ul> |
| Weight      | <ul style="list-style-type: none"> <li>• [0-25) = 0</li> <li>• [25-50) = 1</li> <li>• [50-75) = 2</li> <li>• [75-100) = 3</li> <li>• [100-125) = 4</li> <li>• [125-150) = 5</li> <li>• [150-175) = 6</li> <li>• [175-200) = 7</li> </ul>                                       |
| Diagnosis 1 | O diagnóstico primário (codificado como os três primeiros dígitos do CID9); 848 valores distintos  |
| Diagnosis 2 | Diagnóstico secundário (codificado como os três primeiros dígitos do CID9); 923 valores distintos  |



|                             |   |
|-----------------------------|---|
| Diagnosis 3                 | Diagnóstico secundário adicional (codificado como os três primeiros dígitos do CID9); 954 distinto valores  |
| max_glu_serum               | <ul style="list-style-type: none"> <li>• None = 0</li> <li>• Norm = 1</li> <li>• &gt;200 = 2</li> <li>• &gt;300 = 3</li> </ul>  |
| A1c test result             | <ul style="list-style-type: none"> <li>• None = 0</li> <li>• Norm = 1</li> <li>• &gt;8 = 2</li> <li>• &gt;7 = 3</li> </ul>  |
| Change of medications       | <ul style="list-style-type: none"> <li>• No = 0</li> <li>• Ch = 1</li> </ul>  |
| Diabetes medications        | <ul style="list-style-type: none"> <li>• No = 0</li> <li>• Yes = 1</li> </ul>   |
| 24 features for medications | Para os nomes genéricos: metformina, repaglinida, nateglinida, clorpropamida, glimepirida, acetohexamida, glipizida, gliburida, tolbutamida, pioglitazona, rosiglitazona, acarbose, miglitol, troglitazona, tolazamida, examida, sitagliptina, insulina, gliburida-metformina, glipizida-metformina, glimepirida-pioglitazona, metformina-rosiglitazona e |

|            |  |
|------------|--|
|            | <p>metformina-pioglitazona, o recurso indica se o medicamento foi prescrito ou houve alteração na posologia. Valores: “up” se a dosagem foi aumentada durante o encontro, “down” se a dosagem foi diminuída, “steady” se a dosagem não mudou e “não” se o medicamento não foi prescrito. Variando conforme Imagem 13 e 12.</p> |
| Readmitted | <ul style="list-style-type: none"> <li>• NO = 0</li> <li>• &lt;30 = 1</li> <li>• &gt;30 = 1</li> </ul>   |

Fonte: própria autor

Mesmo após a atribuição de valores ainda garanto a inexistência de valores nulos, dropando-os e verificando que ainda restaram mais de mil instâncias, o que corrobora para uma base sólida de dados para se aplicar ML.

Imagem 10. Drop

```
df = df.dropna()
```

```
df.shape
```

```
(1021, 44)
```

Fonte: Própria autor

Ademais para a colunas as seguintes colunas: *diag\_1*, *diag\_2* e *diag\_3*; deu-se um tratamento diferente, pois de acordo com o artigo existem o seguintes intervalos de valores para cada patologia, que são: circulatória, respiratória, digestiva, diabetes, injurias, musculoesquelética, geniturinário, neoplasmática e outras. Cada intervalo de valor dentro das patologias respectivas foi substituído pelo nome da patologia no *dataset*.

Imagem 11: Descrição colunas diagnósticas

| Group name      | icd9 codes             | Number of encounters | % of encounter | Description   |
|-----------------|------------------------|----------------------|----------------|---|
| Circulatory     | 390–459, 785           | 21,411               | 30.6%          | Diseases of the circulatory system  |
| Respiratory     | 460–519, 786           | 9,490                | 13.6%          | Diseases of the respiratory system  |
| Digestive       | 520–579, 787           | 6,485                | 9.3%           | Diseases of the digestive system  |
| Diabetes        | 250.xx                 | 5,747                | 8.2%           | Diabetes mellitus   |
| Injury          | 800–999                | 4,697                | 6.7%           | Injury and poisoning  |
| Musculoskeletal | 710–739                | 4,076                | 5.8%           | Diseases of the musculoskeletal system and connective tissue                            |
| Genitourinary   | 580–629, 788           | 3,435                | 4.9%           | Diseases of the genitourinary system  |
| Neoplasms       | 140–239                | 2,536                | 3.6%           | Neoplasms   |
|                 | 780, 781, 784, 790–799 | 2,136                | 3.1%           | Other symptoms, signs, and ill-defined conditions                                       |
|                 | 240–279, without 250   | 1,851                | 2.6%           | Endocrine, nutritional, and metabolic diseases and immunity disorders, without diabetes |
|                 | 680–709, 782           | 1,846                | 2.6%           | Diseases of the skin and subcutaneous tissue  |
|                 | 001–139                | 1,683                | 2.4%           | Infectious and parasitic diseases   |
| Other (17.3%)   | 290–319                | 1,544                | 2.2%           | Mental disorders  |
|                 | E–V                    | 918                  | 1.3%           | External causes of injury and supplemental classification                               |
|                 | 280–289                | 652                  | 0.9%           | Diseases of the blood and blood-forming organs  |
|                 | 320–359                | 634                  | 0.9%           | Diseases of the nervous system  |
|                 | 630–679                | 586                  | 0.8%           | Complications of pregnancy, childbirth, and the puerperium                              |
|                 | 360–389                | 216                  | 0.3%           | Diseases of the sense organs  |
|                 | 740–759                | 41                   | 0.1%           | Congenital anomalies  |

Fonte: Strack, et al., 2014 [4]

Para cada patologia, menos a categoria *outras* e *diabetes*, criou uma lista através da biblioteca *Numpy*, onde atribui-se os valores referentes na tabelas do artigo, tratando-as para que seu valores tivessem saída no formato de *string*.

Imagem 12. Atribuição valores variáveis

```

: df['glipizide-metformin'] = df['glipizide-metformin'].map({'No': 0}) #[ 'No' ]

: df['change'] = df['change'].map({'No': 0, 'Ch': 1}) #[ 'Ch' 'No' ]

: df['tolbutamide'] = df['tolbutamide'].map({'No': 0}) #[ 'No' ]

: df['examide'] = df['examide'].map({'No': 0}) #[ 'No' ]

: df['pioglitazone'] = df['pioglitazone'].map({'No': 0, 'Steady': 1, 'Up': 2, 'Down': 3}) #[ 'No' 'Steady' 'Up' 'Down' ]

: df['diabetesMed'] = df['diabetesMed'].map({'No': 0, 'Yes': 1}) #[ 'Yes' 'No' ]

: df['citoglipton'] = df['citoglipton'].map({'No': 0}) #[ 'No' ]

: df['insulin'] = df['insulin'].map({'Steady': 0, 'No': 1, 'Down': 2, 'Up': 3}) #[ 'Steady' 'No' 'Down' 'Up' ]

: df['troglitazone'] = df['troglitazone'].map({'No': 0}) #[ 'No' ]

: df['glinepiride-pioglitazone'] = df['glinepiride-pioglitazone'].map({'No': 0}) #[ 'No' ]

: df['glyburide-metformin'] = df['glyburide-metformin'].map({'No': 0}) #[ 'No' ]

: df['metformin-pioglitazone'] = df['metformin-pioglitazone'].map({'No': 0}) #[ 'No' ]

: df['miglitol'] = df['miglitol'].map({'No': 0}) #[ 'No' ]

: df['glyburide'] = df['glyburide'].map({'No': 0, 'Steady': 1, 'Up': 2, 'Down': 3}) #[ 'No' 'Steady' 'Up' 'Down' ]

: df['tolazamide'] = df['tolazamide'].map({'No': 0}) #[ 'No' ]

: df['max_glu_serum'] = df['max_glu_serum'].map({'None': 0, 'Norm': 1, '>200': 2, '>300': 3}) #[ 'None' '>300' 'Norm' '>200' ]

: df['nateglinide'] = df['nateglinide'].map({'No': 0, 'Steady': 1}) #[ 'No' 'Steady' ]

```

Fonte: própria autor

Imagem 13. Atribuição valores variáveis restantes

```

df['metformin-rosiglitazone'] = df['metformin-rosiglitazone'].map({'No': 0}) #['No']

df['glinepiride'] = df['glinepiride'].map({'No': 0, 'Steady': 1, 'Down': 2, 'Up': 3}) #['No' 'Steady' 'Down' 'Up']

df['metformin'] = df['metformin'].map({'No': 0, 'Steady': 1, 'Up': 2, 'Down': 3}) #['No' 'Steady' 'Up' 'Down']

df['A1Cresult'] = df['A1Cresult'].map({'None': 0, 'Norm': 1, '>8': 2, '>7': 3}) #['None' 'Norm' '>8' '>7']

df['chlorpropamide'] = df['chlorpropamide'].map({'No': 0}) #['No']

df['acetoexamide'] = df['acetoexamide'].map({'No': 0}) #['No']

df['rosiglitazone'] = df['rosiglitazone'].map({'No': 0, 'Steady': 1})

df['glipizide'] = df['glipizide'].map({'Steady': 0, 'No': 1, 'Up': 2, 'Down': 3})

df['acarbose'] = df['acarbose'].map({'No': 0, 'Steady': 1})

df['repaglinide'] = df['repaglinide'].map({'No': 0, 'Steady': 1, 'Up': 2})

df['readmitted'] = df['readmitted'].map({'NO': 0, '<30': 1, '>30': 1})

df['gender'] = df['gender'].map({'Male': 0, 'Female': 1})

df['age'] = df['age'].map({'[0-10)': 0, '[10-20)': 1, '[20-30)': 2, '[30-40)': 3, '[40-50)': 4, '[50-60)': 5, '[60-70)': 6, '[70-80)': 7})

df['race'] = df['race'].map({'Caucasian': 0, 'AfricanAmerican': 1, 'Asian': 2, 'Hispanic': 3, 'Other': 4})

df['weight'] = df['weight'].map({'[0-25)': 0, '[25-50)': 1, '[50-75)': 2, '[75-100)': 3, '[100-125)': 4, '[125-150)': 5, '[150-175)': 6})

```

Fonte: própria autor

Imagem 14. Atribuição valores patologias

```

Circulatory = np.arange(390,460).tolist() + [785]
Circulatory_str = [str(elemento) for elemento in Circulatory]

Respiratory = np.arange(460,520).tolist() + [786]
Respiratory_str = [str(elemento) for elemento in Respiratory]

Digestive = np.arange(520,580).tolist() + [787]
Digestive_str = [str(elemento) for elemento in Digestive]

Injury = np.arange(800,1000).tolist() + [786]
Injury_str = [str(elemento) for elemento in Injury]

Musculoskeletal = np.arange(710,740).tolist() + [786]
Musculoskeletal_str = [str(elemento) for elemento in Musculoskeletal]

Genitourinary = np.arange(580,630).tolist() + [788]
Genitourinary_str = [str(elemento) for elemento in Genitourinary]

Neoplasms = np.arange(140,240).tolist()
Neoplasms_str = [str(elemento) for elemento in Neoplasms]

```

Fonte: própria autor

Para a categoria *diabetes*, como se inicia com o valor “250” atribui-se a seguinte lógica:

Imagem 15. Atribuição valor fixo de “250”

```
df.loc[df['diag_1'].str.startswith('250'), 'diag_1'] = 'Diabetes'
df.loc[df['diag_2'].str.startswith('250'), 'diag_2'] = 'Diabetes'
df.loc[df['diag_3'].str.startswith('250'), 'diag_3'] = 'Diabetes'
```

Fonte: própria autor

No código a seguir a lógica do preenchimento das colunas com as listas de patologias:

Imagem 16. Lista patologias

```
df.loc[df['diag_1'].isin(Circulatory_str), 'diag_1'] = 'Circulatory'
df.loc[df['diag_1'].isin(Respiratory_str), 'diag_1'] = 'Respiratory'
df.loc[df['diag_1'].isin(Digestive_str), 'diag_1'] = 'Digestive'
df.loc[df['diag_1'].isin(Injury_str), 'diag_1'] = 'Injury'
df.loc[df['diag_1'].isin(Musculoskeletal_str), 'diag_1'] = 'Musculoskeletal'
df.loc[df['diag_1'].isin(Genitourinary_str), 'diag_1'] = 'Genitourinary'
df.loc[df['diag_1'].isin(Neoplasms_str), 'diag_1'] = 'Neoplasms'

df.loc[df['diag_2'].isin(Circulatory_str), 'diag_2'] = 'Circulatory'
df.loc[df['diag_2'].isin(Respiratory_str), 'diag_2'] = 'Respiratory'
df.loc[df['diag_2'].isin(Digestive_str), 'diag_2'] = 'Digestive'
df.loc[df['diag_2'].isin(Injury_str), 'diag_2'] = 'Injury'
df.loc[df['diag_2'].isin(Musculoskeletal_str), 'diag_2'] = 'Musculoskeletal'
df.loc[df['diag_2'].isin(Genitourinary_str), 'diag_2'] = 'Genitourinary'
df.loc[df['diag_2'].isin(Neoplasms_str), 'diag_2'] = 'Neoplasms'

df.loc[df['diag_3'].isin(Circulatory_str), 'diag_3'] = 'Circulatory'
df.loc[df['diag_3'].isin(Respiratory_str), 'diag_3'] = 'Respiratory'
df.loc[df['diag_3'].isin(Digestive_str), 'diag_3'] = 'Digestive'
df.loc[df['diag_3'].isin(Injury_str), 'diag_3'] = 'Injury'
df.loc[df['diag_3'].isin(Musculoskeletal_str), 'diag_3'] = 'Musculoskeletal'
df.loc[df['diag_3'].isin(Genitourinary_str), 'diag_3'] = 'Genitourinary'
df.loc[df['diag_3'].isin(Neoplasms_str), 'diag_3'] = 'Neoplasms'
```

Fonte: própria autor

Para o preenchimento da coluna *outras* criou-se a lista *patology*, que contém as patologias e caso algum valor não esteja nas colunas, dentre os valores da lista, é preenchido com o valor *Other*. E também se exclui a coluna *encounter\_id* que servia de chave para aplicar os cortes e manipulações no *dataframe*.

Imagem 17. Lista “Other”

```

patology = ["Circulatory", "Respiratory", "Digestive", "Injury", "Musculoskeletal", "Genitourinary", "Neoplasms"]

df.loc[~df['diag_1'].isin(patology), 'diag_1'] = 'Other'

df.loc[~df['diag_2'].isin(patology), 'diag_2'] = 'Other'

df.loc[~df['diag_3'].isin(patology), 'diag_3'] = 'Other'

colunas_excluidas_2 = ['encounter_id']

df = df.drop(colunas_excluidas_2, axis = 1)

```

Fonte: própria autor

Para a última etapa do tratamento, fazendo com que os dados estejam todos com valores numéricos, aplica-se o seguinte dicionário com valores respectivos:

Imagem 18. Iteração nas colunas *diag\_1*, *diag\_2* e *diag\_3*

```

diags = ['diag_1', 'diag_2', 'diag_3']
for diag in diags:
    df[diag] = df[diag].map({'Circulatory': 0, 'Respiratory': 1, 'Digestive': 2, 'Injury': 3,
                             'Musculoskeletal': 4, 'Genitourinary': 5, 'Neoplasms': 6, 'Other': 7})

```

Fonte: própria autor

E para confirmar que se a quantidade de valores unitários dentre as colunas *diag\_1*, *diag\_2* e *diag\_3*, aplicou-se a seguinte lógica, obtendo-se o valor *true*:

Imagem 19. Verifica das colunas

```

len(df["diag_1"].unique()) == len(df["diag_2"].unique()) == len(df["diag_3"].unique())

True

```

Fonte: própria autor

Todo esse tratamento foi para obter-se uma base de dados sólida e que facultasse a aplicação dos algoritmos de *machine learning*. Ratificando mais uma vez que o *dataframe*, mesmo com a exclusão de valores nulos ou sem significado, contém mais de mil instâncias, possibilitando uma varredura satisfatória do algoritmo.

Imagem 20. *Dataframe*

df

|        | race | gender | age | weight | time_in_hospital | num_lab_procedures | num_procedures | num_medications | number_outpatient |
|--------|------|--------|-----|--------|------------------|--------------------|----------------|-----------------|-------------------|
| 27068  | 0    | 1      | 7   | 3      | 10               | 65                 | 1              | 28              | 1                 |
| 27128  | 0    | 0      | 8   | 2      | 6                | 73                 | 0              | 16              | 0                 |
| 27147  | 0    | 0      | 6   | 4      | 2                | 58                 | 3              | 12              | 0                 |
| 27186  | 0    | 0      | 4   | 3      | 3                | 33                 | 0              | 7               | 4                 |
| 27236  | 0    | 0      | 5   | 4      | 2                | 5                  | 4              | 11              | 0                 |
| ...    | ...  | ...    | ... | ...    | ...              | ...                | ...            | ...             | ...               |
| 101238 | 1    | 0      | 7   | 3      | 1                | 69                 | 1              | 20              | 0                 |
| 101330 | 0    | 0      | 3   | 4      | 4                | 49                 | 1              | 7               | 0                 |
| 101340 | 0    | 0      | 6   | 4      | 3                | 48                 | 0              | 11              | 0                 |
| 101449 | 0    | 1      | 7   | 2      | 13               | 82                 | 1              | 22              | 7                 |
| 101456 | 1    | 0      | 6   | 3      | 2                | 53                 | 4              | 14              | 0                 |

1021 rows × 43 columns

Fonte: própria autor

#### 4. Análise e Exploração dos Dados

A etapa de exploração baseia-se no uso de duas bibliotecas: *collections* e *matplotlib*. A *collections* possibilita tanto a contagem dos valores quanto cada evento selecionado aparece. Já a *matplotlib* permite a visualização gráfica das regras e valores selecionados.

Imagem 21. Importação biblioteca para análise

```
from collections import Counter
import matplotlib.pyplot as plt
```

Fonte: própria autor

Para uma exploração mais fidedigna concentrou-se em analisar características intrínsecas dos pacientes, como: gênero, peso, idade e raça. Sob esse ponto de vista verifica-se o comportamento, pela visibilidade gráfica, o comportamento também da variável alvo, *readmitted*.

Primeiramente no *dataframe* verificou-se a quantidade de valores pelas faixas atribuídas anteriormente no tratamento. Para isso pode utilizar da função *counter*,



importada anteriormente, e nela aplica-se o *slice* da variável a ser contada. Realizando essa contagem percebe-se que o número de pacientes que foram e os que não foram readmitidos estão com valores próximos.

Imagem 22. Contagem de atributos

```
gender = Counter(df["gender"])
gender
```

```
Counter({1: 508, 0: 513})
```

```
age = Counter(df["age"])
age
```

```
Counter({7: 305, 8: 171, 6: 255, 4: 68, 5: 177, 2: 18, 3: 25, 0: 1, 1: 1})
```

```
weight = Counter(df["weight"])
weight
```

```
Counter({3: 431, 2: 270, 4: 220, 5: 46, 1: 26, 6: 16, 7: 6, 0: 6})
```

```
readmitted = Counter(df["readmitted"])
readmitted
```

```
Counter({1: 572, 0: 449})
```

```
race = Counter(df["race"])
race
```

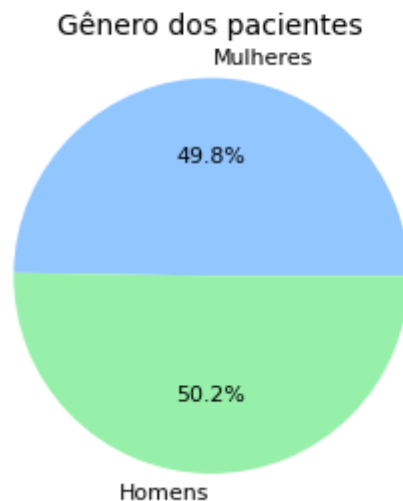
```
Counter({0: 977, 1: 30, 4: 13, 2: 1})
```

Fonte: própria autor

A seguir vê-se o gênero dos pacientes pelo gráfico de pizza, percebe-se também que há um valor equiparado. Com a leve presença masculina como pacientes, em torno de 50%.

Imagem 23. Gráfico divisão do sexo

```
plt.style.use('seaborn-pastel')
plt.pie(gender.values(), labels = ["Mulheres", "Homens"],
autopct = '%1.1f%%', textprops={'fontsize': 11})
plt.axis("image")
plt.title("Gênero dos pacientes", fontsize=14, pad =12)
plt.show()
```



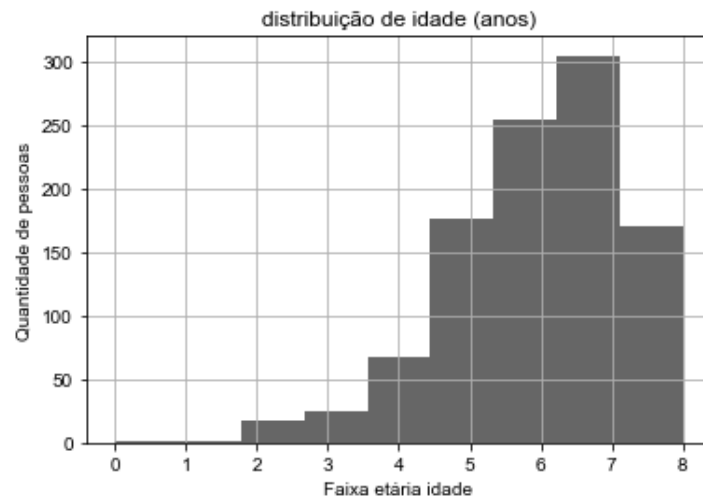
Fonte: própria autor

Outra característica discriminativa que pode ser obtida no conjunto de dados é a distribuição de valores na forma de histograma. Essa maneira ajuda a analisar onde se enquadram certas faixas de valores. No próximo histograma vê-se a distribuição de pessoas dentre os pacientes com uma faixa de idade mais avançada.

Imagem 24. Distribuição de idades

```
df.age.hist(bins=9, color = 'black', alpha = 0.6)
plt.style.use('seaborn-whitegrid')
plt.xlabel("Faixa etária idade")
plt.ylabel("Quantidade de pessoas")
plt.title("distribuição de idade (anos)")

plt.show()
```

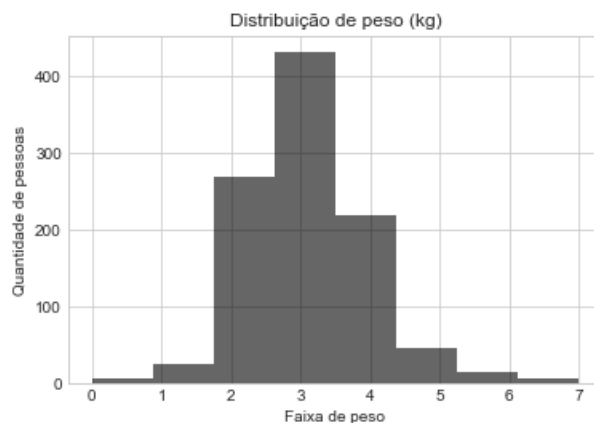


Fonte: própria autor

Outro fator importante de se analisar é a distribuição de peso, percebendo que houve um comportamento regular, enquadrando na curva normal, assim definido que a maior contração da faixa de peso dos pacientes está no valores centrais.

Imagem 25. Distribuição peso

```
df.weight.hist(bins=8, color = 'black', alpha = 0.6)
plt.style.use('seaborn-whitegrid')
plt.xlabel("Faixa de peso")
plt.ylabel("Quantidade de pessoas")
plt.title("Distribuição de peso (kg)")
plt.show()
```



Fonte: própria autor

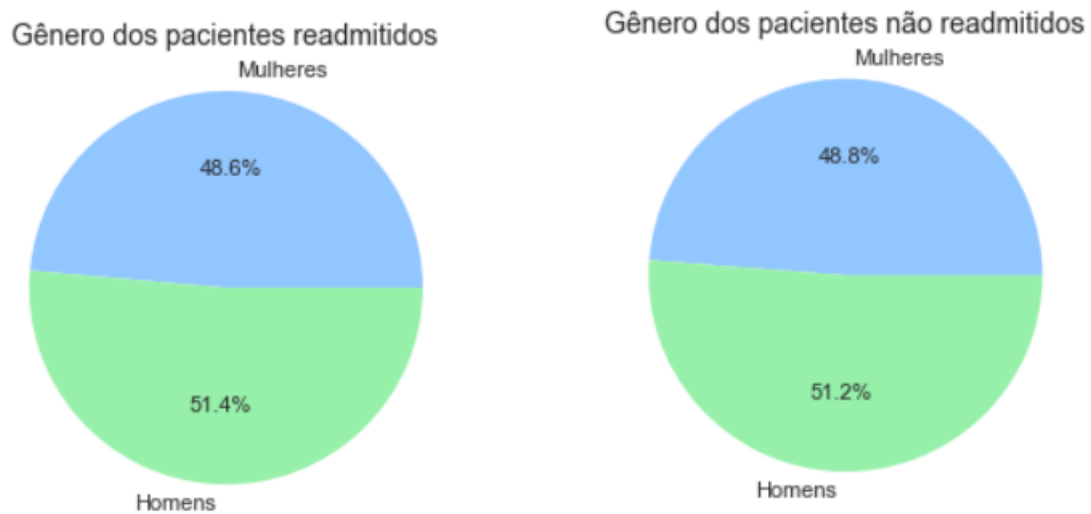
Seguindo o mesmo modelo de raciocínio o *dataframe* foi dividido em dois, com referência pela variável alvo, sendo: *df\_readmitted* e o *df\_no\_readmitted*. O primeiro encontra-se valores de todas as variáveis de pacientes que foram readmitidos, o segundo os que não foram.

Imagem 26. Divisão do *dataframe*

```
df_readmitted = df.loc[df['readmitted'] == 1 ]
df_not_readmitted = df.loc[df['readmitted'] == 0]
```

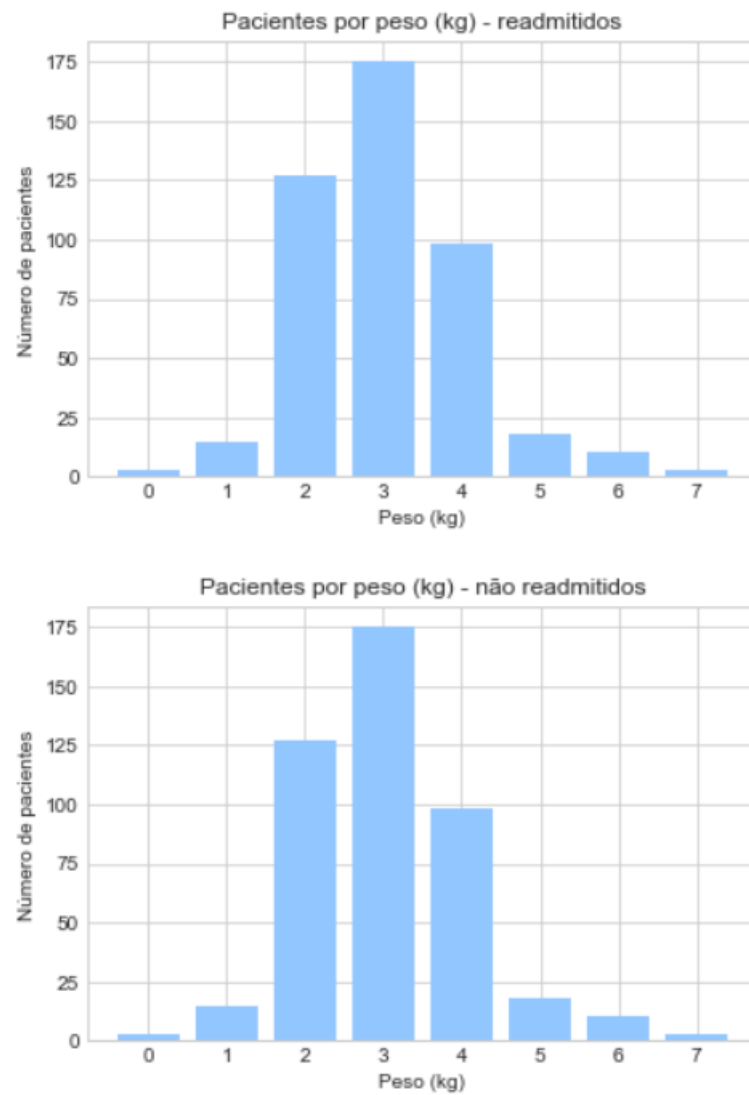
Fonte: própria autor

Com essa subdivisão conseguimos avaliar mais precisamente o comportamento da variável *target*, mostrando um comparativo entre os dois *dataframes*. A seguir compara-se o gênero dos pacientes, entre os readmitidos e não readmitidos. Percebe-se que a proporção entre os sexos está bem próxima, sendo que os homens predominam com a faixa de 51% em ambos os gráficos.

Imagem 27. Gênero por *dataframe*

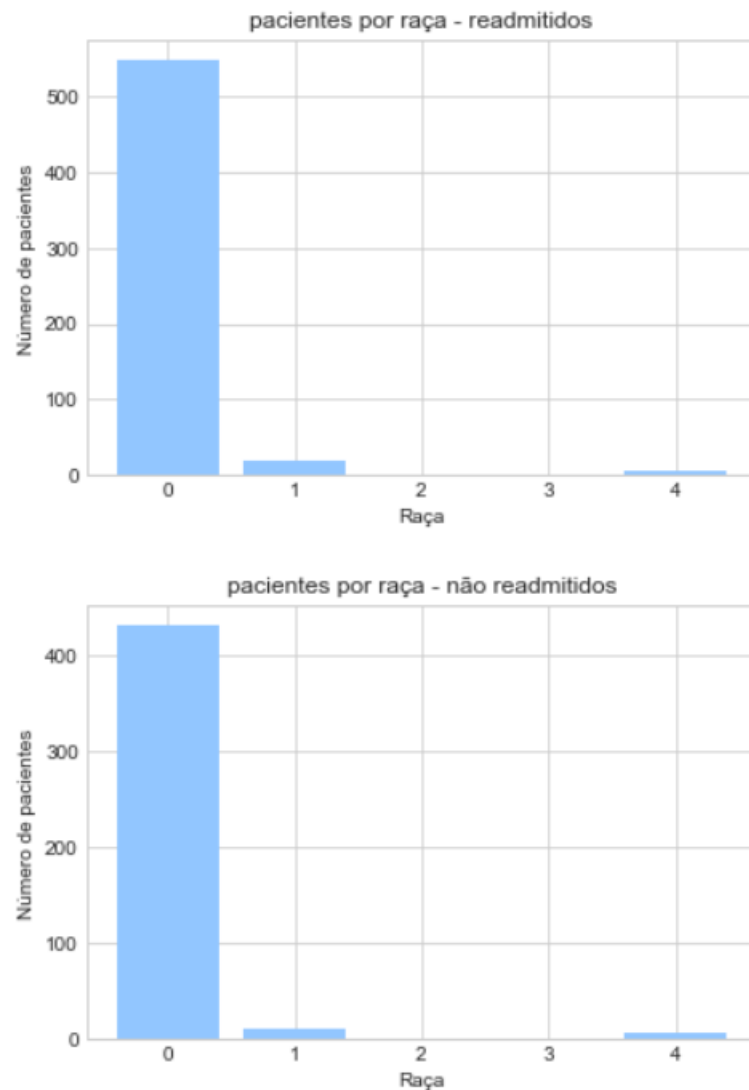
Fonte: Própria autor

Também é possível analisar a destruição de peso entre os dois conjuntos de dados, analisando que paciente com uma faixa central de peso estão em predominância, que é em torno de 50 a 125 *pounds*.

Imagem 28. Quantidade de peso dos pacientes por *datasets*

Fonte: própria autor

Outro fator interessante de se analisar é como se distribui as raças pelos pacientes, vendo que, segundo os dados, a maioria dos pacientes, readmitidos ou não, são declarados caucasianos.

Imagem 29. Quantidade de pacientes por raça nos *datasets*

Fonte: própria autor

## 5. Criação de Modelos de Machine Learning

Para essa parte do trabalho, a finalidade é a aplicar finalmente os algoritmos de *machine learning*, tendo a base de dados já tratada e analisada, possibilitando melhores parâmetros para verificar a variável alvo *readmitted*. Reforçando que tal variável implica o retorno ou não do paciente com diabetes devido a determinadas circunstâncias clínicas expostas do conjunto de dados.

Tendo a ideia primordial que é avaliar o retorno ou não, tende-se a aplicar o conceito de classificação e regressão. Tal conceito define-se a partir de bases definidas anteriormente e subdivide-se, ou classifica, em grupos ou subconjuntos, os valores de interesse, que se assemelham. Portanto a classificação dos pacientes pode ser analisada por esse viés, sabendo que ao fim do processo o interesse é definir os grupos de interesse de retorno ou não. Sendo o conceito básico de regressão que encontrar uma reta, tal qual se ajusta nos parâmetros observados.

Primeiramente aplicou-se o algoritmo K-Nearest Neighbors (KNN) (“O que é e como funciona o algoritmo KNN?”. [6], sendo uma técnica de classificação de dados que calcula a probabilidade de um ponto de dados pertencer a um grupo ou outro com base nos pontos de dados mais próximos a ele. Este algoritmo é fácil de implementar, resistente a variações nos dados de treinamento e é eficaz quando lidando com grandes conjuntos de dados de treinamento. No entanto, uma desvantagem significativa do algoritmo é que ele requer muitos recursos computacionais para ser executado.

Outro algoritmo que se aplicou, dentro da ideia de classificação, foi o Naive Bayes (“O que é Naive Bayes e como funciona esse algoritmo de classificação”, [7]). O algoritmo de Naive Bayes é construído sobre o teorema de Bayes, presumindo que cada par de valores seja independente. A característica fundamental deste algoritmo, que justifica a inclusão do termo “*naive*” (ingênuo) em seu nome, é a sua completa desconsideração da correlação existente entre as variáveis. As principais vantagens deste algoritmo incluem sua capacidade de estimar parâmetros com um volume de dados de treinamento relativamente pequeno e sua velocidade extremamente alta em relação a métodos mais sofisticados. No entanto, uma desvantagem significativa é que ele não é considerado um bom avaliador.

Por fim se usou o gradiente descendente (“Gradiente descendente”, [8]). O algoritmo de gradiente descendente é um dos algoritmos mais bem-sucedidos em problemas de aprendizado de máquina, que funciona encontrando iterativamente os valores dos parâmetros que minimizam uma função de interesse. As principais vantagens deste algoritmo são sua eficiência e facilidade de implementação, no entanto, ele necessita de diversos hiper parâmetros e é sensível ao dimensionamento de recursos utilizados.

Adentrando-se nos pontos de análises principais para cada algoritmo acima citando, têm-se: acurácia, precisão, *recall* e *F1-score*. Acurácia sendo a quantidade de acertos dividido pelo total da amostra. Precisão resume-se ao acertos que realmente foram



verdadeiros, dentre os verdadeiros classificados, dividido pelo total de verdadeiro positivos mais os falsos positivos. *Recall* são verdadeiros positivos divididos pelos mesmo, somando-se aos falsos negativos. E por fim o *F1-score*, média ponderada da precisão e acurácia.

A seguir define-se os *datasets* de treino e teste, além de importar as biblioteca do *sklearn* para análise, definindo o tamanho do teste para 30%.

Imagem 30. Importação e separação dos dados de treino e testes

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

X = df.drop('readmitted',axis=1).values
y = df['readmitted'].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

Fonte: própria autor

Após a definição dos *dataframes* de treinamento e teste e a escolha das medidas de avaliação, o próximo passo é a aplicação dos algoritmos previamente selecionados. O procedimento para todos eles serão o mesmo, começando pela importação do algoritmo correspondente e realizando o treinamento com a função *fit* nas duas bases de treinamento, seguido do registro da sua precisão com a função *score*. Em seguida, a função *predict* será utilizada na base de teste *X\_teste* e sua saída será comparada com a série *y\_teste*, gerando medidas de avaliação por meio das funções *accuracy\_score* e *classification\_report*.

Imagem 31. Algoritmo KNN

## knn

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn = knn.fit(X_train, y_train)
print("Acurácia: ", knn.score(X_train, y_train))
tp_knn = knn.predict(X_test)
print("Acurácia de previsão: ", accuracy_score(y_test, tp_knn))
print(classification_report(y_test, tp_knn))
```

```
Acurácia: 0.7058823529411765
Acurácia de previsão: 0.46579804560260585
      precision    recall  f1-score   support

0         0.41         0.44         0.42         136
1         0.52         0.49         0.50         171

 accuracy          0.47         307
 macro avg         0.46         0.46         0.46         307
weighted avg         0.47         0.47         0.47         307
```

Fonte: própria autor

Imagem 32. Algoritmo Naive Bayes

## bayes

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
nb = GaussianNB()
nb = nb.fit(X_train, y_train)
print("Acurácia: ", nb.score(X_train, y_train))
tp_nb = nb.predict(X_test)
print(classification_report(y_test, tp_nb))
```

```
Acurácia: 0.6386554621848739
      precision    recall  f1-score   support

0         0.48         0.63         0.55         136
1         0.61         0.46         0.53         171

 accuracy          0.54         307
 macro avg         0.55         0.55         0.54         307
weighted avg         0.56         0.54         0.54         307
```

Fonte: própria autor

Imagem 33. Algoritmo Gradiente Descendente

## gradiente descendente

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd = sgd.fit(X_train, y_train)
print("Acurácia: ", sgd.score(X_train, y_train))
tp_sgd = sgd.predict(X_test)
print(classification_report(y_test, tp_sgd))
```

```
Acurácia: 0.6428571428571429
              precision    recall  f1-score   support

         0         0.52      0.43      0.47        136
         1         0.60      0.68      0.64        171

 accuracy          0.57        307
 macro avg          0.56        307
 weighted avg       0.56        307
```

Fonte: própria autor

## 6. Interpretação dos Resultados e Conclusão

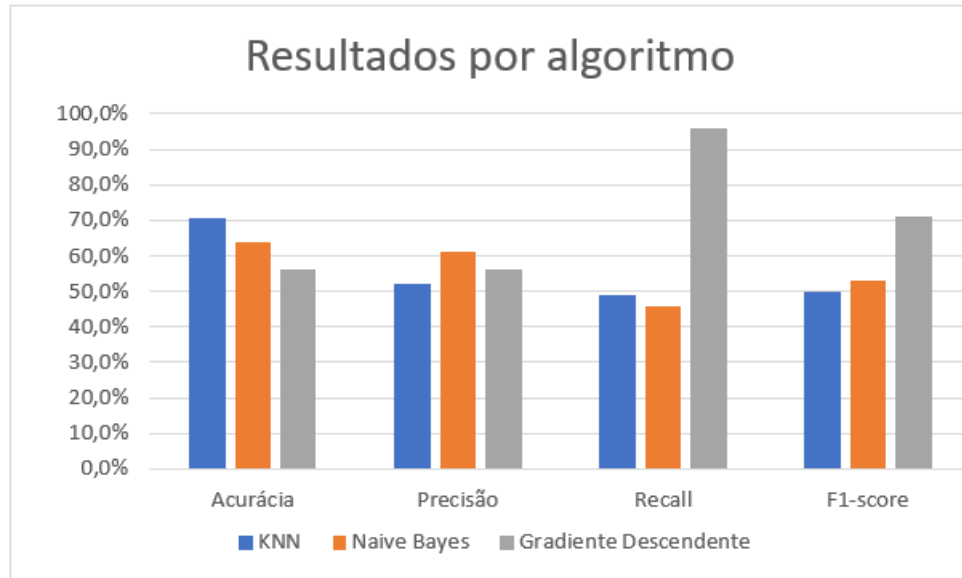
A seguir no quadro estão as principais métricas de cada algoritmo.

Imagem 34. Valores dos resultados dos algoritmos

| Algoritmo             | Acurácia | Precisão | Recall | F1-score |
|-----------------------|----------|----------|--------|----------|
| KNN                   | 70,5%    | 52%      | 49%    | 50%      |
| Naive Bayes           | 63,8%    | 61%      | 46%    | 53%      |
| Gradiente Descendente | 56,0%    | 56%      | 96%    | 71%      |

Fonte: própria autor

Imagem 35. Gráfico dos resultados por métrica



Fonte: própria autor

Percebe-se que o algoritmo KNN teve uma performance melhor em relação aos demais, nos valores de acurácia, 70,5 %. Já a melhor precisão foi do algoritmo Naive Bayes, alcançando 61 %. Já a pior média entre precisão e acurácia foi do modelo utilizando gradiente descendente.

Esses valores obtidos dão um norte programático em que se pode ajustar valores, modelagens e técnicas para se alcançar melhores resultados para a classificação dos pacientes. Logo dentre esses algoritmos é preferível trabalhar-se com o Naive Bayes e o KNN, podendo definir melhores métricas, uma quantidade ótima de vizinhos, entre outros fatores que contribuam para a performance.

Com isso podemos ter uma base teórica que os algoritmos de classificação apresentaram boa performance ao classificar se pacientes diabéticos podem ou não ter retorno ao hospital. Isso pode ser aplicado com ênfase na sociedade hospitalar, para que sirva de ferramenta médica, prevenindo e antecipando tratamentos.

## 7. Links

Vídeo:

<https://www.youtube.com/watch?v=XyQwpdyMoqs>

Github:

<https://github.com/RodrigoLeite96/TCC-PUCMINAS-POS-GRADUACAO.git>

## REFERÊNCIAS

- [1] **O que é Big Data?** Disponível em: <<https://www.oracle.com/br/big-data/what-is-big-data/>>. Acesso em: 30 mar. 2023.
- [2] **O que é Machine Learning?** Disponível em: <<https://www.oracle.com/br/artificial-intelligence/machine-learning/what-is-machine-learning/>>. Acesso em: 30 mar. 2023.
- [3] **Algoritmos de Classificação: o que são e como funcionam.** Disponível em: <[https://awari.com.br/algoritmos-de-classificacao/?utm\\_source=blog](https://awari.com.br/algoritmos-de-classificacao/?utm_source=blog)>. Acesso em: 30 mar. 2023.
- [4] **Diabetes 130-US hospitals for years 1999-2008 Data Set.** Disponível em: <<https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008>>. Acesso em: 30 mar. 2023.
- [4] Strack, et al., 2014. **Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records.** Internacional, Hindawi Publishing Corporation, 2014.
- [5] **Jupyter Project Documentation.** Disponível em: <<https://docs.jupyter.org/en/latest/>>. Acesso em: 30 mar. 2023.
- [6] **O que é e como funciona o algoritmo KNN?** Disponível em: <<https://didatica.tech/o-que-e-e-como-funciona-o-algoritmo-knn/#:~:text=O%20KNN%20%C3%A9%20muito%20utilizado,rela%C3%A7%C3%A3o%20aos%20vizinhos%20mais%20pr%C3%B3ximos.>>. Acesso em: 30 mar. 2023.
- [7] **O que é Naive Bayes e como funciona esse algoritmo de classificação.** Disponível em: <<https://rockcontent.com/br/blog/naive-bayes/>>. Acesso em: 30 mar. 2023.
- [8] **Gradiente descendente.** Disponível em: <[https://pt.d2l.ai/chapter\\_optimization/gd.html](https://pt.d2l.ai/chapter_optimization/gd.html)>. Acesso em: 30 mar. 2023.

## APÊNDICE

### Programação/Scripts

```
#importação das bibliotecas para leitura e exploração
import pandas as pd
import requests
from zipfile import ZipFile
from io import BytesIO
import numpy as np
pd.set_option('display.max_columns', None)

response = requests.get("https://archive.ics.uci.edu/ml/machine-learning-
databases/00296/dataset_diabetes.zip")
files = ZipFile(BytesIO(response.content))
df = pd.read_csv(files.open("dataset_diabetes/diabetic_data.csv"))

#variavel para pegar as colunas do df em fomra de lista
cols = df.columns

#variavel para selecionar colunas numéricas do df
num_cols = df._get_numeric_data().columns

#variavel que diminui as colunas numéricas das colunas, obtendo-se as colunas categóricas
cat_col = list(set(cols) - set(num_cols))

#laço for para valores únicos categóricos
for coluna in cat_col:
    print(coluna + str(df[coluna].unique()) + '\n')

#laço for para valores únicos numéricos
for col in num_cols:
    print( str(type(col)) + col + str(df[col].unique()) + '\n')

#valores a serem excluídos
drop_values = ["?", "Unknown/Invalid"]
```

```

#lógica para excluir linhas que possuem valores na lista drop_values
df = df[~df.isin(drop_values).any(axis=1)]

cols = df.columns

#variavel para selecionar colunas numéricas do df
num_cols = df._get_numeric_data().columns

#variavel que diminui as colunas numéricas das colunas, obtendo-se as colunas categóricas
cat_col = list(set(cols) - set(num_cols))

#laço for para valores únicos categóricos
for coluna in cat_col:
    print(coluna + str(df[coluna].unique()) + '\n')

#laço for para valores únicos numéricos
for col in num_cols:
    print( str(type(col)) + col + str(df[col].unique()) + '\n')

df.shape

#lista para colunas a serem excluídas
colunas_excluidas = ['payer_code'
, 'medical_specialty'
, 'admission_type_id'
, 'admission_source_id'
, 'discharge_disposition_id'
, 'patient_nbr']

df = df.drop(colunas_excluidas, axis = 1)

#colunas diagnosticos para o tipo string
df['diag_1'] = df['diag_1'].astype(pd.StringDtype())
df['diag_2'] = df['diag_2'].astype(pd.StringDtype())
df['diag_3'] = df['diag_3'].astype(pd.StringDtype())

#atribuição do dicionário de valores para equalizar numericamente as variáveis

```



```

df['glipizide-metformin'] = df['glipizide-metformin'].map({'No': 0}) #['No']
df['change'] = df['change'].map({'No': 0, 'Ch': 1}) #['Ch' 'No']
df['tolbutamide'] = df['tolbutamide'].map({'No': 0}) #['No']
df['examide'] = df['examide'].map({'No': 0}) #['No']
df['pioglitazone'] = df['pioglitazone'].map({'No': 0, 'Steady': 1, 'Up': 2, 'Down': 3}) #['No' 'Steady' 'Up' 'Down']
df['diabetesMed'] = df['diabetesMed'].map({'No': 0, 'Yes': 1}) #['Yes' 'No']
df['citoglipton'] = df['citoglipton'].map({'No': 0}) #['No']
df['insulin'] = df['insulin'].map({'Steady': 0, 'No': 1, 'Down': 2, 'Up': 3}) #['Steady' 'No' 'Down' 'Up']
df['troglitazone'] = df['troglitazone'].map({'No': 0}) #['No']
df['glimepiride-pioglitazone'] = df['glimepiride-pioglitazone'].map({'No': 0}) #['No']
df['glyburide-metformin'] = df['glyburide-metformin'].map({'No': 0}) #['No']
df['metformin-pioglitazone'] = df['metformin-pioglitazone'].map({'No': 0}) #['No']
df['miglitol'] = df['miglitol'].map({'No': 0}) #['No']
df['glyburide'] = df['glyburide'].map({'No': 0, 'Steady': 1, 'Up': 2, 'Down': 3}) #['No' 'Steady' 'Up' 'Down']
df['tolazamide'] = df['tolazamide'].map({'No': 0}) #['No']
df['max_glu_serum'] = df['max_glu_serum'].map({'None': 0, 'Norm': 1, '>200': 2, '>300': 3}) #['None' '>300'
'Norm' '>200']
df['nateglinide'] = df['nateglinide'].map({'No': 0, 'Steady': 1}) #['No' 'Steady']
df['metformin-rosiglitazone'] = df['metformin-rosiglitazone'].map({'No': 0}) #['No']
df['glimepiride'] = df['glimepiride'].map({'No': 0, 'Steady': 1, 'Down': 2, 'Up': 3}) #['No' 'Steady' 'Down' 'Up']
df['metformin'] = df['metformin'].map({'No': 0, 'Steady': 1, 'Up': 2, 'Down': 3}) #['No' 'Steady' 'Up' 'Down']
df['A1Cresult'] = df['A1Cresult'].map({'None': 0, 'Norm': 1, '>8': 2, '>7': 3}) #['None' 'Norm' '>8' '>7']
df['chlorpropamide'] = df['chlorpropamide'].map({'No': 0}) #['No']
df['acetohexamide'] = df['acetohexamide'].map({'No': 0}) #['No']
df['rosiglitazone'] = df['rosiglitazone'].map({'No': 0, 'Steady': 1})
df['glipizide'] = df['glipizide'].map({'Steady': 0, 'No': 1, 'Up': 2, 'Down': 3})
df['acarbose'] = df['acarbose'].map({'No': 0, 'Steady': 1})
df['repaglinide'] = df['repaglinide'].map({'No': 0, 'Steady': 1, 'Up': 2})
df['readmitted'] = df['readmitted'].map({'NO': 0, '<30': 1, '>30': 1})
df['gender'] = df['gender'].map({'Male': 0, 'Female': 1})
df['age'] = df['age'].map({'[0-10)': 0, '[10-20)': 1, '[20-30)': 2, '[30-40)': 3, '[40-50)': 4, '[50-60)': 5, '[60-70)': 6,
'[70-80)': 7, '[80-90)': 8, '[90-100)': 9})
df['age'] = df['age'].map({'[0-10)': 0, '[10-20)': 1, '[20-30)': 2, '[30-40)': 3, '[40-50)': 4, '[50-60)': 5, '[60-70)': 6,
'[70-80)': 7, '[80-90)': 8, '[90-100)': 9})
df['race'] = df['race'].map({'Caucasian': 0, 'AfricanAmerican': 1, 'Asian': 2, 'Hispanic': 3, 'Other': 4})
df['weight'] = df['weight'].map({'[0-25)': 0, '[25-50)': 1, '[50-75)': 2, '[75-100)': 3, '[100-125)': 4, '[125-150)': 5,
'[150-175)': 6, '[175-200)': 7})

```

```

df[df['age'].isnull()]
df = df.dropna()
df.shape

#lista de atribuição das patologias
Circulatory = np.arange(390,460).tolist() + [785]
Circulatory_str = [str(elemento) for elemento in Circulatory]

Respiratory = np.arange(460,520).tolist() + [786]
Respiratory_str = [str(elemento) for elemento in Respiratory]

Digestive = np.arange(520,580).tolist() + [787]
Digestive_str = [str(elemento) for elemento in Digestive]

Injury = np.arange(800,1000).tolist() + [786]
Injury_str = [str(elemento) for elemento in Injury]

Musculoskeletal = np.arange(710,740).tolist() + [786]
Musculoskeletal_str = [str(elemento) for elemento in Musculoskeletal]

Genitourinary = np.arange(580,630).tolist() + [788]
Genitourinary_str = [str(elemento) for elemento in Genitourinary]

Neoplasms = np.arange(140,240).tolist()
Neoplasms_str = [str(elemento) for elemento in Neoplasms]

Other = np.arange(240,250).tolist() + np.arange(251,280).tolist() +
[780,781,784,790,791,792,793,794,795,796,797,798,799]
Other_str = [str(elemento) for elemento in Other]

#atribuição das patologias dentre as colunas de diagnostico
df.loc[df['diag_1'].str.startswith('250'), 'diag_1'] = 'Diabetes'
df.loc[df['diag_2'].str.startswith('250'), 'diag_2'] = 'Diabetes'
df.loc[df['diag_3'].str.startswith('250'), 'diag_3'] = 'Diabetes'

df.loc[df['diag_1'].isin(Circulatory_str), 'diag_1'] = 'Circulatory'
df.loc[df['diag_1'].isin(Respiratory_str), 'diag_1'] = 'Respiratory'
df.loc[df['diag_1'].isin(Digestive_str), 'diag_1'] = 'Digestive'

```

```

df.loc[df['diag_1'].isin(Injury_str), 'diag_1'] = 'Injury'
df.loc[df['diag_1'].isin(Musculoskeletal_str), 'diag_1'] = 'Musculoskeletal'
df.loc[df['diag_1'].isin(Genitourinary_str), 'diag_1'] = 'Genitourinary'
df.loc[df['diag_1'].isin(Neoplasms_str), 'diag_1'] = 'Neoplasms'

df.loc[df['diag_2'].isin(Circulatory_str), 'diag_2'] = 'Circulatory'
df.loc[df['diag_2'].isin(Respiratory_str), 'diag_2'] = 'Respiratory'
df.loc[df['diag_2'].isin(Digestive_str), 'diag_2'] = 'Digestive'
df.loc[df['diag_2'].isin(Injury_str), 'diag_2'] = 'Injury'
df.loc[df['diag_2'].isin(Musculoskeletal_str), 'diag_2'] = 'Musculoskeletal'
df.loc[df['diag_2'].isin(Genitourinary_str), 'diag_2'] = 'Genitourinary'
df.loc[df['diag_2'].isin(Neoplasms_str), 'diag_2'] = 'Neoplasms'

df.loc[df['diag_3'].isin(Circulatory_str), 'diag_3'] = 'Circulatory'
df.loc[df['diag_3'].isin(Respiratory_str), 'diag_3'] = 'Respiratory'
df.loc[df['diag_3'].isin(Digestive_str), 'diag_3'] = 'Digestive'
df.loc[df['diag_3'].isin(Injury_str), 'diag_3'] = 'Injury'
df.loc[df['diag_3'].isin(Musculoskeletal_str), 'diag_3'] = 'Musculoskeletal'
df.loc[df['diag_3'].isin(Genitourinary_str), 'diag_3'] = 'Genitourinary'
df.loc[df['diag_3'].isin(Neoplasms_str), 'diag_3'] = 'Neoplasms'

patology = ["Circulatory", "Respiratory", "Digestive", "Injury", "Musculoskeletal", "Genitourinary", "Neoplasms"]

df.loc[~df['diag_1'].isin(patology), 'diag_1'] = 'Other'
df.loc[~df['diag_2'].isin(patology), 'diag_2'] = 'Other'
df.loc[~df['diag_3'].isin(patology), 'diag_3'] = 'Other'

#verificação da atribuição de valores
colunas_excluidas_2 = ['encounter_id']
df = df.drop(colunas_excluidas_2, axis = 1)
len(df["diag_1"].unique()) == len(df["diag_2"].unique()) == len(df["diag_3"].unique())

#laço for para atribuição
diags = ['diag_1', 'diag_2', 'diag_3']
for diag in diags:
    df[diag] = df[diag].map({'Circulatory': 0, 'Respiratory': 1, 'Digestive': 2, 'Injury': 3
        , 'Musculoskeletal': 4, 'Genitourinary': 5, 'Neoplasms': 6, 'Other': 7})

```

```

df['age'] = df['age'].astype(int)

#importação da biblioteca para plotagem e exploração
from collections import Counter
import matplotlib.pyplot as plt

#slice das variaveis
gender = Counter(df["gender"])
gender
age = Counter(df["age"])
age
weight = Counter(df["weight"])
weight
readmitted = Counter(df["readmitted"])
readmitted
race = Counter(df["race"])
race

plt.style.use('seaborn-pastel')
plt.pie(gender.values(), labels = ["Mulheres", "Homens"],
autopct = '%1.1f%%', textprops={'fontsize': 11})
plt.axis("image")
plt.title("Gênero dos pacientes", fontsize=14,pad =12)
plt.show()

df.age.hist(bins=9, color = 'black', alpha = 0.6)
plt.style.use('seaborn-whitegrid')
plt.xlabel("Faixa etária idade")
plt.ylabel("Quantidade de pessoas")
plt.title("distribuição de idade (anos)")
plt.show()

df.weight.hist(bins=8, color = 'black', alpha = 0.6)
plt.style.use('seaborn-whitegrid')
plt.xlabel("Faixa de peso")
plt.ylabel("Quantidade de pessoas")
plt.title("Distribuição de peso (kg)")
plt.show()

```

```

#slice dataframe pela variavel target
df_readmitted = df.loc[df['readmitted'] == 1 ]
df_not_readmitted = df.loc[df['readmitted'] == 0]

#slice do genero nos dataframes
gender_not_readmitted = Counter(df_not_readmitted["gender"])
gender_not_readmitted
gender_readmitted = Counter(df_readmitted["gender"])
gender_readmitted

plt.style.use('seaborn-pastel')
plt.pie(gender_readmitted.values(), labels = ["Mulheres", "Homens"],
autopct = '%1.1f%%', textprops={'fontsize': 11})
plt.axis("image")
plt.title("Gênero dos pacientes readmitidos", fontsize=14,pad =12)
plt.show()

plt.style.use('seaborn-pastel')
plt.pie(gender_not_readmitted.values(), labels = ["Mulheres", "Homens"],
autopct = '%1.1f%%', textprops={'fontsize': 11})
plt.axis("image")
plt.title("Gênero dos pacientes não readmitidos", fontsize=14,pad =12)
plt.show()

race_readmitted = Counter(df_readmitted["race"])
race_readmitted
race_not_readmitted = Counter(df_not_readmitted["race"])
race_not_readmitted
weight_readmitted = Counter(df_not_readmitted["weight"])
weight_readmitted
weight_not_readmitted = Counter(df_not_readmitted["weight"])
weight_not_readmitted

plt.style.use('seaborn-whitegrid')
plt.bar(['M', 'F'], gender_readmitted.values())
plt.ylabel('Número de pacientes')

```

```
plt.xlabel('Raça')
plt.title('pacientes por sexo - readmitidos')
plt.show()
```

```
plt.style.use('seaborn-whitegrid')
plt.bar(['M','F'], gender_not_readmitted.values())
plt.ylabel('Número de pacientes')
plt.xlabel('Raça')
plt.title('pacientes por sexo - não readmitidos')
plt.show()
```

```
plt.style.use('seaborn-whitegrid')
plt.bar(list(race_readmitted.keys()), race_readmitted.values())
plt.ylabel('Número de pacientes')
plt.xlabel('Raça')
plt.title('pacientes por raça - readmitidos')
plt.show()
```

```
plt.style.use('seaborn-whitegrid')
plt.bar(list(race_not_readmitted.keys()), race_not_readmitted.values())
plt.ylabel('Número de pacientes')
plt.xlabel('Raça')
plt.title('pacientes por raça - não readmitidos')
plt.show()
```

```
plt.style.use('seaborn-whitegrid')
plt.bar(['M','F'], gender_readmitted.values())
plt.ylabel('Número de pacientes')
plt.xlabel('Raça')
plt.title('pacientes por sexo - readmitidos')
plt.show()
```

```
plt.style.use('seaborn-whitegrid')
plt.bar(['M','F'], gender_not_readmitted.values())
plt.ylabel('Número de pacientes')
plt.xlabel('Raça')
plt.title('pacientes por sexo - não readmitidos')
plt.show()
```

```
plt.style.use('seaborn-whitegrid')
plt.bar(list(weight_readmitted.keys()), weight_readmitted.values())
plt.ylabel('Número de pacientes')
plt.xlabel('Peso (kg)')
plt.title('Pacientes por peso (kg) - readmitidos')
plt.show()
```

```
plt.style.use('seaborn-whitegrid')
plt.bar(list(weight_not_readmitted.keys()), weight_not_readmitted.values())
plt.ylabel('Número de pacientes')
plt.xlabel('Peso (kg)')
plt.title('Pacientes por peso (kg) - não readmitidos')
plt.show()
```

```
#importação das bibliotecas do sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
```

```
#separação dados de treino e teste
X = df.drop('readmitted',axis=1).values
y = df['readmitted'].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

```
#aplicação algoritmo KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn = knn.fit(X_train, y_train)
print("Acurácia: ", knn.score(X_train, y_train))
tp_knn = knn.predict(X_test)
print("Acurácia de previsão: ", accuracy_score(y_test, tp_knn))
print(classification_report(y_test, tp_knn))
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
print(f"Acurácia: {round(accuracy_score(y_test, tp_knn), 2)}")
print(f"Precisão: {round(precision_score(y_test, tp_knn), 2)}")
print(f"Recall: {round(recall_score(y_test, tp_knn), 2)}")
print(f"F1_score: {round(f1_score(y_test, tp_knn), 2)}")
```

#aplicação algoritmo Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
nb = GaussianNB()
nb = nb.fit(X_train, y_train)
print("Acurácia: ", nb.score(X_train, y_train))
tp_nb = nb.predict(X_test)
print(classification_report(y_test, tp_nb))
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
print(f"Accuracy: {round(accuracy_score(y_test, tp_nb), 2)}")
print(f"Precision: {round(precision_score(y_test, tp_nb), 2)}")
print(f"Recall: {round(recall_score(y_test, tp_nb), 2)}")
print(f"F1_score: {round(f1_score(y_test, tp_nb), 2)}")
```

#aplicação algoritmo gradiente descendente

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd = sgd.fit(X_train, y_train)
print("Acurácia: ", sgd.score(X_train, y_train))
tp_sgd = sgd.predict(X_test)
print(classification_report(y_test, tp_sgd))
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
print(f"Accuracy: {round(accuracy_score(y_test, tp_sgd), 2)}")
print(f"Precision: {round(precision_score(y_test, tp_sgd), 2)}")
print(f"Recall: {round(recall_score(y_test, tp_sgd), 2)}")
print(f"F1_score: {round(f1_score(y_test, tp_sgd), 2)}")
```