

# **Introdução à uma prática de análise de dados com Python**



Autor: Victor Miguel de Souza Soares

RECIFE  
2024



## Introdução: Por Que Usar Python para Análise de Dados?

Nos últimos anos, a análise de dados emergiu como uma das competências mais valiosas no mundo digital, impulsionando decisões estratégicas em diversas indústrias, desde tecnologia até finanças e saúde. Em meio a esse cenário, Python tem se destacado como uma das linguagens de programação mais populares e eficientes para conduzir análises de dados.

Python oferece uma combinação única de simplicidade e poder, permitindo que tanto iniciantes quanto especialistas consigam realizar tarefas complexas de maneira intuitiva. Com uma vasta coleção de bibliotecas dedicadas, como Pandas para manipulação de dados, Matplotlib e Seaborn para visualização, e Scikit-learn para aprendizado de máquina, Python se torna uma ferramenta robusta que facilita o processo de transformar dados brutos em insights valiosos.

Além disso, a comunidade ativa de Python significa que há uma infinidade de recursos, tutoriais e fóruns disponíveis, tornando a resolução de problemas e o aprendizado contínuo mais acessíveis. A natureza de código aberto da linguagem também garante que qualquer pessoa possa contribuir para o seu desenvolvimento, o que continuamente aprimora e expande suas capacidades.

Em resumo, Python não é apenas uma linguagem de programação; é um ecossistema completo que possibilita a utilização de análises de dados de ponta, de forma acessível e eficiente. Este eBook explora um exemplo prático de como utilizar Python para extrair, limpar, analisar e visualizar dados, demonstrando porque essa linguagem se tornou a escolha mais usada para profissionais de dados.

### Ambiente e ferramentas de trabalho

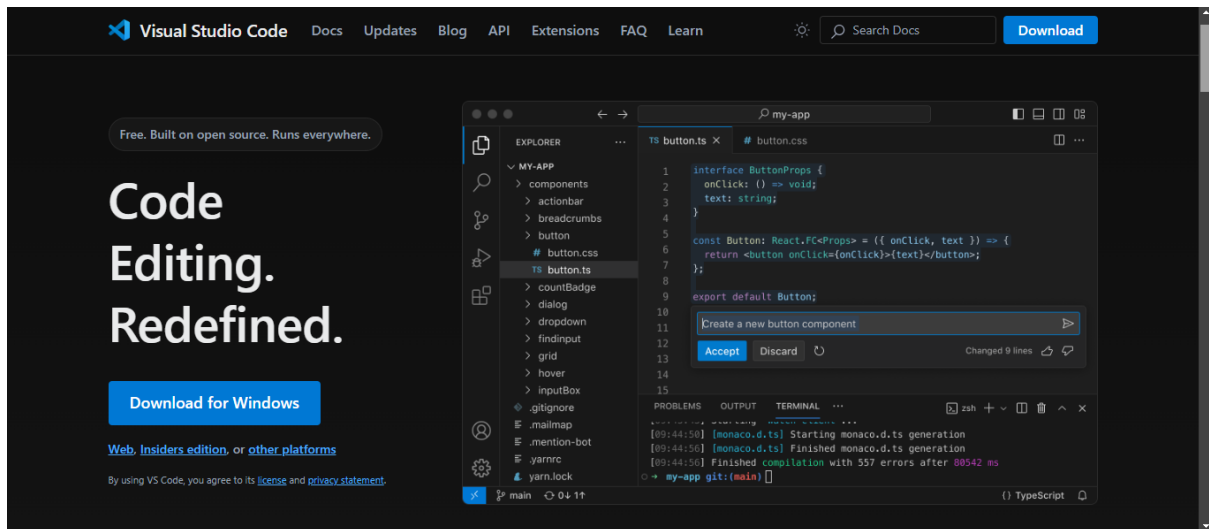
A escolha do ambiente de desenvolvimento é um passo crucial para garantir uma experiência leve e produtiva durante a análise de dados. Dois dos ambientes mais populares entre os cientistas de dados e desenvolvedores Python são o **Visual Studio Code (VS Code)**, que utilizaremos, e o **Jupyter Notebook**, que apenas faremos uso do

seu formato de arquivo. Ambos oferecem funcionalidades únicas que facilitam o desenvolvimento, depuração e visualização de dados.

### VS Code



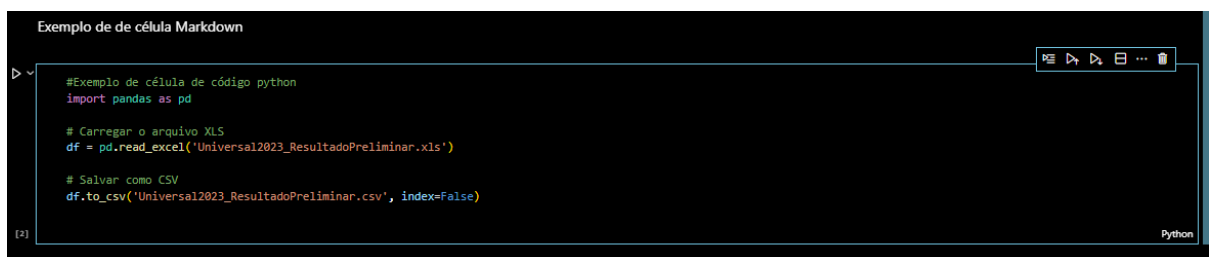
O VS Code é um editor de código-fonte gratuito e altamente extensível, desenvolvido pela Microsoft. Ele se destaca por sua leveza, velocidade e flexibilidade, oferecendo uma experiência de desenvolvimento integrada e poderosa. Além disso, possui **Extensões:** uma vasta biblioteca de extensões, permitindo personalizar o ambiente de acordo com as necessidades do projeto. Por exemplo, a extensão Python do VS Code adiciona suporte avançado para depuração, auto-completar, e linting, facilitando a escrita e otimização de código. Complementando esse conceito vantajoso, está a capacidade do VS Code de suportar arquivos de extensão Ipynb (que significa *Interactive Python Notebook*), que será o formato dos arquivos que iremos utilizar. Além disso, possui **Integração com Git:** O VS Code integra-se nativamente com o Git, tornando o controle de versão e a colaboração mais eficientes. O seu download fica disponível no link: <https://code.visualstudio.com/>



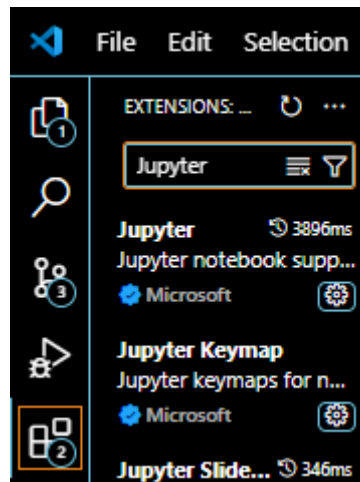
## Extensão Jupyter Nootbook

Um arquivo ipynb contém não apenas código Python, mas também texto explicativo, visualizações, e outros elementos interativos. Esses arquivos são organizados em células, que podem ser de diferentes tipos:

1. **Células de Código:** Onde você escreve e executa código Python. Os resultados do código (como saídas de texto ou gráficos) são exibidos logo abaixo da célula.
2. **Células de Markdown:** Onde você pode escrever texto formatado, incluindo títulos, listas, links, imagens, e equações matemáticas. O Markdown permite criar documentação e explicações junto ao código, facilitando a leitura e compreensão do notebook.
3. **Células de Saída:** Onde são exibidos os resultados das execuções das células de código, como tabelas, gráficos e outros outputs.



O seu download pode ser feito através da barra de pesquisa de extensões, contida, no lado esquerdo superior do VS Code, no quarto ícone, pesquisando pela palavra Jupyter, sendo o primeiro resultado da busca.



### Os 4 tipos de análises de dados:

As análises de dados podem ser classificadas em diferentes tipos, cada uma com um propósito específico e métodos apropriados. Vamos entender cada um:

## OS QUATRO PRINCIPAIS TIPOS DE ANÁLISE DE DADOS



### Análise Descritiva

- **Objetivo:** Resumir e descrever os principais aspectos de um conjunto de dados, fornecendo uma visão geral dos dados.
- **Métodos:** Estatísticas descritivas, como médias, medianas, modas, desvio padrão, e visualizações como gráficos de barras, histogramas e boxplots.
- **Exemplo:** Analisar as vendas mensais de uma empresa para identificar tendências ou padrões de crescimento ao longo do tempo.

## 2. Análise Diagnóstica

- **Objetivo:** Explorar os dados para entender as causas ou fatores subjacentes a certos padrões ou resultados observados na análise descritiva.
- **Métodos:** Análise de correlação, regressão, e drill-down em dados para investigar relações e causas.
- **Exemplo:** Após identificar uma queda nas vendas em um mês específico, a análise diagnóstica pode investigar se houve mudanças no mercado, campanhas de marketing, ou outros fatores externos que possam ter contribuído.

## 3. Análise Preditiva

- **Objetivo:** Usar dados históricos e técnicas de modelagem estatística para prever resultados futuros.
- **Métodos:** Modelos de regressão, aprendizado de máquina, séries temporais, entre outros.
- **Exemplo:** Prever as vendas futuras com base em padrões históricos, sazonalidade e outras variáveis relevantes.

## 4. Análise Prescritiva

- **Objetivo:** Sugerir ações ou estratégias com base nos dados analisados para alcançar um resultado desejado.
- **Métodos:** Otimização, simulação, e algoritmos de decisão.
- **Exemplo:** Sugerir a melhor estratégia de marketing para maximizar as vendas com base em previsões de demanda e orçamento disponível.

## Prática

### Estudo de Caso: Prevendo o Diagnóstico de Diabetes com Machine Learning

Esse desafio pode ser encontrado no site Kaggle, onde foi extraída a base de dados: <https://www.kaggle.com/datasets/priyamchoksi/100000-diabetes-clinical-dataset/data>

### *Sobre o Dataset*

Neste exemplo prático, vamos trabalhar com um dataset detalhado que contém dados de saúde e demográficos de 100.000 indivíduos. Este conjunto de dados foi criado para facilitar pesquisas relacionadas ao diabetes e modelagem preditiva. As informações contidas no dataset incluem variáveis como gênero, idade, localização, raça,

hipertensão, doenças cardíacas, histórico de tabagismo, IMC, nível de HbA1c, nível de glicose no sangue, e o status de diabetes.

Para este desafio, iremos realizar uma **Modelagem Preditiva**: Construir modelos de Machine Learning que preveem a probabilidade de uma pessoa ter diabetes com base em características demográficas e de saúde.

Contextualizando, nesta prática, são utilizados dois tipos de análises de dados, **Análise Preditiva**: O objetivo principal dessa análise é prever se uma pessoa tem diabetes com base em suas características ou dados médicos, como idade, IMC, níveis de glicose, histórico familiar, entre outros. E ainda, a **Análise Diagnóstica**: Além de prever o diagnóstico, essa análise pode ser usada para entender os fatores que contribuem mais fortemente para o risco de diabetes. A análise diagnóstica pode ajudar a identificar quais variáveis (por exemplo, glicemia em jejum, IMC) têm maior impacto no diagnóstico de diabetes.

### O que é Machine Learning?

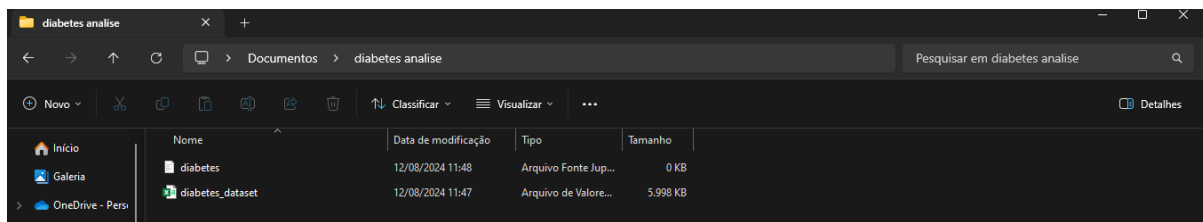
E por fim, como último pré-requisito para entender como realizar o desafio, é entender o que é Machine Learning. Segundo a empresa multinacional **Oracle**: “O machine learning (ML) é o subconjunto da inteligência artificial (IA) que se concentra na construção de sistemas que aprendem, ou melhoram o desempenho, com base nos dados que consomem. A inteligência artificial é um termo amplo que se refere a sistemas ou máquinas que imitam a inteligência humana. O machine learning e a IA são frequentemente abordados juntos, e os termos às vezes são usados de forma intercambiável, mas não significam a mesma coisa. Uma distinção importante é que, embora todo machine learning seja IA, nem toda IA é machine learning.

Hoje, o machine learning funciona ao nosso redor. Quando interagimos com bancos, fazemos compras on-line ou usamos mídias sociais, os algoritmos de machine learning entram em ação para tornar nossa experiência eficiente, suave e segura. O machine learning e a tecnologia em torno dele estão se desenvolvendo rapidamente, e estamos apenas começando a arranhar a superfície de seus recursos.”

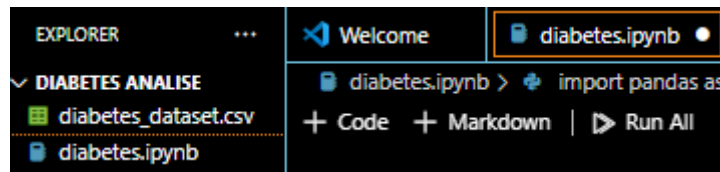
### Iniciando prática

#### Passo 1:

O primeiro passo é criar uma pasta contendo o nosso dataset baixado do Kaggle, e o arquivo .ipynb criado para manipulação.



## Passo 2: Abrir pasta dos arquivos dentro VS Code



## Passo 3: Importar bibliotecas



### Pandas

- **Por que usar:** Pandas é a biblioteca principal para manipulação e análise de dados em Python. Ela facilita a manipulação de dados tabulares, como DataFrames, permitindo filtragem, agregação, e transformação de dados de maneira simples e intuitiva.
- **Aplicação:** Usaremos Pandas para carregar, limpar, e explorar o dataset, além de preparar os dados para análise e modelagem.

### Matplotlib

- **Por que usar:** Matplotlib é uma biblioteca de visualização que permite criar gráficos 2D com alta customização. Embora seja um pouco mais básica, é extremamente flexível, o que a torna uma excelente escolha para criar visualizações detalhadas.
- **Aplicação:** Aplicaremos Matplotlib para gerar gráficos como histogramas, scatter plots, e gráficos de linha, ajudando a visualizar os dados e identificar padrões.

### Seaborn

- **Por que usar:** Seaborn é construída sobre o Matplotlib e é especializada em criar visualizações estatísticas mais atraentes e informativas. Ela oferece uma

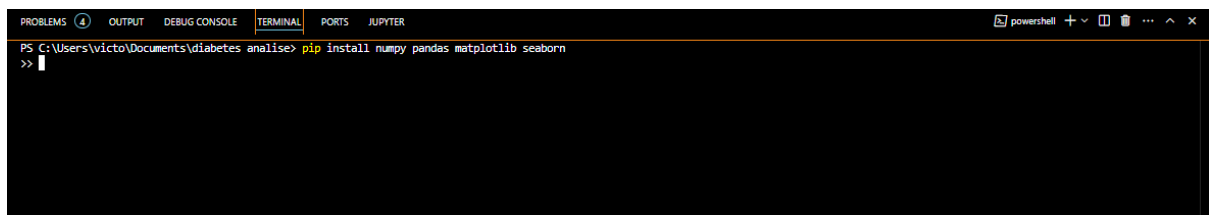


interface de alto nível que simplifica a criação de gráficos complexos e facilita a visualização de padrões e correlações entre variáveis.

- **Aplicação:** Usaremos Seaborn para criar gráficos avançados, como mapas de calor (heatmaps) e gráficos de distribuição, que nos ajudarão a explorar as relações entre diferentes variáveis do dataset.

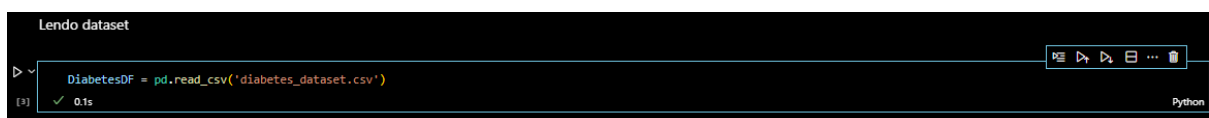
Para instalação das bibliotecas, utilizar dentro do terminal do VS Code, o comando:

*pip install pandas matplotlib seaborn*



```
PS C:\Users\victo\Documents\diabetes analise> pip install numpy pandas matplotlib seaborn
>>
```

#### Passo 4: Leitura do arquivo csv através do Pandas

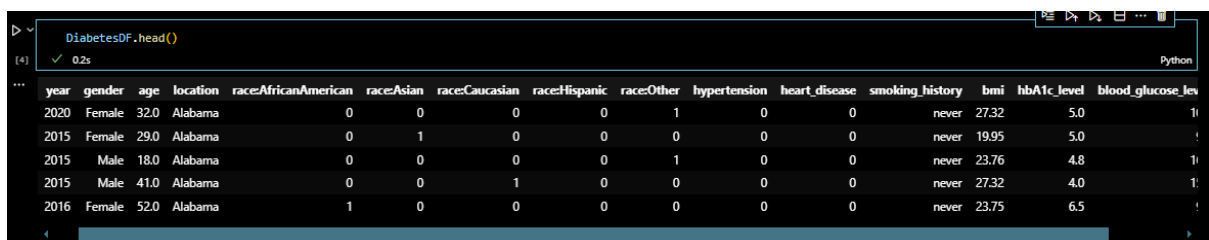


```
DiabetesDF = pd.read_csv('diabetes_dataset.csv')
```

Salvaremos dentro da variável DiabetesDF, através do módulo pd (Apelido dado à biblioteca Pandas), e da função read\_csv, recebendo como parâmetro o nome do arquivo, a leitura do arquivo csv.

#### Passo 5: visualização da estrutura do Data Frame

Utilizando o método head(), podemos ver o nome das colunas, e como estão estruturadas as suas 5 primeiras linhas.



```
DiabetesDF.head()
```

year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian	race:Hispanic	race:Other	hypertension	heart_disease	smoking_history	bmi	hbA1c_level	blood_glucose_level
2020	Female	32.0	Alabama	0	0	0	0	1	0	0	never	27.32	5.0	11
2015	Female	29.0	Alabama	0	1	0	0	0	0	0	never	19.95	5.0	11
2015	Male	18.0	Alabama	0	0	0	0	1	0	0	never	23.76	4.8	11
2015	Male	41.0	Alabama	0	0	1	0	0	0	0	never	27.32	4.0	11
2016	Female	52.0	Alabama	1	0	0	0	0	0	0	never	23.75	6.5	11

#### Passo 6: Analisar detalhadamente estrutura das colunas do DataFrame

```
DiabetesDF.info()
[5] ✓ 0.3s

... <class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   year                  100000 non-null  int64
1   gender                100000 non-null  object
2   age                   100000 non-null  float64
3   location              100000 non-null  object
4   race:AfricanAmerican  100000 non-null  int64
5   race:Asian            100000 non-null  int64
6   race:Caucasian        100000 non-null  int64
7   race:Hispanic         100000 non-null  int64
8   race:Other            100000 non-null  int64
9   hypertension          100000 non-null  int64
10  heart_disease         100000 non-null  int64
11  smoking_history       100000 non-null  object
12  bmi                   100000 non-null  float64
13  hbA1c_level           100000 non-null  float64
14  blood_glucose_level   100000 non-null  int64
15  diabetes              100000 non-null  int64
dtypes: float64(3), int64(10), object(3)
memory usage: 12.2+ MB
```

Percebemos que possuímos, um total de 16 colunas, podemos ver que estão divididas em 2 tipos, os de objeto (que são, basicamente, formas de texto) e as de tipo numérico, além disso, é possível notar que elas não possuem valores nulos, o que é muito bom. Valores nulos, quase sempre precisam ser tratados, e se fossem um problema para essa atividade, era possível checá-los mais detalhadamente desta maneira:

```
DiabetesDF.isnull().sum()
[7] ✓ 0.0s

... year                0
gender                0
age                  0
location             0
race:AfricanAmerican  0
race:Asian           0
race:Caucasian       0
race:Hispanic        0
race:Other           0
hypertension         0
heart_disease        0
smoking_history      0
bmi                  0
hbA1c_level          0
blood_glucose_level  0
diabetes             0
dtype: int64
```

E como estratégia de tratamento, poderíamos excluí-los (opção mais comum) ou substituí-los pela mediana da coluna, dependendo da situação enfrentada.

## Passo 7: Tratamento das colunas

Observa-se que as colunas do tipo objeto podem ser classificadas como categóricas e as de tipo numérico, como o próprio nome diz, numéricas. O problema enfrentado é o seguinte: Modelos de machine learning só podem usar colunas numéricas porque eles realizam cálculos matemáticos para identificar padrões e fazer previsões. Variáveis categóricas, como texto ou rótulos, precisam ser convertidas em números para que o modelo possa processá-las.

E podemos fazer isso da seguinte maneira:

```
colunas_numericas = DiabetesDF.select_dtypes(include='number').columns
colunas_categoricas = DiabetesDF.select_dtypes(exclude='number').columns
```

[28] ✓ 0.0s Python

Ou seja, salvamos nas variáveis "colunas\_numericas" e "colunas\_categoricas" a divisão do em duas classes numéricas e categóricas do Data Frame.

### Exclusão de colunas

Porém, ao avaliarmos as colunas, nota-se que duas delas são passíveis de exclusão, a coluna "year", pois o ano que foi feita catalogação não vai interferir no diagnóstico de uma diabetes do paciente e a outra é a coluna "location", pois por ser uma coluna categórica e possuir muitos valores diferentes, ficará muito complexa uma conversão eficaz para valores numéricos, fugindo da ideia desse e-book.

```
DiabetesDF.unique().sort_values()
```

[8] ✓ 0.1s

...	
race:AfricanAmerican	2
race:Asian	2
race:Caucasian	2
race:Hispanic	2
race:Other	2
hypertension	2
heart_disease	2
diabetes	2
gender	3
smoking_history	6
year	7
hbA1c_level	18
blood_glucose_level	18
location	55
age	102
bmi	4247
dtype: int64	

"location" possuindo 55 valores distintos

```
DiabetesDF = DiabetesDF.drop(columns='year')
DiabetesDF = DiabetesDF.drop(columns='location')
```

[15] ✓ 0.1s

Exclusão das colunas "year" e "location"

```
colunas_numericas
[20] ✓ 0.0s Python
... Index(['age', 'race:AfricanAmerican', 'race:Asian', 'race:Caucasian',
        'race:Hispanic', 'race:Other', 'hypertension', 'heart_disease', 'bmi',
        'hbA1c_level', 'blood_glucose_level', 'diabetes'],
        dtype='object')

colunas_categoricas
[20] ✓ 0.0s Python
... Index(['gender', 'smoking_history'], dtype='object')
```

Exibição de como ficaram armazenadas as variáveis

Transformação das colunas categóricas em valores True ou False:

```
df_dummies = pd.get_dummies(DiabetesDF, colunas_categoricas, drop_first= True)
```

Através do método `get_dummies` do pandas, conseguimos transformar as colunas categóricas em dummies (valores True ou False), salvando o novo Data Frame na variável “`df_dummies`”, além de eliminar as colunas da pré-transformação.

```
> ~
df_dummies = pd.get_dummies(DiabetesDF, colunas_categoricas, drop_first= True)
DiabetesDF.head()
[21] ✓ 0.1s Python
...
  gender  age  race:AfricanAmerican  race:Asian  race:Caucasian  race:Hispanic  race:Other  hypertension  heart_disease  smoking_history  bmi  hbA1c_level  blood_glucose_level  diabetes
0  Female  32.0                0         0         0         0         1         0         0         never  27.32         5.0         100         0
1  Female  29.0                0         1         0         0         0         0         0         never  19.95         5.0         90         0
2   Male  18.0                0         0         0         0         1         0         0         never  23.76         4.8         160         0
3   Male  41.0                0         0         1         0         0         0         0         never  27.32         4.0         159         0
4  Female  52.0                1         0         0         0         0         0         0         never  23.75         6.5         90         0

df_dummies.head()
[23] ✓ 0.0s Python
...
  hbA1c_level  blood_glucose_level  diabetes  gender_Male  gender_Other  smoking_history_current  smoking_history_ever  smoking_history_former  smoking_history_never  smoking_history_not
i
!         5.0             100         0         False         False             False             False             False             True             False
i         5.0             90         0         False         False             False             False             False             True             False
i         4.8            160         0         True         False             False             False             False             True             False
!         4.0            159         0         True         False             False             False             False             True             False
i         6.5             90         0         False         False             False             False             False             True             False
```

Comparativo entre os dois Data Frames

Nota-se, que antes a coluna “`gender`” continha os valores “Male”, “Female” e “Other”, agora, ao ser transformada, nas colunas “`gender_Male`”, estão os valores True se for homem, e False se não for; e na coluna “`gender_Other`”, também está sendo True ou False para os valores. Caso você esteja se perguntando onde está a coluna para Females, ela não é necessária. Pois, caso uma linha da tabela seja False para ambas as colunas, se entente que se trata de uma mulher. Enquanto a coluna “`smoking_history`”, subdividiu-se nas últimas 5 colunas da imagem, recebendo valores True ou False.

## Passo 8: Preparação para criação do modelo

Separar as features (variáveis independentes) do target (variável dependente) é essencial para garantir que o modelo de machine learning aprenda corretamente. Isso evita que o modelo tenha acesso à resposta correta durante o treinamento, o que resultaria em um modelo que “memoriza” os dados em vez de aprender a generalizar

para novos exemplos. Assim, ao separar as features do target, garantimos que o modelo treine de forma robusta e possa fazer previsões precisas em dados novos e não vistos.

```
▶ ▾  
# Features  
X = df_dummies.drop('diabetes',axis=1)  
  
# Target(alvo)  
  
y = df_dummies['diabetes']  
[24] ✓ 0.0s
```

Nesse caso, separamos nossa variável alvo (“diabetes”), pois nela contém os nossos valores dummies, para classificar o paciente com a doença ou não.

### Passo 9: Importação da biblioteca scikit-learn

A **scikit-learn** é uma biblioteca essencial para machine learning em Python. Ela oferece uma ampla gama de algoritmos de aprendizado supervisionado e não supervisionado, além de ferramentas para pré-processamento de dados, validação cruzada, e avaliação de modelos. Sua simplicidade e eficiência tornam-na ideal tanto para iniciantes quanto para profissionais, facilitando a implementação rápida e eficaz de modelos de machine learning.

```
▶ ▾  
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)  
[25] ✓ 0.1s
```

Uma prática essencial ao construir modelos de machine learning é dividir os dados disponíveis em dois conjuntos principais: **conjunto de treinamento** e **conjunto de teste**. Essa separação é crucial para avaliar a capacidade do modelo de generalizar para novos dados. Aqui, estamos importando a função `train_test_split` do módulo `model_selection` da biblioteca `scikit-learn`. Essa função é responsável por dividir os dados em conjuntos de treinamento e teste.

- X e y são as variáveis que representam as características (features) e os rótulos (labels) dos dados, respectivamente.
- X\_train e y\_train são os conjuntos de dados que serão utilizados para treinar o modelo.
- X\_test e y\_test são os conjuntos de dados que serão utilizados para testar o desempenho do modelo após o treinamento.

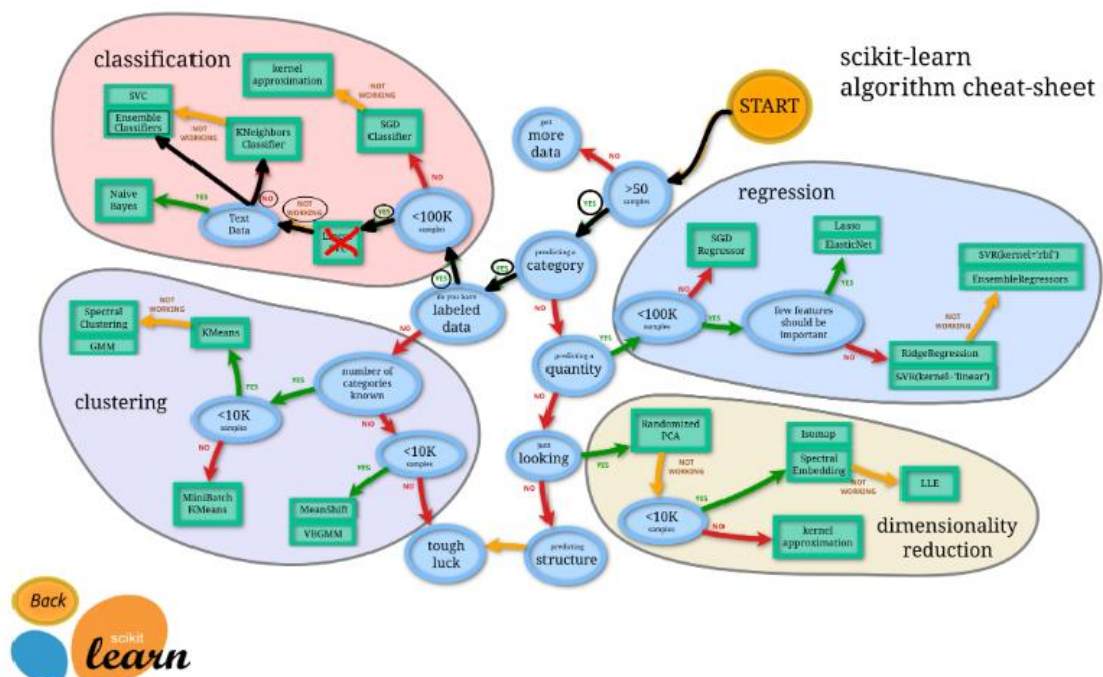
“**test\_size=0.2**”: Este parâmetro define a proporção dos dados que serão reservados para o conjunto de teste. Neste caso, 20% dos dados serão utilizados para o teste, enquanto os outros 80% serão usados para o treinamento. “**random\_state=42**”: O parâmetro `random_state` é utilizado para garantir que a divisão dos dados seja

reproduzível. Ao definir um número específico (neste caso, 42), estamos garantindo que, a cada execução do código, os mesmos dados serão selecionados para o treinamento e teste.

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
✓ 0.0s
((80000, 18), (80000,), (20000, 18), (20000,))
```

Confirmação das divisões entre 20% e 80%

## Passo 10: Escolher qual tipo de modelo utilizar, Classificação ou Regressão?



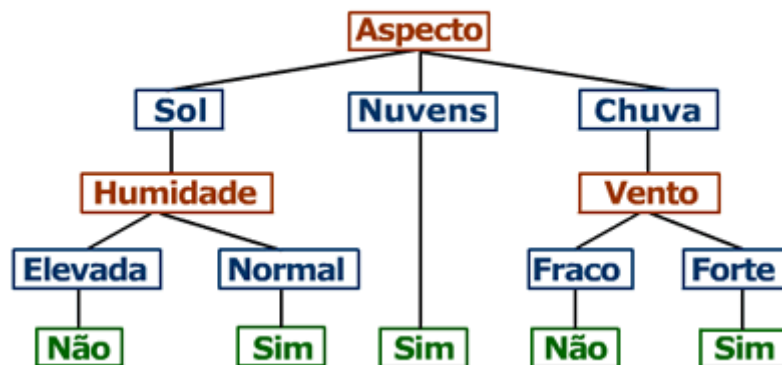
Modelos de **classificação** e **regressão** são usados em machine learning para diferentes tipos de previsões. **Classificação** prevê categorias ou classes discretas, como "diabético" ou "não diabético". **Regressão**, por outro lado, prevê valores contínuos, como o preço de uma casa ou o nível de glicose no sangue. Em resumo, a classificação lida com escolhas entre opções predefinidas, enquanto a regressão prevê valores numéricos em um intervalo contínuo.

## Passo 11: Importando modelo XGboost e entendendo-o

```
import xgboost as xgb
✓ 2.5s
```

O modelo Xgboost é um modelo de classificação baseado em várias árvores de decisão em sequência, mas o que é uma árvore de decisão?

### Árvore de Decisão para Jogar Tênis



Segundo o site da IBM: “Modelos de árvore de decisão para desenvolver sistemas de classificação que prevejam ou classifiquem observações futuras com base em um conjunto de regras de decisão. Se você tiver dados divididos em classes de seu interesse (por exemplo, empréstimos de alto versus baixo risco, assinantes versus não assinantes, votantes versus não votantes, ou tipos de bactérias), seus dados poderão ser usados para construir regras que possam ser utilizadas para classificar casos novos ou antigos com máxima precisão. Por exemplo, é possível construir uma árvore que classifica o risco de crédito ou intenções de compra com base na idade e em outros fatores. Esta abordagem, às vezes conhecida como indução de regra, tem várias vantagens. Primeiro, o processo de raciocínio por trás do modelo é evidente quando navegar pela árvore. Isso está em contraste com outras técnicas de modelagem caixa preta em que a lógica interna pode ser difícil de trabalhar. Segundo, o processo inclui automaticamente em sua regra apenas os atributos que realmente forem importantes para uma tomada de decisão. Os atributos que não contribuírem com a precisão da árvore são ignorados. Isso pode produzir informações muito úteis sobre os dados e também ser utilizado para reduzir os dados para campos relevantes antes de treinar outra técnica de aprendizado, como uma rede neural.”

```
▶ xgb_classifier = xgb.XGBClassifier()
  xgb_classifier.fit(X_train, y_train)
[30] ✓ 0.5s
```

- **xgb.XGBClassifier()** inicializa um modelo de classificação usando o algoritmo XGBoost
- **xgb\_classifier.fit(X\_train, y\_train)** treina o modelo com os dados de treinamento, ajustando-o para que possa prever as classes com base nas características fornecidas em X\_train e nos rótulos y\_train.

```
▶ y_pred = xgb_classifier.predict(X_test)
[31] ✓ 0.0s
```

O código `y_pred = xgb_classifier.predict(X_test)` utiliza o modelo treinado `xgb_classifier` para fazer previsões. Ele gera as previsões (`y_pred`) para o conjunto de teste `X_test`, determinando a classe (por exemplo, "diabético" ou "não diabético") para cada entrada no conjunto de teste. Essas previsões podem então ser comparadas com os valores reais para avaliar a precisão do modelo.

## Passo 12: Escolher métrica de precisão

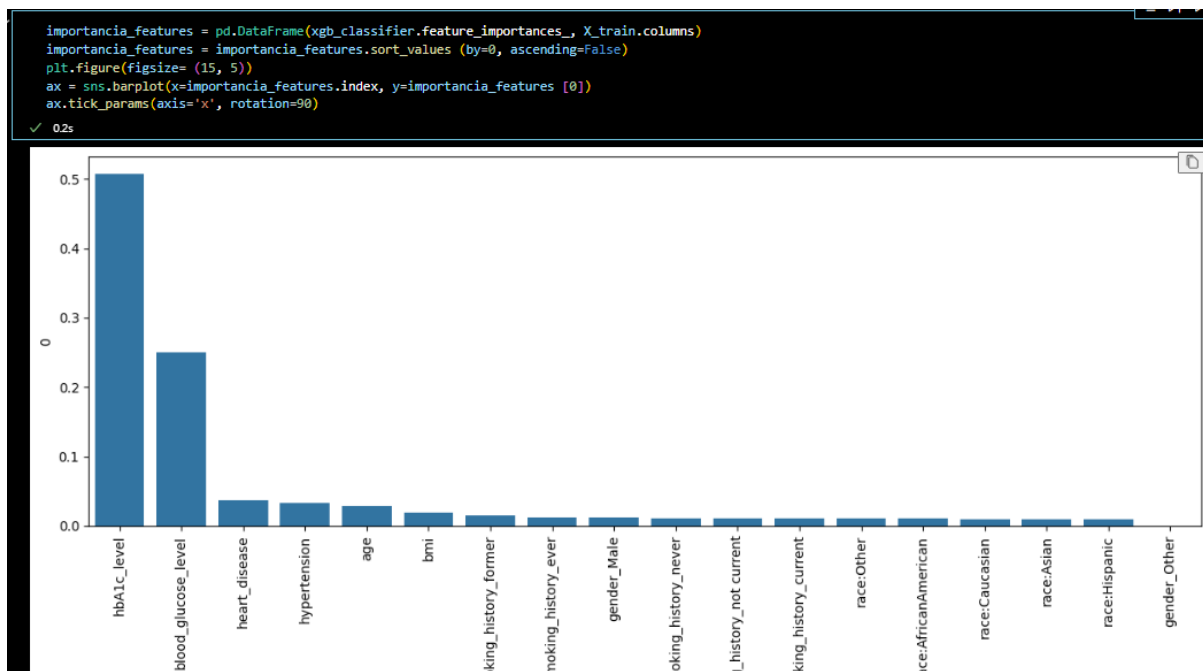
Para este caso, foi considerada mais adequada a métrica de **Acurácia**, pois é uma métrica de avaliação usada para medir o desempenho de modelos de classificação. Ela representa a proporção de previsões corretas feitas pelo modelo em relação ao total de previsões. Em outras palavras, é a razão entre o número de classificações corretas e o número total de exemplos avaliados. Embora seja uma métrica útil, a acurácia pode ser enganosa em casos de classes desbalanceadas, onde uma classe é muito mais frequente que a outra.

```
from sklearn.metrics import accuracy_score
acuracia = accuracy_score(y_test, y_pred)
print(acuracia)
[32] ✓ 0.0s
... 0.97145 Python
```

Neste caso, importamos a métrica de acurácia do scikit-learn, e utilizamos a função `accuracy_score` comparando os parâmetros `y_test` e `y_pred`, chegando à precisão de 97,14% do modelo.

## Passo 13: Descobrir quais features foram mais relevantes para predição do modelo





O código avaliou a importância das características no modelo de classificação XGBoost. Primeiramente, a importância das características é extraída do modelo e organizada em um DataFrame com os nomes das colunas de `X_train`. O DataFrame é então ordenado com base na importância das características, em ordem decrescente. Em seguida, é gerado um gráfico de barras usando a biblioteca Seaborn, onde cada barra representa a importância de uma característica no modelo. A rotação dos rótulos do eixo x é ajustada para melhorar a legibilidade. Este processo ajuda a identificar que o **nível de HbA1c** e o **nível de glicose no sangue** foram os **mais influentes** na construção do modelo.

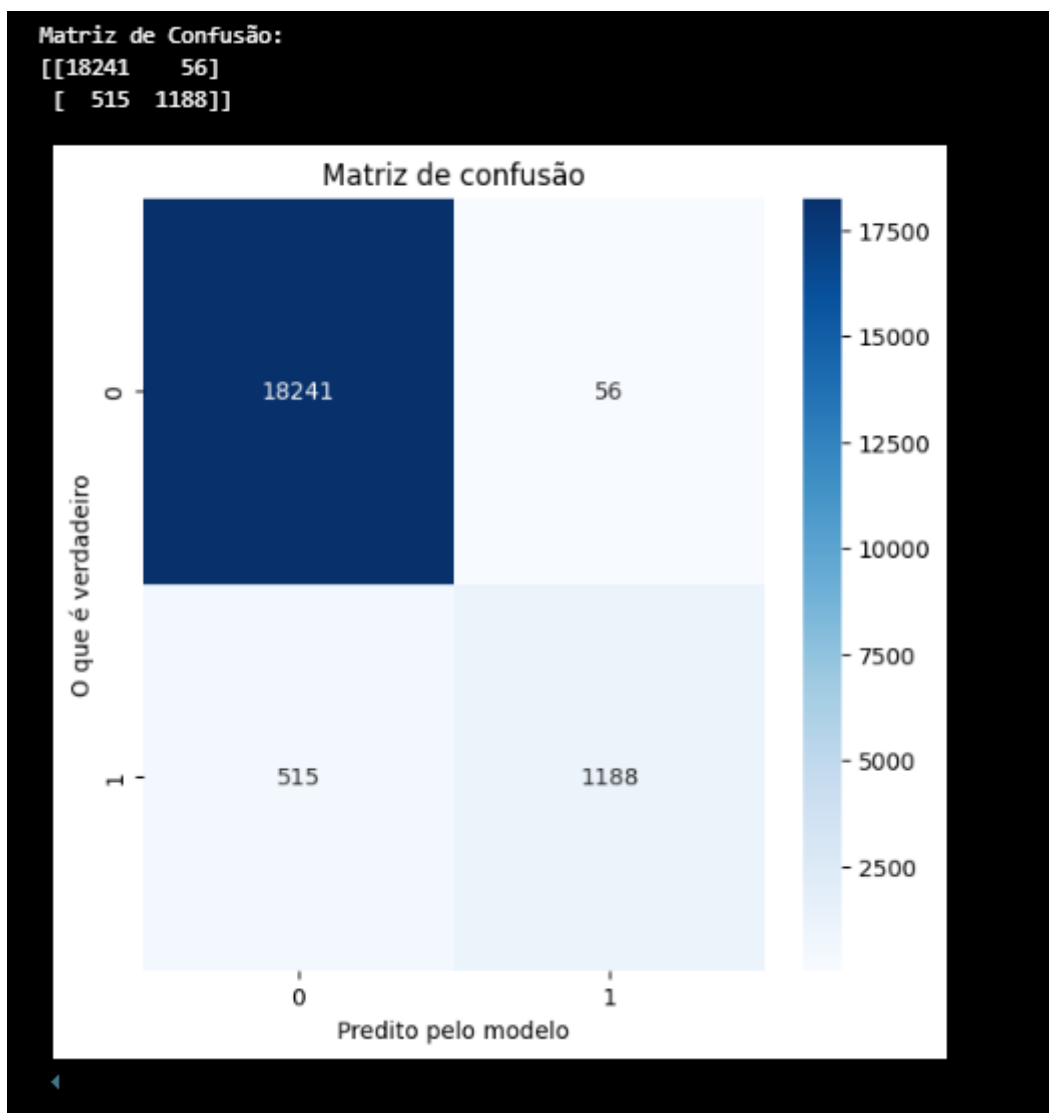
#### Passo 14: Entender o resultado e concluir sobre o modelo

Para entendermos o que significa esses 97,14%, podemos criar um gráfico de uma matriz de confusão. A **matriz de confusão** é uma ferramenta de avaliação de modelos de classificação que mostra o desempenho do modelo em termos de suas previsões corretas e incorretas. Ela é organizada da seguinte forma:

- **Verdadeiros Positivos (TP):** Casos onde o modelo corretamente previu a classe positiva.
- **Falsos Positivos (FP):** Casos onde o modelo previu a classe positiva incorretamente.
- **Verdadeiros Negativos (TN):** Casos onde o modelo corretamente previu a classe negativa.
- **Falsos Negativos (FN):** Casos onde o modelo previu a classe negativa incorretamente.

```
#Gerar matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)
print('Matriz de Confusão:')
print(conf_matrix)
# Gerar gráfico da matriz
plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=DiabetesDF['diabetes'].unique(), yticklabels=DiabetesDF['diabetes'].unique())
plt.xlabel("Predito pelo modelo")
plt.ylabel("O que é verdadeiro")
plt.title("Matriz de confusão")
plt.show()
```

No input, O código gera e visualiza a matriz de confusão para avaliar o desempenho do modelo. Primeiro, calcula a matriz de confusão comparando os rótulos reais (y\_test) com as previsões (y\_pred). Em seguida, usando as bibliotecas do seaborn e matplotlib, é criado um mapa de calor para criar uma visualização clara dessa matriz, mostrando os valores em cada célula com anotações e rótulos nos eixos para facilitar a interpretação das previsões corretas e incorretas.



Concluindo sobre o output, basicamente, **515** pacientes foram preditos inadequadamente com diabetes, **1188** foram preditos adequadamente com diabetes, **18241** foram preditas adequadamente livres da doença e **56** pessoas

**foram preditos inadequadamente livres da doença, que é o cenário mais grave.** A soma de todos esses fatores resultou na acurácia de cerca de 97% do nosso modelo, sendo considerado, na maioria dos casos, um modelo eficiente. Porém, por se tratar de um modelo de uso medicinal, seu resultado ainda é insatisfatório, pois, nesses casos, o número de falhas deve estar próximo, ao máximo possível do valor zero. Como este projeto é de uso acadêmico, o resultado do modelo é mais que satisfatório.

## Referências:

ORACLE. What is machine learning? Oracle. Disponível em:

<https://www.oracle.com/br/artificial-intelligence/machine-learning/what-is-machine-learning/>. Acesso em: 12 ago. 2024.

IBM. Decision tree models. In: \*IBM SPSS Modeler Documentation\*. Versão 18.4.0.

Disponível em: <https://www.ibm.com/docs/pt-br/spss-modeler/18.4.0?topic=trees-decision-tree-models>. Acesso em: 12 ago. 2024.

Universidade dos Dados. YouTube. Disponível em:

<https://www.youtube.com/@universidade-dos-dados>. Acesso em: 12 ago. 2024.

SCIKIT-LEARN. \*scikit-learn: Machine learning in Python\*. Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 12 ago. 2024.

Material disponível no seguinte repositório:

<https://github.com/victor7miguel7/Previs-o-de-diabetes-para-ebook>