



“Almirante Tamandaré”
São Bernardo do Campo
sbc.sp.senai.br

Fundamentos de *Big Data & Data Analytics* com Python

Prof. Vinícius B. Suterio

Serviço Nacional de Aprendizagem Industrial
Departamento Regional de São Paulo
Área: Tecnologia da Informação

vinicius.suterio@sp.senai.br

2o. Semestre
2023

1.1.1. FIC: *Big Data & Data Analytics* com Python

Objetivo:

- O Curso de Aperfeiçoamento Profissional Fundamentos de *Big Data* e *Data Analytics* com Python tem por objetivo o **desenvolvimento de competências relativas à extração e transformação de dados utilizando softwares específicos**, aplicando técnicas de *machine learning* e gerando informações para **tomada de decisão** seguindo procedimentos e normas com análise estatística.

Requisitos:

- Ter concluído o Ensino Médio;
- Ter, no mínimo, 16 anos;
- Ter conhecimento em linguagem de programação.

Perfil da Qualificação Profissional:

- Solução de problemas com extração e transformação de dados por meio de softwares específicos, usando técnicas de *machine learning* e estatística para auxílio em tomadas de decisões.

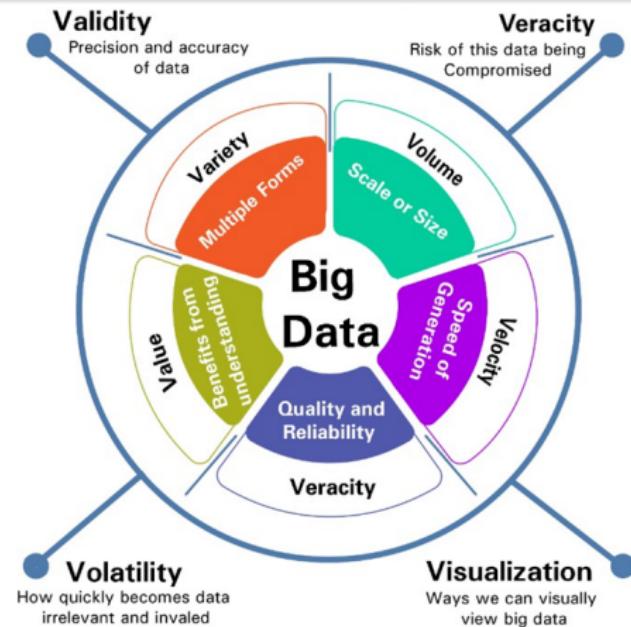
1.2.1. Organização curricular: *Big Data & Data Analytics*

LEGISLAÇÃO	UNIDADES CURRICULARES	CARGA HORÁRIA TOTAL (HORAS)
Lei Federal nº 9.394/96 e Decreto Federal nº 5.154/04	Fundamentos de Big Data e Data Analytics com Python	60
	Carga Horária Total	60

1.3.1. Ementário: *Big Data & Data Analytics*

Big Data & Data Analytics - 60h:

- *Big Data:*
 - Histórico e definição;
 - Aplicações;
 - Características dos 5Vs:
 - Volume;
 - Velocidade;
 - Valor;
 - Variedade;
 - Veracidade.
 - Etapas dos processos
- *Banco de dados:*
 - Definição;
 - Características;
 - Tipos:
 - SQL;
 - NoSQL.



target_date	target_time	select_target_date	target_time	server_target_time	remote_time	remote_ip	remote_port
2016-12-26	02:29:30	14827371902					
2016-12-26	02:32:29	14827371902					
2016-12-26	02:32:29	14827371902					
2016-12-26	02:35:39	14827371902					
2016-12-26	02:39:39	14827371902					
2016-12-26	02:39:39	14827371902					
2016-12-26	02:39:39	14827371902					
2016-12-26	02:41:30	14827371902					
2016-12-26	02:41:30	14827371902					
2016-12-26	02:44:29	14827370510					
2016-12-26	02:44:29	14827370510					



1.3.1.1. Ementário: *Big Data & Data Analytics*

Big Data & Data Analytics - 60h:

- Comandos em SQL, Spark e PySpark;

- *Machine Learning:*

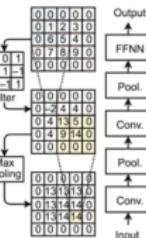
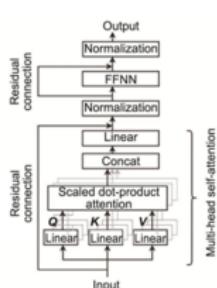
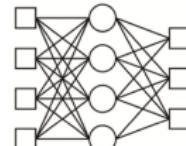
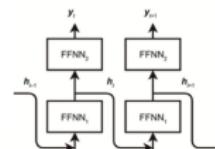
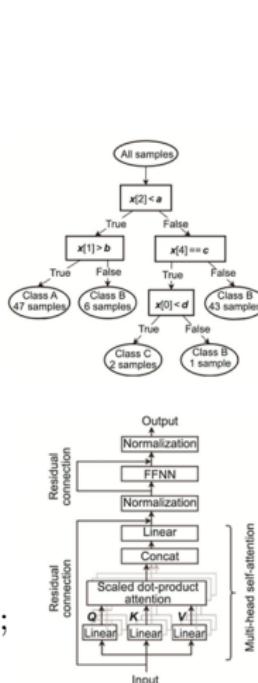
- Regressão linear;
- Árvore de decisão.

- *Dashboards:*

- PowerBI e Tableau.

- *Matemática Estatística:*

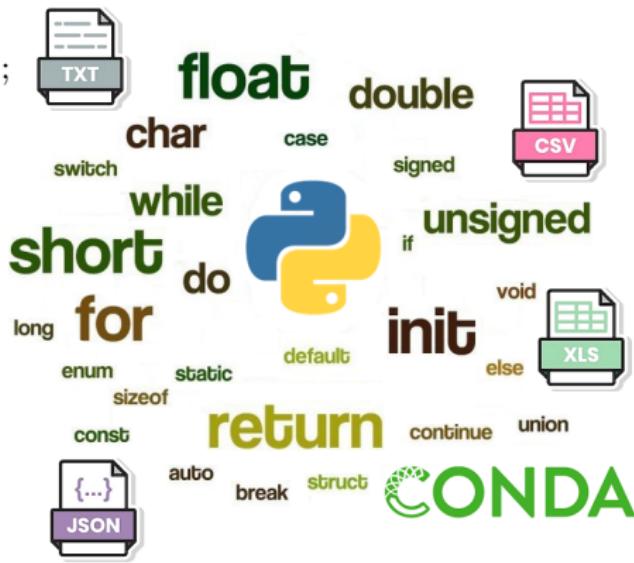
- Definições;
- Funções interativas:
 - Média / moda;
 - Variância;
 - Desvio padrão.
- Estatística descritiva;
- Tabulação de dados;
- Gráficos: linha, coluna, dispersão, etc.



1.3.1.2. Ementário: *Big Data & Data Analytics*

Big Data & Data Analytics - 60h:

- Diretórios e arquivos:
 - *Prompt* de comando e Estrutura;
 - Dados simples (txt, csv, excel e json);
- Linguagem de programação - Python:
 - Histórico;
 - Compiladores/*interpretadores**
 - Bibliotecas: Pip e Conda;
 - Atualizações e repositórios;
- Variáveis e parâmetros:
 - Global/local;
 - *Keywords*, *VarArgs*, *Init* e *Return*;
 - *Self*, herança e dependência;
 - Dados: tipos e classes;
 - Fluxo de controle:
 - *if/else*; *for*; *while*; *break*; *continue*.



1.3.1.3. Ementário: *Big Data & Data Analytics*

Big Data & Data Analytics - 60h:

- Biblioteca Numpy:

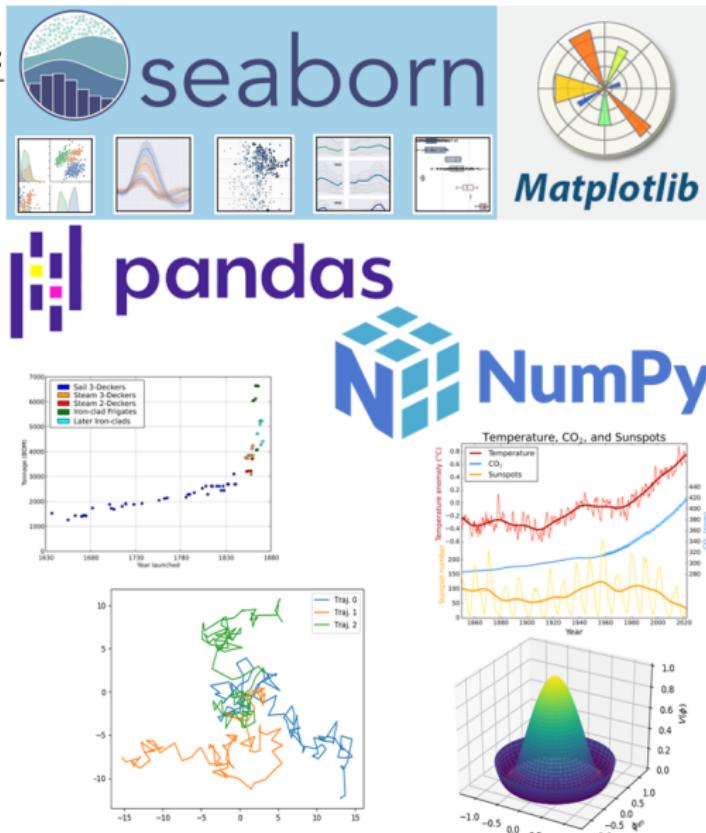
- Comandos;
- Funções;
- Matrizes/equações;
- Gráficos.

- Bibliotecas: Pandas e Seaborn:

- Tabelas/comandos;
- Dados: *import/export*;
- *Dataframes*;
- Dicionários, tuplas e filtros;

- Biblioteca Matplotlib:

- Plotagem;
- Exportação de gráficos;
- KPIs.



1.3.1.7. Ementário: *Big Data & Data Analytics*

Procedimentos de ensino:

- Aulas Teóricas:
 - Lousa/Multimídia;
 - Realização de algoritmos (exemplos e fixação);
 - Elaboração/Análise de projetos utilizando análise de dados com Python.
- Aulas Práticas:
 - Desenvolvimento de lógica utilizando linguagem Python;
 - Familiarizar-se com a linguagem e comandos básicos na modelagem de dados;
 - Manipulação de informações nos bancos de dados (pré-processamento);
 - Obtenção de *features* por meio de análise dos dados utilizando estatística;
 - Técnicas computacionais para avaliação de *features* - criação de indicadores;

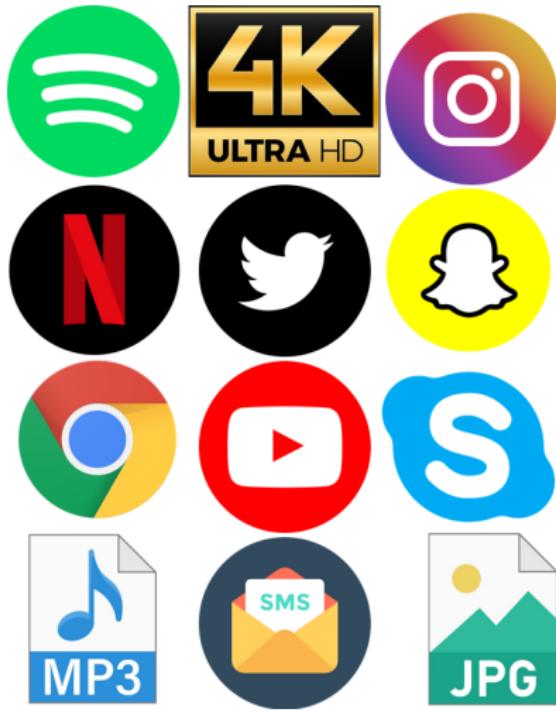
2.1.1. Big Data & Data Analytics

- **Big Data** consiste na área do conhecimento que realiza o **tratamento, análise e obtenção de informações** a partir de um **conjunto de dados** consideravelmente grande;
 - Atualmente cientistas dizem que os dados - *data* - são tão importantes quanto a criação/desenvolvimento de algoritmos;
 - Aproximadamente **2,5 quintilhões** de bytes (**2,5 Ebyte** - $2,5 \times 10^{18}$) são criados diariamente e 90% dos dados mundiais foram criados apenas nos **últimos 2 anos**;
 - Estima-se que a partir de 2025 o fornecimento global de dados atingirá **175 Zbytes** (**175 trilhões de Gbytes**) **anualmente**;
 - Basicamente *data* consiste em **todo tipo de informação** que se deseja tratar/analisar que pode ser uni, bi ou multidimensional.



2.1.1.1. Big Data & Data Analytics

- Para que tenhamos uma pequena noção:
 - Música MP3 ≈ 1 a 2,4 MB/min;
 - Foto JPEG ≈ 8 a 10 MB/foto;
 - Vídeo 4K ≈ 350 MB/min;
 - DVD ≈ 8,5 GB (141 horas de música);
 - HD ≈ 1 TB (84 horas de vídeo 4K).
- O quanto de *data* é gerado por minuto?
 - 473.400 *tweets* enviados;
 - 2.083.333 *Snapchats* compartilhados;
 - 97.222 horas de Netflix;
 - 12.986.111 mensagens de texto enviada;
 - 49.380 Posts de *Instagram*;
 - 176.220 ligações no *Skype*;
 - 750.000 músicas transmitidas no *Spotify*;
 - 3.877.140 Pesquisas no Google;
 - 4.333.560 Vídeos assistidos no YouTube.

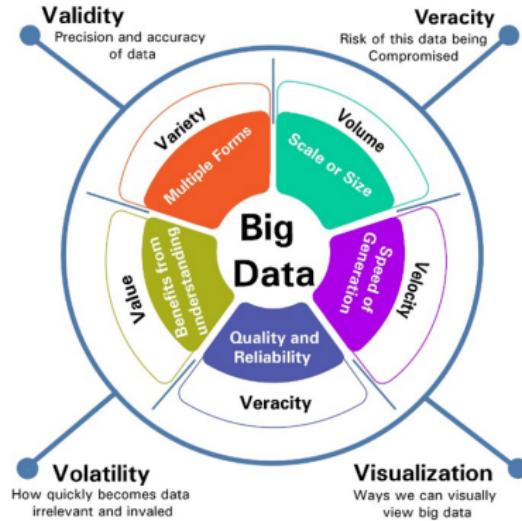


2.1.1.2. Big Data & Data Analytics

- A “explosão” do *Big Data* provavelmente continuará de forma exponencial nos próximos anos e com aproximadamente 50bi de dispositivos conectados por *IoT* gerando “dados”, onde iremos parar?

- Os V's do Big Data:

- **Volume:** Quantidade de dados produzidos/obtidos - podem ou não ter baixa densidade de informação (**relevância**);
- **Velocidade:** O quão rápido os dados são produzidos, coletados e até modificados (tomada de decisão - *real time*);
- **Valor:** O quão relevante são os dados. Mensurar o **grau de significância para o processo**. Quanto irá agregar em valor?
- **Variedade:** Diferentes formatos - alfanuméricos, áudios, vídeos, sensores, etc;
- **Veracidade:** Nível de confiabilidade dos dados. São completos, precisos e reais?



2.1.1.3. Big Data & Data Analytics

- A maior parte dos dados são criados digitalmente em uma variedade de **tipos**, **volumes** extraordinários e movem-se em **velocidades** surpreendentes nas mais diversas aplicações:
 - Desenvolvimento de produtos;
 - Manutenção preditiva;
 - *Machine Learning*;
 - Otimização de processos - performance;
 - Eficiência operacional - estratégia, etc;
- Um dos aplicativos de **Big Data** mais utilizado é o de navegação GPS - **Waze** com 90mi de usuários mensais;
- Os primeiros dispositivos GPS dependiam de estatística, mapas e coordenadas. O Waze, devido sua grande quantidade de dados em tempo real, se ajusta dinamicamente com relação ao mapa, alertas, trânsito, etc.



2.2.1. O que utilizaremos em nosso curso?

Software **ANACONDA** consiste em um conjunto de “bibliotecas” com foco nas linguagens de programação **Python** e R que simplificam o gerenciamento e implantação de pacotes.

- Aplicação principalmente em *Data Science: machine learning*, processamento de dados em larga escala, análise preditiva, etc.

O notebook **JUPYTER** consiste em um ambiente computacional web para criação de documentos da plataforma Jupyter. Tal documento é estruturado no formato JSON.

- Contém uma lista ordenada de células, entrada/saída, pode conter código (algoritmo), texto (Markdown), gráficos, etc.

O **VSCODE** é um editor de código-fonte usado para diversas linguagens de programação e ≠'s Sist. Op. Possui recursos de depuração, destaque de sintaxe, refatoração de código, etc.



2.2.2. Download e instalação do ANACONDA

Link do software **ANACONDA**: <https://www.anaconda.com/download>

The screenshot shows a web browser displaying the Anaconda Distribution download page. The page has a dark green background with white text. At the top, there is a navigation bar with links for 'Enterprise', 'Pricing', 'Resources', and 'About'. On the right side of the navigation bar is a 'Contact Sales' button. Below the navigation bar, the text 'Anaconda Distribution' is displayed in white. The main title 'Free Download' is prominently shown in large white letters. A subtitle below it reads 'Everything you need to get started in data science on your workstation.' To the left of the subtitle is a bulleted list of features, each preceded by a green checkmark. The features listed are: 'Free distribution install', 'Thousands of the most fundamental DS, AI, and ML packages', 'Manage packages and environments from desktop application', and 'Deploy across hardware and software platforms'. At the bottom of the page, there are two green buttons: 'Code in the Cloud' and 'Download'. Below these buttons is a link 'Get Additional Installers' followed by icons for Windows, Mac, and Linux.

2.2.3. Download e instalação do VSCode

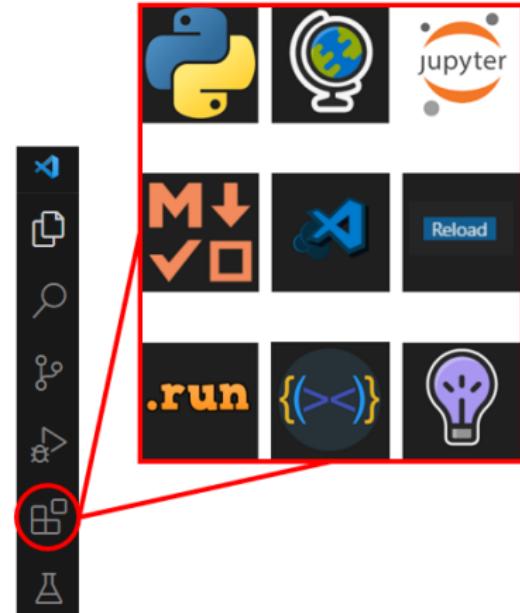
Link do software **VSCode**: <https://code.visualstudio.com/download#>

The screenshot shows the official Visual Studio Code download page at code.visualstudio.com/download#. The page features a dark header with the Visual Studio Code logo and navigation links for Docs, Updates, Blog, API, Extensions, FAQ, Learn, and a prominent blue 'Download' button. A message at the top indicates 'Version 1.83' is available with new features and fixes from September. Below the header, the main title 'Download Visual Studio Code' is displayed, followed by the subtitle 'Free and built on open source. Integrated Git, debugging and extensions.' Three large download buttons are shown: 'Windows' (with icons for Windows 10, 11), 'Linux' (with a Tux icon), and 'Mac' (with an Apple icon). Each button has a sub-section below it listing various installer options and their supported architectures.

Platform	Installer Type	Architectures
Windows	User Installer	x64 x86 Arm64
	System Installer	x64 x86 Arm64
	.zip	x64 x86 Arm64
	CLI	x64 x86 Arm64
Linux	.deb	x64 Arm32 Arm64
	.rpm	x64 Arm32 Arm64
	.tar.gz	x64 Arm32 Arm64
	Snap	Snap Store
Mac	.zip	Intel chip Apple silicon Universal
	.rpm	Intel chip Apple silicon
	.tar.gz	Intel chip Apple silicon
	CLI	x64 Arm32 Arm64

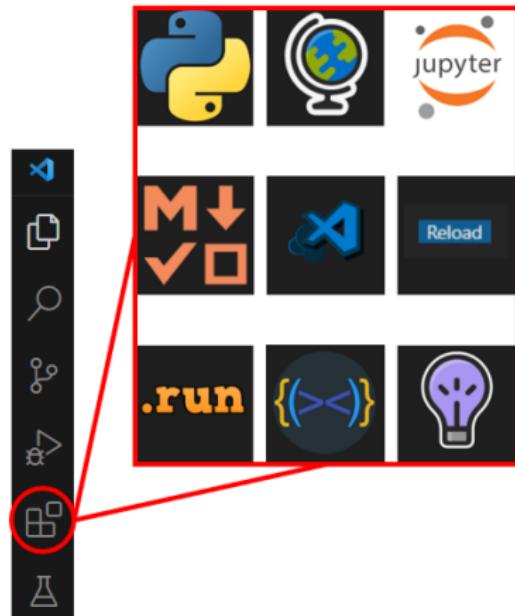
2.2.4. Configurando extensões do VSCode

- *Python, Microsoft*: Permite acesso a diversas funcionalidades para os programadores Phyton usando VSCode - selecionar interpretador, *helper*, etc (essencial);
- *Portuguese (Brazil) Language Pack for Visual Studio, Microsoft*: Pacote português;
- *Jupyter, Microsoft*: Garante a integração com o jupyter notebook e fornece ferramentas que facilitarão o manuseio de dados;
- *Markdown Checkboxes, Matt Bierner*: criação dos checkboxes e interface com formatações que garantem a legibilidade;
- *VSCode-Icons, VSCode Icons Team*: Construção de uma interface mais agradável com um layout mais intuitivo facilitando a busca de arquivos na barra de diretório;



2.2.4.1. Configurando extensões do VSCode

- **Reload:** Facilita o processo de reinicialização do VSCode quando se faz necessário atualizar as configurações;
- **Code Runner, Jun Han:** Execução de trechos de algoritmos em ≠'s linguagens de programação - arquivos com extensão .py;
- **Rainbow Brackets, Mhammed Talhaouy:** formatação intuitiva de parênteses para melhor interpretação do algoritmo;
- **IntelliCode, Microsoft:** IA que sugere o preenchimento das instruções utilizadas (complementa a digitação do programador).



♥ IMPORTANTE ♥

- **Run Code:** Configurar o *Code-runner* → habilitar *Code-runner: Run In Terminal* ✓.
- **Executar Arquivo do Python:** Mudar de *PowerShell* → *Command Prompt*.
- **Configurações:** → habilitar *Jupyter>Interactive Window>Text Editor:Execute Selection* ✓;

2.2.5. Testando o VSCode + Anaconda

Arquivos Python extensão .py → criação de *scripts*:

The screenshot shows the Visual Studio Code (VSCode) interface with the following details:

- File Explorer:** Shows a workspace named "SEM TÍTULO (WORKSPACE)" containing files: "HelloWorld.ipynb", "TesteProgr.ipynb", and "OlaMundo.py". The "OlaMundo.py" file is selected and highlighted with a red box.
- Code Editor:** Displays the content of "OlaMundo.py":

```
1 print('Olá Mundo!')
```
- Terminal:** Shows two sessions:
 - Session 1 (Windows Terminal):

```
(base) C:\TesteProgr>python -u "c:\TesteProgr\OlaMundo.py"
Olá Mundo!
```
 - Session 2 (Windows Terminal):

```
2 Microsoft Windows [versão 10.0.19045.3570]
(c) Microsoft Corporation. Todos os direitos reservados.

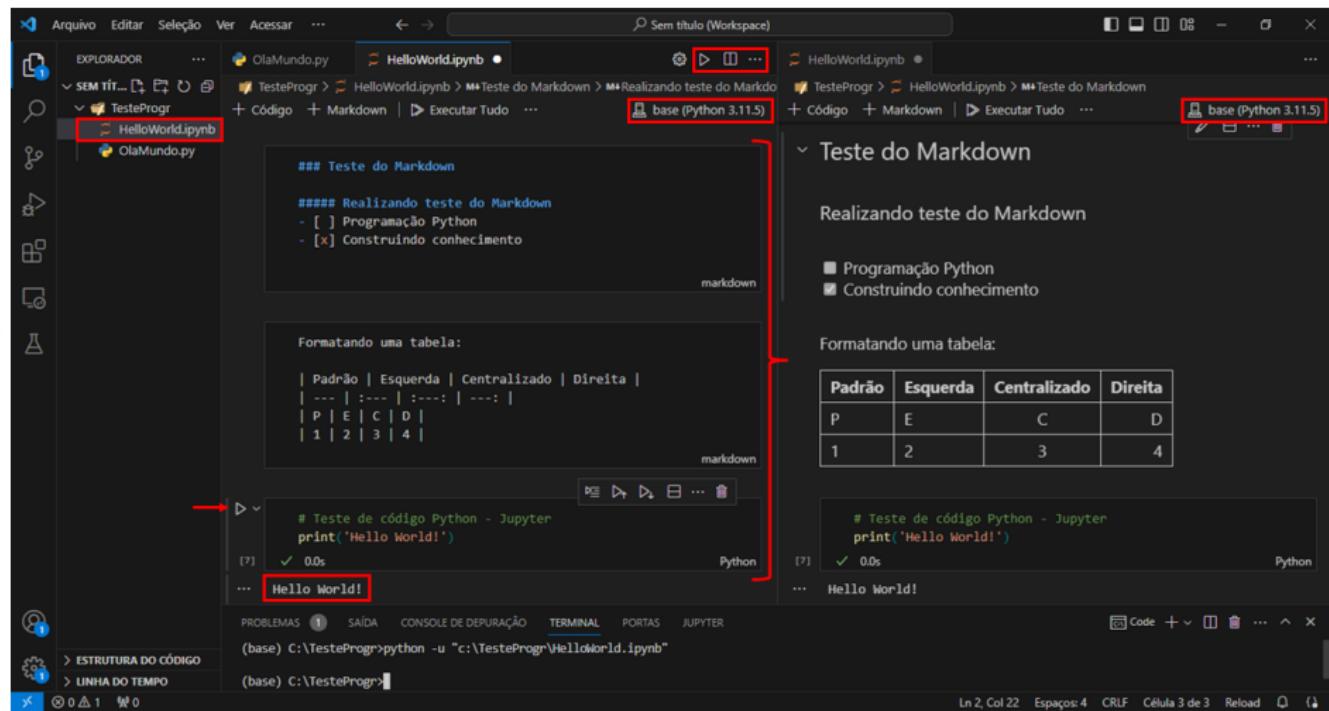
C:\TesteProgr>C:/ProgramData/anaconda3/Scripts/activate

(base) C:\TesteProgr>conda activate base

(base) C:\TesteProgr>C:/ProgramData/anaconda3/python.exe c:/TesteProgr/OlaMundo.py
Olá Mundo!
```
- Contextual Menu:** A context menu is open over the "OlaMundo.py" file in the Explorer. It contains the following options:
 - Run Code (Ctrl+Alt+N)
 - Executar Arquivo do Python
 - Executar Arquivo Python no Terminal Dedicado
 - Depurar Arquivo do Python
 - Executar Arquivo Atual na Janela Interativa
- Status Bar:** Shows the current file path as "3.11.5 (base):conda", along with other status information like line and column numbers (Ln 1, Col 20), character count (Espaços: 4), and encoding (UTF-8).

2.2.5.1. Testando o VSCode + Anaconda

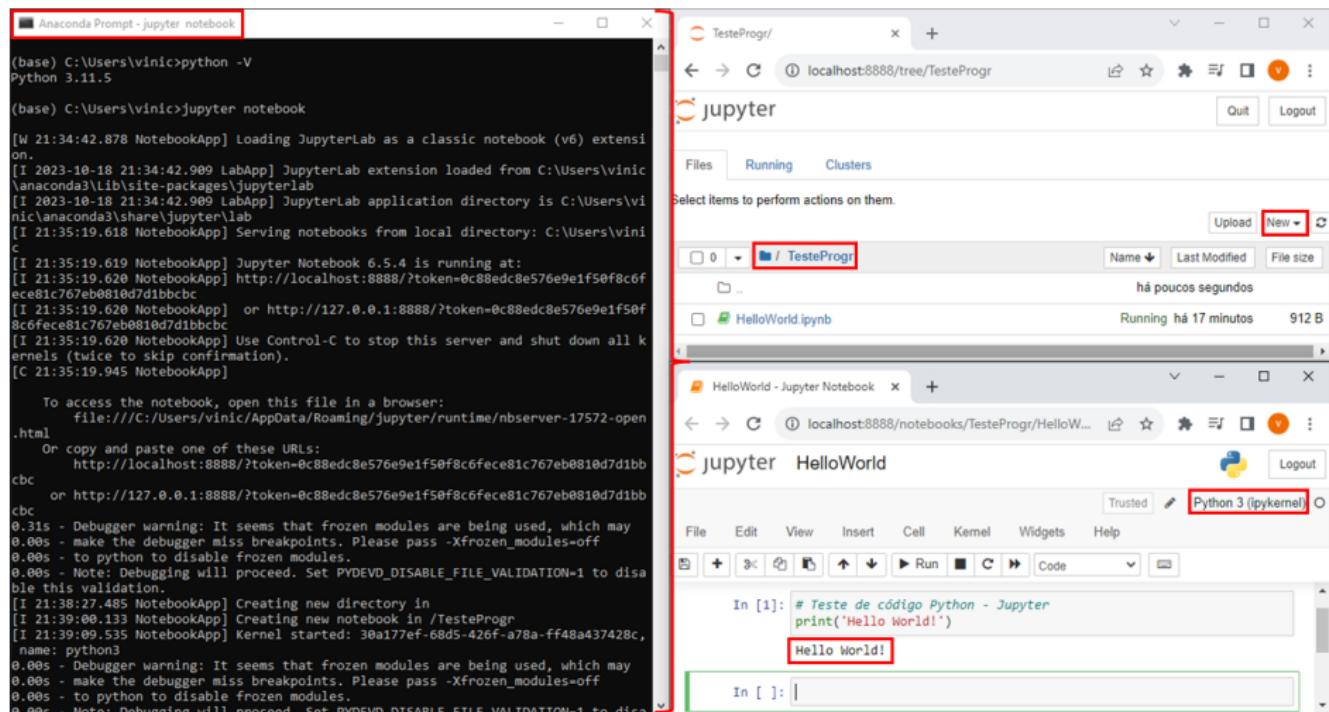
Arquivos Python extensão `.ipynb` → Jupyter notebook:



[...] → Exportar em HTML e gerar o .pdf utilizando CTRL + P

2.2.5.2. Testando o VSCode + Anaconda

Arquivos Python extensão *.ipynb* → Jupyter notebook → *prompt/navegador*:



2.2.5.3. Testando o VSCode + Anaconda

Crie um *script* (*plot.py*) contendo o código apresentado a seguir.

- O algoritmo fará uso de *duas bibliotecas*: *matplotlib.pyplot* e *numpy*;
- Serão criados *dois vetores temporais*: *t* e *s*;
- O algoritmo irá plotar o *gráfico bidimensional* $t \times s$ - avalie os **dados**.

Crie um segundo *script* (*plot3D.py*) contendo o código do próximo slide.

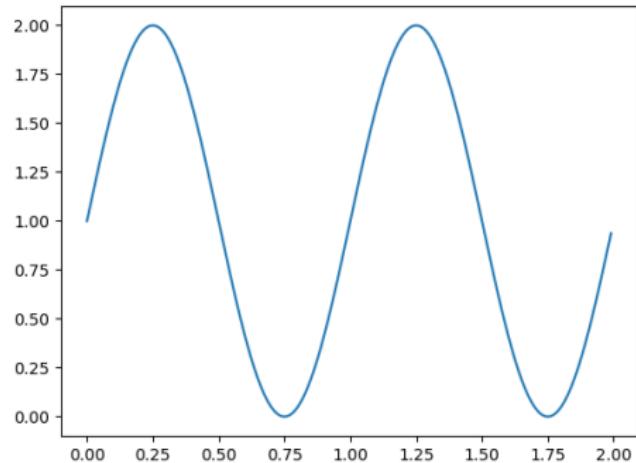
- O algoritmo fará uso das *mesmas bibliotecas*: *matplotlib.pyplot* e *numpy*;
- Serão criados *três matrizes de dados*: *X*, *Y* e *Z*;
- O algoritmo irá plotar o *gráfico tridimensional* $X \times Y \times Z$ - avalie os **dados**.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 t = np.arange(0.0, 2.0, 0.01)
5 s = 1+np.sin(2 *np.pi *t)
6
7 fig, ax =plt.subplots()
8 ax.plot(t,s)
```

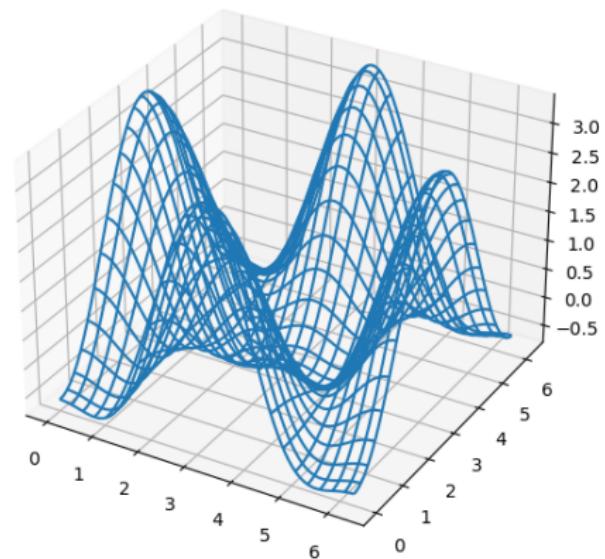
2.2.5.3.1 Testando o VSCode + Anaconda

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 alpha = 0.7
5 phi_ext = 2*np.pi *0.5
6
7 def ColorMap(phi_m,phi_p):
8     return (+alpha -2 * np.cos(phi_p) * np.cos(phi_m)
9             - alpha * np.cos(phi_ext -2*phi_p) )
10
11 phi_m =np.linspace(0, 2*np.pi, 100)
12 phi_p =np.linspace(0, 2*np.pi, 100)
13 X,Y =np.meshgrid(phi_p, phi_m)
14 Z =ColorMap(X, Y).T
15
16 fig = plt.figure(figsize=(8, 6))
17
18 ax =fig.add_subplot(1, 1, 1, projection = '3d')
19
20 p =ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)
```

2.2.5.3.2. Testando o VSCode + Anaconda



(a) Gráfico 2D - *plot.py*



(b) Gráfico 3D - *plot3D.py*

2.3.1. Contextualização: Linguagem Python

- Apesar do **Python** ter sido criado ao final da década de 80, como um sucessor da linguagem de programação ABC, foi por volta de 2019/20 que ela se “tornou tendência” mantendo-se em alta até os dias de hoje!
- Python** consiste em uma linguagem de **alto nível** (próximo à linguagem humana), quando comparada com **Assembly - baixo nível** → “linguagem de máquina” → zeros e uns;
- Para que um algoritmo seja executado se faz necessário um compilador/interpretador que recebe o código legível pelo usuário e convertem-no para o código “legível” pela máquina;
 - Na linguagem **compilada** (C, C++, Haskell, etc) a máquina traduz o programa diretamente, enquanto a **interpretada** (PHP, Python, JavaScript, etc) se faz necessário um programa que irá ler e executar o código em questão.

```

1  NULL EQU 0
2  STD_OUT_HANDLE EQU -11
3
4  extern ExitProcess
5  extern GetStdHandle
6  extern WriteFile
7
8  global inicio
9
10 section .data
11 mensagem db "HELLO WORLD", 0xA,0xD,0
12 tamanho_mensagem EQU $-mensagem
13
14 section .text
15
16 inicio:
17
18     mov rcx, STD_OUT_HANDLE
19     call GetStdHandle
20
21     mov rcx,rax
22     mov rdx,mensagem
23     mov r8,tamanho_mensagem
24     mov r9,0
25     call WriteFile
26
27     mov rax,NULL
28     call ExitProcess

```

A screenshot of a Jupyter Notebook cell. The code is:

```
print("Hello World")
```

The output shows:

```
[1] ✓ 0.0s
```

At the bottom, it says:

```
... Hello World
```

Python

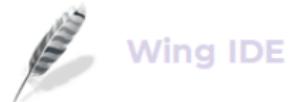
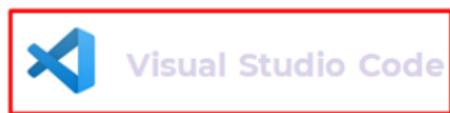
2.3.2. Compiladores - IDE's

Os compiladores/interpretadores são programas que realizam a “tradução” de um algoritmo, que geralmente é criado em alto nível, para um programa equivalente em linguagem de máquina permitindo que um processador possa executá-lo.

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```



```
01011011010101010
1010101010101010
1010101010101010
1010111111110000
10101011111000101
```



2.3.3. Gerenciadores de pacotes e repositórios → ❤

- Os **Gerenciadores de pacotes**, são ferramentas que auxiliam um desenvolvedor a fazer o download de **pacotes/libs/frameworks** permitindo assim com que eles possam ser inseridos e utilizados nos projetos.
- **Repositórios**, segundo o dicionário, consiste em um *“lugar onde se guarda, arquiva e/ou coleciona alguma coisa.”* Nos repositórios é possível armazenar versões do projeto e partilhar conteúdo/ideias/algoritmos❤.

PIP

CONDA®

 GitHub

2.3.4. Vantagens da linguagem Python

Vantagens de se escolher Python:

- Aprendizado relativamente fácil:
 - Linguagem gratuita com código aberto, fóruns com suporte aos membros da comunidade e grande quantidade de material didático.
- Sintaxe relativamente simples:
 - Linguagem simples, fácil e intuitiva, quantidade reduzida de códigos na elaboração projetos (\uparrow produtividade \leftrightarrow \downarrow erros).
- Versatilidade:
 - Compatível com diversas plataformas, sistemas operacionais e processadores. Pode ser utilizado em diferentes tipos de ambientes/interfaces, cumpre bem com o propósito.
- Ciência de dados:
 - Aplicação nas mais diversas áreas com análise de dados para definição de novos objetivos: maximização de lucro, minimização de custos.

Funções

- Desenvolver, sistemas, aplicações e soluções para web
- Desenvolver protótipos e coordenar testes
- Manejo de bibliotecas e frameworks

Skills

- Manipulação de sistemas embarcados com Python
- Programação orientada a objetos
- Frameworks e bibliotecas Python

Formação

- Computação, engenharia de sistemas, matemática, etc.
- Certificações e pós-graduação em programação, desenvolvimento para web, entre outros

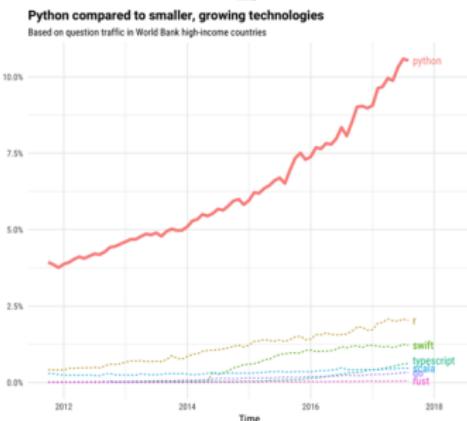
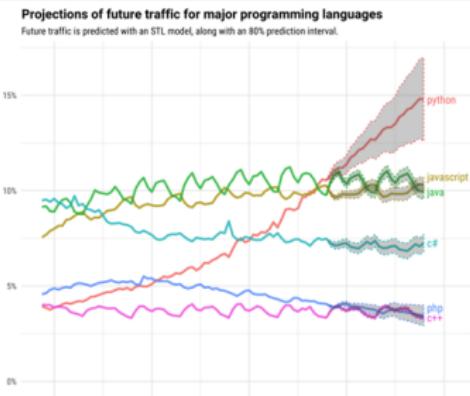
Salário

Estados Unidos: \$ 77.000 / ano
Portugal: € 70.000 / ano
Brasil: R\$ 41.000 / ano

2.3.5. Aplicações da linguagem Python

Aplicações do Python:

- Desenvolvimento Web:
 - Desenvolvimento que inclui funções complexas de *backend* - acesso a banco de dados, outros sites, envio seguro de informações, etc.
- Automação com *scripts* Python:
 - Tarefas cotidianas e exaustivas: renomear/-converter/avaliar arquivos, duplicidade de palavras, enviar e-mails, etc.
- Ciência de dados e *Machine Learning*:
 - Obtenção de informações relevantes em dados analisados por meio da inteligência artificial/computacional: separação de classes, previsão de séries temporais, estatística, etc.
- Desenvolvimento/testes de softwares:
 - Construção, gerenciamento, manutenção, interface gráfica do usuário (GUI), jogos, garantia do funcionamento desejado.



2.4.1. Mas antes, vamos relembrar... O que é lógica?

- O uso corriqueiro da palavra lógica associa-se à coerência e a racionalidade. Frequentemente associada à matemática, porém pode ser aplicada as demais ciências. Lógica consiste na “Ciência das formas de pensamento”;

Exemplo:

- a-) Todo mamífero é um animal.
Todo cavalo é um mamífero.
Portanto, todo cavalo é um animal.
- a-) Kaiton é país do planeta Stix.
Todos os Xinpins são de Kaiton.
Logo, todos os Xinpins são Stixianos.

Exemplos no dia-a-dia:

- a-) A gaveta está fechada.
A caneta está dentro da gaveta.
Precisamos primeiro abrir a gaveta para depois pegar a caneta.
- a-) Anacleto é mais velho que Felisberto.
Felisberto é mais velho que Marivalda.
Portanto, Anacleto é mais velho que Marivalda.

2.4.2. Lógica de programação e raciocínio?

Mas e a lógica de programação?

- O raciocínio é algo abstrato e intangível que todos possuem. Para que possamos utilizá-lo com facilidade, basta treinarmos e ele poderá ser expresso independente do idioma a ser utilizado;
- De forma similar acontece com a **Lógica de Programação**, que pode ser concebida pela mente treinada e pode ser representada em qualquer das inúmeras linguagens de programação existentes.

O que é um algoritmo?

- O objetivo principal do estudo da lógica de programação é a construção de algoritmos;
- Um **algoritmo** pode ser definido como uma sequência de passos que visam atingir um objetivo bem definido;
- Para especificarmos uma sequência de passos, faz-se necessário utilizar ordem, ou seja, ‘pensar com ordem’, portanto precisamos utilizar lógica;
- Apesar do nome não ser tão usual, o algoritmo está mais presente em nossas vidas do que pensamos, como uma **receita de bolo**.

2.4.3. Exemplo de um algoritmo

Algoritmo 1.1: Troca de lâmpada:

- ① Pegar uma escada;
- ② Posicionar a escada embaixo da lâmpada;
- ③ Buscar uma lâmpada nova;
- ④ Subir na escada;
- ⑤ Retirar a lâmpada velha;
- ⑥ Colocar a lâmpada nova.

Este algoritmo é bem objetivo! - *troca de lâmpada*. Mas e se esta lâmpada não estivesse queimada por exemplo? O algoritmo não previu esta possibilidade.

Algoritmo 1.2: Troca de lâmpada com teste:

- ① Pegar uma escada;
- ② Posicionar a escada embaixo da lâmpada;
- ③ Buscar uma lâmpada nova;
- ④ Acionar o interruptor;
- ⑤ Se a lâmpada não acender, então:
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar a lâmpada nova.

2.4.3.1. Exemplo de um algoritmo

Ambos os algoritmos anteriores (1.1 e 1.2) são eficazes, pois realizam o **mesmo objetivo**. A grande diferença entre eles é que no (1.1) todas as ações **SÃO executadas** e no (1.2) há uma **condicional** na 5^a ação que, apenas caso seja verdadeira, realizará a substituição da lâmpada - reduzindo de 6 ações para 5.

Mesmo o algoritmo (1.2) sendo mais eficiente ele ainda pode ser melhorado, pois não sabemos se as ações buscar uma escada e uma lâmpada são necessárias.

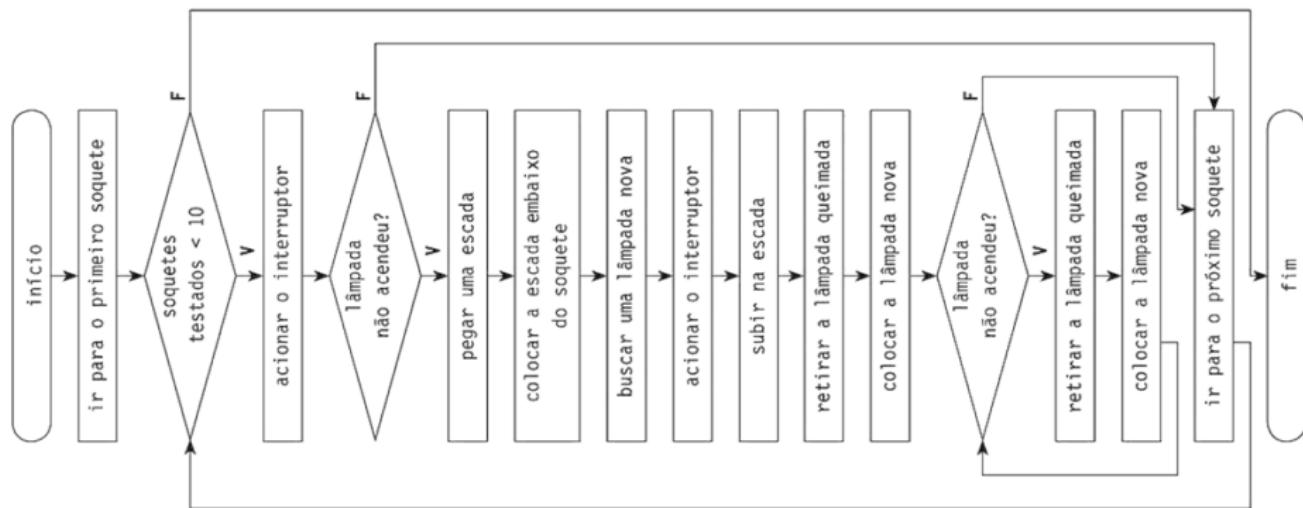
Algoritmo 1.3: Troca de lâmpada com teste no início:

- ① Acionar o interruptor;
- ② Se a lâmpada não acender, então:
 - Pegar uma escada;
 - Posicionar a escada embaixo da lâmpada;
 - Buscar uma lâmpada nova;
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar a lâmpada nova.

Neste caso as primeiras ações foram reduzidas de 5 para 2. Mas e se mesmo assim, a lâmpada não acender? Podemos representar com uma rotina de repetição! Podemos também otimizar nosso algoritmo buscando a lâmpada e a escada ao mesmo tempo - reduzindo uma ação. E assim construímos nosso algoritmo.

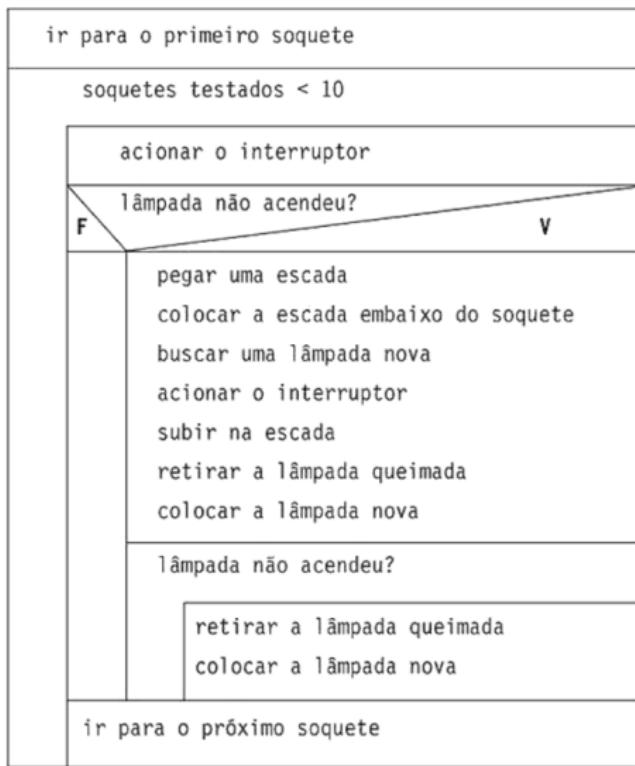
2.4.4. Representação de algoritmo por fluxograma

Um algoritmo pode ser representado por diversas formas, uma delas é bastante usual é o fluxograma. Neste apresenta um algoritmo para realizar a troca de lâmpada com testes para 10 soquetes com repetição:



Mudar a orientação da pagina para vertical no Adobe.

2.4.5. Representação de algoritmo por diagrama de Chapin



2.4.6. Exercícios de fixação: *Google Classroom*

Ex-1) Três senhoras - dona BRANCA, dona ROSA e dona VIOLETA - passeavam pelo parque quando dona ROSA disse:

-Não é curioso que estejamos usando vestidos de cores BRANCA, ROSA e VIOLETA, embora nenhuma de nós esteja usando um vestido de cor igual ao seu próprio nome?

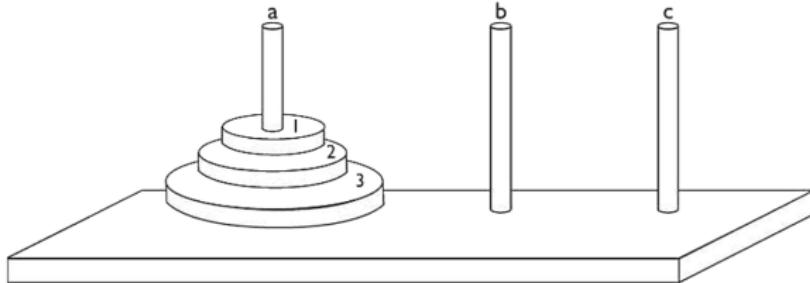
- Uma simples coincidência! - respondeu a senhora com o vestido VIOLETA. Qual a cor do vestido de cada senhora?

Ex-2) Um homem precisa atravessar um rio com um barco que possui capacidade apenas para carregar ele mesmo e mais uma de suas três cargas, que são: um lobo, um bode e um maço de alfafa. O que o homem deve fazer para conseguir atravessar o rio sem perder suas cargas? Escreva um algoritmo mostrando a resposta, ou seja, indicando todas as ações necessárias para efetuar a travessia segura.

2.4.6.1. Exercícios de fixação: *Google Classroom*

Ex-3) Três jesuítas e três canibais precisam atravessar um rio; para tal, dispõe de um barco com capacidade para duas pessoas. Por medida de segurança, não se deve permitir que em alguma margem a quantidade de jesuítas seja inferior à de canibais. Qual solução para efetuar a travessia com segurança? Elabore um algoritmo mostrando a resposta, indicando as ações que concretizam a solução deste problema.

Ex-4) Elabore um algoritmo que move três discos de uma Torre de Hanói, que consiste em três hastes ($a - b - c$), uma das quais serve de suporte para três discos de tamanhos diferentes (1 – 2 – 3), os menores sobre os maiores. Pode-se mover um disco de cada vez para qualquer haste, contanto que nunca seja colocado um disco maior sobre um menor. O objetivo é transferir os três discos para outra haste.



2.5.1. Tipos primitivos

- O computador manipula informações utilizando quatro tipos primitivos, que são utilizados para construção de algoritmos. Tais tipos são:
 - Inteiro: informação numérica que pertença ao conjunto dos números inteiros (negativo, nulo ou positivo). Ex:
 - Gilmar tem 15 irmãos;
 - Aquela escada possui 8 degraus;
 - Meu vizinho comprou 2 carros.
 - Real: informação numérica que pertença ao conjunto dos números reais (negativo, nulo ou positivo). Ex:
 - Ela tem 1,73 metros de altura;
 - Meu saldo bancário é de R\$215,10;
 - No momento estou pesando 82,5Kg
 - Caracter: informação composta de um conjunto de caracteres alfanuméricos: (0...9), alfabeticos (A...Z, a...z) e especiais (#, ?, !, @). Ex:
 - Constava na prova: “Use somente caneta!”;
 - O parque municipal estava repleto de placas: “Não pise na grama”;
 - O nome do vencedor é Felisberto Laranjeira.
 - Lógico: informação composta que pose assumir apenas duas situações (bistáveis). Ex:
 - A porta pode estar aberta ou fechada;
 - A lâmpada pode estar acesa ou apagada.

2.5.2. Informações técnicas

Constantes: dado que não sofre nenhuma variação no decorrer do tempo, ou seja, seu valor é constante desde o início até o fim da execução do algoritmo;

Variável: pode ser alterado em algum instante no decorrer do tempo, ou seja, durante a execução do algoritmo o valor do dado pode sofrer alterações;

Formação de identificadores: os “nomes” das informações são os identificadores e devem seguir as seguintes regras:

- Começar com um caracter alfanumérico;
- Podem ser seguidos por mais caracteres alfanumérico ou numéricos;
- Não devem ser usados caracteres especiais;
- Ex: Alpha, X, BJ156, Notas, Media, 5X, E(13), Nota/2, AWQ*, P&AA.

Declaração de variáveis: as informações são guardadas em dispositivos eletrônicos, **memória**. Tais memórias possuem ‘Subespaços’ responsáveis por armazenar as variáveis de tipos iguais juntas. Porém, cada uma no seu lugar - guardadas individualmente. Ex:

- **Inteiro**: X;
- **Caracter**: Nome, Endereco, Data;
- **Real**: ABC, XPTO, Peso, Dolar;
- **Logico**: resposta, H186;

2.5.3. Operadores matemáticos

Tabela 2.1 Operadores aritméticos

Operador	Função	Exemplos
+	Adição	2 + 3, X + Y
-	Subtração	4 - 2, N - M
*	Multiplicação	3 * 4, A * B
/	Divisão	10/2, X1/X2

Tabela 2.2 Potenciação e radiciação

Operador	Função	Significado	Exemplos
pow(x,y)	Potenciação	x elevado a y	pot(2,3)
sqrt(x)	Radiciação	Raiz quadrada de x	rad(9)

Tabela 2.3 Operador de resto e quociente de divisão inteira

Operador	Função	Exemplos
A % B	Resto da divisão	9 mod 4 resulta em 1 27 mod 5 resulta em 2
A / B	Quociente da divisão	9 div 4 resulta em 2 27 div 5 resulta em 5

Tabela 2.4 Precedência entre os operadores aritméticos

Prioridade	Operadores
1 ^a	parênteses mais internos
2 ^a	pow sqrt
3 ^a	* / % Alguns operadores mudam de linguagem para linguagem
4 ^a	+ -

2.5.4. Operadores lógicos

Tabela 2.5 Operadores relacionais

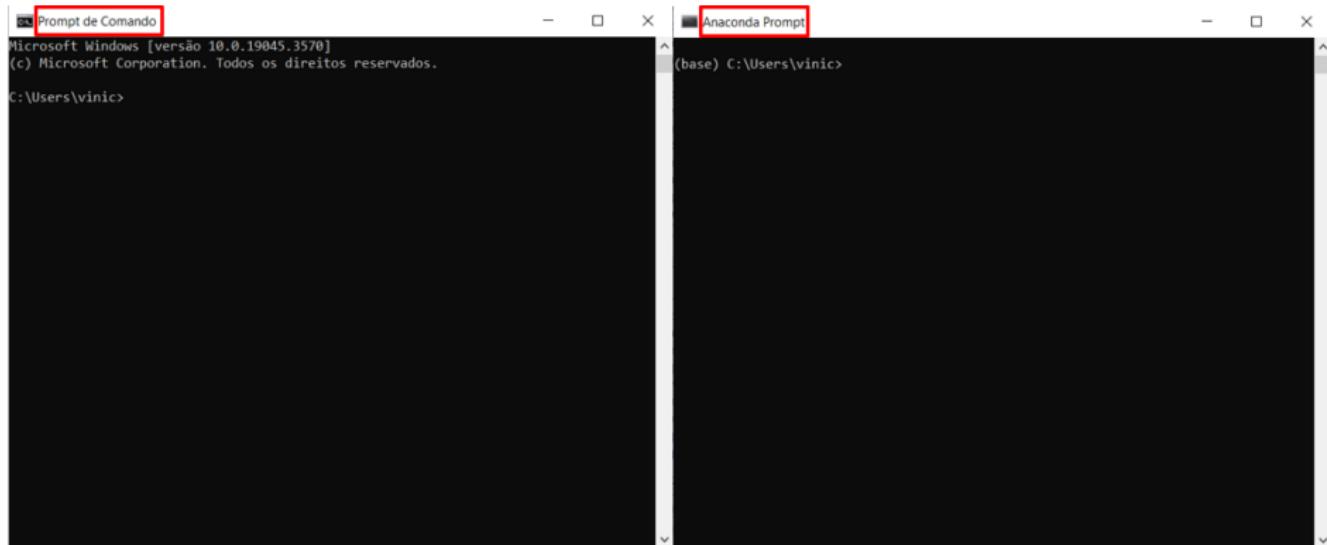
Operador	Função	Exemplos
<code>==</code>	Igual a	<code>3 == 3, X == Y</code>
<code>></code>	Maior que	<code>5 > 4, X > Y</code>
<code><</code>	Menor que	<code>3 < 6, X < Y</code>
<code>>=</code>	Maior ou igual a	<code>5 >= 3, X >= Y</code>
<code><=</code>	Menor ou igual a	<code>3 <= 5, X <= Y</code>
<code>!=</code>	Diferente de	<code>8 != 9, X != Y</code>

Tabela 2.6 Operadores lógicos

Operador	Função
<code>!</code>	Negação
<code>&&</code>	Conjunção (e)
<code> </code>	Disjunção (ou)

3.2.1. O que é e o que faz um *prompt* de comandos ?

- O *prompt* de comandos consiste em um programa que emula o campo de entrada em uma tela de interface do usuário **baseada em texto**. Tal ferramenta possui aparência da interface gráfica do usuário (GUI) do *Windows*.



3.2.1.1. O que é e o que faz um *prompt* de comandos ?

Anaconda Prompt

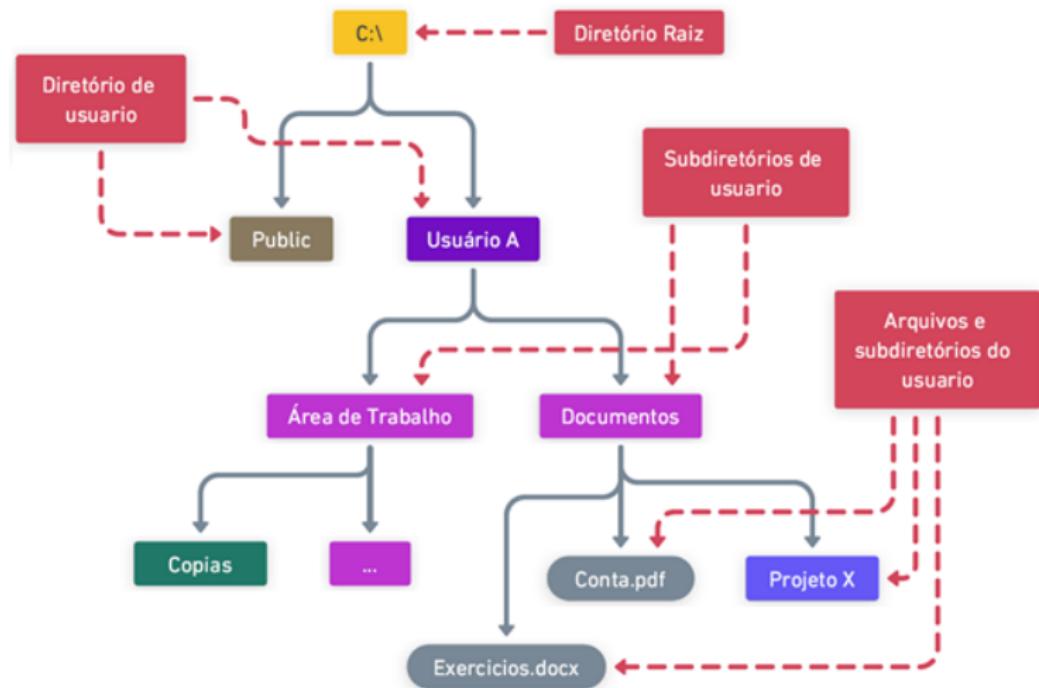
```
(base) C:\Users\vinic>python
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 10
>>> print("Hello World!")
Hello World!
>>> a
10
>>> import numpy as np
>>> t = np.arange(0, 10, 1)
>>> t
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> print(a[1,2])
6
>>> a[1][2]=100
>>> a
array([[ 1,   2,   3],
       [ 4,   5, 100],
       [ 7,   8,   9]])
>>> t[2]=200
>>> t
array([ 0,   1, 200,   3,   4,   5,   6,   7,   8,   9])
>>> exit()

(base) C:\Users\vinic>
```

Comandos básicos: DIR, COPY, MOVE, MKDIR, CD, RD, CLS, etc... → [HELP](#)

3.2.2. Estrutura de diretórios

Exemplo de hierarquia de diretórios do *Windows*



3.2.3. Dados simples



Txt - Arquivo de texto “cru” sem formatação (.txt)

A screenshot of a Windows Notepad window titled "dados - Bloco de Notas". The window contains the following text:

```
Arquivo Editar Formatar Exibir Ajuda
Aluno
aluno@email.com
senha: 123456
```

The window has standard Windows-style controls (minimize, maximize, close) at the top right. At the bottom, there are status bars showing "Ln 3, Col 14", "100%", "Windows (CRLF)", and "UTF-8". A vertical scroll bar is visible on the right side of the window.

3.2.3.1. Dados simples



Csv - Comma Separated Values - valores separados por vírgula: arquivo de texto estruturado, onde seus campos são separados por vírgulas (.csv)



```
1 Nome;Idade;Genero;Email;  
2 Joao;15;M;joaopedefeijao@hotmail.com;  
3 Maria;16;F;maryacomym@gmail.com;  
4 Cleberson;22;M;cleocleo@uol.com;  
5
```

3.2.3.2. Dados simples



Excel - Pasta de trabalho com várias planilhas (.xls .xlsx .xlsm etc...)

The screenshot shows a Microsoft Excel spreadsheet titled "Planilha1". The data is organized in a table with columns labeled "Nome", "Idade", "Genero", and "Email". The rows contain the following data:

	Nome	Idade	Genero	Email
1	Joao	15	M	joaopedefeijsao@hotmail.com
2	Maria	16	F	maryacomym@gmail.com
3	Cleberson	22	M	cleocleo@uol.com
4				
5				
6				
7				
8				
9				
10				
11				

The Excel ribbon is visible at the top, showing tabs like Arquivo, Página Inicial, Inserir, Layout da Página, Fórmulas, Dados, Revisão, Exibir, Desenvolvedor, and Ajuda. The formula bar shows "D16". The status bar at the bottom indicates "Planilha1" and "Acessibilidade: não disponível".

3.2.3.3. Dados simples



Json - Java Script Object Notation: é um arquivo leve de troca de informações/dados entre sistemas → possui estrutura definida (.json)

```
1  {
2      "produtos": [
3          {
4              "id": 1,
5              "name": "caneca",
6              "price": 5.45,
7              "color": "white"
8          },
9          {
10             "id": 2,
11             "name": "garrafa",
12             "price": 75,
13             "color": "white"
14         }
15     ]
16 }
```

3.2.4. Python - Keywords

Python

>>>

```
>>> help("keywords")
```

Here is a list of the Python keywords. Enter any keyword to get more help.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

3.3.1. Qual a definição de um banco de dados?



Banco de dados (BD) consiste em uma **coleção organizada de informações/dados estruturados**.

Normalmente **armazenados eletronicamente em um sistema de computador e controlado por um sistema de gerenciamento de banco de dados**.

As características de um BD são:

- Tem um propósito específico;
- O conjunto deve ser modelável;
- Representa um aspecto do mundo real;
- Pode variar em tamanho e complexidade;
- Deve ter controle de redundância;
- Deve permitir o uso de dados simultâneos;
- Deve ser padronizado.

3.4.1. Primeiro contato com banco de dados em Python

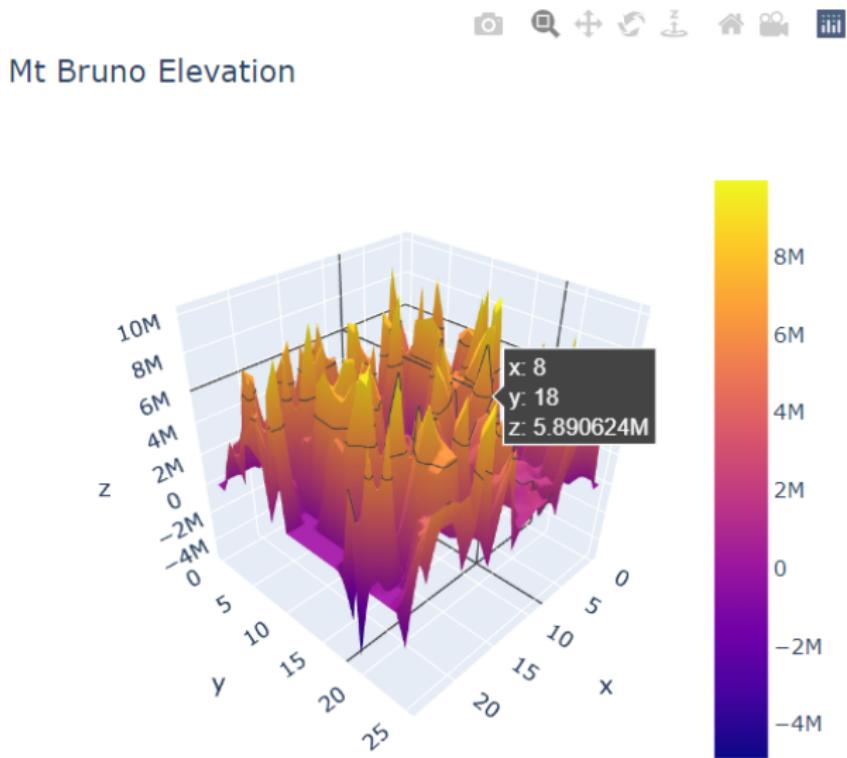
O banco de dados contido no link a seguir consiste em dados de latitude, longitude e elevações - *Mt Bruno Elevation*. Realize seu download e importe os dados e avaliando a plotagem do gráfico obtido a partir dos mesmos.

```
1 import plotly.graph_objects as go
2 import pandas as pd
3
4 # Leitura de dados .csv
5 z_dataCSV =pd.read_csv('DataCSV.csv')
6
7 # Apresentação dados em CSV
8 fig =go.Figure(data=[go.Surface(z=z_dataCSV.values)])
9
10 fig.update_layout(title='Mt Bruno Elevation – .csv', autosize=False,
11                   width=500, height=500,
12                   margin=dict(l=90, r=50, b=65, t=90))
13
14 fig.show()
```

Link para o banco de dados - *Mt Bruno Elevation*:

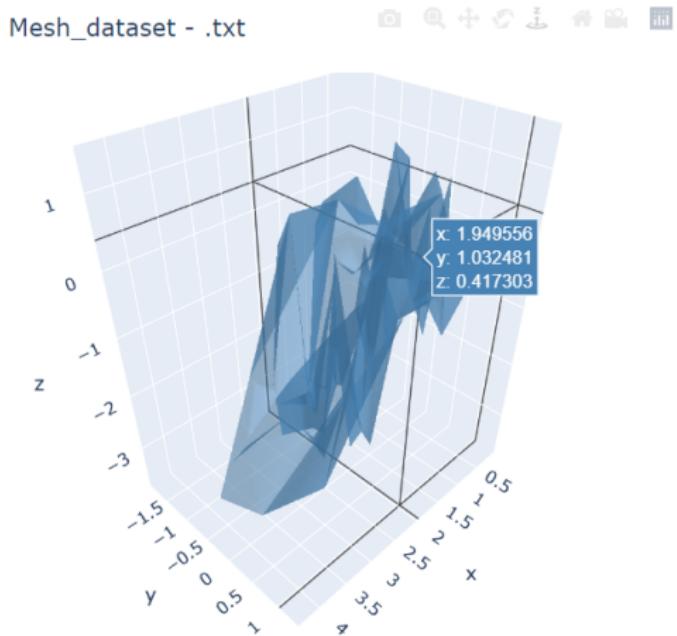
https://raw.githubusercontent.com/plotly/datasets/master/api_docs/mt_bruno_elevation.csv

3.4.1.1. Primeiro contato com banco de dados em Python



3.4.1.2. Primeiro contato com banco de dados em Python

Faça as pesquisas necessárias na internet para realizar a importação de dados na extensão .txt em Python e fazendo uso da mesma biblioteca: *plotly.graph_objects*, plote o gráfico apresentado a seguir.



https://raw.githubusercontent.com/plotly/datasets/master/mesh_dataset.txt

3.5.1. Definições banco de dados SQL e NoSQL



Structured Query Language:

“Linguagem de consulta estruturada”

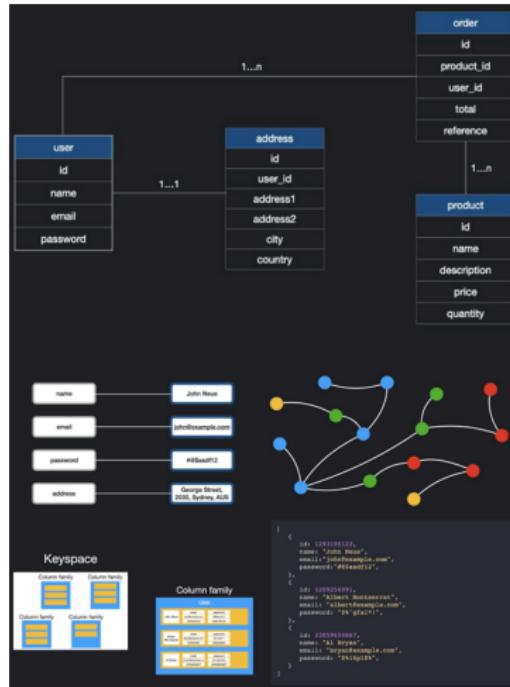
SQL consiste em uma linguagem de programação usada por grande parte dos bancos de dados **relacionais** para consultar, manipular, definir e fornecer controle de acesso. Ou seja, gerenciar dados relacionais em um sistema de BD ou para processamento de fluxo de dados.

No Only Structured Query Language:

“Não somente Linguagem de consulta estruturada”

NoSQL é uma linguagem de programação usada para BDs **não relacionais**. Ele pode conter SQL, porém há outras formas de manipular os dados. Fornece um mecanismo para armazenamento e recuperação de dados modelados de formas diferentes das relações tabulares (BD relacionais).

3.5.1.1. Definições banco de dados SQL e NoSQL



Afinal, o que é um banco relacional?

É um formato de banco rigidamente estruturado, baseado em tabelas. Os campos tem relacionamento entre si.

E o que é um banco não relacional?

É qualquer banco de dados que não segue o modelo relacional fornecido pelos sistemas tradicionais de gerenciamento de banco de dados relacionais (SGBDR). Apresentam os seguintes tipos:

- *Key-value stores;*
- *Graph stores;*
- *Column stores;*
- *Document stores.*

3.5.2. Alguns comandos muito utilizados em SQL

Alguns comandos em SQL:

- **CREATE**: cria novas tabelas em um banco de dados;
- **ALTER**: alterar uma tabela já criada;
- **INSERT**: adiciona registros a uma tabela;
- **UPDATE**: atualiza os registros já inseridos;
- **DELETE**: exclui registros de uma tabela;
- **SELECT**: busca registros na tabela;
- **GRANT**: permite acesso a objetos do banco de dados;
- **REVOKE**: remove o acesso a objetos do banco de dados;
- **DENY**: bloqueia o acesso para objetos e usuários;específicos
- **DROP**: exclui uma tabela do banco de dados.

3.5.3. Exemplos de comandos em SQL

CREATE: criar novas tabelas em um BD.

```
CREATE TABLE usuarios (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome VARCHAR(50),
    email VARCHAR(100),
    senha VARCHAR(50));
```

ALTER: alterar uma tabela já criada em um BD.

```
ALTER TABLE usuarios ADD data_nascimento VARCHAR(10);
```

INSERT: adicionar registros a uma tabela de um BD.

```
INSERT INTO usuarios(nome, email, senha, data_nascimento)
VALUES ('Joao','joao_e_maria@gmail.com','Maria123','21/05/1989');
```

UPDATE: atualizar os registros de uma tabela já criada em um BD.

```
UPDATE usuarios SET senha="Ana456" WHERE nome = "Joao"
```

DELETE: excluir os registros de uma tabela já criada em um BD.

```
DELETE FROM usuarios WHERE email="joao_e_maria@gmail.com"
```

3.5.4. Aplicação comando *CREATE*

```
1 import sqlite3  
2  
3 # Inicializa a conexão com o banco de dados  
4 BD =sqlite3.connect("MyBD_py.db")  
5  
6 # Inicializa o cursor onde sera feita as Querys  
7 c=BD.cursor()  
8  
9 # Define a Query e executa  
10 c.execute("""CREATE TABLE IF NOT EXISTS usuarios (  
11     id INTEGER PRIMARY KEY AUTOINCREMENT,  
12     nome VARCHAR(50),  
13     email VARCHAR(100),  
14     senha VARCHAR(50)  
15 )""")  
16  
17 print ("Tabela Usuarios Criada com sucesso")  
18  
19 # Grava no banco  
20 BD.commit()  
21  
22 # Fechar a conexão com banco de dados  
23 BD.close()
```

3.5.5. Aplicação comando *ALTER*

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""ALTER TABLE usuarios ADD data_nascimento VARCHAR(10)""")
11
12 print ("Tabela Usuarios Alterada com sucesso")
13
14 # Grava no banco
15 BD.commit()
16
17 # Fechar a conexão com banco de dados
18 BD.close()
```

3.5.6. Aplicação comando *INSERT*: Versão 1

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento)
11 VALUES ('Joao','joao_e_maria@gmail.com','Maria123','21/05/1989')""")
12 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento)
13 VALUES ('Roberto','beto@gmail.com','Beto123','15/11/2000')""")
14 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento)
15 VALUES ('Marcia','Marinha@gmail.com','Marcinha123','07/01/1993')""")
16
17 # Mostra mensagem para o usuario
18 print("Usuarios Adicionados com sucesso V1")
19
20 # Grava no banco
21 BD.commit()
22
23 # Fechar a conexão com banco de dados
24 BD.close()
25
26 # Para visualizar os dados no BD (comentar da linha 9 à 26)
27 #c.execute("SELECT * FROM usuarios")
28 #print(c.fetchall())
```

3.5.6.1. Aplicação comando *INSERT*: Versão 2

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect ("MyBD.py.db", timeout=60)
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Cria a lista de usuarios
10 lista = [
11     ('Fernando', 'Nando89@gmail.com', 'Nando123', '21/05/1989'),
12     ('Gilmar', 'Gil@gmail.com', 'Gilmar123', '07/12/1999'),
13     ('Carlos', 'carlinhos1988@gmail.com', 'Carlinhos123', '18/06/1988')
14 ]
15
16 # Define a Query e executa
17 c.executemany ("""INSERT INTO usuarios(nome, email, senha, data_nascimento) VALUES (?,?,?,?,?)""", lista )
18
19 # Mostra mensagem para o usuario
20 print ("Usuarios Adicionados com sucesso V2")
21
22 # Grava no banco
23 BD.commit()
24
25 # Fechar a conexão com banco de dados
26 BD.close()
```

3.5.6.2. Aplicação comando *INSERT*: Versão 3

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect ("MyBD.py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Carrega as informações do usuario
10 p_nome =input ('Nome: ')
11 p_email = input ('Email: ')
12 p_senha = input ('Senha: ')
13 p_data_de_nascimento = input ('Data de Nascimento (dd/mm/yyyy): ')
14
15
16 # Define a Query e executa
17 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento) VALUES (?,?,?,?,?)""",
18     (p_nome, p_email, p_senha, p_data_de_nascimento))
19
20 # Mostra mensagem para o usuario
21 print ("Usuarios Adicionados com sucesso V3")
22
23 # Grava no banco
24 BD.commit()
25
26 # Fechar a conexão com banco de dados
27 BD.close()
```

3.5.7. Aplicação comando *UPDATE*: Versão 1

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect ("MyBD.py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Carrega as informações do usuario
10 id_usuario =1
11 novo_nome ="Luiza"
12 novo_email ="luiza@uol.com.br"
13 nova_senha ="souLinda123"
14 nova_data_de_nascimento ="02/02/2002"
15
16 # Define a Query e executa
17 c.execute("""UPDATE usuarios SET nome=? , email=? , senha=? , data_nascimento=? WHERE id=? """,
18     (novo_nome, novo_email, nova_senha, nova_data_de_nascimento, id_usuario))
19
20 # Grava no banco
21 BD.commit()
22
23 # Mostra mensagem para o usuario
24 print (f" Usuarios { id_usuario } modificado com sucesso V1")
25
26 # Fechar a conexão com banco de dados
27 BD.close()
```

3.5.7.1. Aplicação comando *UPDATE*: Versão 2

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD = sqlite3.connect("MyBD.py.db")
5
6 # Inicializa o cursor onde sera feita as Queries
7 c = BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""SELECT * FROM usuarios""")
11
12 # Mostra os dados da tabela para o usuario
13 for linha in c.fetchall():
14     print(linha)
15
16 # Carrega as informações do usuario
17 id_usuario = input('Id do usuario: ')
18 novo_nome = input('Nome: ')
19 novo_email = input('Email: ')
20 nova_senha = input('Senha: ')
21 nova_data_de_nascimento = input('Data de Nascimento (dd/mm/yyyy): ')
22
23 # Define a Query e executa
24 c.execute("""UPDATE usuarios SET nome=?, email=?, senha=?, data_nascimento=? WHERE id=?""",
25             (novo_nome, novo_email, nova_senha, nova_data_de_nascimento, id_usuario))
26
27 # Grava no banco
28 BD.commit()
29
30 # Mostra mensagem para o usuario
31 print(f"Usuarios {id_usuario} modificado com sucesso V2")
32
33 # Fechar a conexão com banco de dados
34 BD.close()
```

3.5.8. Aplicação comando *DELETE*

```
1 import sqlite3  
2  
3 # Inicializa a conexão com o banco de dados  
4 BD =sqlite3.connect("MyBD_py.db")  
5  
6 # Inicializa o cursor onde sera feita as Querys  
7 c =BD.cursor()  
8  
9 # Carrega as informações do usuario  
10 id_usuario = "5"  
11  
12 # Define a Query e executa  
13 c.execute("""DELETE FROM usuarios WHERE id= ? """, (id_usuario))  
14  
15 # Grava no banco  
16 BD.commit()  
17  
18 # Mostra mensagem para o usuario  
19 print(f'Usuario { id_usuario } deletado com sucesso')  
20  
21 # Fechar a conexão com banco de dados  
22 BD.close()
```

3.5.9.4. Exemplo de comandos em SQL

SELECT: busca registros de uma tabela já criada em um BD.

`SELECT * FROM usuarios`

`SELECT * FROM usuarios WHERE nome = "Ana"`

`SELECT nome FROM usuarios WHERE senha = "Joao123"`

GRANT: permite acesso a objetos de uma tabela já criada em um BD.

`GRANT SELECT, INSERT UPDATE ON usuarios TO usuario_db_01`

REVOKE: remove o acesso a objetos de uma tabela já criada em um BD.

`REVOKE SELECT ON usuarios FROM usuario_db_01`

DENY: bloqueia o acesso a objetos de uma tabela já criada em um BD.

`DENY SELECT ON usuarios TO usuario_db_01`

DROP: exclui uma tabela de um BD ou o próprio BD.

`DROP TABLE usuarios → c.execute(" " " DROP TABLE usuarios" " ")`

`DROP DATABASE db`

3.5.10.6.1. Aplicação comando *SELECT*

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""SELECT * FROM usuarios""")
11
12 # Mostra mensagem para o usuario
13 for linha in c.fetchall():
14     print(linha)
15
16 # Fechar a conexão com banco de dados
17 BD.close()
```

3.5.11. Exercício de fixação

Desenvolva um banco de dados (*BD.ipynb*) no qual deverá ser criado utilizando os comandos aprendidos anteriormente. O conteúdo do banco de dados será escolhido por você. Por exemplo:

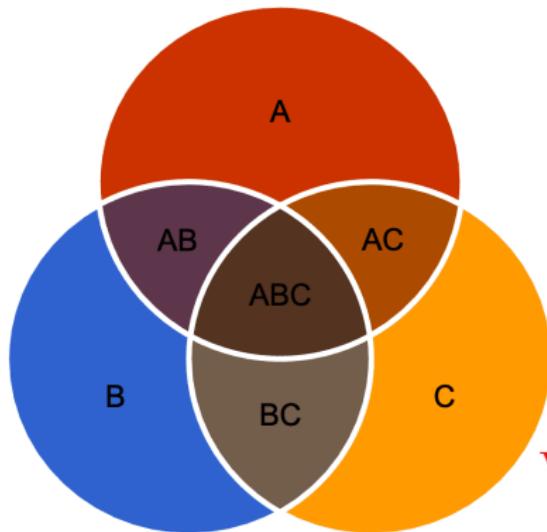
- Estoque de uma loja;
- Controle de clientes;
- Dados obtidos a partir de uma linha de produção;
- Dados médicos de pacientes;
- Fabricação de veículos, etc.

Elabore um *Markdown* descrevendo as etapas do código, e por fim, elabore um arquivo em .pdf.

4.2.1. Variáveis globais × Variáveis locais

Variáveis globais:

- Consiste em uma variável **acessível em todos** os escopos de um algoritmo;
- Tal variável é **definida fora** do corpo de uma função específica;
- Geralmente considerada “inadequada” devido estar **sujeita a alterações** em qualquer local e parte de um *script*;
- Muito utilizadas em **programação concorrente** para que as informações possam permeiar em diferentes seções do código que não possuem relação entre si → *threads* e sinais.



Variáveis locais:

- Consiste em uma variável **acessível apenas** dentro da função na qual foi declarada;
- Só pode ser **alterada dentro do referido escopo** em que foi definida.

4.2.1.1. Variáveis globais × Variáveis locais

```
1 nome = "Jose" # José como variável global
2
3 def EscreveGlob():
4     print(f"{nome} esta em aula!")
5
6 def EscreveInfoGlob():
7     print(f"{nome} esta estudando")
8
9 EscreveGlob()
10 EscreveInfoGlob()
11 nome = "Fernanda" # Mudando uma variável global
12
13 def EscreveLoc():
14     nome = "Maria" # Maria como variável local
15     print(f"{nome} esta assistindo aula no SENAI!")
16
17 def EscreveInfoLoc():
18     nome = "Maria" # Maria como variável local
19     print(f"{nome} esta na aula de Data Analytics com Python!")
20
21 EscreveLoc()
22 EscreveInfoLoc()
23
24 print(nome + ' É uma variável global e Maria é local!')
```

4.2.2. Varargs - argumentos variáveis e comando *return*

Varargs:

```
def calcular_imposto(valor, perc_ir):
    ir = valor * perc_ir
    iss = valor * 0.05
    csll = valor * 0.0375
    pis = valor * 0.03
    return ir + iss + csll + pis

print(calcular_imposto(1000, perc_ir=0.275))

def calcular_imposto(valor, *args):
    total_imposto = 0
    for item in args:
        total_imposto += valor * item
    return total_imposto

print(calcular_imposto(1000, 0.275, 0.05, 0.0375, 0.03))
```

392.5

- Quando em um método **não se sabe ao certo quantos parâmetros serão passados**: vetor, lista, objetos, etc, pode-se definir um determinado parâmetro do tipo *varargs*;
- Pode ser considerado um “facilitador” devido a sua versatilidade em um *script*;
- Ao invés de criar um *array* ou *lista* e colocar os valores dentro dele é possível chamá-lo diretamente passando ‘n’ valores e os **parâmetros são automaticamente adicionados** em um *array* do mesmo tipo do *varargs*.

Return:

- É um comando que retorna algum valor que deseja ser obtido após a execução de uma **função**, por exemplo. Uma função que calcula Pitágoras deve retornar a hipotenusa.

4.2.2.1. Varargs - argumentos variáveis e comando *return*

```
1 def soma(*args):
2     resultado = 0
3
4     for x in args:
5         resultado+= x
6
7     print(resultado)
8
9 soma(1,2,3,4,5,6)
10 soma(1,3)
11
12 import numpy as np
13 cont = 0
14 def Matrix3_3(cont, *args):
15     Mat = np.zeros([int(len(args)/3),int(len(args)/3)])
16     for x in range(int(len(args)/3)):
17         for y in range(int(len(args)/3)):
18             Mat[x][y] = args[cont]
19             cont += 1
20     print(Mat)
21
22 Matrix3_3(cont, 22,32,123,553,22,12,56,23,98)
```

4.2.2.1. Varargs - argumentos variáveis e comando *return*

```
1 import math
2
3 def MenorNumero(*args):
4
5     oMenor = math.inf
6     for x in args:
7         if (oMenor >= x):
8             oMenor = x
9
10    return (oMenor)
11
12 print('O menor número é: ' + str(MenorNumero(14,5,99,4,9,63)))
13
14 def MaiorNumero(*args):
15
16     oMaior = -math.inf
17     for x in args:
18         if (oMaior <= x):
19             oMaior = x
20
21    return (oMaior)
22
23 print('O maior número é: ' + str(MaiorNumero(14,5,99,4,9,63)))
```

4.2.3. Init e Self no Python

```
# Vamos criar uma classe carro
class carro():
    # método init ou construtor
    def __init__(self, modelo, cor):
        self.modelo = modelo
        self.cor = cor

    def mostrar(self):
        print("O Modelo é", self.modelo )
        print("A Cor é", self.cor )

# cada objeto possui diferentes self com atributos
audi = carro("audi a4", "azul")
ferrari = carro("ferrari 488", "verde")
audi.mostrar()
ferrari.mostrar()

O Modelo é audi a4
A Cor é azul
O Modelo é ferrari 488
A Cor é verde

[10] carro.mostrar(audi)

O Modelo é audi a4
A Cor é azul

[11] carro.mostrar(ferrari)

O Modelo é ferrari 488
A Cor é verde
```

__init__ :

- Também chamada de “método construtor” é responsável por criar o objeto de uma determinada classe;
- Ou seja, é uma função especial utilizada para inicializar o objeto quando vai criar uma instância daquela classe.

self:

- É um parâmetro que faz referência a instância de uma determinada classe;
- É responsável por vincular os atributos com os argumentos enviados para uma função ou método → **1º argumento a ser passado.**

4.2.3.1. Init e Self no Python

```
1 class Pessoa:  
2     def __init__(self, nome, idade):  
3         self.nome = nome  
4         self.idade = idade  
5  
6     def apresenta(self):  
7         print(f'{self.nome} tem {self.idade} anos!')  
8  
9 Pessoa1 = Pessoa("Joao","15")  
10 Pessoa2 = Pessoa("Maria","18")  
11  
12 print(Pessoa1.nome)  
13 print(Pessoa1.idade)  
14  
15 Pessoa.apresenta(Pessoa2)
```

4.2.3.2. Init e Self no Python

```
1 class Fruta:
2     def __init__(self, tipo, cor) -> None:
3         self.tipo = tipo
4         self.cor = cor
5
6     def MostrarPropriedades(self):
7         print(f"Sou um(a) {self.tipo} e minha cor é {self.cor}!")
8
9 NovaFruta = Fruta("Banana", "Amarela")
10 NewFruit = Fruta("Melancia", "Verde")
11
12 NovaFruta.MostrarPropriedades()
13 NewFruit.MostrarPropriedades()
```

4.2.4. Break / Continue no Python

```
→ while <expr>:  
    <statement>  
    <statement>  
  
        break ←  
        <statement>  
        <statement>  
  
        continue  
        <statement>  
        <statement>  
  
<statement> ←
```

Break :

- É uma instrução que encerra o *loop*, no qual está inserido, e retoma na execução **da próxima instrução**;
- Ou seja, o *program counter* é redirecionado para o **próximo statement**.

Continue:

- É uma instrução que retorna o *program counter* **para o início** do laço de repetição no qual está contido;
- Ou seja, ela rejeita todas as instruções restantes da interação atual do *loop* e volta para o **início**.

4.2.4.2. Break / Continue no Python

```
1  for i in range(20):
2      print(f"Inicializando a iteração -> {i}")
3      if i > 8:
4          print(f"Finalizando a instrução {i}")
5          break
6          print("Ação nunca vista")
7      if (i > 4):
8          print(f"Continuando a iteração -> {i}")
9          continue
10         print("Ação nunca vista")
11     print(f"Loop normal -> iteração {i}")
12     print(f"i^2 é {i**2}")
13
14     print("Próxima instrução...")
```

Realize a análise desse *script* utilizando o *debug* e identifique o comportamento desse algoritmo para compreender melhor os *statements* - *Break/Continue*

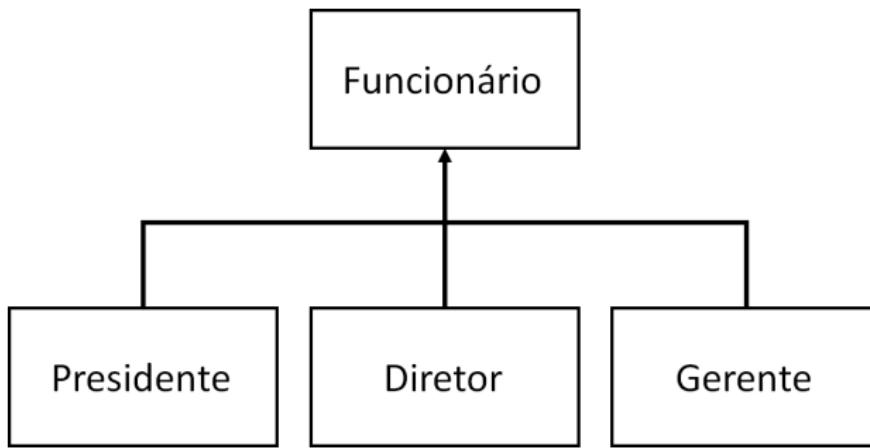
4.2.5. Herança e dependência no Python

Herança:

- A **herança** é um conceito do paradigma da **orientação à objetos** que determina que uma **classe (filha)** pode herdar atributos e métodos de uma outra **classe (pai)** e, assim, evitar que haja muita repetição de código.

Dependência:

- É a **referência de uma biblioteca, objeto, arquivo, etc**, que se faz necessário para que haja o correto funcionamento de uma função/classe/objeto, etc.



4.2.5.1. Herança e dependência no Python

```
1 class Animal():
2     def __init__(self, nome, cor):
3         self.__nome = nome
4         self.__cor = cor
5
6     def comer(self):
7         print(f"O {self.__nome} {self.__cor} está comendo")
8
9 class Gato(Animal):
10    def __init__(self, nome, cor):
11        super().__init__(nome, cor)
12
13 class Cachorro(Animal):
14    def __init__(self, nome, cor):
15        super().__init__(nome, cor)
16
17 class Coelho(Animal):
18    def __init__(self, nome, cor):
19        super().__init__(nome, cor)
20
21
22 gato = Gato("Bichano", "Branco")
23 cachorro = Cachorro("Totó", "Preto")
24 coelho = Coelho("Pernalonga", "Cinza")
25
26 gato.comer()
27 cachorro.comer()
28 coelho.comer()
```

```
1 # Dependência
2
3 import math
4
5 print(math.pi)
6 print(math.cos(0))
```

4.3.1. Dashboard / Tableau: uma interface gráfica

Dashboards / Tableau:

- Em **tecnologia da informação**, pode-se dizer que consiste em um painel composto por uma interface gráfica (*knobs*, mostradores, *bar chart*, etc) que fornece visualizações rápidas dos principais indicadores de desempenho relevantes para um objetivo ou processo de negócios específico;
- Em outras palavras o “Painel de bordo” ou *dashboard* é utilizado como um painel de indicadores que fornece uma representação ilustrada do desempenho dos negócios em toda a organização;
- Um exemplo de aplicação são os **indicadores chave de desempenho (KPIs)** dos quais ajudam as empresas a **medir, monitorar e até gerenciar seu desempenho, alcançando assim seus objetivos**.



4.3.2.1. Dashboard / Tableau: uma interface gráfica

Dashboards / Tableau:

- Para que seja possível criar *Dashboards* em **Python** se faz necessário a instalação da biblioteca **streamlit** via **CMD** por meio do comando:

```
pip install streamlit
```

- Após a instalação da biblioteca e criação do *script* referente ao projeto em questão, novamente, via **CMD**, é possível obter o *dashboard* pelo comando:

```
python -m streamlit run nomeScript.py
```

```
(base) C:\Users\vinic\OneDrive\Professor2023_1\SENAI\BigDataPython\SlideAulasBigDataPython\Algoritmos>pip install streamlit
Requirement already satisfied: streamlit in c:\programdata\anaconda3\lib\site-packages (1.28.2)
Requirement already satisfied: altair<6,>4.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (5.1.2)
Requirement already satisfied: blinker<2,>1.0.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (1.7.0)
Requirement already satisfied: cachetools<6,>4.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (5.3.2)
Requirement already satisfied: click<9,>7.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (8.0.4)
Requirement already satisfied: importlib-metadata<7,>1.4 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (6.0.0)
Requirement already satisfied: numpy<2,>1.19.3 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (1.24.3)
Requirement already satisfied: packaging<24,>=16.8 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (23.1)
Requirement already satisfied: pandas<3,>=1.3.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (2.0.3)
Requirement already satisfied: pillow<11,>7.1.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (9.4.0)
Requirement already satisfied: protobuf<5,>=3.20 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (4.25.1)
Requirement already satisfied: pyarrow<6.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (11.0.0)
Requirement already satisfied: python-dateutil<3,>=2.7.3 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (2.8.2)
Requirement already satisfied: requests<3,>=2.27 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (2.31.0)
Requirement already satisfied: rich<14,>=10.14.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (13.7.0)
:
Requirement already satisfied: pyrsistent!=0.17.0,!0.17.1,!0.17.2,>0.14.0 in c:\programdata\anaconda3\lib\site-packages (from jsonschema)=3.0->altair<6,>4.0->streamlit) (0.18.0)
Requirement already satisfied: mdurl~0.1 in c:\programdata\anaconda3\lib\site-packages (from markdown-it-py)>=2.2.0->rich<14,>=10.14.0->streamlit) (0.1.0)

(base) C:\Users\vinic\OneDrive\Professor2023_1\SENAI\BigDataPython\SlideAulasBigDataPython\Algoritmos>streamlit run Dashboarduber.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8502
Network URL: http://192.168.0.220:8502
```

4.3.3.2. Dashboard / Tableau: uma interface gráfica

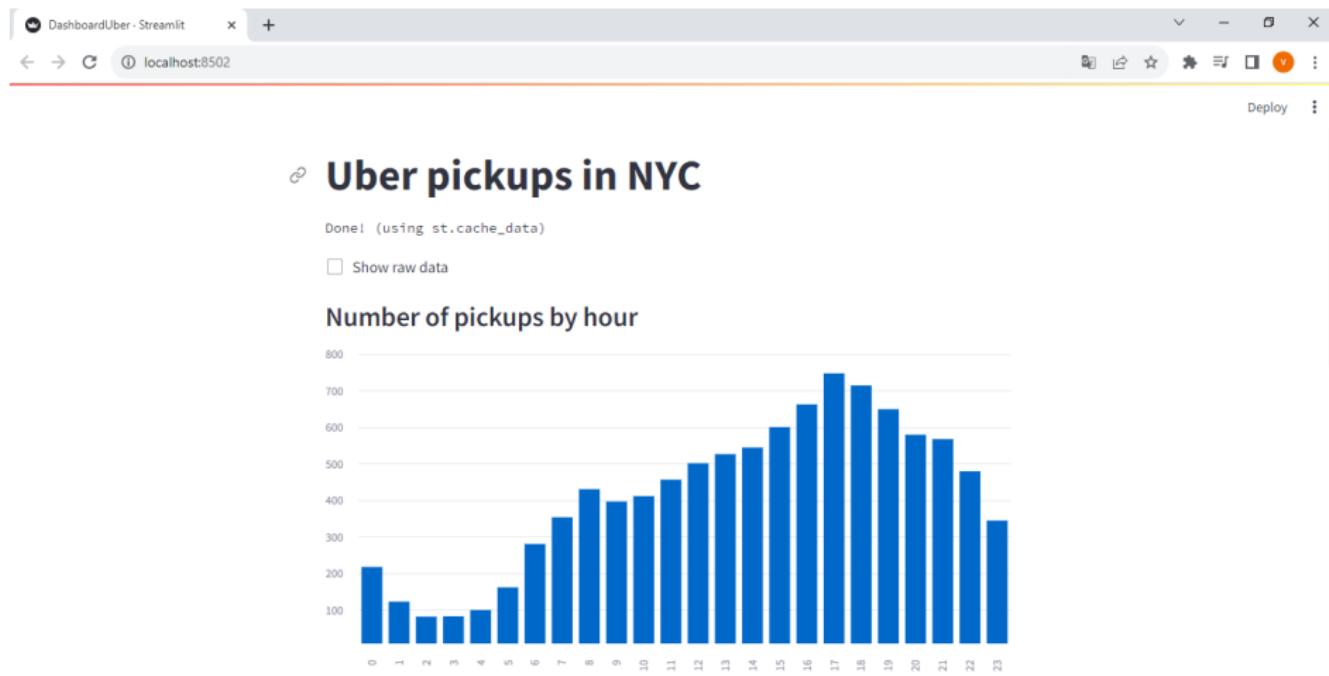
Dashboards / Tableau:

- O link a seguir: <https://docs.streamlit.io/> fornece a documentação do *streamlit* desde a instalação até exemplos de aplicação prática;
- O *script* a seguir apresenta um *dashboard* da utilização de Ubers na cidade de Nova York por meio de um banco de dados .csv durante as 24h do dia;

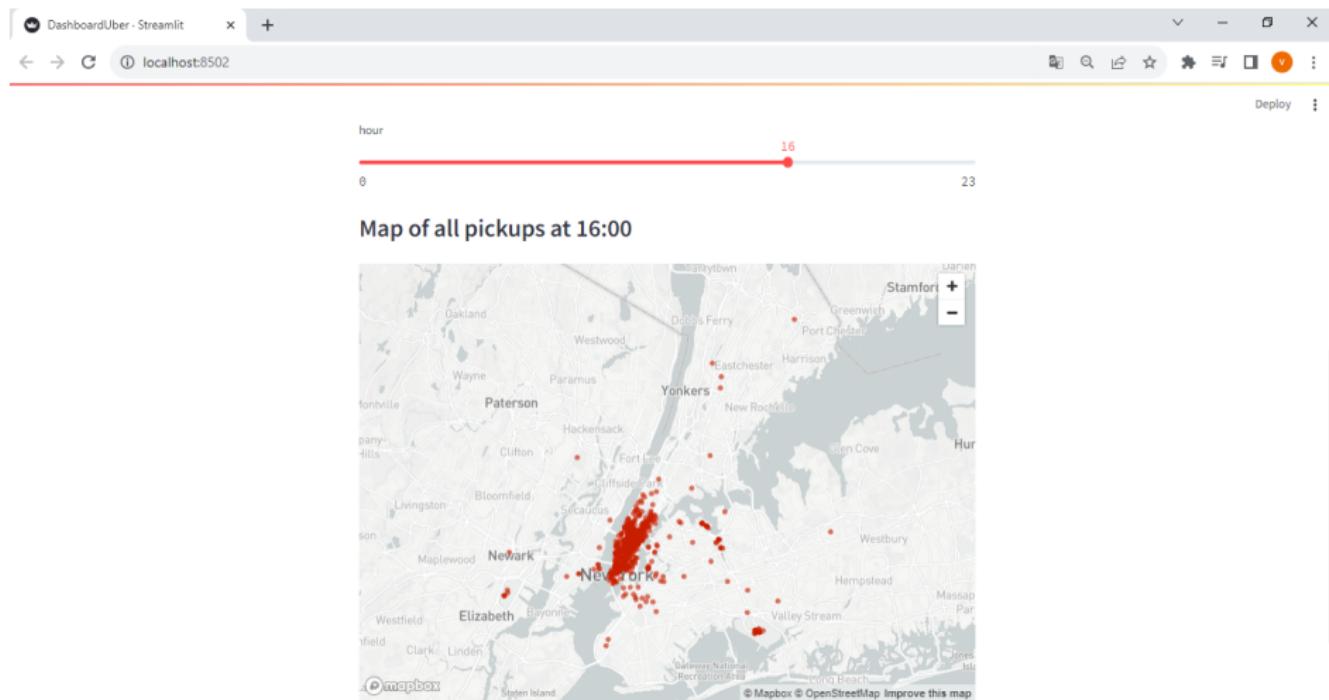
The screenshot shows a Streamlit application running in a browser window. The title of the dashboard is "Uber pickups in NYC". Below the title, there is a message "Done! (using st.cache_data)". There is a checked checkbox labeled "Show raw data". A section titled "Raw data" contains a table with 10 rows of data. The columns are "date/time", "lat", "lon", and "base". The data is as follows:

	date/time	lat	lon	base
0	2014-09-01 00:01:00	40.2201	-74.0021	B02512
1	2014-09-01 00:01:00	40.75	-74.0027	B02512
2	2014-09-01 00:03:00	40.7559	-73.9864	B02512
3	2014-09-01 00:06:00	40.745	-73.9889	B02512
4	2014-09-01 00:11:00	40.8145	-73.9444	B02512
5	2014-09-01 00:12:00	40.6735	-73.9918	B02512
6	2014-09-01 00:15:00	40.7471	-73.6472	B02512
7	2014-09-01 00:16:00	40.6613	-74.2691	B02512
8	2014-09-01 00:32:00	40.3745	-73.9999	B02512
9	2014-09-01 00:33:00	40.7633	-73.9773	B02512

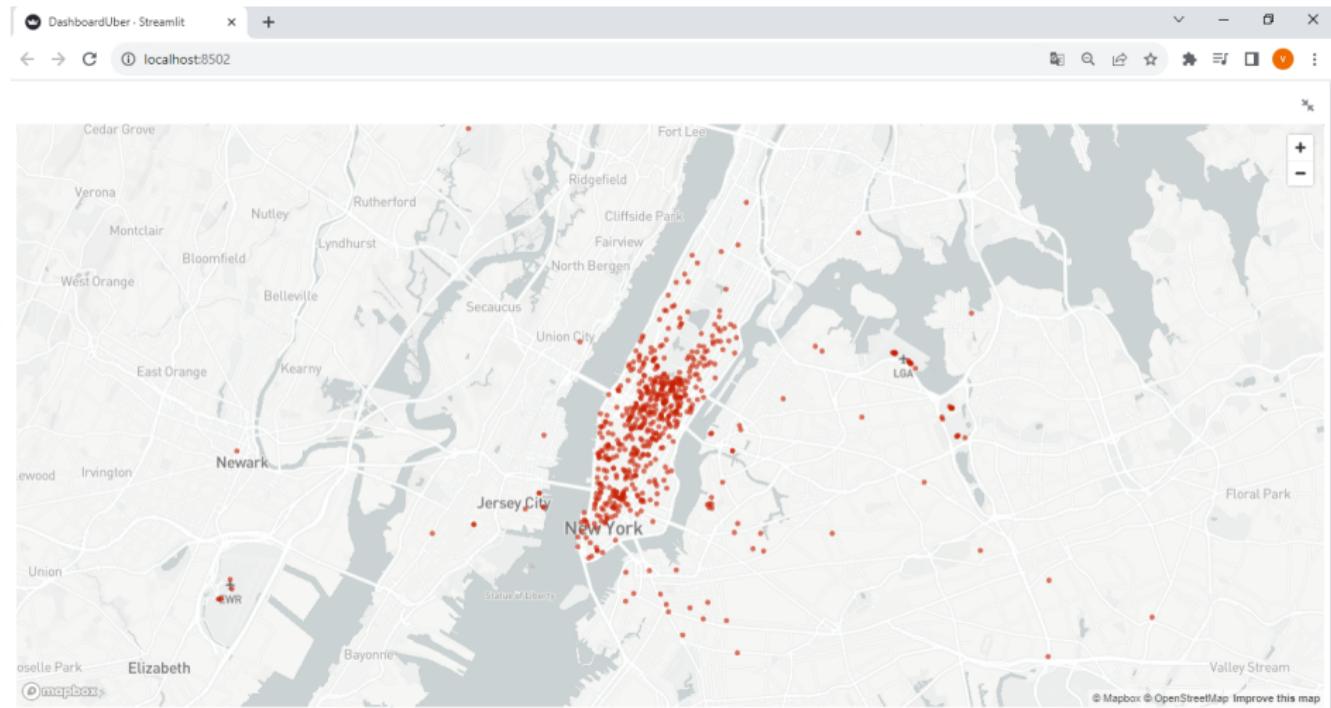
4.3.4.2.1. Dashboard / Tableau: uma interface gráfica



4.3.5.2.2. Dashboard / Tableau: uma interface gráfica



4.3.6.2.2.1. Dashboard / Tableau: uma interface gráfica

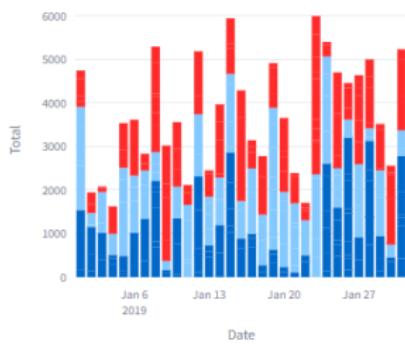


4.3.6.6.1. Dashboard / Tableau: uma interface gráfica

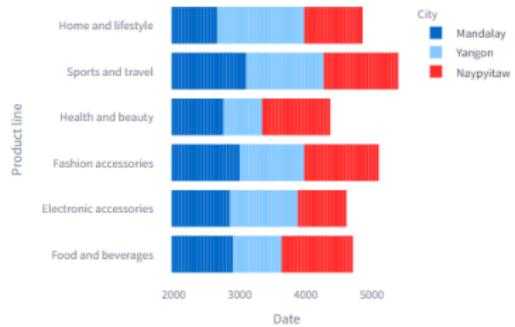
```
1 import streamlit as st
2 import pandas as pd
3 import numpy as np
4
5 st.title('Uber pickups in NYC')
6
7 DATE_COLUMN = 'date/time'
8 DATA_URL = ('https://s3-us-west-2.amazonaws.com/'
9             'streamlit-demo-data/uber-raw-data-sep14.csv.gz')
10
11 @st.cache_data
12 def load_data(nrows):
13     data = pd.read_csv(DATA_URL, nrows=nrows)
14     lowercase = lambda x: str(x).lower()
15     data.rename(lowercase, axis='columns', inplace=True)
16     data[DATE_COLUMN] = pd.to_datetime(data[DATE_COLUMN])
17     return data
18
19 data_load_state = st.text('Loading data ... ')
20 data = load_data(10000)
21 data_load_state.text("Done! (using st.cache_data)")
22
23 if st.checkbox('Show raw data'):
24     st.subheader('Raw data')
25     st.write(data)
26
27 st.subheader('Number of pickups by hour')
28 hist_values = np.histogram(data[DATE_COLUMN].dt.hour, bins=24, range=(0, 24))[0]
29 st.bar_chart(hist_values)
30
31 # Some number in the range 0–23
32 hour_to_filter = st.slider('hour', 0, 23, 17)
33 filtered_data = data[data[DATE_COLUMN].dt.hour == hour_to_filter]
34
35 st.subheader('Map of all pickups at %s:00' % hour_to_filter)
36 st.map(filtered_data)
```

4.3.7. Dashboard no banco: supermarket_sales.csv

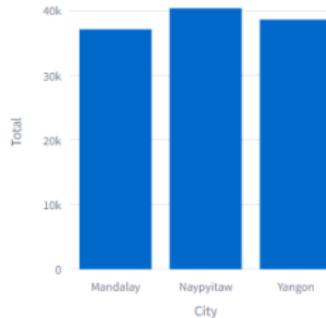
Faturamento por dia



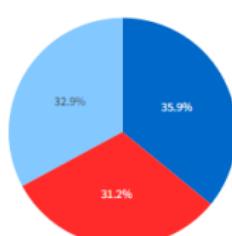
Faturamento por tipo de produto



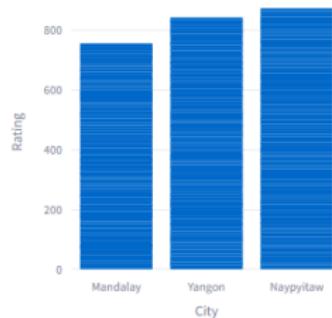
Faturamento por filial



Faturamento por tipo de pagamento



Avaliação



4.4.1. Definições de matemática estatística

Estatística:



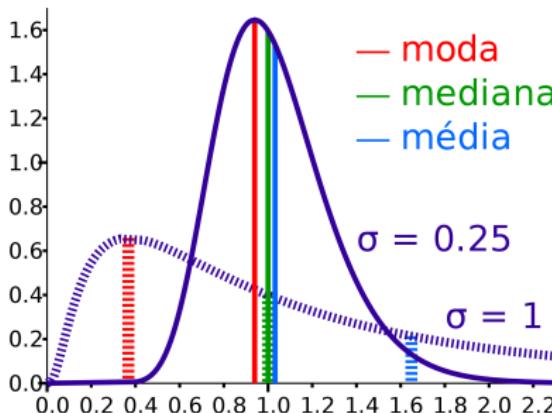
- É a ciência que realiza coleta, análise e até interpretação dos dados com informações pertinentes, das quais deseja-se avaliar;
- Pode-se dizer que a estatística utiliza-se das teorias probabilísticas: medidas de informação, métodos estocásticos, dentre outros, para modelar o comportamento de um determinado objeto de estudo quanto a sua aleatoriedade, incerteza, previsões, etc;
- O propósito para utilização de tal técnica pode ser desde a elaboração de uma pesquisa até mesmo um direcionamento para refinar/novos objetivos. Ou seja, “filtragem” de informações, correlações e consequente modelagem do comportamento das variáveis matematicamente criando indicadores para tomada de ação.

4.4.2. Estatística: média, moda e mediana

Média (μ):

- Consiste no **valor de concentração de dados em uma distribuição** - ponto de equilíbrio;
- Existem alguns tipos de média: aritmética, geométrica, harmônica e ponderada;
- Em outras palavras, pode-se dizer que a média aritmética (MA) é a **medida central de tendência de uma série de dados**.
- Uma MA pode ser modelada por:

$$MA = \frac{X_1 + X_2 + X_3 + \dots + X_n}{n}$$



Exemplo: A média da idade dos funcionários de uma empresa: {28, 30, 29, 32}

$$MA = \frac{38 + 21 + 42 + 23}{4} = 31 \text{ anos}$$

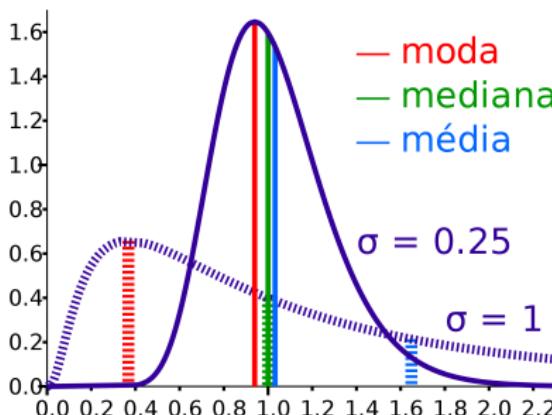
4.4.2.1. Estatística: média, moda e mediana

```
1 #Cálculo da média composta por 4 notas
2
3 nota01=float(input("Digite a nota do PRIMEIRO bimestre: "))
4 nota02=float(input("Digite a nota do SEGUNDO bimestre: "))
5 nota03=float(input("Digite a nota do TERCEIRO bimestre: "))
6 nota04=float(input("Digite a nota do QUARTO bimestre: "))
7 media = (nota01 + nota02 + nota03 + nota04)/4
8 print("A nota do 1º Bimestre é: ",nota01)
9 print("A nota do 2º Bimestre é: ",nota02)
10 print("A nota do 3º Bimestre é: ",nota03)
11 print("A nota do 4º Bimestre é: ",nota04)
12 print("A média final é: ",media)
13 if media >= 7:
14     print("Aprovado")
15 else:
16     print("Não Aprovado")
17
18 # Biblioteca estatística
19 import statistics as st
20
21 print('**-* Cálculo da média com a biblioteca statistics **-**')
22
23 # Calculando a média dos conjuntos
24 print(st.mean([1, 3, 5, 7, 9, 11, 13]))
25 print(st.mean([1, 3, 5, 7, 9, 11]))
26 print(st.mean([-11, 5.5, -3.4, 7.1, -9, 22]))
```

4.4.2.2. Estatística: média, moda e mediana

Moda:

- Pode ser definida também como **moda amostral e populacional**;
 - Trata do valor que ocorre com maior frequência ou o valor mais comum contido em um conjunto de dados;
 - Ou seja, o **valor que mais se repete** em uma série de dados. Exemplo:
- Lista = {4, 11, 15, 37, 37, 40, 58, 58, 58, 58, 9}*
- O número 37 se repete 2 vezes enquanto o 58 se repete 4 vezes. Os demais números aparecem apenas uma única vez. Dessa forma, 58 é a moda do conjunto apresentado.



4.4.2.2.1. Estatística: média, moda e mediana

```
1 # Biblioteca estatística
2 import numpy as np
3 import statistics as st
4
5 print(st.mode([1, 1, 2, 3, 3, 3, 3, 4]))
6 print(st.mode(["red", "blue", "blue", "red", "green", "red", "red"]))
7 print(st.multimode('aabbbbccddddeeffffgg'))
8
9
10 # Biblioteca para statistica de arrays
11 from scipy import stats
12 res1 = stats.mode(np.array([1,0,0,0,1,0,0,0,1]))
13 arr = np.array([[1, 2, 3, 4, 5],
14                 [1, 2, 2, 2, 2],
15                 [4, 5, 7, 9, 4],
16                 [6, 7, 8, 9, 2],
17                 [2, 3, 4, 8, 6]])
18 res2 = stats.mode(arr)
19 res1.mode
20 res1.count
21 print(res2)
```

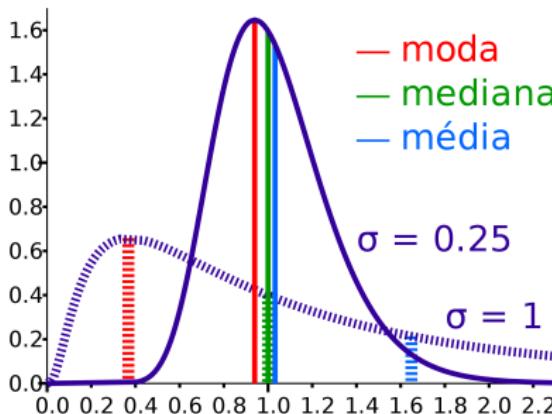
Avalie os resultados obtidos e note as diferenças dentre as bibliotecas utilizadas.

4.4.2.3. Estatística: média, moda e mediana

Mediana:

- Consiste no valor cujo separa a metade maior da metade menor de uma amostra, população ou distribuição de probabilidade;
- Ou seja, a mediana pode ser o valor do meio de um conjunto de dados;
- Exemplo:

$$\text{Lista} = \{1, 2, 3, 3, 4, 5, \textcolor{red}{6}, 7, 8, 9, 9, 6, 5, \}$$
- O número 6 está alocado exatamente na metade do conjunto de dados. Caso a sequência seja par realiza-se a média dos números centrais: $\text{Lista} = \{1, 2, 3, \textcolor{red}{6}, \textcolor{red}{7}, 8, 9, 9, \}$



$$MD = \frac{6 + 7}{2} = 6,5$$

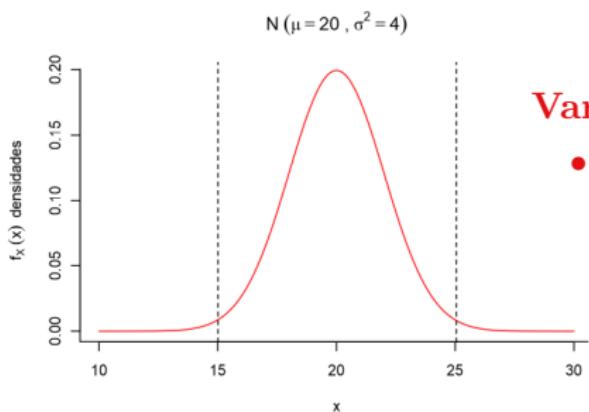
4.4.2.3.1. Estatística: média, moda e mediana

```
1 import statistics as st
2
3 print(st.median([1, 3, 5]))
4 print(st.median([1, 3, 5, 7]))
5
6 print(st.median_low([1, 3, 5]))
7 print(st.median_low([1, 3, 5, 7]))
8
9 print(st.median_high([1, 3, 5]))
10 print(st.median_high([1, 3, 5, 7]))
```

4.4.3. Estatística: variância e desvio padrão

Desvio padrão (σ):

- É uma medida de dispersão em torno da média populacional de uma variável aleatória estudada.



Variância (σ^2):

- Na teoria da probabilidade tal grandeza, de uma variável aleatória ou processo estocástico, consiste em uma medida da sua dispersão estatística, indicando o “quão longe” em geral os seus valores se encontram daquele esperado. Ex.: $L = \{3, 7, 6, 5, 4\}$

$$\sigma^2 = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}} \leftrightarrow \sqrt{\sigma^2} = \sigma$$

$$\sigma^2 = \frac{(3-5)^2 + (7-5)^2 + (6-5)^2 + (5-5)^2 + (4-5)^2}{5-1} = 2 \leftrightarrow \sqrt{\sigma^2} \approx 1,4142$$

4.4.3.3.1. Estatística: variância e desvio padrão

```
1 # Biblioteca estatística
2 import statistics as st
3
4 print(st.variance([3, 7, 6, 5, 4]))
5 print(st.pvariance([3, 7, 6, 5, 4]))
6
7 print(st.stdev([3, 7, 6, 5, 4]))
8 print(st.pstdev([3, 7, 6, 5, 4]))
```

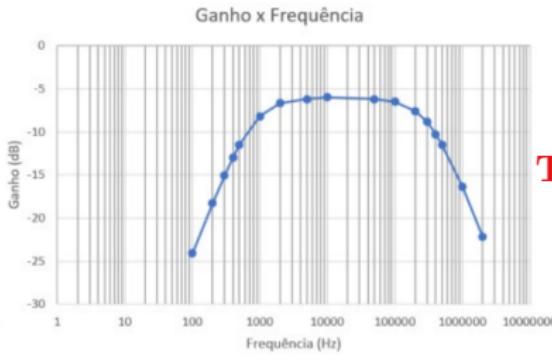
Identifique as diferenças, de cada uma das funções, para os resultados de σ e σ^2 .

4.4.4. Estatística descritiva e tabulação de dados

Estatística descritiva:

- A estatística descritiva tem como principal objetivo sintetizar uma série de valores, de mesma natureza, permitindo dessa forma que se tenha uma visão global da variação desses valores. Os dados podem estar em: tabelas, gráficos e medidas descritivas.

- Ou seja, é um ramo da estatística que permite de forma sistemática a aplicação de várias técnicas com intuito de descrever e resumir um conjunto de dados expressando de forma visual valores numéricos, de maneira diferente, facilitando assim a compreensão.

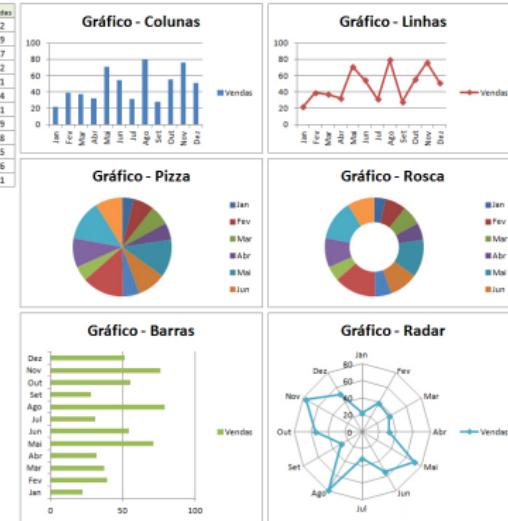


Tabulação de dados:

- Tabular consiste em “colocar dados em colunas ou tabelas”. Em pesquisas empíricas a tabulação é uma etapa fundamental para organizar e sistematizar os dados.

4.4.5. Gráficos: uma ferramenta visual

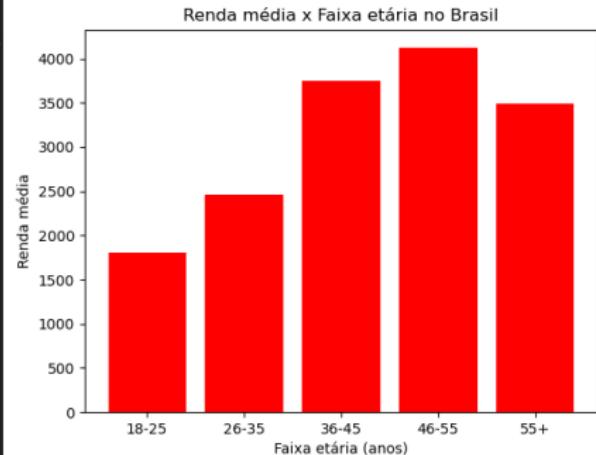
Gráficos:



- Os gráficos permitem a realização de uma modelagem de um grupo de dados de forma abrangente. Ou seja, fornecem uma visão panorâmica da problemática em questão;
- Consistem em ferramentas visuais oferecendo ao usuário a obtenção de informações em fração de segundos;
- As grandezas expressas graficamente podem ser obtidas de um banco de dados extenso ou até mesmo de tabelas com o intuito de sintetizar o conteúdo simplificando e facilitando a compreensão;
- Dentre os tipos existentes tem-se: barras, dispersão, coluna, linhas, pizza, etc.

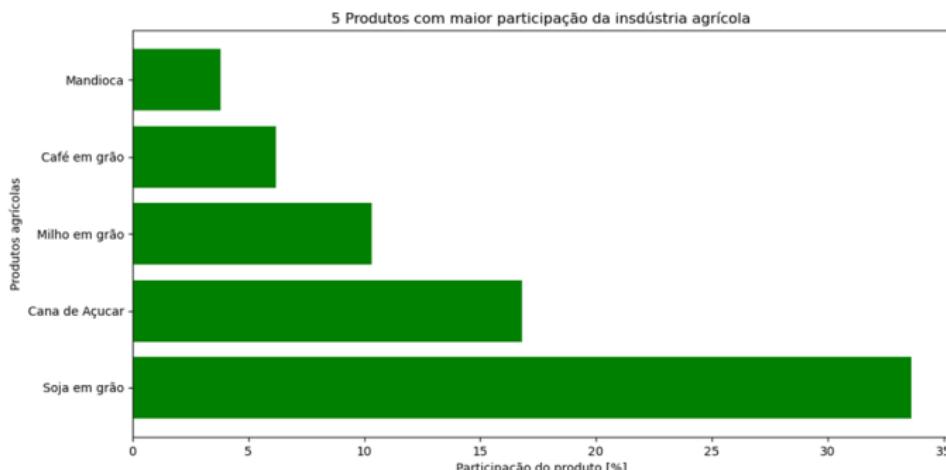
4.4.6. Gráfico de colunas

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Vetores de dados
5 faixaEtaria = np.array(['18-25', '26-35', '36-45', '46-55', '55+'])
6 renda = np.array([1805.45, 2458.12, 3752.12, 4120.89, 3486.22])
7
8 # Criação do gráfico de barras
9 plt.bar(faixaEtaria, renda, color="red")
10
11 # Dados numéricos no eixo X
12 plt.xticks(faixaEtaria)
13
14 # Nome do eixo Y
15 plt.ylabel('Renda média')
16
17 # Nome do eixo X
18 plt.xlabel('Faixa etária (anos)')
19
20 # Título do gráfico
21 plt.title('Renda média x Faixa etária no Brasil')
22
23 # Apresentação da figura
24 plt.show()
```



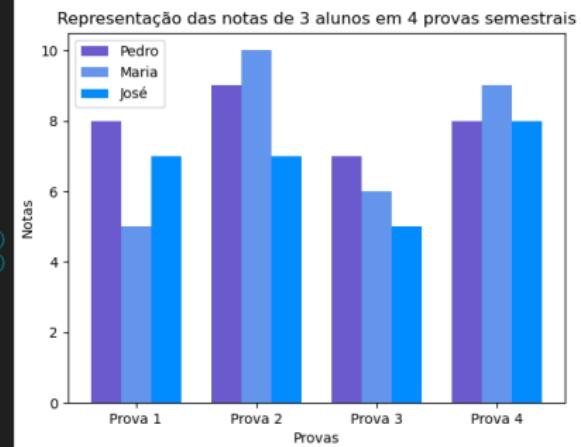
4.4.7. Gráfico de barras horizontais

```
1 import matplotlib.pyplot as plt
2
3 produtos = ['Soja em grão', 'Cana de Açucar', 'Milho em grão', 'Café em grão', 'Mandioca']
4
5 quantidade_pct = [33.6, 16.8, 10.3, 6.2, 3.8]
6
7 plt.figure(figsize=(12, 6))
8 plt.barh(produtos, quantidade_pct, color='green')
9
10 # Inserindo nome nos eixos e gráfico
11 plt.ylabel("Produtos agrícolas")
12 plt.xlabel("Participação do produto [%]")
13 plt.title("5 Produtos com maior participação da insdústria agrícola")
14
15 plt.show()
```



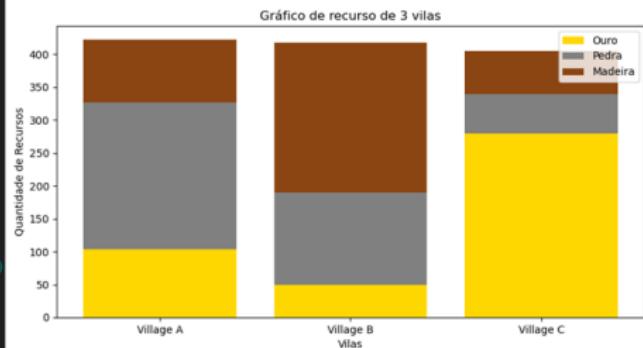
4.4.8. Gráfico de barras agrupadas

```
2 import matplotlib.pyplot as plt
3
4 notas_pedro = [8, 9, 7, 8]
5 notas_maria = [5, 10, 6, 9]
6 notas_jose = [7, 7, 5, 8]
7
8
9 barWidth = 0.25
10
11 r1 = np.arange(len(notas_pedro))
12 r2 = [x + barWidth for x in r1]
13 r3 = [x + barWidth for x in r2]
14
15 plt.bar(r1, notas_pedro, color="#6A5ACD", width=barWidth, label='Pedro')
16 plt.bar(r2, notas_maria, color="#6495ED", width=barWidth, label='Maria')
17 plt.bar(r3, notas_jose, color="#008BFF", width=barWidth, label='José')
18
19 plt.xlabel('Provas')
20 plt.xticks([r + barWidth for r in range(len(notas_pedro))],
21           ['Prova 1', 'Prova 2', 'Prova 3', 'Prova 4'])
22 plt.ylabel('Notas')
23 plt.title('Representação das notas de 3 alunos em 4 provas semestrais')
24
25 plt.legend()
26 plt.show()
```



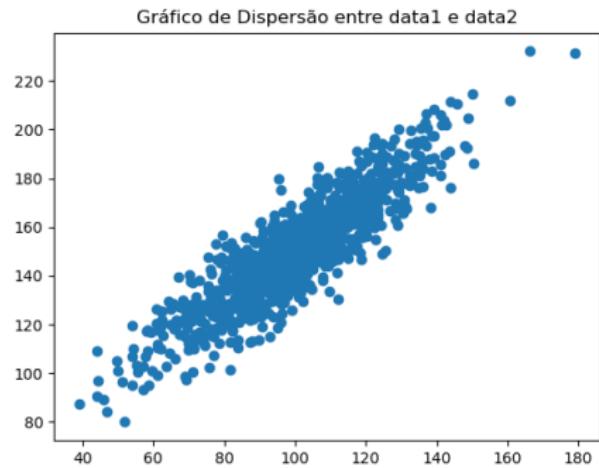
4.4.9. Gráfico de barras empilhadas

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 gold = np.array([104, 50, 280])
5 stone = np.array([223, 140, 60])
6 wood = np.array([95, 228, 65])
7
8 resources = ['Ouro', 'Pedra', 'Madeira']
9
10 villages = ['Village A', 'Village B', 'Village C']
11
12 plt.figure(figsize=(10,5))
13
14 plt.bar(villages, gold, color='gold')
15 plt.bar(villages, stone, color='grey', bottom = gold)
16 plt.bar(villages, wood, color='saddlebrown', bottom = gold + stone)
17
18 plt.xlabel('Vilas')
19 plt.ylabel('Quantidade de Recursos')
20 plt.title('Gráfico de recurso de 3 vilas')
21
22 plt.legend(['Ouro', 'Pedra', 'Madeira'])
23
24 plt.show()
```



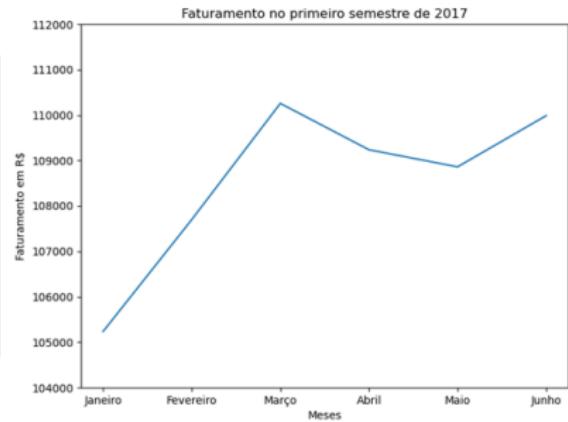
4.4.10. Gráfico de dispersão

```
1  from numpy import mean
2  from numpy import std
3  from numpy import correlate
4  from numpy.random import randn
5  from numpy.random import seed
6  from matplotlib import pyplot
7
8  # semente para gerar números aleatórios
9  seed(1)
10 # preparação:
11 data1 = 20 * randn(1000) + 100
12 data2 = data1 + (10 * randn(1000) + 50)
13
14 # media e desvio padrao de data1
15 print('média de data1: ', str(round(mean(data1),2)))
16 print('desvio padrão de data1: ', str(round(std(data1),2)))
17
18 # media e desvio padrao de data2
19 print('média de data2: ', str(round(mean(data2),2)))
20 print('desvio padrão de data1: ', str(round(std(data2),2)))
21
22 # plot
23 pyplot.scatter(data1, data2)
24 pyplot.title('Gráfico de Dispersão entre data1 e data2')
25 pyplot.show()
```



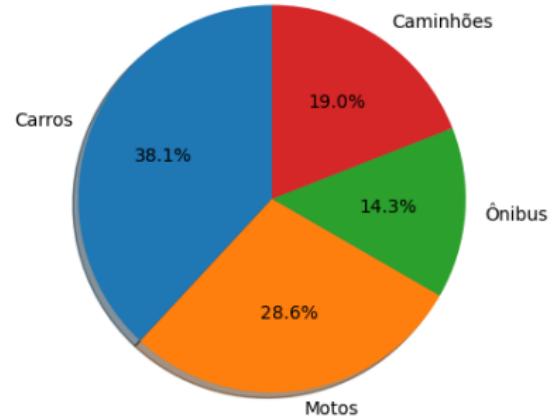
4.4.11. Gráfico de linha

```
1 import matplotlib.pyplot as plt
2
3 meses = ['Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho']
4 valores = [105235, 107697, 110256, 109236, 108859, 109986]
5
6 plt.figure(figsize=(8, 6))
7 plt.plot(meses, valores)
8 plt.ylim(104000, 112000)
9 plt.title('Faturamento no primeiro semestre de 2017')
10 plt.xlabel('Meses')
11 plt.ylabel('Faturamento em R$')
12 plt.show()
```



4.4.12. Gráfico de setor/pizza

```
1 import matplotlib.pyplot as plt
2
3 # Conjunto de dados a serem plotados
4 labels = 'Carros', 'Motos', 'Ônibus', 'Caminhões'
5 sizes = [40, 30, 15, 20]
6
7 # Criando a área de plotagem
8 fig, ax1 = plt.subplots()
9
10 # Criando o gráfico
11 ax1.pie(sizes, labels=labels, autopct='%.1f%%',
12         shadow=True, startangle=90)
13
14 # Apresentar a figura
15 plt.show()
```



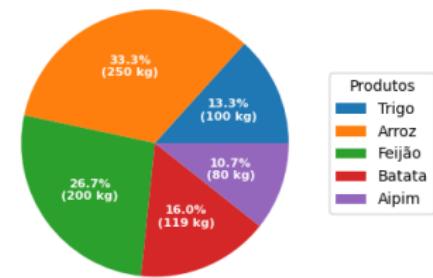
4.4.12.1. Gráfico de setor/pizza

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 fig, ax = plt.subplots(figsize=(6, 4), subplot_kw=dict(aspect="equal"))
5
6 recipe = ["Trigo",
7             "Arroz",
8             "Feijão",
9             "Batata",
10            "Aipim"]
11
12 data = [100, 250, 200, 120, 80]
13
14 def func(pct, allvals):
15     absolute = int(pct/100.*np.sum(allvals))
16
17     return "{:.1f}\n({:d} kg)".format(pct, absolute)
18
19 wedges, texts, autotexts = ax.pie(data, autopct=lambda pct: func(pct, data),
20                                     textprops=dict(color="w"))
21
22 ax.legend(wedges, recipe,
23            title="Produtos",
24            loc="center left",
25            bbox_to_anchor=(1, 0, 0.5, 1))
26
27 plt.setp(autotexts, size=8, weight="bold")
28
29 ax.set_title("Quantidade de itens do estoque em kg:")
30
31 plt.show()

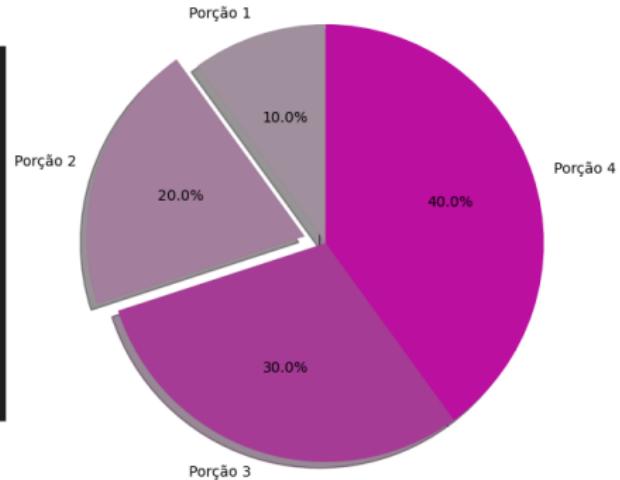
```

Quantidade de itens do estoque em kg:



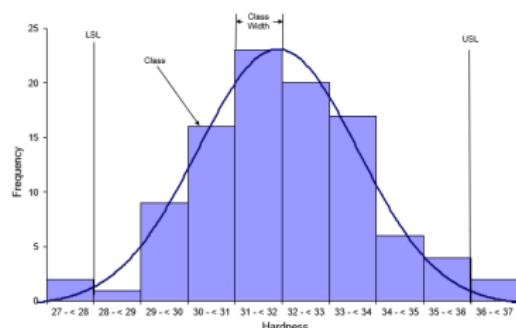
4.4.12.2. Gráfico de setor/pizza

```
1 import matplotlib.pyplot as plt
2
3 labels = 'Porção 1', 'Porção 2', 'Porção 3', 'Porção 4'
4 sizes = [10, 20, 30, 40]
5
6 explode = (0, 0.1, 0, 0)
7
8 fig1, ax1 = plt.subplots(figsize=(6, 6))
9
10 c = ['#A08F9D', '#A37E9D', '#A63B95', '#BB0FA0']
11
12 ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
13         shadow=True, startangle=90, colors=c)
14
15 plt.show()
```

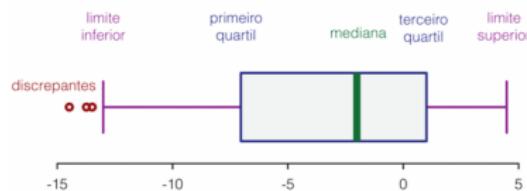


4.4.13. Histograma e *boxplot*

Histograma:



- Também conhecido como distribuição de frequências e sete ferramentas da qualidade;
- Consiste em uma representação gráfica dispostas em colunas ou barras de um conjunto de dados previamente tabulado e dividido em classes uniformes ou não;
- A base de cada retângulo representa uma classe e a altura representa a quantidade ou frequência absoluta com que o valor ocorre no conjunto de dados.



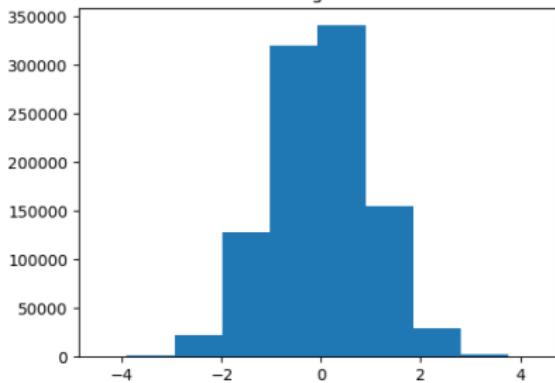
Boxplot:

- Em estatística descritiva também como diagrama de caixa e diagrama de extremo e quartis é uma **ferramenta utilizada** para analisar e comparar a variação de uma variável entre diferentes grupos de dados.

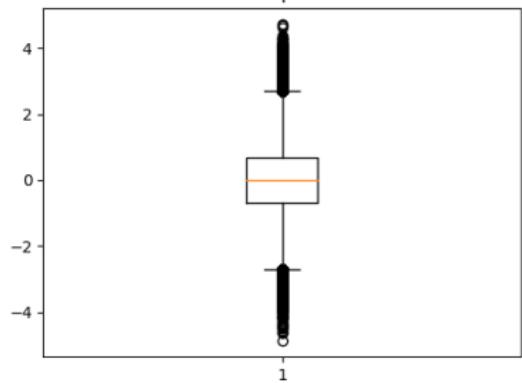
4.4.13.1. Histograma e *boxplot*

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n = np.random.randn(1000000)
5
6 fig, axes = plt.subplots(1, 2, figsize=(12, 4))
7
8 axes[0].hist(n)
9 axes[0].set_title('Histograma')
10 axes[0].set_xlim((min(n), max(n)))
11
12 axes[1].boxplot(n)
13 axes[1].set_title('Boxplot')
14
15 plt.show()
```

Histograma

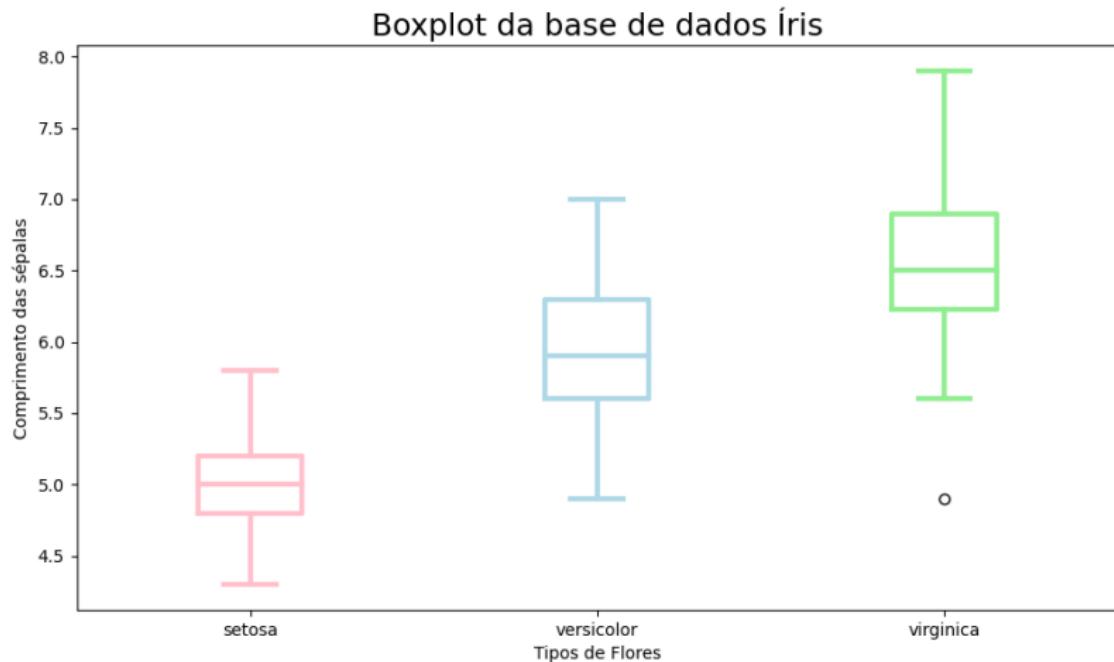


Boxplot



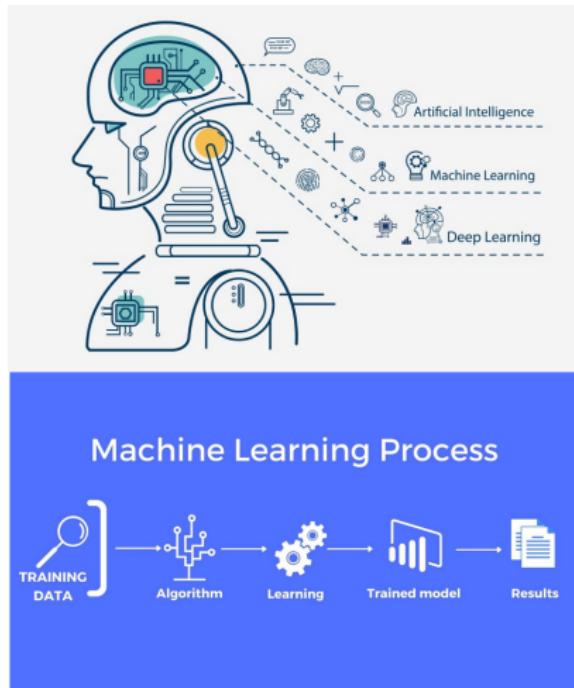
4.4.13.2. Histograma e *boxplot*

Utilizando o comando da biblioteca *seaborn*: `sns.load_dataset('iris')` realize o *upload* do banco de dados: *IrisDataSet* e elabore um *boxplot*, como Ex. a seguir.



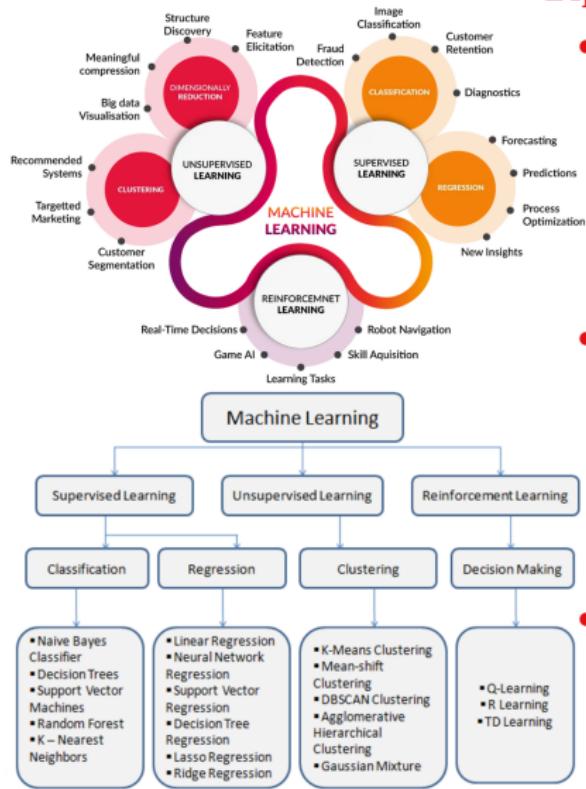
5.2.1. Técnicas de *machine learning*

Machine learning:



- O *machine learning* - aprendizado de máquina - é uma ramificação da engenharia/ciência da computação que evoluiu do estudo de *pattern recognition* e da teoria de aprendizado computacional em IA;
- Tal conceito baseia-se na habilidade dos computadores aprenderem sem serem explicitamente programados. Ou seja, aprendem com seus erros e são capazes de generalizar conceitos realizando por exemplo, até a previsão sobre dados desconhecidos;
- O conceito de *machine learning*, embutido na inteligência computacional, cujo é uma ramificação da IA, detém de um raciocínio indutivo responsável por extração de padrões de grandes conjuntos de dados.

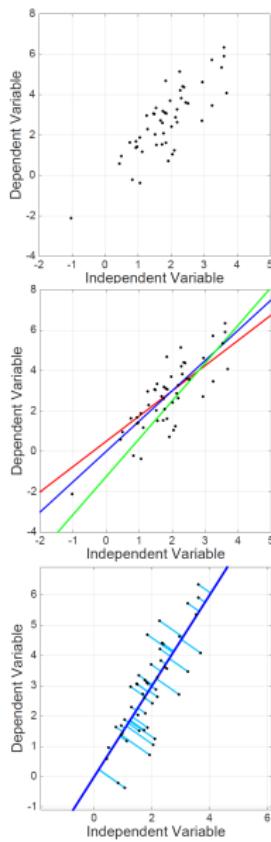
5.2.1.1. Técnicas de *machine learning*



Tipos de aprendizagem:

- Aprendizagem supervisionada:** função na qual mapeia uma entrada para uma saída com base em pares entrada-saída fornecidas como exemplo. Ou seja, **para cada amostra de entrada tem-se a respectiva saída desejada (rótulo)**. Ex: RNA's, SVM's, etc;
- Aprendizagem não-supervisionada:** o modelo deve, por conta própria, **encontrar padrões ou estruturas em conjuntos de dados não rotulados**. Ou seja, não há o conjunto de rótulos como na aprendizagem supervisionada. Ex: *k-means*, PCA, etc;
- Aprendizagem por reforço:** a máquina tenta aprender qual a melhor ação a ser tomada tomando como base circunstâncias na qual essa ação será executada: conceito de $\max(\text{"recompensa"})$ Ex: Q/R/TD learning

5.2.2. Machine learning: Regressão linear

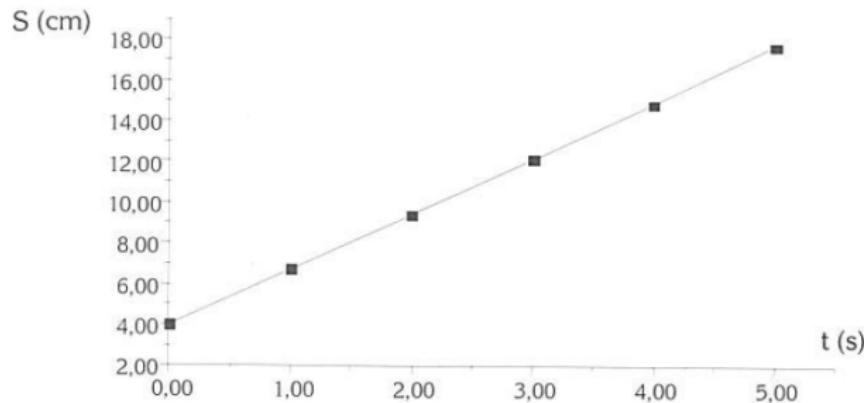


- Em estatística tal conceito consiste em uma equação para se estimar a condicional (valor esperado) de uma determinada variável y (imagem) a partir de valores de outras variáveis x (domínio);
- Existem modelos que realizam a regressão linear utilizando o **método dos mínimos quadrados**, por exemplo;
- Tal técnica prevê o valor de dados desconhecidos usando outro valor de dados relacionado e conhecido por meio da abordagem de **minimização de uma penalização**;
- Pode-se dizer que deve-se mensurar um possível erro cujo mediria o quanto incorreta seria a previsão de uma função linear para um conjunto de pares de variáveis independentes e dependentes conhecidas.

5.3.1.2. Relembrando: obtenção de informações via gráfico

Exemplo: Os pontos experimentais na tabela abaixo representam a posição de um objeto em MRU ao longo do tempo e sua representação gráfica também apresenta-se a seguir:

S (cm)	4,00	6,71	9,43	12,18	14,87	17,70
t (s)	0,00	1,00	2,00	3,00	4,00	5,00



5.3.2. Relembrando: equação da reta

Como apresentado no gráfico anterior, a curva que representa o MRU é uma reta. A partir da geometria analítica tem-se:

$$y = \textcolor{blue}{a}x + \textcolor{red}{b}$$

onde a e b são os coeficientes angular e linear, respectivamente. O próximo passo é obter os parâmetros $\textcolor{blue}{a}$ e $\textcolor{red}{b}$ que ajustam os pontos experimentais à Eq. da reta:

$$\textcolor{blue}{a} = \frac{y_2 - y_1}{x_2 - x_1}$$

onde (x_1, y_1) e (x_2, y_2) são dois pontos quaisquer que **pertencem à reta**. Sempre se devem escolher pontos de fácil determinação.

No exemplo anterior, foram escolhidos no gráfico os seguintes pontos, para dois casos distintos. Um dos pares são pontos presentes na reta mas não experimentais e o outro são pontos experimentais:

- 1) $P_1 = (0, 75; 6, 00)$ e $P_2 = (4, 50; 16, 30)$;
- 2) $P_3 = (2, 00; 9, 43)$ e $P_4 = (3, 00; 12, 18)$

5.3.2.1. Relembrando: equação da reta

Desta forma, resolvendo as contas com ambos os casos, tem-se:

$$1) \quad a = \frac{16,30 - 6,00}{4,50 - 0,75} = \frac{10,30}{3,75} = 2,7467 \text{ cm/s}$$

$$2) \quad a = \frac{12,18 - 9,43}{3,00 - 2,00} = \frac{2,45}{1} = 2,75 \text{ cm/s}$$

Nunca se deve esquecer das unidades: a unidade de a sempre é $[Y]/[X]$ e a de b é a mesma da variável dependente. Desta forma, para a determinação do parâmetro linear b , escolhe-se qualquer ponto da reta, como por exemplo o P_2 , para o caso 1 e P_4 para o caso 2, e substitui-se os valores de t e S na equação da reta com o valor já determinado de a :

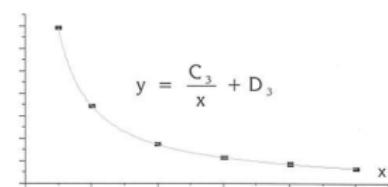
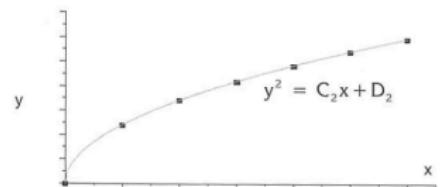
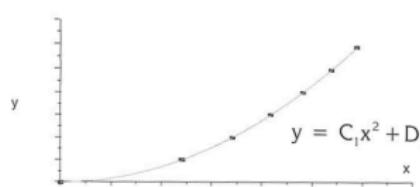
$$1) \quad y = ax + b \rightarrow 16,30 = 2,7467 \cdot 4,50 + b \rightarrow b = 3,925 \rightarrow \boxed{y = 2,7467 \cdot x + 3,925}$$

$$2) \quad y = ax + b \rightarrow 12,18 = 2,75 \cdot 3,00 + b \rightarrow b = 3,93 \rightarrow \boxed{y = 2,75 \cdot x + 3,93}$$

Desta forma é possível encontrar qualquer valor de imagem para um dado domínio

5.3.3. Relembrando: linearização de gráficos

Muitas vezes, ao coletar os dados experimentais para a construção de uma curva, nota-se, ao construir o gráfico, que os pontos não obedecem a uma equação de reta, como os exemplos a seguir.



Como proceder agora para determinar os parâmetros C_1 , D_1 , C_2 , D_2 , C_3 e D_3 ? Deve-se recordar que não necessariamente a curva passa sobre os pontos experimentais: além disso, muitas vezes a equação da curva é desconhecida.

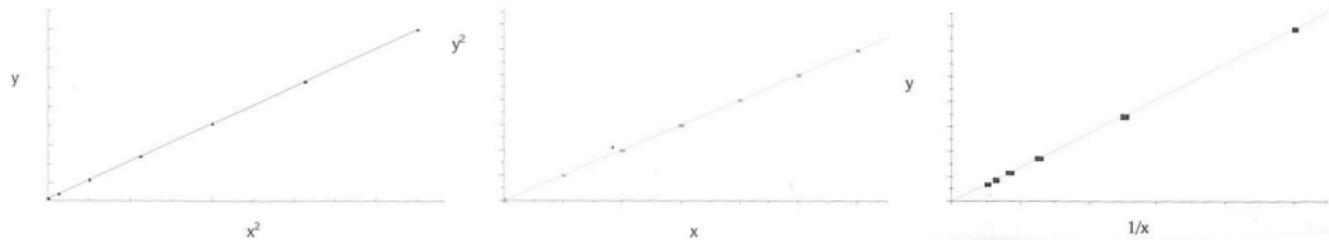
Para isso, utiliza-se o artifício chamado *linearização de gráficos*, que nada mais é do que, por meio de uma mudança de variáveis, construir um novo gráfico que seja representado agora por uma reta.

5.3.3.1. Relembrando: linearização de gráficos

A primeira curva, representada por $y = C_1x^2 + D_1$, caracterizada por um comportamento exponencial deverá ser comparada com $Y = ax + b$ (linear);

- $Y = y$;
- $x = x^2$;
- $a = C_1$;
- $b = D_1$

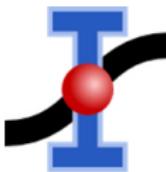
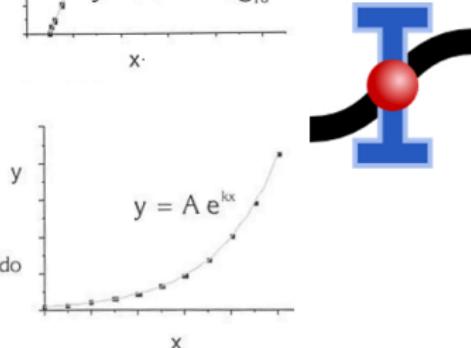
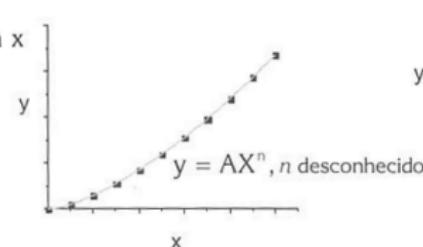
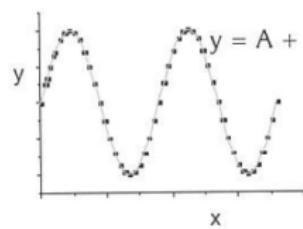
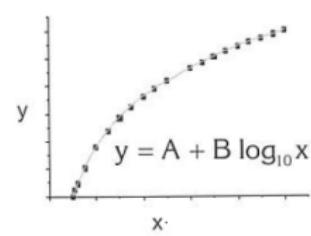
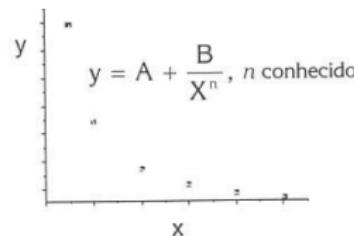
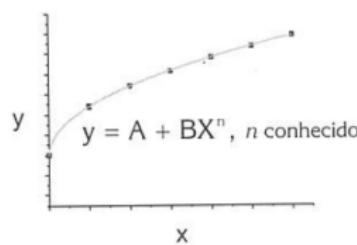
Desta forma, para construir uma reta a partir da equação $y = C_1x^2 + D_1$, basta construir o gráfico $y \times x^2$, como na Figura a seguir:



5.3.3.2. Relembrando: linearização de gráficos

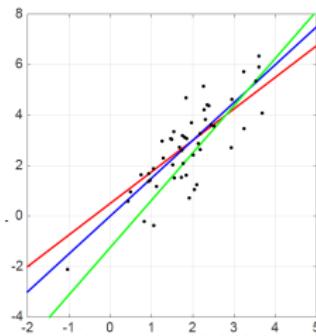
Existem situações nas quais a equação da curva é totalmente desconhecida. Quando isso ocorre, deve-se proceder da seguinte maneira:

- Traçar a curva dos pontos experimentais. Ex: software SciDAVis;
- Analisar a curva obtida e propor uma equação experimental; para isso, é necessário conhecer os tipos de gráficos das equações mais comuns;
- Traçar um novo gráfico com a equação mais adequada e já linearizada.



5.4.1. Regressão linear - mínimos quadrados

Ao se obter uma sucessão de pontos experimentais que, representados em um gráfico, apresentam comportamento linear, poderão ser encontradas diferentes retas, ou seja, diferentes valores para a e b . Qual a melhor reta?



Para responder essa questão, utilizam-se as equações dos mínimos quadrados. Essas equações fornecem os melhores coeficientes linear e angular para a reta:

$$a = \frac{N \sum(XY) - \sum X \sum Y}{N \sum X^2 - (\sum X)^2} \quad b = \frac{\sum Y \sum X^2 - \sum X \sum (XY)}{N \sum X^2 - (\sum X)^2}$$

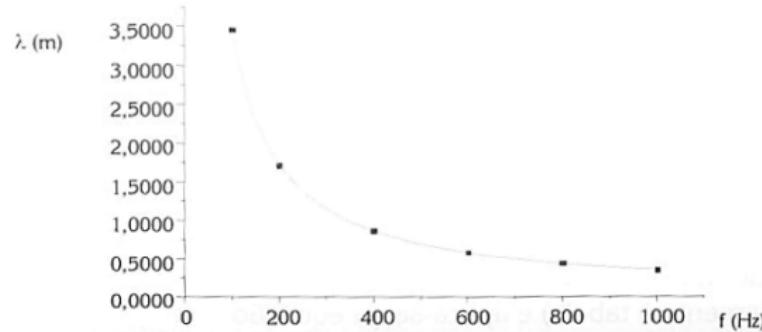
Observação: caso o conjunto de pontos experimentais corresponda a uma equação não linear, deve-se primeiro linearizá-la e, como decorrência, refazer a tabela dos pontos experimentais (antes da utilização das relações anteriores).

5.4.1. Regressão linear - mínimos quadrados

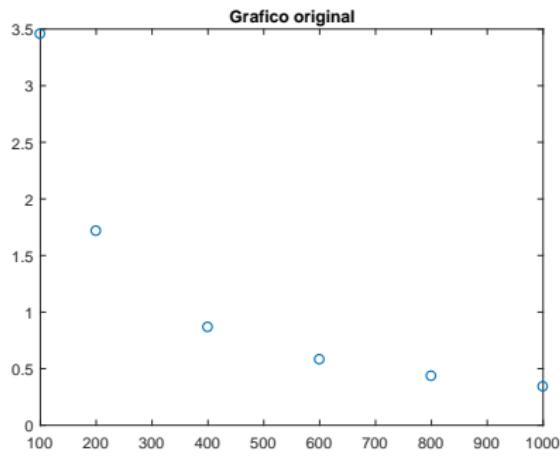
Em uma experiência para a determinação da velocidade do som no ar (v), na qual se mediou o comprimento de onda (λ) em função da frequência (f), foram obtidos os dados da tabela a seguir. A expressão que rege o comportamento é $v = \lambda f$. Isolando λ em função de f , obtém-se: $\lambda = \frac{v}{f}$:

f (Hz)	1000	800	600	400	200	100
λ (m)	0,3405	0,4340	0,5800	0,8655	1,7155	3,4556

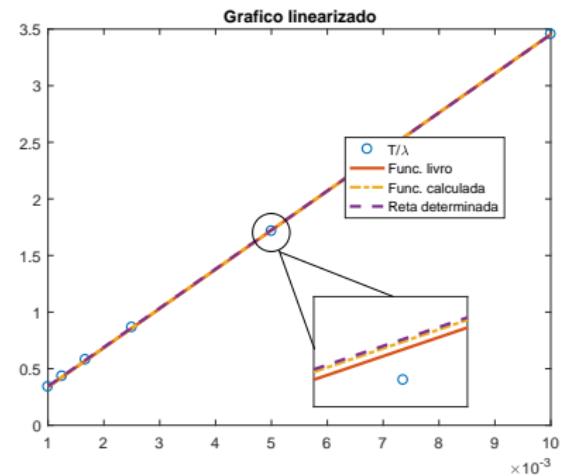
Os pontos da tabela colocados diretamente no papel milimetrado fornecem o seguinte gráfico:



5.4.1. Regressão linear - mínimos quadrados



(c)



(d)

5.4.1.3. Regressão linear - mínimos quadrados

Calculando os termos das equações para a e b :

$$\sum Y = 0,3405 + 0,4340 + 0,5800 + 0,86655 + 1,7155 + 0,34556 = 7,3911$$

$$\sum X = (1 + 1,25 + 1,67 + 2,5 + 5 + 10) \times 10^{-3} = 21,42 \times 10^{-3}$$

$$(\sum X)^2 = 4,588164 \times 10^{-4}$$

$$\begin{aligned} \sum X^2 &= (1 \times 10^{-3})^2 + (1,25 \times 10^{-3})^2 + (1,67 \times 10^{-3})^2 + (2,5 \times 10^{-3})^2 + \\ &\quad + (5 \times 10^{-3})^2 + (10 \times 10^{-3})^2 \times 10^{-6} = 1,366014 \times 10^{-4} \end{aligned}$$

$$\begin{aligned} \sum(XY) &= (1 \times 10^{-3} \times 0,3405) + (1,25 \times 10^{-3} \times 0,4340) + (1,67 \times 10^{-3} \times 0,58) + \\ &\quad + (2,5 \times 10^{-3} \times 0,8655) + (5 \times 10^{-3} \times 1,7155) + (10 \times 10^{-3} \times 3,4556) \\ &= 4,714885 \times 10^{-2} \end{aligned}$$

Com $N = 6$ (número de pontos), tem-se:

$$a = \frac{6 \times 4,414653 \times 10^{-2} - 2,1416 \times 10^{-2} - 7,3911}{6 \times 1,366014 \times 10^{-4} - 4,588164 \times 10^{-4}} = 345,23987$$

$$b = \frac{7,3911 \times 1,366014 \times 10^{-4} - 2,142 \times 10^{-2} \times 4,714855 \times 10^{-2}}{6 \times 1,366014 \times 10^{-4} - 4,588164 \times 10^{-4}} = -8,1420724 \times 10^{-4}$$

5.4.1.2. Regressão linear - mínimos quadrados

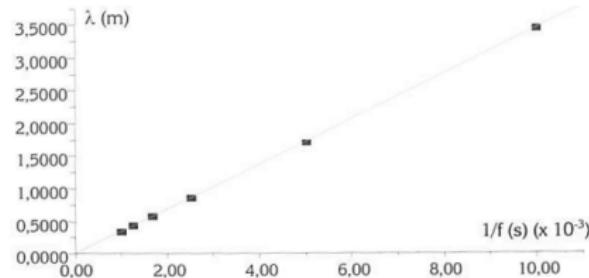
Desta forma, tem-se:

$$\lambda = -0,0008 + \frac{345,2}{f}$$

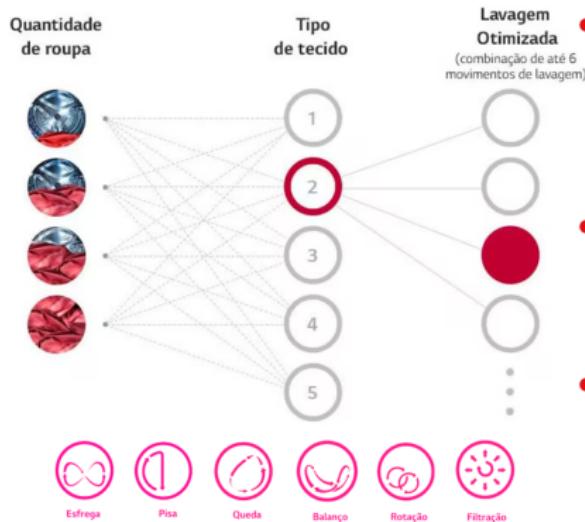
Quando não se faz necessário uma grande precisão, utiliza-se a equação diretamente do gráfico, porém quando uma confiabilidade maior é requerida é fundamental o emprego dos mínimos quadrados;

Convém salientar que durante os cálculos dos somatórios para a aplicação dos mínimos quadrados **não é feito nenhum arredondamento**. Caso fosse utilizado, ocorreria um afastamento da reta em relação aos pontos experimentais;

Uma vez obtido os coeficientes angular e linear da melhor reta, para um determinado conjunto de pontos, é necessário representá-la em um gráfico, juntamente com os pontos experimentais cuja tendência ela representa.



5.5.1. Árvore de decisão



- Consiste em uma representação de uma tabela de decisão em formato de árvore. Ou seja, é uma técnica para tomada de decisão;
- Tal técnica é amplamente utilizada em *machine learning*, porém encontra-se também em áreas como a pesquisa operacional (métodos/técnicas de otimização);
- Por serem modelos de aprendizado supervisionado são empregados em tarefas tanto de regressão quanto classificação;
- O objetivo é criar uma árvore que aprende com os dados por meio de regras básicas: *if-then rules*. Tal modelo identifica qual o melhor atributo para separar os dados utilizando critérios escolhidos que podem ser medidas de informação (estatística), por exemplo.

5.6.1. Redes Neurais Artificiais

Ponto de vista da engenharia:

- Modelos **computacionais inspirados** nos mecanismos de **aprendizagem do cérebro humano**;
- Emulam a forma como o cérebro humano **resolve problemas**;
- Fundamenta-se na **observação** do funcionamento de uma rede neural **biológica**;
- Possuem **capacidade de aquisição e manutenção do conhecimento** (baseado em informações).

Inspiração no cérebro humano:

- O processamento de informações no cérebro humano é altamente **complexo, não-linear e paralelo**;
- O cérebro é constituído por aproximadamente **100bi** de neurônios (10^{11});
- Cada neurônio é interligado em média com outros **6.000** neurônios: 600 trilhões de **sinapses**.

Podem ser definidas como um **conjunto de unidades de processamento** (neurônios artificiais) **interconectadas** por intermédio de um grande número de conexões (sinapses).

5.1.2. Computador × cérebro humano

Tipo de processamento:

- **Computador** → sequencial (serial);
- **Cérebro** → paralelo.

Quantidade e complexidade:

- **Computador** → um ou poucos processadores;
- **Cérebro** → $\approx 10^{11}$ neurônios; $\approx 100^{12}$ conexões (trilhões).

Velocidade de processamento:

- **Computador** → $\approx 10^{-9}$ segundos;
- **Cérebro** → $\approx 10^{-6}$ segundos.

Eficiência energética:

- **Computador** → $\approx 10^{-16}$ Joules (por operação/segundo);
- **Cérebro** → $\approx 10^{-6}$ Joules (por operação/segundo).

5.1.3. Principais características

Adaptação por experiência:

- Parâmetros **internos** da rede são **ajustados** a partir da apresentação sucessiva de **exemplos** (amostras e medidas).

Capacidade de Aprendizado:

- Aplicação de método de treinamento possibilita a rede **extrair** o **relacionamento** existente entre variáveis que compõem a aplicação.

Habilidade de Generalização:

- Após o processo de treinamento, a rede é capaz de generalizar o conhecimento adquirido, possibilitando a **estimação de soluções** que eram até então **desconhecidas**.

Organização de Dados:

- Baseada em características marcantes de um conjunto de dados a respeito do processo, a rede é capaz de realizar sua **organização** interna para **agrupamento** de amostras que são similares/comuns.

5.1.3.1. Principais características

Tolerância a Falhas:

- Devido ao elevado nível de **interconexões** entre neurônios artificiais, a rede torna-se um sistema **tolerante** a falhas quando parte de sua estrutura interna for **sensivelmente** comprometida.

Armazenamento Distribuído:

- O conhecimento dentro da rede é realizado de forma **distribuída** entre as **sinapses** dos neurônios artificiais, permitindo-se então robustez frente a eventuais neurônios que se tornaram **inoperantes**.

Facilidade de Prototipagem:

- Após o processo de treinamento, os seus resultados são **normalmente** obtidos por algumas operações **matemáticas elementares**.

5.2.1. Aproximador de funções e controle de processos

Aproximador de Funções:

- Mapeiam o relacionamento entre variáveis de um sistema a partir de um **conjunto conhecido** de seus valores representativos;
- Envolvem normalmente o mapeamento de processos cuja modelagem por técnicas convencionais são de **difícil obtenção**;
- Exemplo de aplicações: resolução de diversos tipos de problemas (em diferentes áreas do conhecimento).

Controle de Processos:

- Consistem em identificar ações de **controle** que permitam o alcance dos requisitos de **qualidade**, de eficiência e de **segurança** do processo;
- Exemplo de aplicações: controles empregados em robótica, aeronaves, elevadores, eletrodomésticos, satélites, etc.

5.2.2. Classificador de padrões e *clustering*

Classificação de Padrões:

- O objetivo desta aplicação consiste de **associar** um padrão de entrada (amostra) para uma das **classes** previamente **definidas**;
- O problema a ser tratado possui um conjunto **discreto** e **conhecido** das **possíveis** saídas desejadas;
- Exemplos de aplicações: reconhecimento de imagens, voz, escrita, etc.

Clustering:

- O alvo aqui consiste da **identificação** e **detecção de similaridade** se particularidades entre as diversas amostras do processo, objetivando-se **agrupamento** dos mesmos;
- Exemplos de aplicações: identificação automática de classes (como em problemas de diagnóstico médico), compressão de dados e garimpagem de dados (*data mining*).

5.2.3. Previsão e otimização de sistemas

Previsão:

- O objetivo consiste em **estimar** valores futuros de um processo levando-se em consideração diversas **medidas prévias** observadas em seu domínio;
- Exemplos de aplicações: previsão de demanda de energia, previsões de mercados financeiros, previsões climáticas, séries temporais, etc.

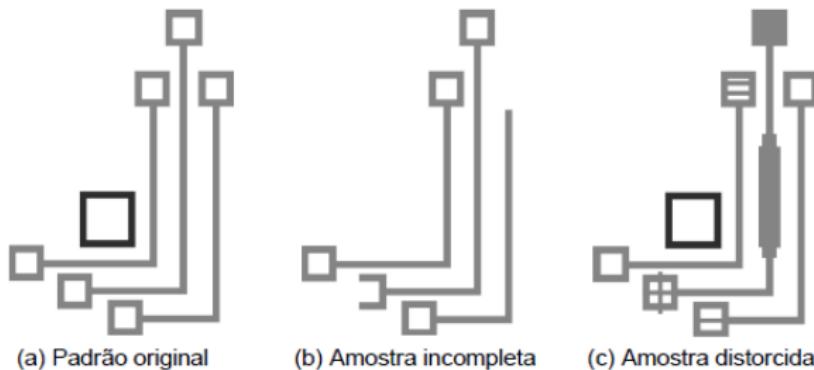
Otimização:

- O alvo consiste em **minimizar** ou **maximizar** uma **função objetivo** (custo), obedecendo-se também eventuais **restrições** que são impostas para o correto mapeamento do problema;
- Exemplos de aplicações: problemas de otimização restrita, otimização combinatorial, programação dinâmica, problemas de sequenciamento de produção, etc.

5.2.4. Memórias associativas

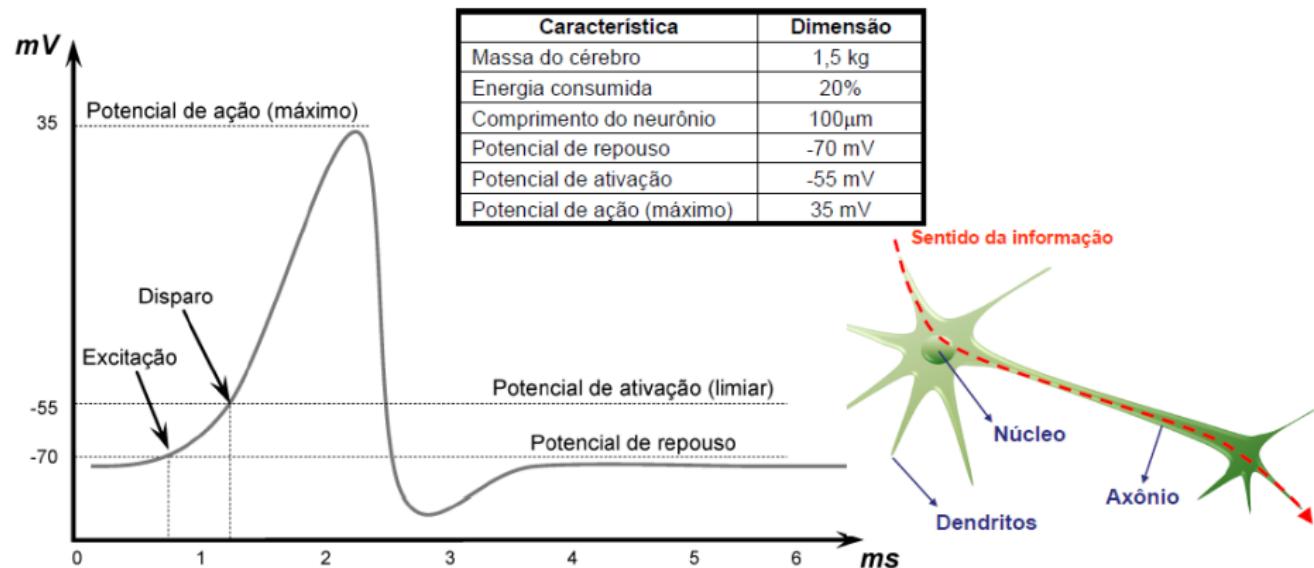
Memórias Associativas:

- A missão consiste em **recuperar** padrões corretos mesmo se os seus elementos constituintes forem apresentados de forma **incompleta** ou **distorcida**;
- Exemplos de aplicações: processamento de imagens, transmissão de sinais, identificação de caracteres manuscritos, etc.



5.3.1. Características elétricas e da célula nervosa

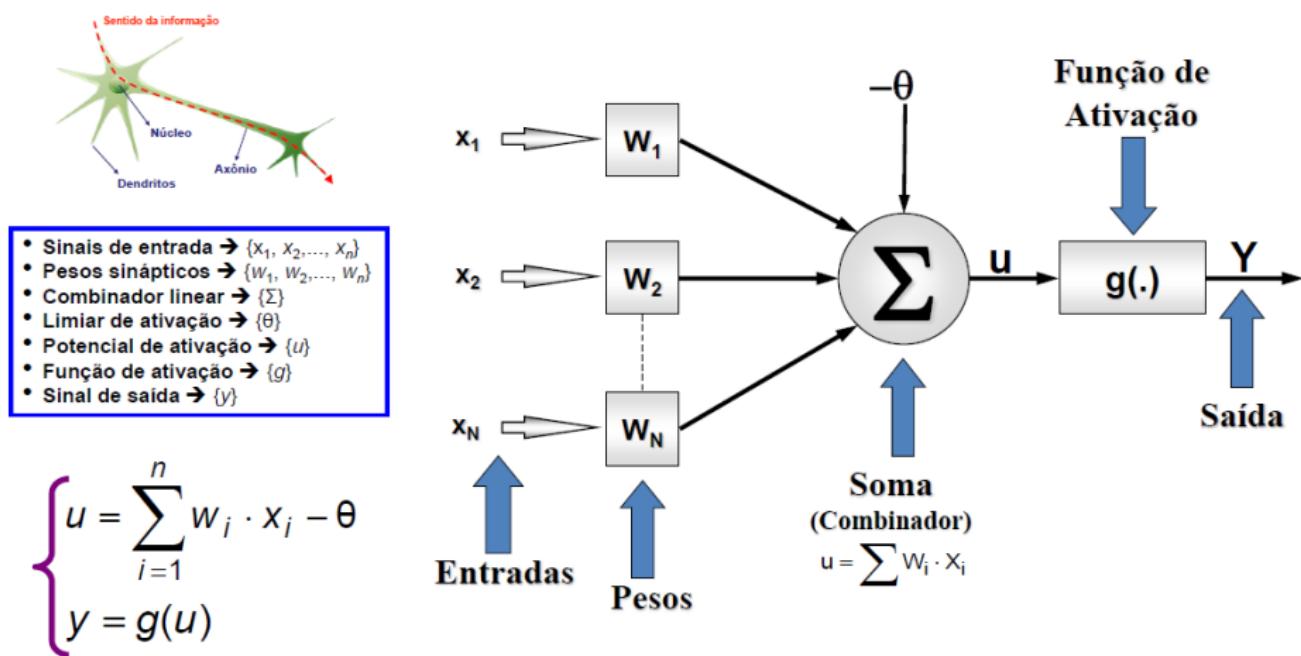
- Etapas de variação (potencial de ação):



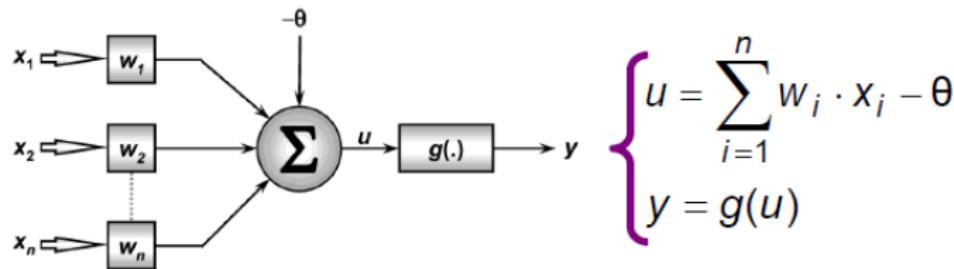
5.4.1. Representação e definições

O neurônio artificial consiste em um modelo proposto a partir dos estudos da geração e propagação de impulsos pelo neurônio biológico.

- O modelo de McCulloch & Pitts (1943) é um dos mais utilizados:



5.4.2. Resumo do funcionamento: obtenção da saída

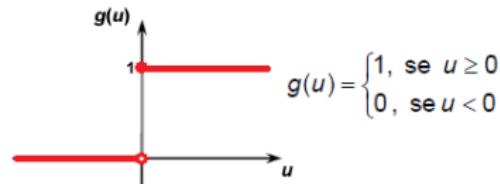


- ① Apresentação de um conjunto de valores que representam as variáveis de entrada do neurônio;
- ② Multiplicação de cada entrada do neurônio pelo seu respectivo peso sináptico;
- ③ Obtenção do potencial de ativação produzido pela soma ponderada dos sinais de entrada, subtraindo-se o limiar de ativação;
- ④ Aplicação de uma função de ativação apropriada, tendo-se como objetivo limitar a saída do neurônio;
- ⑤ Compilação da saída a partir da aplicação da função de ativação neural em relação ao seu potencial de ativação.

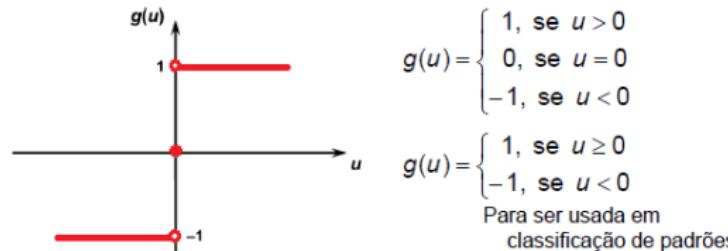
5.5.1. Funções parcialmente diferenciáveis - descontínuas

As **funções de ativação** tem como principal objetivo **limitar a saída do neurônio** de modo que dada uma excitação suficiente, maior que o limiar de ativação, produza uma **saída respectiva a ação do neurônio**.

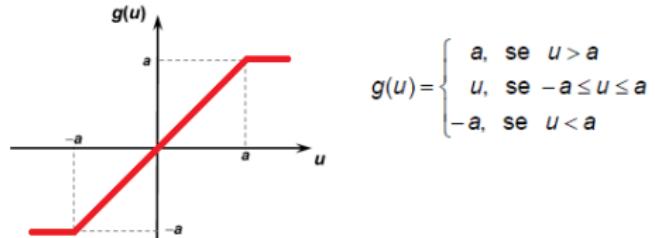
Função Degrau (Heavyside):



Função Degrau Bipolar (Sinal):



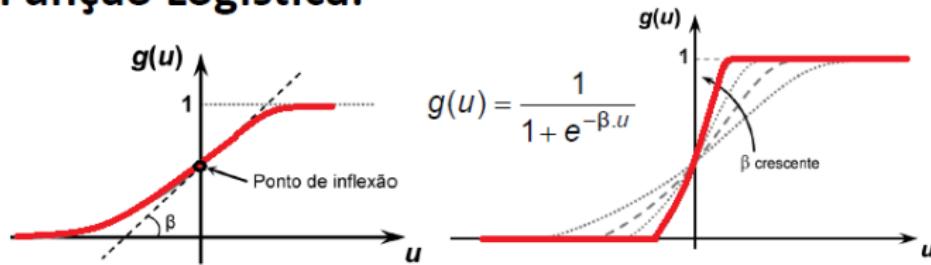
Função Rampa Simétrica:



Os valores retornados são iguais aos próprios valores dos potenciais de ativação quando os mesmos estão definidos no intervalo $[-a, a]$, limitando-se aos valores limites em caso contrário.

5.5.2. Funções totalmente diferenciáveis - contínuas

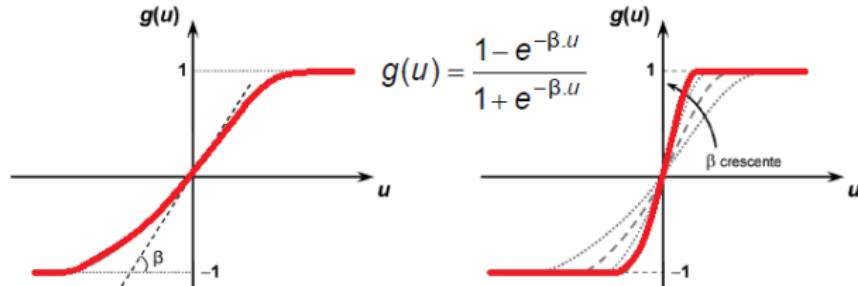
Função Logística:



- Parâmetro β é uma constante real relacionada à inclinação da função logística frente ao seu ponto de inflexão;
- Resultado de saída assumirá sempre valores reais entre 0 e 1;
- Formato geométrico tende a ser similar àquele da função degrau quando β for muito elevado (tender ao infinito).

5.5.2.1. Funções totalmente diferenciáveis - contínuas

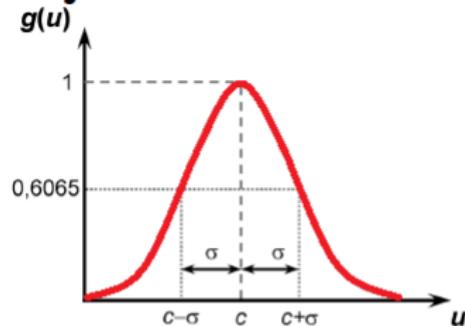
Função Tangente Hiperbólica:



- Parâmetro β está também relacionado à inclinação da função frente ao seu ponto de inflexão;
- Resultado de saída assumirá sempre valores reais entre -1 e 1;
- Formato geométrico tende a ser similar àquele da função degrau quando β for muito elevado (tender ao infinito).

5.5.2.2. Funções totalmente diferenciáveis - contínuas

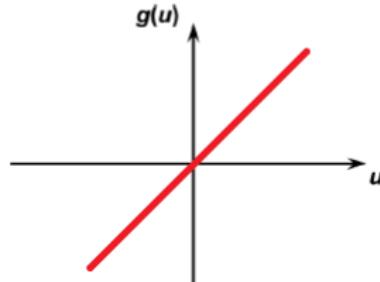
Função Gaussiana:



$$g(u) = e^{-\frac{(u-c)^2}{2\sigma^2}}$$

- Parâmetro c define o centro (média) da função.
- Parâmetro σ^2 especifica a variância (espalhamento).

Função Linear (Identidade):



$$g(u) = u$$

- Produz resultados idênticos aos valores do potencial de ativação $\{u\}$.

5.6.1. Arquitetura, topologia e treinamento

Arquitetura:

- Define a forma como os seus **diversos** neurônios estão **arranjados**, ou **dispostos**, uns em relação aos outros;
- Os arranjos são **essencialmente** estruturados através do direcionamento do **fluxo** sináptico.

Topologia:

- Define as **diferentes** formas de **composições** estruturais que uma rede poderá assumir, frente a uma determinada arquitetura;
- Exemplo: dada uma determinada arquitetura, uma das redes pode ser composta de 10 neurônios e a outra de 20 neurônios.

Treinamento:

- Consiste da aplicação de um conjunto de **passos ordenados** com o intuito de ajustar os **pesos** e os **limiares** de seus neurônios;
- Visa então “**sintonizar**” a rede para que as suas respostas estejam próximas dos valores **desejados**.

5.6.2. Estruturas de camadas

Camada de Entrada:

- Responsável pelo **recebimento** de informações (dados), sinais, características ou medições advindas do meio externo;
- Valores recebidos são geralmente **normalizados** em relação aos valores **limites** produzidos pelas funções de **ativação**.

Camadas Ocultas, Escondidas ou Intermediárias:

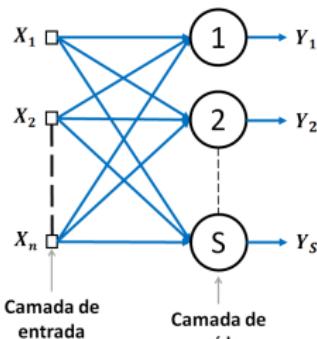
- São aquelas camadas **compostas de neurônio** se que possuem a responsabilidade de **extrair** as características associadas ao **processo** ou **sistema** a ser inferido.

Camada de Saída:

- Esta camada é também constituída de neurônios, sendo responsável por **produzir/apresentar** os resultados finais da rede.

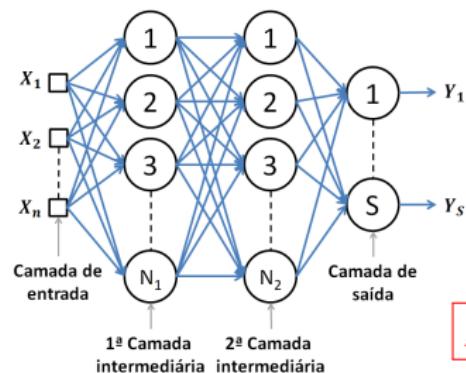
5.6.3. Arquitetura *FeedForward*

Camada simples:



- Constituída de uma camada de entrada e única camada de neurônios, que é a própria camada de saída;
- Fluxo de informações segue sempre numa única direção (unidirecional);
- **Exemplos:** Perceptron simples e Adaline;
- **Aplicações:** classificação de padrões, filtragem.

Camada múltiplas:

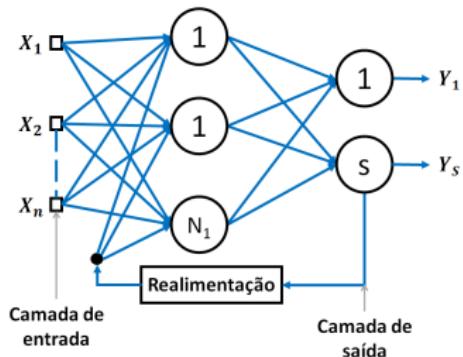


- Presença de uma ou mais camadas neurais escondidas (camadas intermediárias);
- Quantidade de camadas escondidas e de neurônios dependem, sobretudo, do tipo e complexidade do problema a ser mapeado;
- **Exemplos:** Perceptron multicamadas (PMC), redes de base radial (RBF);
- **Aplicações:** aproximação de funções, classificação de padrões, controle de processos, otimização, etc.

FeedForward: utilizadas em 80% das aplicações práticas!

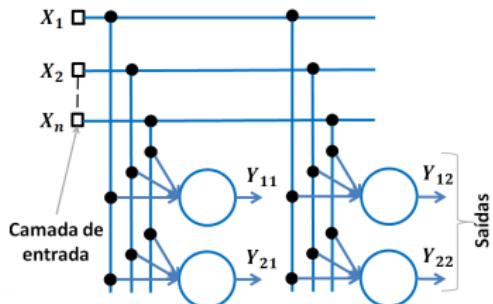
5.6.4. Arquitetura recorrente e reticulada

Arquitetura recorrente:



- Saídas dos neurônios são realimentadas como sinais de entrada para outros neurônios;
- Realimentação as qualificam para processamento dinâmico de informações (*feedback*);
- **Exemplos:** *Hopfield*, *Perceptron* multicamadas com realimentação;
- **Aplicações:** sistemas de previsão, otimização, controle de processos, etc..

Arquitetura reticulada:



- Levam em consideração a forma em que a disposição espacial dos neurônios está organizada;
- Visa propósitos de extração de características;
- Normalmente são redes auto-organizáveis;
- **Exemplos:** *Kohonen*;
- **Aplicações:** problemas de agrupamento, classificação de padrões, otimização de sistemas, etc..

5.7.1. Aspectos de treinamento/teste

Finalidade do Treinamento:

- Consiste da aplicação de **passos** ordenados a fim de **ajustar** (sintonizar) os parâmetros livres (pesos sinápticos e limiares) dos neurônios;
- Após treinamento, a rede está apta para **generalizar soluções** (as quais não eram conhecidas).
 - Uma rede neural **treinada adequadamente** possui **capacidade de generalização**. O que significa que a rede será capaz de produzir saídas próximas as esperadas, a partir de quaisquer sinais inseridos em suas entradas.

Conjuntos de Treinamento/Teste:

- **Conjunto Total de Amostras** → Representa todos os dados disponíveis sobre o comportamento do processo a ser mapeado. Será **dividido em subconjuntos** de **treinamento** e de **teste**:
 - **Subconjunto de Treinamento** → Usado essencialmente para o processo de aprendizado da rede. Composto aleatoriamente com cerca de 60 a 90% das amostras do conjunto total;
 - **Subconjunto de Teste** → Usado para verificar se a generalização de soluções por parte da rede já estão em patamares aceitáveis. Composto de 10 a 40% das amostras do conjunto total.

5.7.2. Pré e Pós-processamento

Pré-processamento:

- Sempre que possível, é importante atentar para que os casos críticos **máximos e mínimos**, estejam no **conjunto de treinamento** a fim de possibilitar o melhor aprendizado da rede (**normalização**);
- Deve-se atentar também para que **todo espaço amostral** esteja **adequadamente representado** no conjunto de treinamento;
- Muitas vezes opta-se por realizar diversos tipos de pré-processamento nos dados de entrada da rede a fim de eliminar não linearidades, e grandes diferenças de magnitude entre as variáveis (escalonamento/*outliers*).

Pós Processamento:

- Uma vez realizado o **pré-processamento (normalização)** para que a rede possa avaliar a dispersão dos dados no espaço amostral e executar com precisão sua função, algumas vezes faz-se necessário o **pós-processamento (desnormalização)**, que seria o retorno dos dados aos originais. Como por exemplo em redes que realizam a classificação de padrões, para o processamento, os dados da classe A e classe B são identificados como 1 e -1 respectivamente. Após o processamento, os dados precisam ser reconvertidos em A e B novamente.

5.7.3. Treinamento supervisionado

Treinamento Supervisionado:

- Consiste em se ter **disponível**, considerando cada **amostra** dos sinais de entrada, as respectivas **saídas desejadas**;
- Cada amostra de treinamento é então **composta pelos sinais de entradas e suas correspondentes saída**;
- Comporta como se houvesse um “professor” **ensinando** para a rede qual seria a resposta **correta** para cada amostra apresentada.

Conjuntos de Passos:

- ① Apresente uma amostra de treinamento;
- ② Calcule a saída produzida pela rede;
- ③ Compare com a saída desejada:
 - A. Então → Termine o processo de aprendizado;
 - B. Senão → Ajuste os pesos sinápticos e limiares e volte ao passo 1.

5.7.4. Treinamento não-supervisionado

Treinamento Não-Supervisionado:

- Diferentemente do supervisionado, as respectivas saídas desejadas **são inexistentes**;
- A rede deve se **auto-organizar** em relação às **particularidades** existentes entre os elementos do conjunto total de amostras, identificando subconjuntos (*clusters*) que contenham **similaridades**.

Conjuntos de Passos:

- ① Apresente todas as amostras de treinamento;
- ② Obtenha as características que marcam as amostras de treinamento;
- ③ Agrupe todas as amostras com características em comum;
- ④ Coloque as amostras comuns em classes.

5.7.5. Treinamento *online* e *off-line*

Off-line:

- Os ajustes efetuados nos pesos e **limiares** dos neurônios só são **efetivados** após a **apresentação de todo** o conjunto de treinamento;
- Cada passo de ajuste leva em consideração o **total** de desvios observados nas amostras de treinamento frente aos respectivos valores desejados para as suas saídas.

Online:

- Ao contrário da *off-line*, os ajustes nos pesos e limiares dos neurônios são efetuados após a **apresentação de cada** amostra de treinamento;
- É normalmente utilizada quando o **comportamento** do sistema a ser **mapeado** varia de forma bastante **rápida**.

5.8.1. Quando devemos utilizar uma RNA?

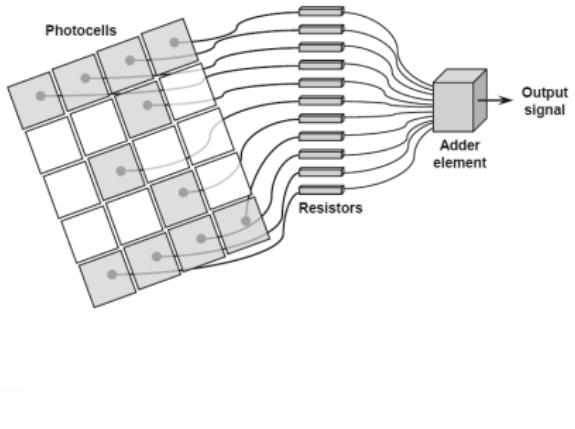
- A tecnologia computacional atual se mostra inadequada?
- O problema depende de múltiplos parâmetros interdependentes?
- A solução do problema é derivada a partir de parâmetros interdependentes que não possuem quantificação precisa?
- Os dados estão prontamente disponíveis, no entanto, estão intrinsecamente sujeitos a ruídos e erros de diversas origens?
- Alguns dados podem estar corrompidos ou perdidos?
- O fenômeno é tão complexo que outras abordagens se mostram ineficazes, muito complicadas ou excessivamente caras?
- Existe uma quantidade de dados derivados de exemplos específicos, suficientemente grande, para modelar o problema em questão?
- O tempo de desenvolvimento de projeto é muito curto, no entanto, suficiente para o treinamento de uma rede neural?

Não recomendado quando:

- O problema pode ser resolvido utilizando uma metodologia computacional convencional;
- Uma altíssima precisão é requerida;
- Rigor matemático e provas dedutivas são necessárias nas análises.

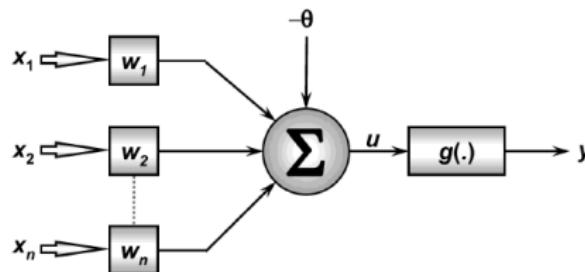
5.9.1. Histórico: *Perceptron*

- Idealizada por Rosenblatt (1958) → forma mais **simples** de uma RNA;
- Possui **uma** camada → **um neurônio**;
- Propósito inicial era implementar um **modelo** computacional inspirado na retina → identificar padrões geométricos.



- ① Sinais elétricos advindos de fotocélulas mapeando padrões geométricos eram **ponderados** por resistores **sintonizáveis**;
- ② Os resistores eram **ajustados** durante o processo de treinamento;
- ③ Um **somador** efetuava a composição de todos os sinais;
- ④ Em consequência, a *Perceptron* poderia **reconhecer** diversos padrões geométricos, tais como letras e números.

5.10.1. Aspectos topológicos

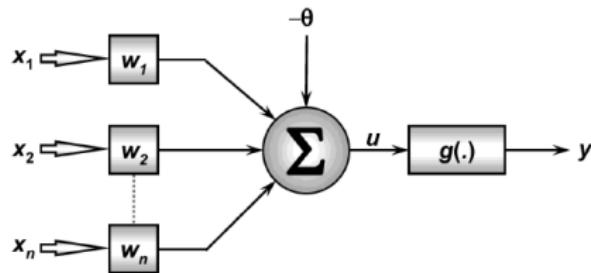


- Sinais de entrada → $\{x_1, x_2, \dots, x_n\}$
- Pesos sinápticos → $\{w_1, w_2, \dots, w_n\}$
- Combinador linear → $\{\Sigma\}$
- Limiar de ativação → $\{\theta\}$
- Potencial de ativação → $\{u\}$
- Função de ativação → $\{g\}$
- Sinal de saída → $\{y\}$

- ① Embora seja uma rede **simples**, a *Perceptron* teve potencial de **atrair**, devido sua proposição, diversos pesquisadores que aspiravam investigar essa **promissora** área de pesquisa;
- ② Recebeu ainda **especial** atenção da comunidade **científica** que também trabalhava com **inteligência computacional**;
- ③ A *Perceptron* é **tipicamente** utilizada em problemas de “**Classificação de Padrões**”.

A *Perceptron* é um classificador usado para classes **linearmente separáveis**

5.11.1. Passos para obtenção da resposta

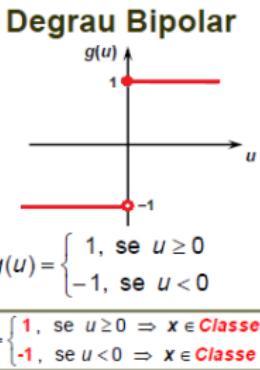
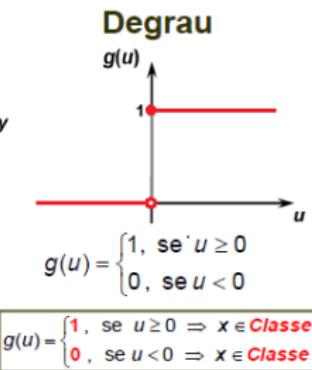
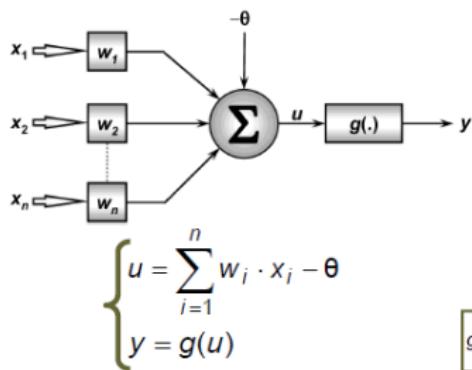


- ① Apresentar o conjunto de entrada;
- ② Multiplicar cada entrada pelo seu respetivo peso sináptico;
- ③ Obter o potencial de ativação subtraindo-se o limiar de ativação;
- ④ Limitar a saída do neurônio com uma função de ativação apropriada;
- ⑤ Compilação da saída a partir da aplicação da função de ativação neural em relação ao seu potencial de ativação.

$$\begin{cases} u = \sum_{i=1}^n w_i \cdot x_i - \theta \\ y = g(u) \end{cases}$$

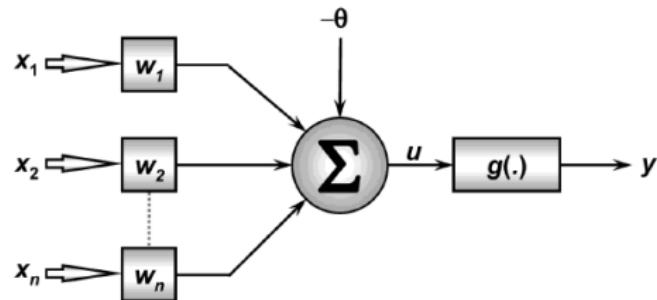
5.11.2. Princípio de funcionamento: classificador de padrões

- A rede perceptron recebe um vetor de **variáveis de entrada**, ponderados pelos **pesos sinápticos**, quantifica-se a importância de cada entrada frente ao objetivo funcional pretendido ao neurônio;
- A **composição** (soma) das entradas ponderadas é **descontada do limiar de ativação** e então repassada à **função de ativação** que irá gerar a **saída**;
- Tipicamente as funções de ativação utilizadas na rede perceptron são a função **degrau** ou **degrau bipolar** para fins de **classificação de padrões**;
- Desta forma, tem-se apenas “duas possibilidades” de valores a serem produzidos pela sua saída: Degrau $\rightarrow 0$ ou 1 e Degrau bipolar $\rightarrow -1$ ou 1 .



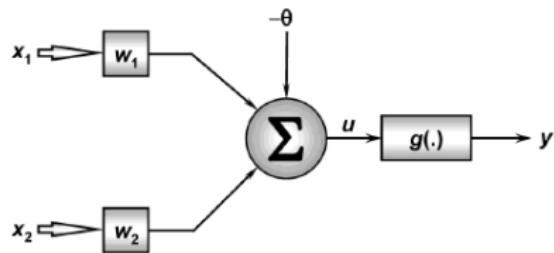
5.11.3. Parâmetros característicos

Parâmetro	Variável Representativa	Tipo Característico
Entradas	x_i (i -ésima entrada)	Reais ou Binária
Pesos Sinápticos	w_i (associado a x_i)	Reais (iniciados aleatoriamente)
Limiar	θ	Real (iniciado aleatoriamente)
Saída	y	Binária
Função de Ativação	$g(\cdot)$	Degrau ou Degrau Bipolar
Processo de Treinamento	-----	Supervisionado
Regra de Aprendizado	-----	Regra de Hebb

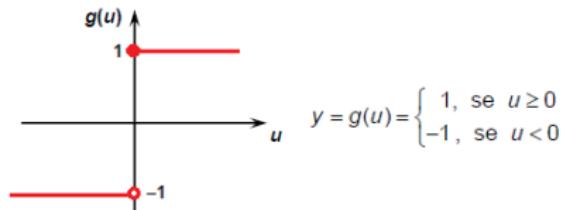


- Sinais de entrada → $\{x_1, x_2, \dots, x_n\}$
- Pesos sinápticos → $\{w_1, w_2, \dots, w_n\}$
- Combinador linear → $\{\Sigma\}$
- Limiar de ativação → $\{\theta\}$
- Potencial de ativação → $\{u\}$
- Função de ativação → $\{g\}$
- Sinal de saída → $\{y\}$

5.11.4. Análise matemática: exemplo com duas entradas



$$\begin{cases} u = \sum_{i=1}^{n=2} w_i \cdot x_i - \theta \Rightarrow w_1 \cdot x_1 + w_2 \cdot x_2 - \theta \\ y = g(u) \end{cases}$$



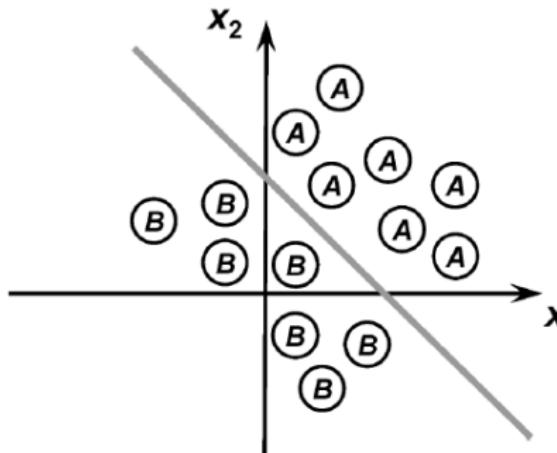
- Usando como ativação a função **degrau**, a saída da *Perceptron* será dada por:

$$y = g(u) = \begin{cases} 1, & \text{se } \sum w_i \cdot x_i - \theta \geq 0 \Leftrightarrow w_1 \cdot x_1 + w_2 \cdot x_2 - \theta \geq 0 \\ -1, & \text{se } \sum w_i \cdot x_i - \theta < 0 \Leftrightarrow w_1 \cdot x_1 + w_2 \cdot x_2 - \theta < 0 \end{cases}$$

- A *Perceptron* é um classificador **linear** que pode ser usado para classificar duas classes **linearmente separáveis!**;
- É possível observar que as **desigualdades** são representadas por uma expressão de **primeiro grau** (linear);
- A **fronteira** de decisão para esta instância (*Perceptron* de duas entradas) será então uma reta cuja equação é definida por:

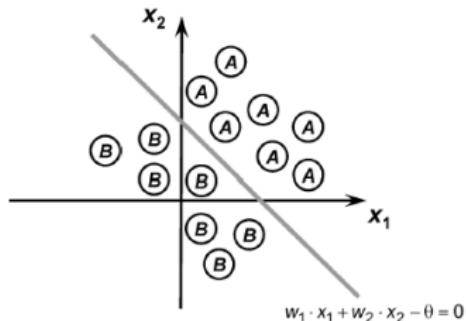
Reta de separabilidade: $W_1 \cdot x_1 + W_2 \cdot x_2 - \theta = 0$

5.11.4.1. Análise matemática: exemplo com duas entradas



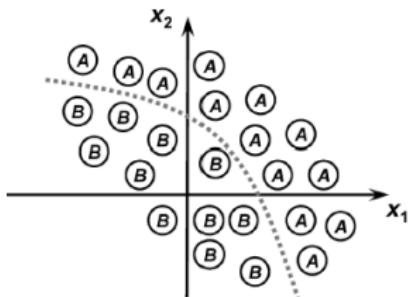
- Em suma, para a circunstância ao lado, a *Perceptron* consegue então dividir **duas classes linearmente separáveis**;
- Se a saída do mesmo for 1 significa que os padrões (Classe A) estão localizados acima da fronteira (reta) de separação; caso contrário, quando a saída for -1 indica que os padrões (Classe B) estão abaixo desta fronteira.

5.11.5. Classificador de padrões: hiperplanos



Problema linearmente separável

- Se a *Perceptron* fosse constituída de **três** entradas (três dimensões), a fronteira de separação seria representada por um **plano**;
- Se a *Perceptron* fosse constituído de quatro ou mais entradas, suas fronteiras seriam **hiperplanos**;
- Conclusão: A condição necessária para que a *Perceptron* de camada simples possa ser utilizado como um classificador de padrões é que as classes do problema a ser mapeado sejam **linearmente separáveis**.



Problema não-linearmente separável

5.12.1. Treinamento e regra de Hebb

- O limiar de ativação (θ) será considerado como um peso sináptico (W_0) a ser ajustado durante o processo:
 - No **início do treinamento**, geralmente, todos os **pesos sinápticos** são inicializados aleatoriamente com **valores pequenos**.
- Comumente a **regra de Hebb** (proposta por Donald Hebb em 1949 → atualização dos pesos sinápticos) é utilizada no **treinamento supervisionado** da rede *Perceptron* para fins de **classificação** de padrões.

$$w_i^{novo} = w_i^{antigo} + \eta(d_k - y_k)x_{i(k)}, i = 0, \dots, n$$

↓ ↓ ↓ ↓
 TAXA DE APRENDIZAGEM RESPOSTA DESEJADA PARA A AMOSTRA K (SUPERVISÃO) SAÍDA DA REDE PARA A AMOSTRA K ELEMENTO I DO VETOR DE ENTRADAS DA AMOSTRA K
 $0 < \eta < 1$

$$W^{novo} = W^{antigo} + \eta(d_k - y_k)X_k$$

A escolha de η deve ser realizada com cautela para evitar a instabilidade do processo de treinamento.

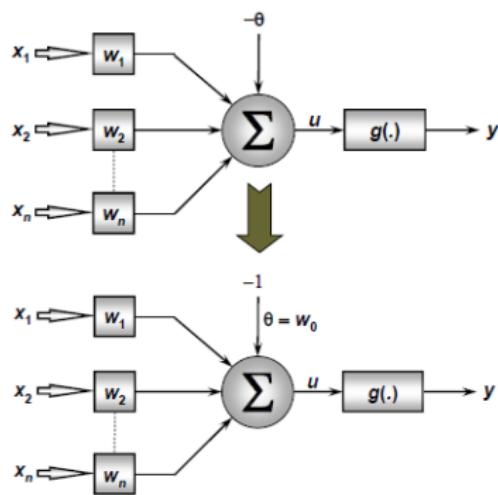
5.12.1.1. Treinamento e regra de Hebb

$$\begin{cases} W_i^{\text{atual}} = W_i^{\text{anterior}} + \eta \cdot (d^{(k)} - y) \cdot \mathbf{x}^{(k)} \\ \theta_i^{\text{atual}} = \theta_i^{\text{anterior}} + \eta \cdot (d^{(k)} - y) \end{cases}$$

- A **regra de Hebb** conduz a rede *perceptron* ao seguinte comportamento:
 - Incrementar os pesos sinápticos sempre que há **concordância** entre a resposta da rede e a saída desejada (**ajuste excitatório**);
 - Decrementar os pesos sinápticos sempre que há **discordância** entre a resposta da rede e a saída desejada (**ajuste inibitório**).
- Este processo é repetido **sequencialmente**, para **todas as amostras de treinamento**, até que a saída produzida pela *Perceptron* seja similar à saída **desejada** de cada amostra.

- W_i são os pesos sinápticos.
- θ é limiar do neurônio.
- $\mathbf{x}^{(k)}$ é o vetor contendo a k -ésima amostra de treinamento
- $d^{(k)}$ é saída desejada para a k -ésima amostra de treinamento.
- y é a saída do *Perceptron*.
- η é a taxa de aprendizagem da rede.

5.12.2. Implementação computacional



- Torna-se mais conveniente tratar as expressões anteriores em sua forma **vetorial**;
- Como a mesma regra de ajuste é aplicada tanto para os pesos W_i como para o limiar θ , pode-se então inserir θ dentro do vetor de pesos:

$$W = [\theta \quad W_1 \quad W_2 \quad W_3 \quad \dots \quad W_N]^T$$

- De fato, o valor do **limiar** é também uma variável a ser **ajustada** a fim de se realizar o treinamento da *Perceptron*.

$$\begin{cases} W_i^{atual} = W_i^{anterior} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)} \\ \theta_i^{atual} = \theta_i^{anterior} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)} \end{cases}$$

notação vetorial

$$W^{atual} = W^{anterior} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)}$$

notação algorítmica

$$w \leftarrow w + \eta \cdot (d^{(k)} - y) \cdot x^{(k)}$$

$$x^{(k)} = [-1 \quad x_1^{(k)} \quad x_2^{(k)} \cdots x_n^{(k)}]^T$$

5.12.3. Montagem dos conjuntos de treinamento

- Problema a ser mapeado pela *Perceptron* com três entradas: $\{x_1, x_2, x_3\}$;
- Assume-se que se tem quatro amostras com os seguintes valores de entrada:
 - Amostra 1 → Entrada: $[0, 1 \ 0, 4 \ 0, 7]^T$ → Saída desejada: [1];
 - Amostra 2 → Entrada: $[0, 3 \ 0, 7 \ 0, 2]^T$ → Saída desejada: [0];
 - Amostra 3 → Entrada: $[0, 6 \ 0, 9 \ 0, 8]^T$ → Saída desejada: [0];
 - Amostra 4 → Entrada: $[0, 5 \ 0, 7 \ 0, 1]^T$ → Saída desejada: [1];

Então, pode-se converter tais sinais para que os mesmos possam ser usados no treinamento da *Perceptron*:

forma
matricial

$$\Omega^{(x)} = \begin{matrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{matrix} \left[\begin{array}{cccc} x^{(1)} & x^{(2)} & x^{(3)} & x^{(4)} \\ \hline -1 & -1 & -1 & -1 \\ 0,1 & 0,3 & 0,6 & 0,5 \\ 0,4 & 0,7 & 0,9 & 0,7 \\ 0,7 & 0,2 & 0,8 & 0,1 \end{array} \right];$$

$$\Omega^{(d)} = \begin{bmatrix} d^{(1)} \\ d^{(2)} \\ d^{(3)} \\ d^{(4)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

5.12.4. Algoritmo de aprendizagem

Início {Algoritmo Perceptron – Fase de Treinamento}

- <1> Obter o conjunto de amostras de treinamento $\{x^{(k)}\}$;
 - <2> Associar a saída desejada $\{d^{(k)}\}$ para cada amostra obtida;
 - <3> Iniciar o vetor w com valores aleatórios pequenos;
 - <4> Especificar a taxa de aprendizagem $\{\eta\}$;
 - <5> Iniciar o contador de número de épocas $\{\text{época} \leftarrow 0\}$;
 - <6> Repetir as instruções:
- <6.1> $\text{erro} \leftarrow \text{"inexiste"}$;
 - <6.2> Para todas as amostras de treinamento $\{x^{(k)}, d^{(k)}\}$, fazer:
 - <6.2.1> $u \leftarrow w^T \cdot x^{(k)}$;
 - <6.2.2> $y \leftarrow \text{sinal}(u)$;
 - <6.2.3> Se $y \neq d^{(k)}$
- <6.2.3.1> Então $\begin{cases} w \leftarrow w + \eta \cdot (d^{(k)} - y) \cdot x^{(k)} \\ \text{erro} \leftarrow \text{"existe"} \end{cases}$
- <6.3> $\text{época} \leftarrow \text{época} + 1$;
- Até que: $\text{erro} \leftarrow \text{"inexiste"}$

Fim {Algoritmo Perceptron – Fase de Treinamento}

Época: É cada **apresentação completa** de todas as amostras pertencentes ao subconjunto de treinamento, visando, sobretudo, o ajuste dos pesos **sinápticos** e limiares de seus neurônios.

5.12.4.1. Algoritmo de aprendizagem

Critérios de parada para o treinamento:

- Como pode ser observado a partir do algoritmo anterior, o **critério de convergência** (parada) do treinamento é a **inexistência de discrepâncias** entre a saída produzida pela rede e aquela desejada pelo projetista;
- No entanto, em casos onde a rede não possa realizar a classificação, o algoritmo irá entrar em *loop* infinito. Assim, geralmente adota-se um **número máximo de épocas** como critério geral de **divergência** do treinamento.

5.13.1. Algoritmo de operação

Início {Algoritmo Perceptron – Fase de Operação}

- {<1> Obter uma amostra a ser classificada { x };
 <2> Utilizar o vetor w ajustado durante o treinamento;
 <3> Executar as seguintes instruções:
 <3.1> $u \leftarrow w^T \cdot x$;
 <3.2> $y \leftarrow \text{sinal}(u)$;
 <3.3> Se $y = -1$
 <3.3.1> Então: amostra $x \in \{\text{Classe A}\}$
 <3.4> Se $y = 1$
 <3.4.1> Então: amostra $x \in \{\text{Classe B}\}$

Fim {Algoritmo Perceptron – Fase de Operação}

- A “fase de operação” é usada somente **após** a fase de treinamento, pois aqui a rede já está **apta** para ser usada no **processo**;
- Então a etapa de **classificação** é realizada na fase de operação, quando novas **amostras** são apresentadas em suas entradas.

Vamos compreender melhor como funcionam os cálculos passo a passo?

5.13.2. Exemplo *Perceptron*

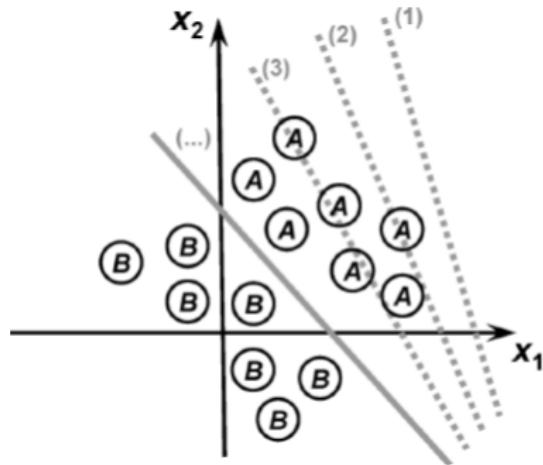
```

1 import numpy as np
2
3 M = np.array([[0.1,0.4,0.7,1],[0.3,0.7,0.2,0],[0.6,0.9,0.8,0],[0.5,0.7,0.1,1]])
4
5 d = (M[:,1:2])
6
7 #Organizando Matriz dados
8 T = np.zeros((M.shape[0], M.shape[1]))
9 T[:,1:] = M[:,1:-1]
10 T[:,0:1] = np.array([(1-np.ones(M.shape[0])).tolist()]).T
11
12 #Peso sinaptico aleatorio
13 w = np.zeros(M.shape[0])
14 w[0] = -1
15 w[1:4] = np.random.rand(3)
16 wi = w; # peso inicial
17
18 print(f'w inicial: ' + str(wi))
19
20 n = 0.5 #estipulando valor de neta taxa de aprendizado
21 epoca = 0 #iniciando o contador de épocas
22
23 def FuncDegrau(q):
24     g = np.zeros(len(q))
25     for k in range(len(q)):
26         if q[k] >= 0:
27             g[k] = 1
28         else:
29             g[k] = 0
30     return g
31
32
33 #Início da fase de treinamento
34 erro = True
35 while erro == True:
36
37     erro = False
38
39     epoca = epoca+1
40
41     for k in range(M.shape[0]):
42         x = T[:,k:k+1]
43         u = w @ x
44
45         y = FuncDegrau(u)
46
47         if y != d[k]:
48
49             w = w + (n*(d[k]-y)*x).T
50
51             erro = True
52     print('Época: ' + str(epoca) + ' -> W: ' + str(w))
53
54     if epoca == 1000: #limite das épocas
55         break #Força a saída do laço
56
57     print('w final: ' + str(w))
58     print('Época para convergência: ' + str(epoca))
59
60     u = w @ T # u será o vetor --> "w - aleatorios" transposto
61     #vezes a linha desse vetor 'x'(treinamento)
62
63     out = np.zeros(u.shape[1])
64     print('Saída pré-rede: ' + str(u))
65     for k in range(u.shape[1]):
66         out[k] = FuncDegrau(u[:,k]) # Aplicação da função bipolar (-1 1)
67
68     print('Resposta Rede ***** Desejado')
69     print([str(out) + '*****' + str(d.tolist())])

```

5.14.1. Ilustração da convergência: Perceptron

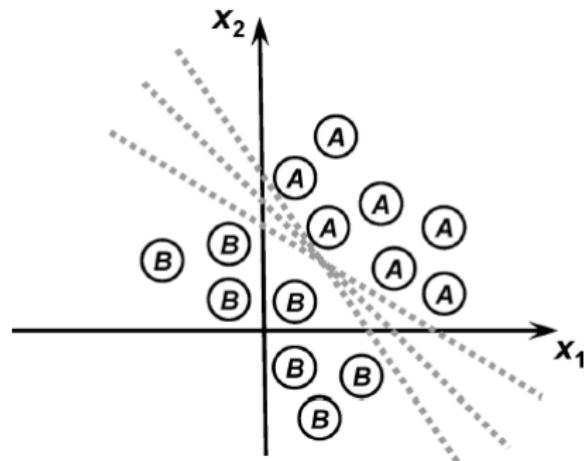
- Processo de treinamento tende a **mover continuamente** o hiperplano de classificação até que seja **alcançada** uma fronteira de **separação** que permite dividir as duas classes;
- Como exemplo, seja uma rede composta de apenas duas entradas $\{x_1 \text{ e } x_2\}$.



- Após a primeira época de treinamento (1), constata-se que o hiperplano está ainda bem longínquo da fronteira de separabilidade das classes.
 - A distância tende ir cada vez mais decrescendo na medida em que se transcorre as épocas de treinamento.
 - Quando o *Perceptron* já estiver convergido, isto estará então significando que tal fronteira foi finalmente alcançada.

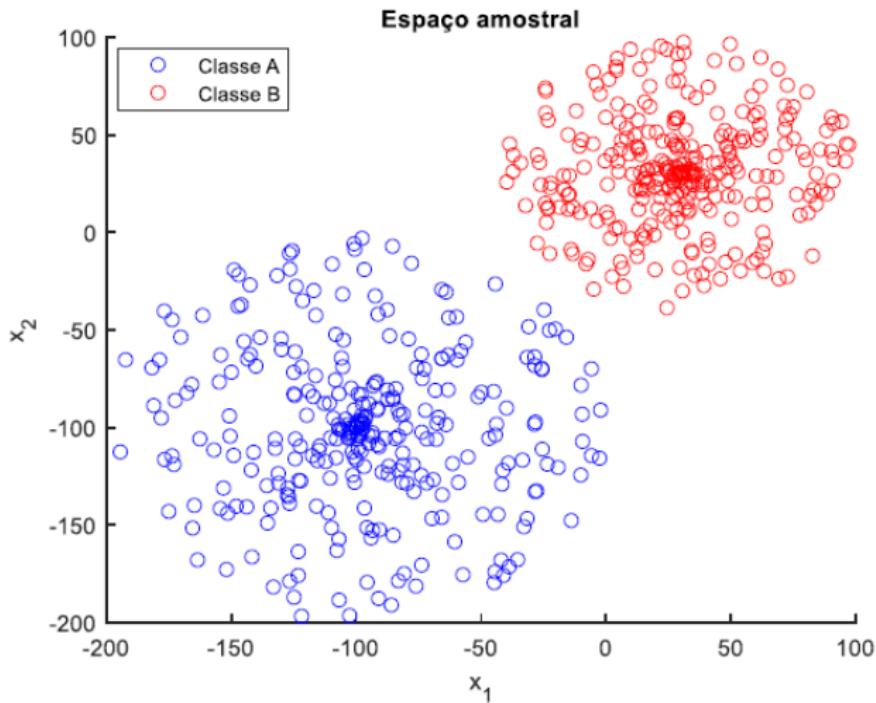
5.14.1.1. Ilustração da convergência: *Perceptron*

- A reta de **separabilidade** a ser produzida após o treinamento do Perceptron **não é única**;
- O número de épocas pode também **variar** de treinamento para treinamento;



5.15.1. Teste de classificação

Agora, vamos utilizar a rede *perceptron* para extrair o padrão de classificação destas classes de dados e inferir detalhes sobre seu processo de treinamento.



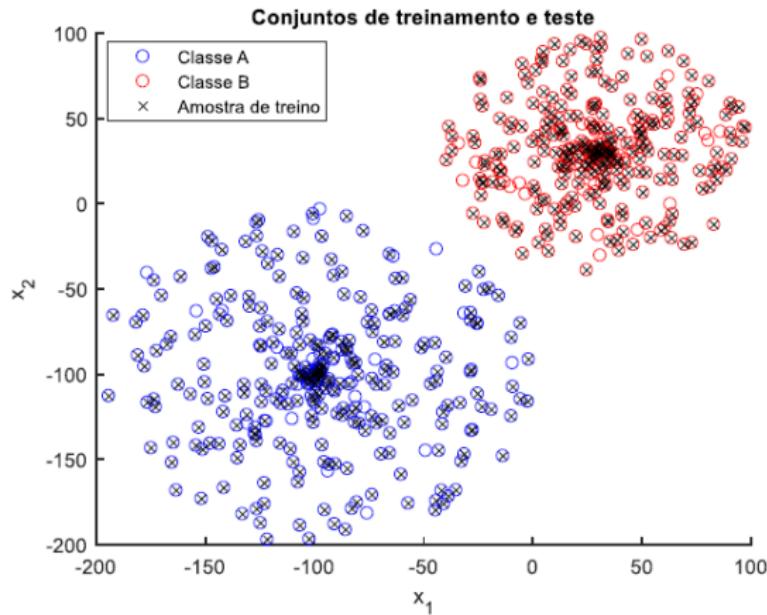
5.15.1.1. Teste de classificação

Cuidados na implementação:

- Lembre-se que o **limiar de ativação** (θ) foi **incluído como um peso sináptico** (W_0) a fim de ser determinado conjuntamente com as outras ponderações da rede durante o treinamento;
- Assim, a **primeira entrada** (x_0) deve receber um **valor igual a -1** para que o **somador realize a subtração** desta entrada, de modo que comumente o conjunto de amostras de uma rede é expresso na forma:

$$\begin{array}{c}
 \text{amostras} = \\
 \left[\begin{array}{c}
 \text{ENTRADAS} \\
 x_0 \quad -1 \\
 x_1 \quad \text{VALORES DAS VARIÁVEIS} \\
 \vdots \\
 x_n
 \end{array} \right] \quad \text{REALIZAÇÕES (MEDIÇÕES)}
 \end{array}$$

5.15.1.2. Teste de classificação



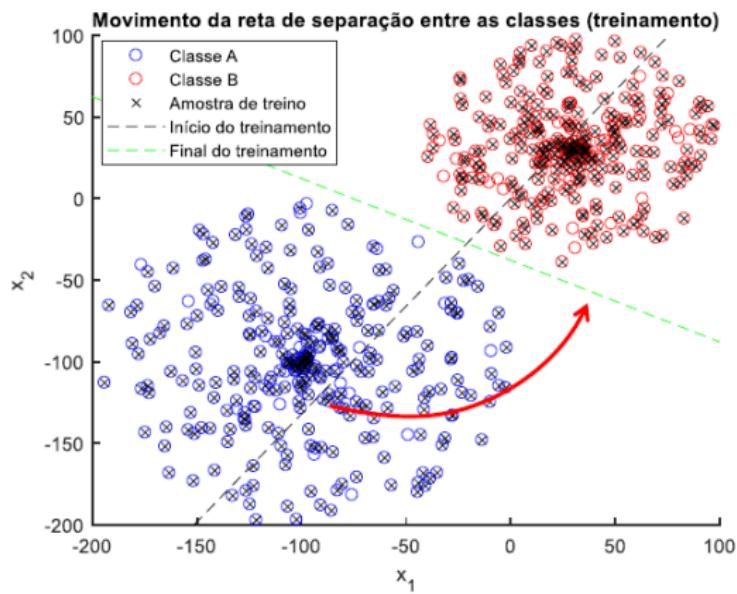
Dado o conjunto de dados, composto por 300 amostras de cada classe, realiza-se separação destas nos subconjuntos de treinamento e teste:

TREINAMENTO = 90%

TESTE = 10%

É importante que as fronteiras dos conjuntos estejam adequadamente representadas no conjunto de treino.

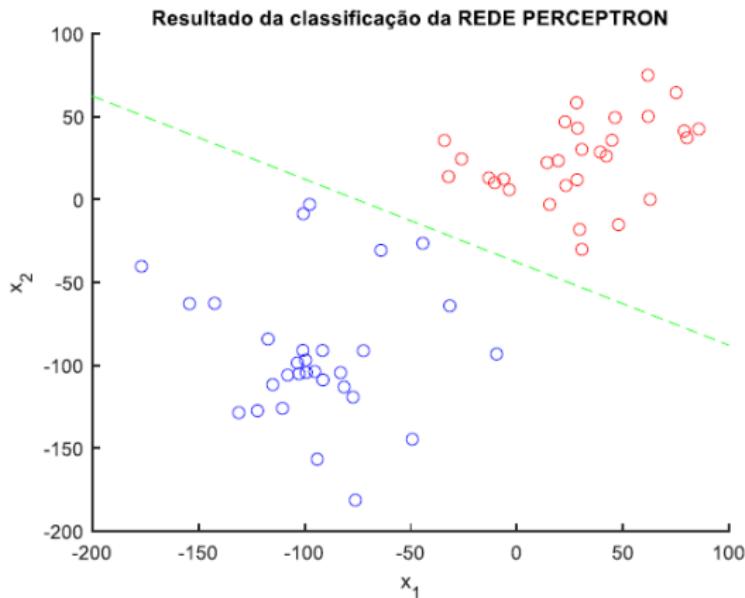
5.15.1.3. Teste de classificação



Utilizando o algoritmo de treinamento supervisionado, baseado na regra de Hebb, com $\eta = 0,05$ foi possível treinar a rede após 19 épocas.

Pode-se observar claramente que a reta de separação entre as classes utilizada pela rede perceptron foi sendo ajustada no decorrer dos passos do treinamento, através dos pesos sinápticos da rede, para promover a separação adequada entre os padrões.

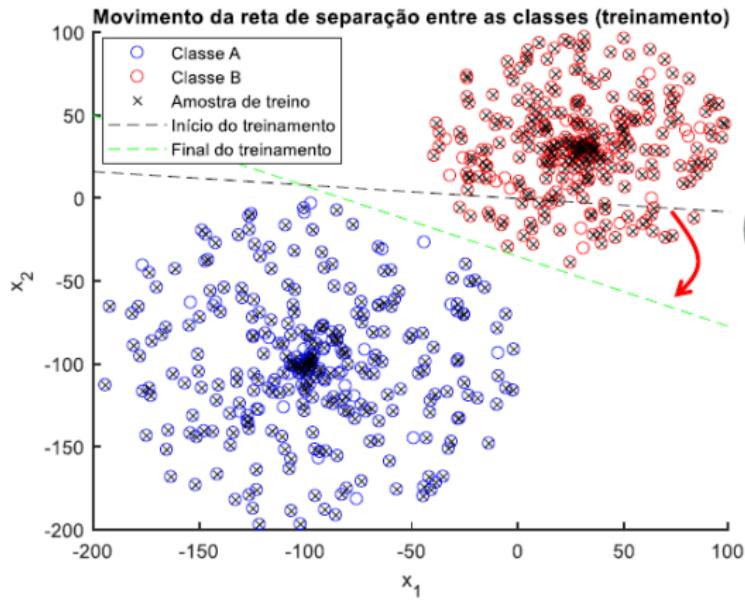
5.15.1.4. Teste de classificação



Após o treinamento verifica-se a capacidade de generalização da rede através da apresentação de amostras de teste cuja saída seja conhecida e conclui-se sobre a acuidade da rede.

ACERTOS NOS TESTES =
100%

5.15.1.5. Teste de classificação



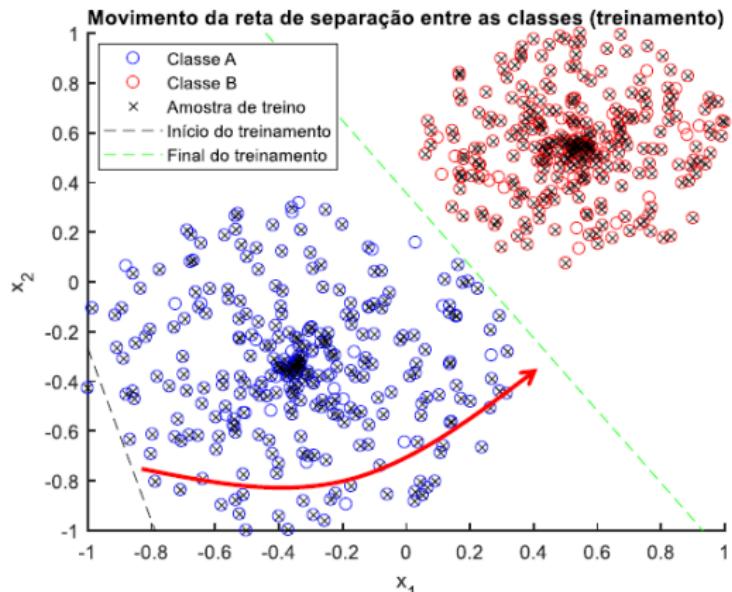
O aumento da taxa de aprendizagem implica em uma maior variação durante a atualização dos pesos sinápticos, e isto nem sempre é vantajoso!

Neste caso utilizou-se no treinamento um novo conjunto de pesos iniciais e $\eta = 0,75$ foi possível treinar a rede após **32 épocas**.

A reta obtida é próxima, porém não é a mesma que a anterior.

ACERTOS NOS TESTES =
100%

5.15.1.6. Teste de classificação



**ACERTOS NOS TESTES =
100%**

Aplicação do teorema de Tales para segmentos proporcionais:

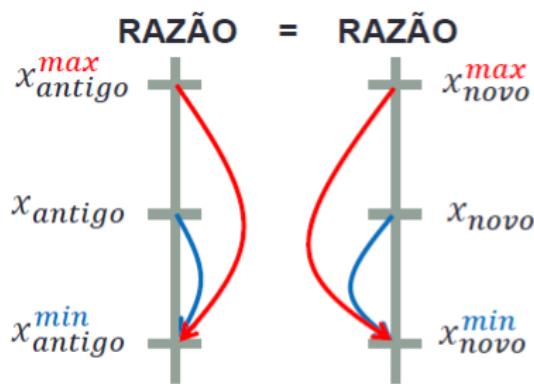
$$x_{\text{novo}} = \frac{2 \cdot x_{\text{antigo}} - (x_{\text{antigo}}^{\max} + x_{\text{antigo}}^{\min})}{x_{\text{antigo}}^{\max} - x_{\text{antigo}}^{\min}}$$

Neste caso utilizou-se uma normalização dos dados de entrada no intervalo [-1,1], com um novo conjunto de pesos iniciais e $\eta = 0,75$ foi possível treinar a rede após **3 épocas**.

Novamente a reta obtida não é a mesma que a anterior.

5.16.1. Normalização dos dados

IMPORTANTE: Este processo deve sempre levar em conta a função de ativação dos neurônios, haja visto que o aumento do desempenho provém o melhor aproveitamento da região dinâmica (variação) desta função. Ou seja a normalização permite que todos os dados sejam alocados no *range* da função de ativação garantindo uma convergência da rede mais eficaz.



$$\frac{x_{antigo} - x_{antigo}^{\min}}{x_{antigo}^{\max} - x_{antigo}^{\min}} = \frac{x_{novo} - x_{novo}^{\min}}{x_{novo}^{\max} - x_{novo}^{\min}}$$

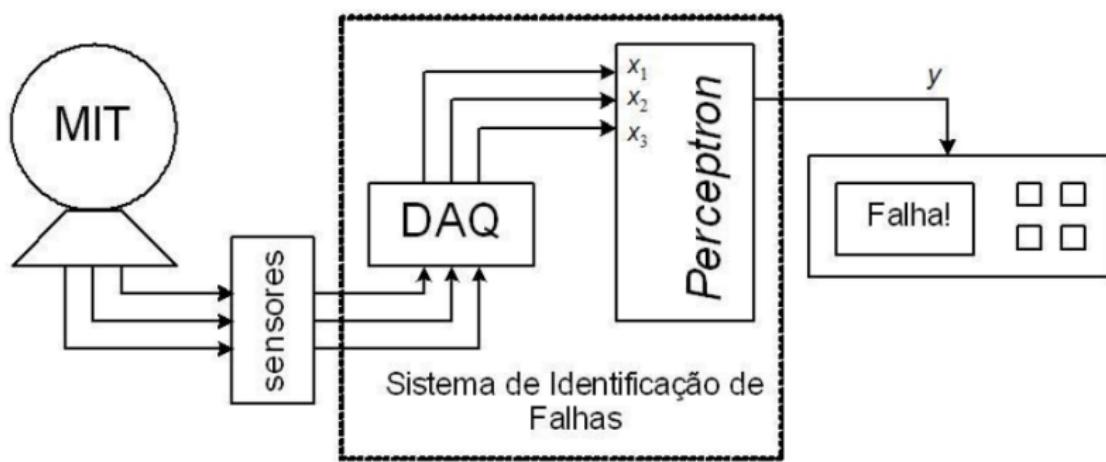
↓

$$x_{novo} = \frac{x_{antigo} \cdot (x_{novo}^{\max} - x_{novo}^{\min}) - x_{novo}^{\max} \cdot x_{antigo}^{\min} + x_{novo}^{\min} \cdot x_{antigo}^{\max}}{x_{antigo}^{\max} - x_{antigo}^{\min}}$$

Aplicação do teorema de Tales para segmentos proporcionais!

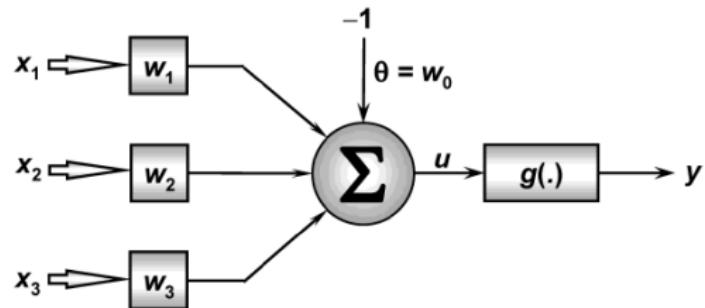
5.17.1. Implementação: Falha em motor de indução 3φ

- Uma equipe de engenheiros determinou que um tipo de **falha**, comumente encontrado em motores de **indução trifásicos** de uma indústria, pode ser **pré-identificada** mediante análises de três grandezas físicas $\{x_1, x_2, x_3\}$;
- A partir de tais grandezas, a equipe pretende aplicar um **Perceptron** para classificar a operação do motor em **duas classes**, ou seja, “**Operação Normal (Classe C1)**” ou “**Inimência de Falha (Classe C2)**”, tendo o intuito de se efetuar manutenção **preventiva** e minimizar o custo operacional da indústria.



5.17.2. Configuração perceptron

- Como existem **três** grandezas físicas que estão sendo medidas, o neurônio constituinte do *Perceptron* terá então **três entradas** $\{x_1, x_2, x_3\}$;
- Consequentemente, a saída $\{y\}$ do *Perceptron* adotará como base três entradas classificando o status da operação do motor em duas situações:
 - Classe C1 → “Operação Normal” → $\{y = 1\}$;
 - Classe C2 → “Iminência de Falha” → $\{y = -1\}$



5.17.3. Configuração da base de treinamento

- A base de dados de treinamento do Perceptron, disponibilizada no arquivo {treinamento.txt}, foi levantada por meio de sucessivos ensaios experimentais e contém o seguinte formato;

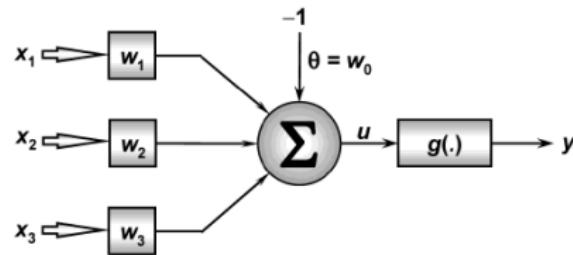
	X_1	X_2	X_3	d
01	1.4715	0.8996	1.1509	-1
02	4.6315	0.3409	2.4209	-1
03	2.9055	3.1858	3.1770	1
04	3.2563	4.3231	3.8009	1
05	0.2798	4.0843	2.2521	1
06	2.6446	3.4718	3.2715	1
07	1.0620	2.7164	1.7854	-1
08	3.5126	4.7822	4.1947	1
09	2.2227	0.427	1.2697	-1
10	0.2867	3.1473	1.8032	1
...	(...)	(...)	(...)	(...)

5.17.4. Configuração da base de treinamento (I)

a) Carregar a matriz de treinamento M usando a seguinte instrução:

- `DataTr = np.loadtxt('treinamento.txt');` {Mostre DataTr para conferência}

x_1	x_2	x_3	d
1.4715	0.8996	1.1509	-1
4.6315	0.3409	2.4209	-1
2.9055	3.1858	3.1770	1
(...)	(...)	(...)	(...)



b) Definir a matriz T : três primeiras colunas de M , inserindo ainda o elemento -1 (relativo ao termo θ) em sua primeira coluna.

- Definir o vetor d , referente aos sinais de saída da Perceptron, que seja composto pela última coluna de M . {Mostre T e d para conferência}

$$T = [-1 \quad 1.4715 \quad 0.8996 \quad 1.1509 \\ -1 \quad 4.6315 \quad 0.3409 \quad 2.4209 \\ -1 \quad 2.9055 \quad 3.1858 \quad 3.1770 \\ (...) \quad (...) \quad (...) \quad (...)]$$

$$d = [-1 \\ -1 \\ 1 \\ (...)]$$

5.17.5. Configuração da base de treinamento (II)

c) Taxa de aprendizagem em 0.01; $\{\eta \leftarrow 0.01\}$

- Contador de épocas em 0; $\{Epoca \leftarrow 0\}$;
- Vetor de pesos $\{W\}$ com valores aleatórios uniformemente distribuídos entre 0 e 1, sendo que cada elemento representará os seguintes parâmetros:
 - $W = [\theta \quad W_1 \quad W_2 \quad W_3]^T$.

d) Implementar a instrução que, dada uma linha k da matriz T , obtenha o potencial de ativação do neurônio, ou seja:

- $x \leftarrow T(k, :)^T$ {onde x conterá a k-ésima linha da matriz $T\}$;
- $u \leftarrow W^T \cdot x$ {realize eventuais transposições que sejam necessárias};
- Teste a sua instrução para $k = 2$, verificando se o valor de retorno está correto.

$$T = [-1 \quad 1.4715 \quad 0.8996 \quad 1.1509 \\ -1 \quad 4.6315 \quad 0.3409 \quad 2.4209 \quad \leftarrow k = 2 \\ -1 \quad 2.9055 \quad 3.1858 \quad 3.1770 \\ (\dots) \quad (\dots) \quad (\dots) \quad (\dots)]$$

5.17.6. Configuração da base de treinamento (III)

(...)

erro ← “existe”

época ← 0

Enquanto (erro = “existe”), faça:

 erro ← “não existe” ;

 época ← época + 1 ;

Para k variando de 1 até a quantidade total de amostras em T , faça:

$x \leftarrow T(k, :)^T$; {atribua amostra k de T ao vetor x // Use instruções do item “d”}

$u \leftarrow w^T * x$; {realize as eventuais transposições que sejam necessárias}

$y \leftarrow \text{sinal}(u)$;

 Se $y \neq d(k)$ então

$w \leftarrow w + \eta * (d(k) - y) * x$;

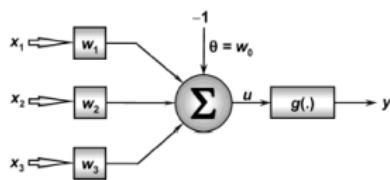
 erro ← “existe” ;

 Fim_se

 Fim_Para

Fim_Enquanto

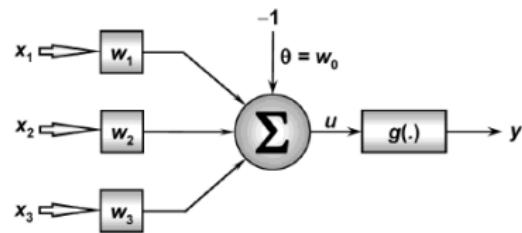
{Execute a rede pelo menos três vezes e analise os números de épocas e os valores finais para o vetor w
imprimir o número de épocas e o valor final do vetor w }



5.17.7. Configuração da base de treinamento (IV)

- e) Após o treinamento do *Perceptron*, aplique-o para efetuar a identificação de falhas para algumas situações coletadas pelos sensores situados na planta industrial. Carregue numa matriz V o arquivo {teste.txt} que contém a relação completa destas situações representadas por medições de x_1 , x_2 e x_3 ;

x_1	x_2	x_3
4.1736	4.5290	4.2580
3.6349	4.5669	2.4343
4.1618	4.4877	3.8373
1.3925	0.7344	2.0922
4.7875	4.8244	4.6913
(...)	(...)	(...)]



- f) Prepare esta matriz V , adicionando os elementos -1 em sua primeira coluna, a fim de ser inserida nas entradas do *Perceptron* já treinado.

5.17.8. Configuração da base de treinamento (V)

- h) Implemente as instruções que permitam a classificação, após a realização do processo de treinamento, usando o algoritmo seguinte. Forneça também os resultados da classificação.

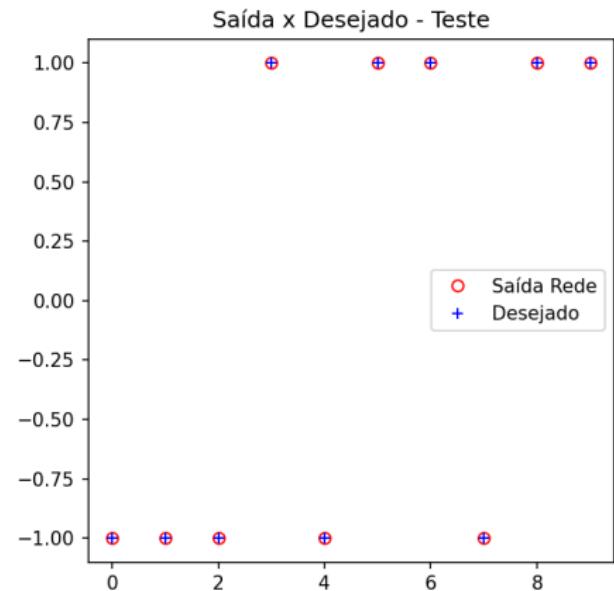
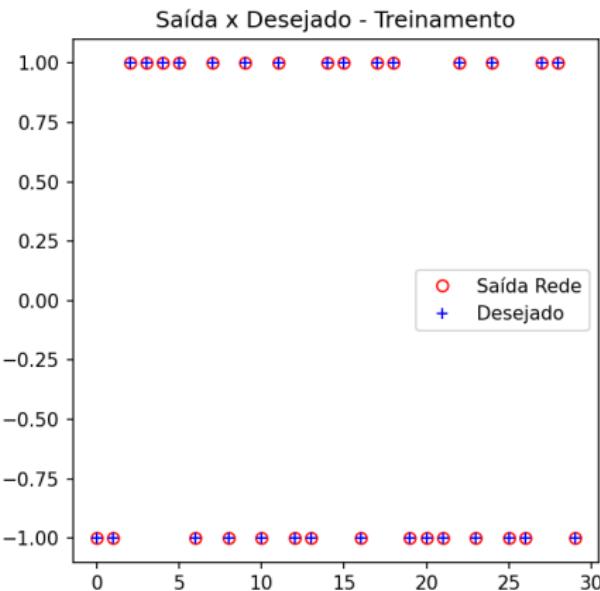
(...)

Para k variando de 1 até a quantidade total de amostras em V , faça:

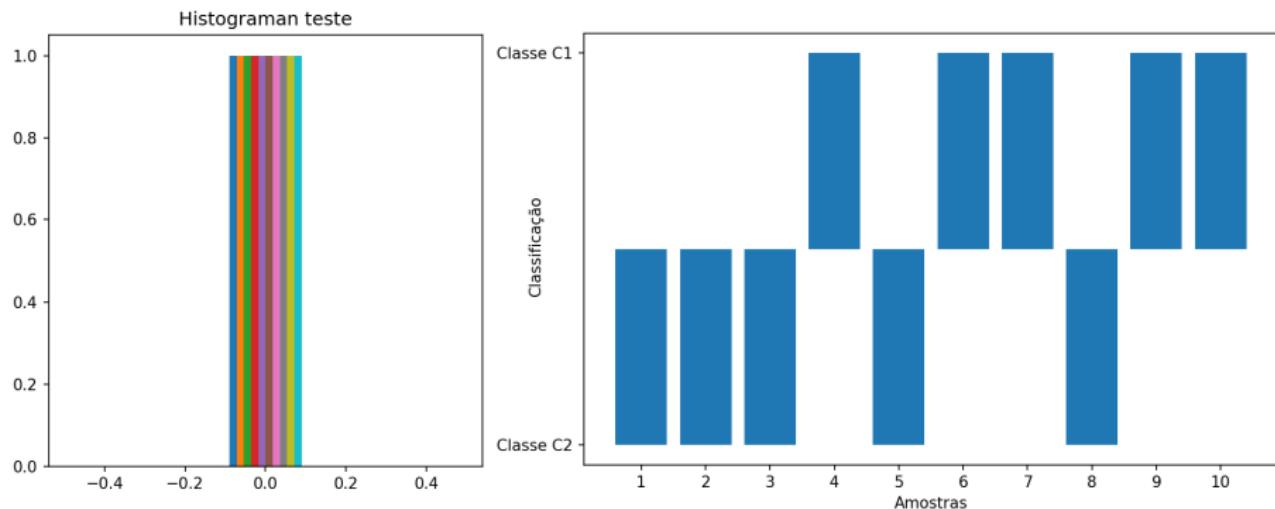
```
x ← V(k, :)T; {atribua a amostra k de V ao vetor x}
u ← wT*x; {realize as transposições que sejam necessárias}
y ← sinal(u);
Imprima(y);
Fim_Para
(...)
```

- ♦ Se $\{y = 1\} \rightarrow$ “Operação Normal” \rightarrow Classe C_1
- ♦ Se $\{y = -1\} \rightarrow$ “Iminência de Falha” \rightarrow Classe C_2

5.17.9. Resultados



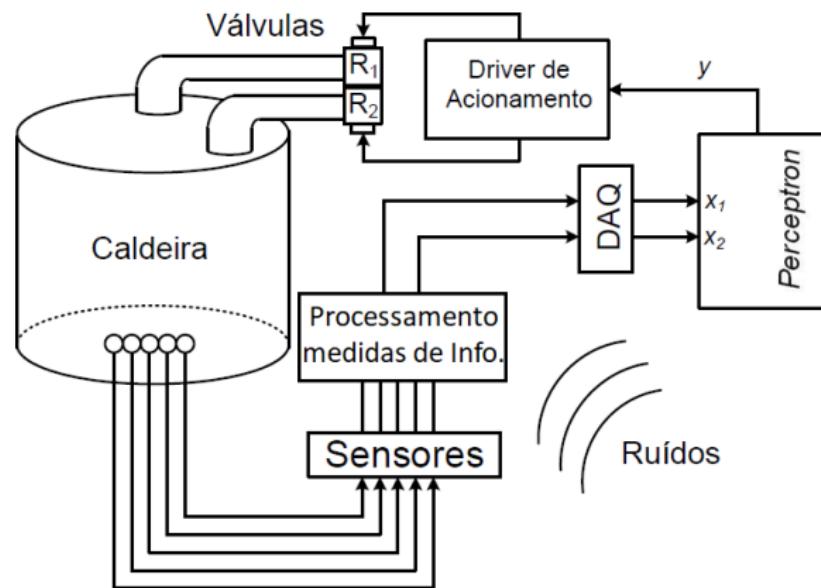
5.17.9.1. Resultados



5.17.10. Implementação usina sucroalcooleira

- Em uma planta de **usina** sucroalcooleira, a eficiência do processo de fabricação do álcool pode ser otimizado a partir do controle adequado da injeção de **dois** tipos de **reagentes** (R1 ou R2) em um determinado estágio do processo de fermentação;
- Assim, uma equipe de **engenheiros**, por meio de pesquisas e ensaios em laboratório, concluíram que os reagentes (R1 ou R2) poderiam ser adicionados ao processo dependendo da **concentração** de algumas **substâncias** (γ_1 , γ_2 e γ_3) e de mais outras **duas** grandezas físico-químicas (σ_1 e σ_2);
- Durante o processo de estudo identificou-se que havia um ruído no ambiente cujo comprometia as variáveis anteriores. Com isso a equipe realizou uma análise estatística da correlação entre tais grandezas na intenção de deixá-las imune ao ruído e duas medidas de informações foram obtidas para realizar a classificação: x_1 e x_2 .
- Por fim, a equipe pretende desenvolver um sistema automático de **acionamento** das válvulas constituintes dos dois tipos de reagentes, cujo diagrama **esquemático** pode ser observado na figura a seguir.

5.17.10.1. Implementação usina sucroalcooleira



Avalie a quantidade de amostras que foram coletadas e separe uma quantidade para treinamento e outra para teste. Lembre-se de embaralhar as amostras para que a rede não “memorize” o sistema. Dependendo das respostas oriundas da rede realize o pré-processamento (normalização) dos dados.

REFERÊNCIAS



[1] José UNPINGCO.

Python Programming for Data Analysis (2020)
Springer.



[2] Paul DEITEL e Harvey DEITEL.

Into to Python for computer science and data science - Learning to program with AI, Big Data and the Cloud (2021)
Pearson Education Limited.



[3] Eric MATTHES.

Curso intensivo de PYTHON - uma introdução prática e baseada em projetos à programação (2016)
Novatec Ltda.



[4] Wes MCKINNEY.

Python for Data Analysis - 1 Ed. (2012)
O'Reilly Media, Inc.



[5] Siegmund BRANDT.

Data Analysis - statistical and computational Methods for Scientists and Engineers
- 4 Ed. (2014).
Springer.



SENAI

Departamento Regional
DE SÃO PAULO
www.sp.senai.br