



Ph.D. Professor Aluisio Igor Rego Fontes

Gustavo de Freitas Rodrigues

José Matheus Bento

**Autor da apostila**

Ph.D. Professor Aluisio Igor Rego Fontes

**Instrutor do curso**

Larissa Jéssica Alves – Analista de Suporte Pedagógico

**Revisão da apostila**

**Autor**

**Aluisio Igor Rêgo Fontes**

Aluisio I. R. Fontes possui graduação em Engenharia da Computação pela Universidade Federal do Rio Grande do Norte (2010), mestrado (2012) e doutorado (2015) em Engenharia Elétrica e Computação pela Universidade Federal do Rio Grande do Norte. Atualmente é professor do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte campus Pau dos Ferros, onde coordena o Laboratório de Análise de Dados e Inteligência Computacional (NADIC). Por mais de 13 anos, vem se dedicando a criação de soluções inovadoras utilizando inteligência artificial e análise de dados para inserir tecnologias em várias organizações. No contexto de sistemas corporativos, desenvolvi soluções com acesso massivo de usuários, alta disponibilidade e robotização de processos. Na pesquisa acadêmica sou autor/co-autor de mais de 15 publicações em periódicos internacionais nas áreas de Inteligência Artificial e teoria da informação, tendo publicado em veículos de grande reputação nestas áreas. Sou revisor de periódicos internacionais (e.g., IEEE Signal Processing, Expert System with Application, IEEE Access e eurasip journal on advances in signal processing). Tenho experiência na área de Engenharia da Computação, com ênfase em Sistemas de Computação, atuando principalmente nos seguintes temas: Processamento digital de sinais, inteligência Computacional, Teoria da Informação, Correntropia, Processamento em Big Data, desenvolvimento de sistemas corporativos e computação de alto desempenho.



**José Matheus Bento**

Fiz curso técnico em informática no IFRN, campus Pau dos Ferros. Atualmente estou cursando Análise e Desenvolvimento de Sistemas no mesmo instituto. Sempre gostei de tecnologia, e assim que aprendi a programar, procurei aprender diversas tecnologias, para me aprimorar e encontrar aquela que mais gostasse. Desde 2020 participo do NADIC onde trabalhei com desenvolvimento backend utilizando o framework Django, em 2021 iniciei um estágio na JusInvestimens ainda focado em backend. No ano de 2022 fui contratado tanto para dar continuidade ao sistema da empresa como também para desenvolver um aplicativo nativo para Android. Tenho muito interesse em redes neurais e aprendizado de máquina, pois desenvolvi um sistema de reconhecimento de placas de veículos utilizando IA.



**Gustavo de Freitas Rodrigues**

Sou técnico de informática no IFRN - Campus Pau dos Ferros, estudante na graduação de Tecnologia da Informação, com interesse em ciências da computação pela UFRN (Universidade Federal do Rio Grande Do Norte). Tem experiência com circuitos e eletrônica, já tendo trabalhado na construção de robôs, utilizando arduino. Participei no NADIC (Núcleo de Análise de Dados e Inteligência Computacional), na área de Deep Learning, Computer Vision e IoT. Atualmente, estou trabalhando e aprofundando meus conhecimentos em ciência de dados e aprendizado de máquina.

**APRESENTAÇÃO**

A presente apostila é um instrumento teórico que complementa o curso de capacitação de Inteligência Artificial. Nela, veremoso primeiro módulo daementa do curso. Este material é baseado em artigos científicos, periódicos, revistas científicas e livros científicos. É extremamente recomendável ao aluno que, ao final da leitura de cada seção, realize os exercícios propostos e acesse os materiais indicados nas referências bibliográficas, para aprofundar a leitura desse material e complementar o que foi lido aqui.

***Boa Leitura !!***

**Sumário**

[**1 Fundamentos de e Inteligência Artificial 6**](#_heading=h.30j0zll)

[**1.1 O que é aprendizado? 7**](#_heading=h.lqxfurnibtbh)

[**1.2 O neurônio e sua inspiração para o perceptron 7**](#_heading=h.pilbfws99z83)

[**1.3 O que é Aprendizado de Máquina? 9**](#_heading=h.116mfqls6s0h)

[**1.4 Tipos de modelos 14**](#_heading=h.xb0k4ktt5oz7)

[**1.5 Classificação e Regressão 20**](#_heading=h.wbc9q7z76863)

[**1.6 Conjunto de dados 21**](#_heading=h.8mf0ln2miwpi)

[**1.6.1 NumPy 23**](#_heading=h.a4lrvcem288f)

[**1.6.2 Pandas 31**](#_heading=h.xokisnfl0cp)

[**1.6.3 Polars 37**](#_heading=h.qxk3vyqaixx5)

[**1.7 Projeto prático I 47**](#_heading=h.iy5roj3hinic)

[**2 Pré-processamento e Transformação de Dados 54**](#_heading=h.qru63u2dzk2u)

[**2.1 O que é o pré-processamento e limpeza de dados 54**](#_heading=h.rmvynn2qw5jy)

[**2.2 Análise Exploratória de Dados 55**](#_heading=h.cwckhdgtse76)

[**2.3 Limpeza dos Dados (com o Pandas) 55**](#_heading=h.2r0nprumqij)

[**2.4 Filtrando DataFrames 55**](#_heading=h.43fsxdcfy6j9)

[**2.5 Redução de dimensionalidade 55**](#_heading=h.crj09tdc3ik4)

[**2.6 Análise dos Componentes Principais**](#_heading=h.7pf4fmqcxjr1) **56**

# Fundamentos de e Inteligência Artificial

A Inteligência Artificial (IA) é uma área da ciência da computação que se dedica ao desenvolvimento de sistemas e algoritmos capazes de executar tarefas que normalmente requerem inteligência humana. O campo da IA busca criar agentes artificiais que podem perceber seu ambiente, tomar decisões, aprender com a experiência e melhorar seu desempenho ao longo do tempo. Uma das subáreas mais proeminentes da IA é a Aprendizagem de Máquina (AM), que desempenha um papel fundamental na realização dessa visão.

A Aprendizagem de Máquina é uma disciplina da IA que se concentra em desenvolver algoritmos e modelos que permitem que os sistemas computacionais aprendam a partir de dados e experiência, em vez de serem explicitamente programados para realizar tarefas específicas. A relação entre IA e AM é intrínseca, pois a AM é frequentemente vista como um dos principais motores da IA moderna.

## O que é aprendizado?

Para buscarmos um bom significado para aprendizado, vamos ver o que significa “aprender”: “Alcançar ou conseguir conhecimento, cognição, educação ou especialidade através da experiência ou estudo”, “Ficar-se competente ou apto em algo”, “Ficar-se eficiente ou capaz, em alguma coisa, de forma gradual”.

Segundo Kolb (1976) aprendizado é um processo de mudança cognitiva que ocorre como resultado de uma experiência. Essa mudança pode envolver a aquisição de novos conhecimentos, habilidades ou atitudes.

Observamos que a aprendizagem vai além da simples memorização de regras ou de qualquer outro conceito. Se fosse assim, tornaria-se trivial ensinar computadores a aprender. A habilidade de memorização é compartilhada por todos, no entanto, a verdadeira complexidade da aprendizagem reside na capacidade de generalizar comportamentos para contextos novos, algo que nosso cérebro realiza de maneira extraordinária.

Para transmitir essa capacidade de generalização aos computadores, foram desenvolvidos programas que aprimoram consideravelmente seu desempenho por meio da experiência. Estes programas empregam técnicas orientadas por dados, que buscam discernir padrões nos dados disponíveis e, a partir disso, gerar hipóteses ou modelos.

## O neurônio e sua inspiração para o *perceptron*

Um neurônio é uma célula do sistema nervoso responsável pelo processamento e transmissão de informações químicas e elétricas. No cérebro humano, os neurônios se conectam entre si formando redes complexas, que permitem o aprendizado e a memória. O processo de aprendizado em um neurônio biológico envolve a alteração da força sináptica, ou seja, a eficiência com que um neurônio influencia o outro.

A relação disso com um *perceptron*, um tipo básico de neurônio artificial, é conceitual e inspiracional. O perceptron foi desenvolvido como um modelo simplificado do neurônio biológico. Ele consiste em unidades de entrada (análogas às dendrites), que recebem sinais; um processo de somatória (similar ao corpo celular); e uma função de ativação (análoga ao axônio) que determina a saída do *perceptron*, essa relação pode ser vista por meio da figura 1.

No *perceptron*, o processo de aprendizado envolve ajustar os pesos das conexões entre os neurônios artificiais, de forma a minimizar a diferença entre a saída esperada e a saída produzida pela rede. Este ajuste é frequentemente realizado por meio de um algoritmo chamado "*backpropagation*", que é uma forma de otimizar esses pesos através de um processo iterativo e gradativo.

Embora a inspiração inicial para redes neurais artificiais tenha vindo dos neurônios biológicos, é importante notar que as redes neurais artificiais são muito mais simplificadas e não replicam muitos dos processos mais complexos do cérebro humano. Contudo, essa conexão conceitual entre os dois ajuda a entender a base da inspiração para o desenvolvimento de algoritmos de machine learning e inteligência artificial. (HAYKIN, 2009)

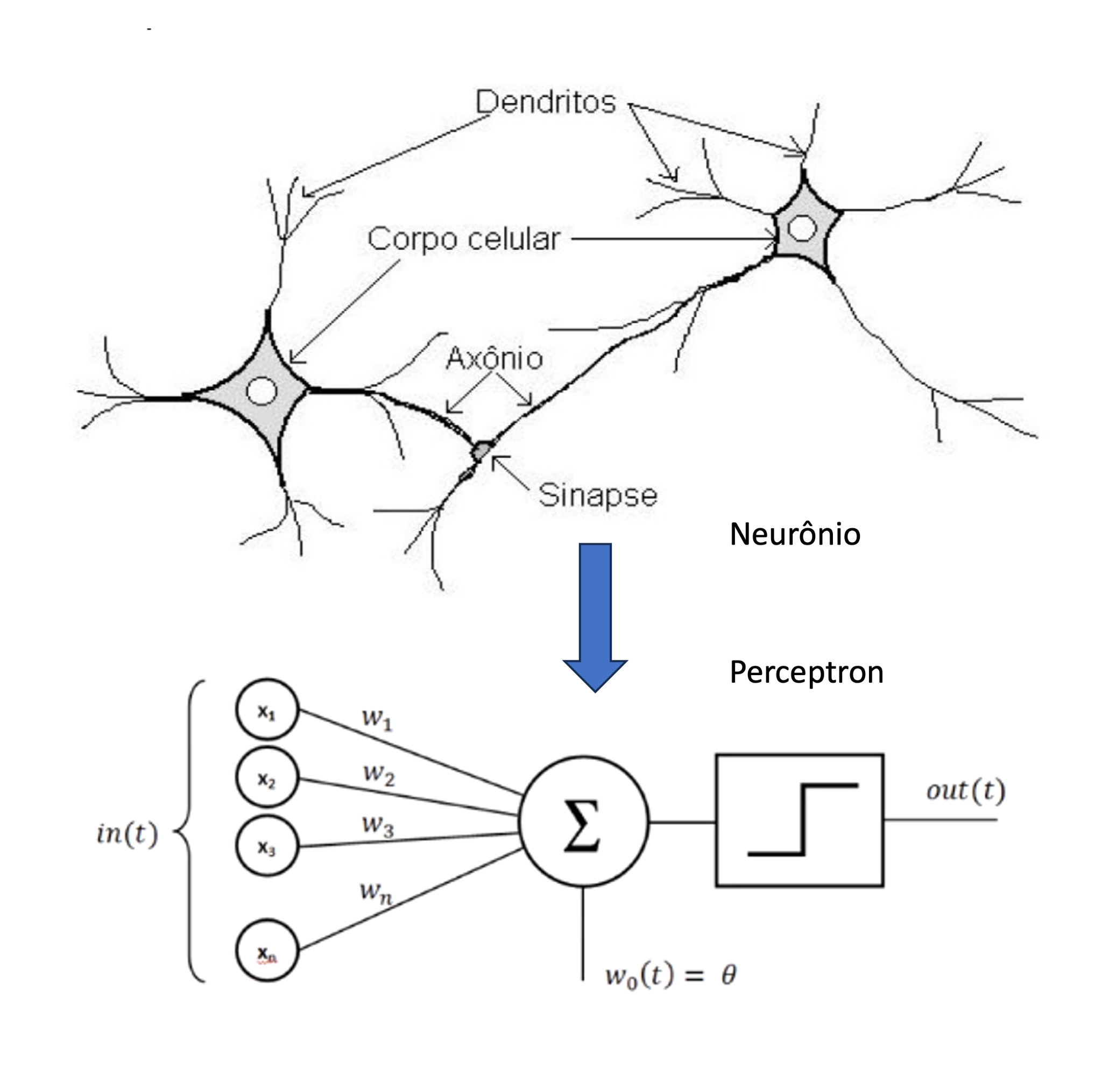
****

Imagem 1 - Representação de uma rede neural de forma artificial

Fonte: Autoria Própria

## O que é Aprendizado de Máquina?

O campo de Aprendizado de Máquina teve suas raízes na investigação do processo de descoberta de conhecimento em bases de dados, conhecido como KDD (Knowledge Discovery in Databases). Esse processo se concentra na tarefa de identificar padrões significativos em vastos conjuntos de dados. A descoberta resultante deve ser relevante, previamente desconhecida e amplamente aplicável, essencialmente capturando a essência dos dados.

Analogamente, pode-se pensar nesse processo como uma espécie de mineração de informações, onde grandes volumes de dados são meticulosamente analisados na busca por insights de valor. Na Figura 2, ilustra-se todas as etapas do KDD de forma clara e concisa.

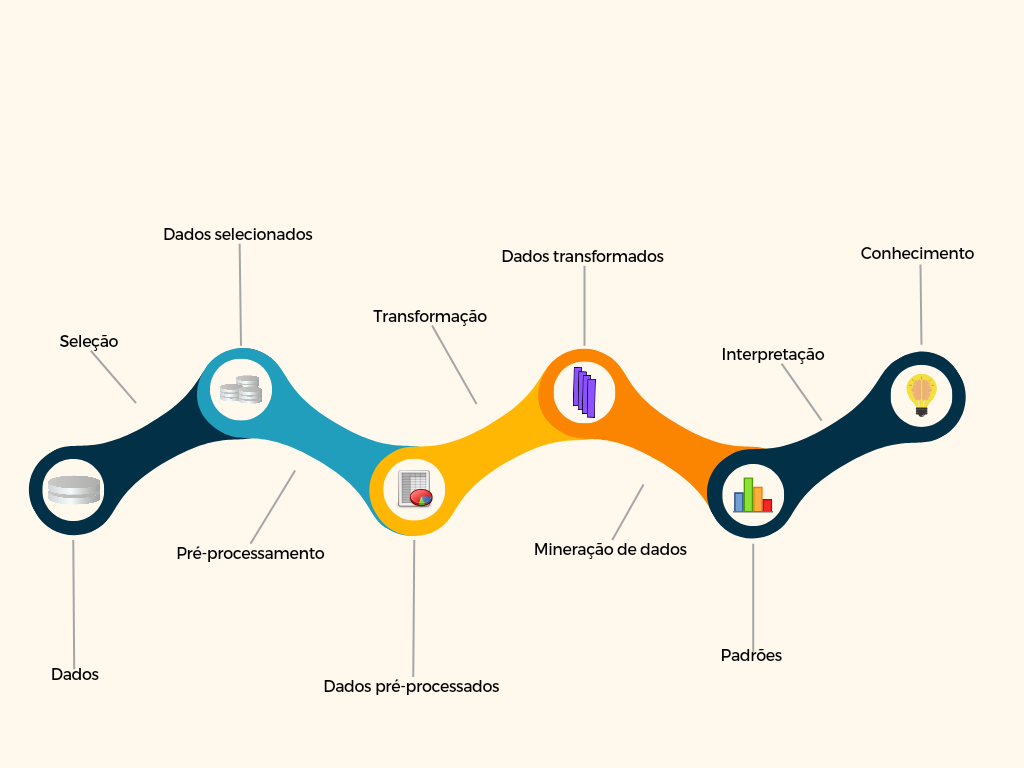


Imagem 2 - Etapas do KDD

Fonte: Autoria Própria

Vemos que nas primeiras versões não aparecia o termo de aprendizado de máquina propriamente dito, contudo, *Data Mining* e Aprendizado de Máquina são termos similares.

No KDD haviam 3 etapas imprescindíveis para a concretização do *Knowledge.* Sendo elas:

1. Pré-Processamento;
2. Aprendizado de Máquina
3. Pós-Processamento.

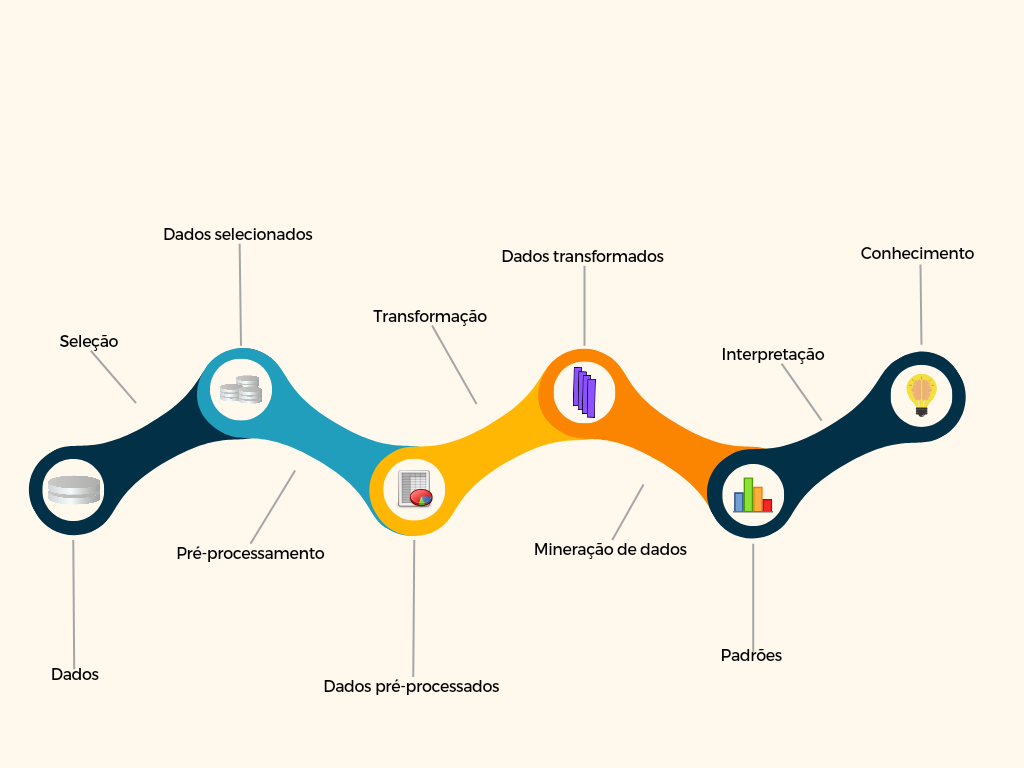
Como podem ser vistas nas imagens 2, 3 e 4. Vale ressaltar que cada etapa será vista com mais detalhes nos posteriores módulos do curso.

Imagem 3 - Pré-Processamento no KDD

Fonte: Autoria Própria

A área demarcada na figura 3 nas etapas do KDD simboliza onde começa e termina a parte do pré-processamento.

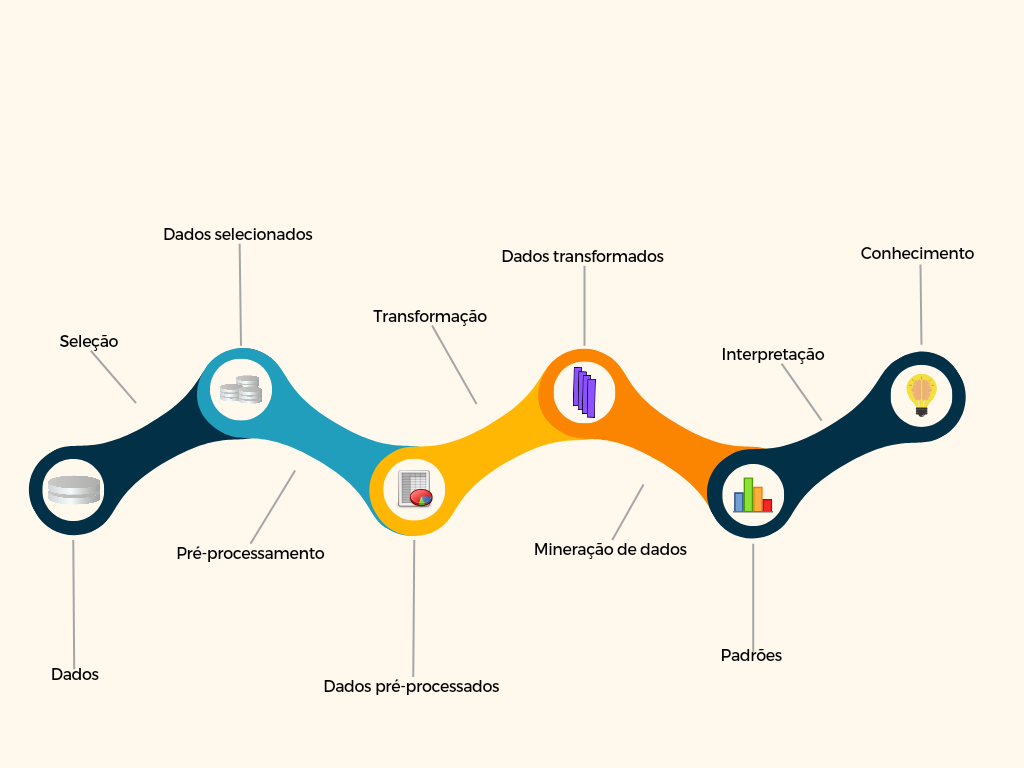


Imagem 4 - Aprendizado de Máquina no KDD

Fonte: Autoria Própria

Na figura 4, podemos visualizar em que momento o aprendizado de máquina entra, de fato, no processo de descobrimentos de novos conhecimentos.

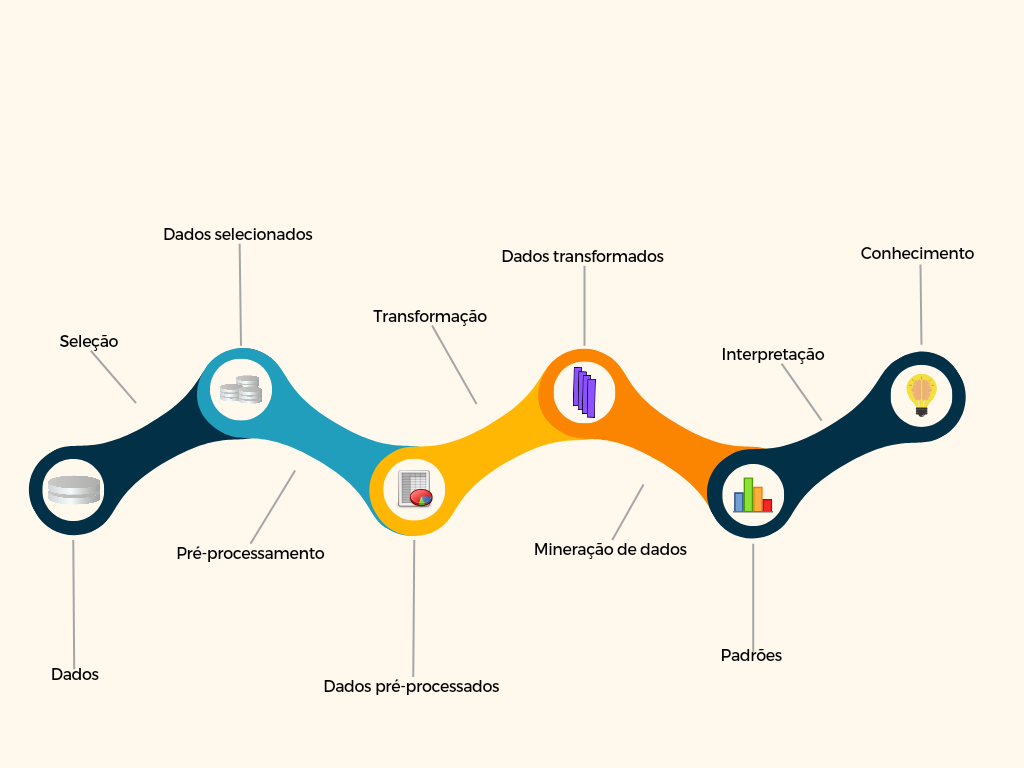


Imagem 5 - Pós - Processamento no KDD

Fonte: Autoria Própria

Na figura 5, vemos onde ocorre o pós-processamento, etapa a qual há a interpretação e a avaliação do aprendizado de máquina.

Buscando na literatura as definições de aprendizado de máquina, tem-se em Arthur Samuel (1959) que "Aprendizado de Máquina é o campo de estudo que dá aos computadores a habilidade de aprender sem ser explicitamente programado."

Já para Tom Mitchell (1997) tem-se uma visão mais técnica que afirma que: "Diz-se que um programa de computador aprende pela experiência **E** em relação a algum tipo de tarefa **T** e alguma medida de desempenho **P** se o seu desempenho em **T**, conforme medido por **P**, melhora com a experiência **E**."

Por exemplo, um algoritmo de envio de anúncios, comum em toda a internet. O objetivo nesse caso é a obtenção de clientes, e a métrica de desempenho é o total de pessoas que viram o anúncio e compraram o produto, ou até mesmo o tempo de visualização de cada anúncio ou área de foco na tela. Cada anúncio seria uma amostra de treinamento, os detalhes da publicação e do usuário que a recebeu estariam nessa amostra. O "Pixel do Facebook" é uma ferramenta de rastreamento e análise fornecida pelo Facebook para proprietários de sites. Ele é um pequeno trecho de código que é adicionado às páginas da web e permite que o Facebook acompanhe as interações dos visitantes com o site. Isso inclui a medição de conversões, como compras online, inscrições em formulários e outras ações importantes.

O *Pixel* do Facebook é uma parte fundamental das estratégias de marketing digital, pois fornece dados valiosos sobre o comportamento dos visitantes do site. Essas informações podem ser usadas para segmentar anúncios de forma mais eficaz, medir o retorno sobre o investimento em publicidade e otimizar campanhas para alcançar os objetivos de negócios. Em resumo, o *Pixel* do Facebook desempenha um papel crucial no marketing online, ajudando as empresas a tomar decisões informadas e melhorar seu desempenho digital (Meta, 2023).

Isso tem diversas vantagens sobre os algoritmos tradicionais que não possuem esse aprendizado, pois desse modo o desenvolvedor teria que levantar todos os tipos de usuários e anúncios, bem como criar métricas para correlacionar o sucesso ou fracasso de seus envios. Ou seja, teria que notar os padrões para em seguida transformá-los em código, para que por fim fosse testada a solução, além de reajustes até um resultado satisfatório. Um processo desgastante e que não garante bons resultados.

## Tipos de modelos

Em aprendizado de máquina, os modelos são algoritmos que adquirem conhecimento a partir de dados. Esses modelos podem ser categorizados em três grupos fundamentais:

**Descritivos**: Estes modelos pertencem à categoria de aprendizado não supervisionado, focando na descrição de padrões e estruturas ocultas nos dados.

**Preditivos**: Enquadram-se na categoria de aprendizado supervisionado, onde os modelos fazem previsões com base em dados de treinamento rotulados.

**Aprendizado por Reforço** (*Reinforcement Learning*): Este tipo de aprendizado opera com base em recompensas, buscando otimizar ações para maximizar ganhos, como ilustrado na Figura 6.

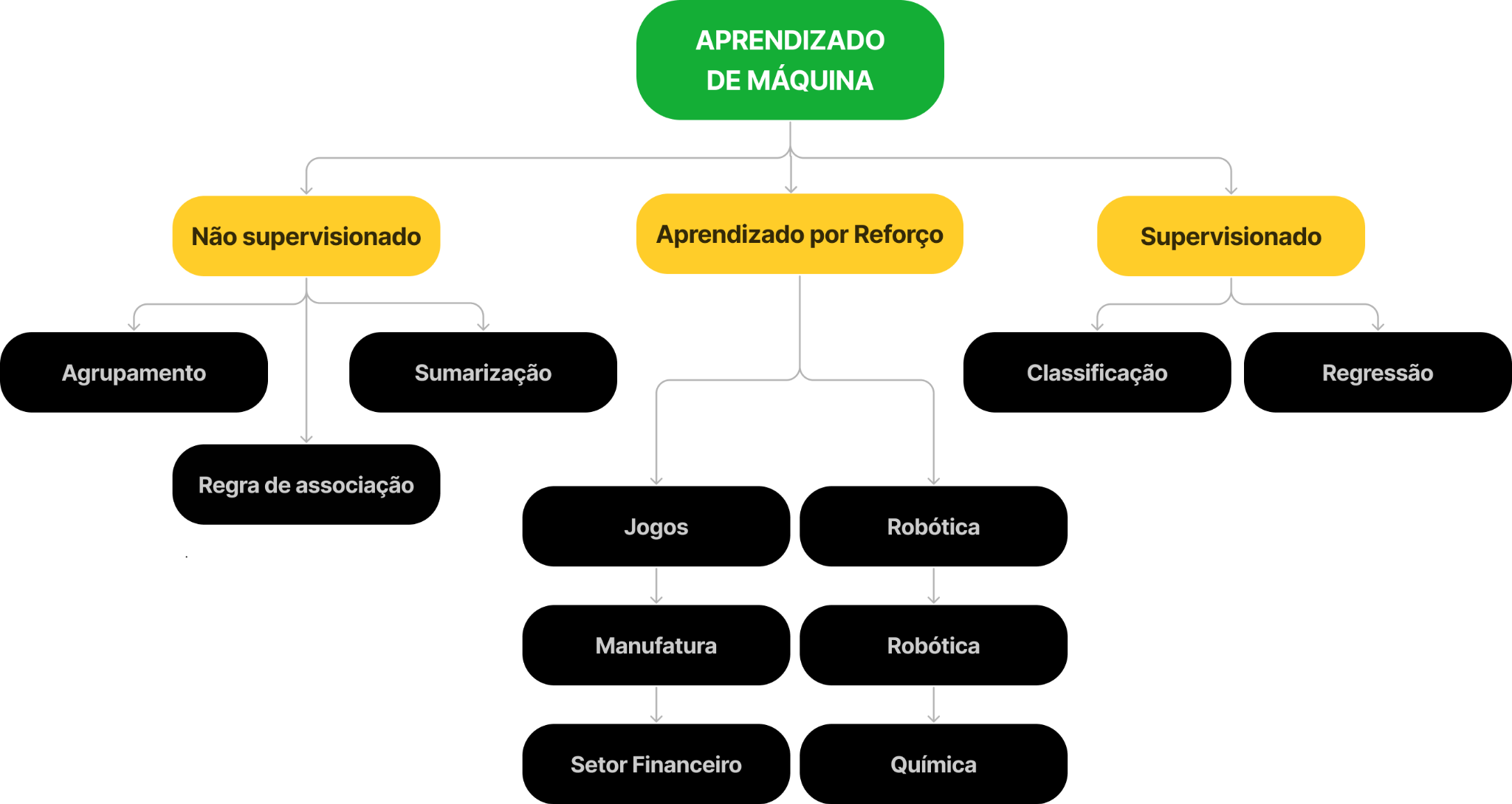


Imagem 6 - Esquema de modelos de aprendizado de máquina

Fonte: Autoria Própria

A Figura 6 ilustra o esquema de modelos de aprendizado de máquina e suas aplicações. Dentro desses modelos, destacam-se:

* **Modelos Descritivos**: São usados em tarefas como agrupamentos, regras de associação e sumarização para desvendar padrões e estruturas em conjuntos de dados.
* **Modelos Preditivos**: Incluem classificações e regressões, sendo úteis para fazer previsões com base em dados de treinamento. São comumente referidos como aprendizado supervisionado, pois analisam exemplos rotulados em busca de padrões que podem ser aplicados a novos dados.
* **Modelos de Reforço:** Encontram aplicação em algoritmos genéticos, redes generativas e processamento de linguagem natural, operando com base em recompensas para otimizar ações.

Um modelo preditivo é uma função ou hipótese extraída de dados de treinamento, usada para prever rótulos ou valores associados a novos exemplos. Em problemas de regressão, esses modelos fornecem valores contínuos ou discretizados, como prever o preço de terrenos ou o consumo semanal de água. Em problemas de Classificação, eles categorizam entradas, como determinar se um smartphone é classificado como "comprido" ou "curto" em vez de fornecer medidas específicas. Esses valores de rótulo pertencem a conjuntos finitos de valores nominais, ordenados ou não, dependendo do contexto.

Essa divisão entre modelos descritivos, preditivos e de reforço ajuda a compreender suas respectivas aplicações e funcionalidades no campo do aprendizado de máquina.

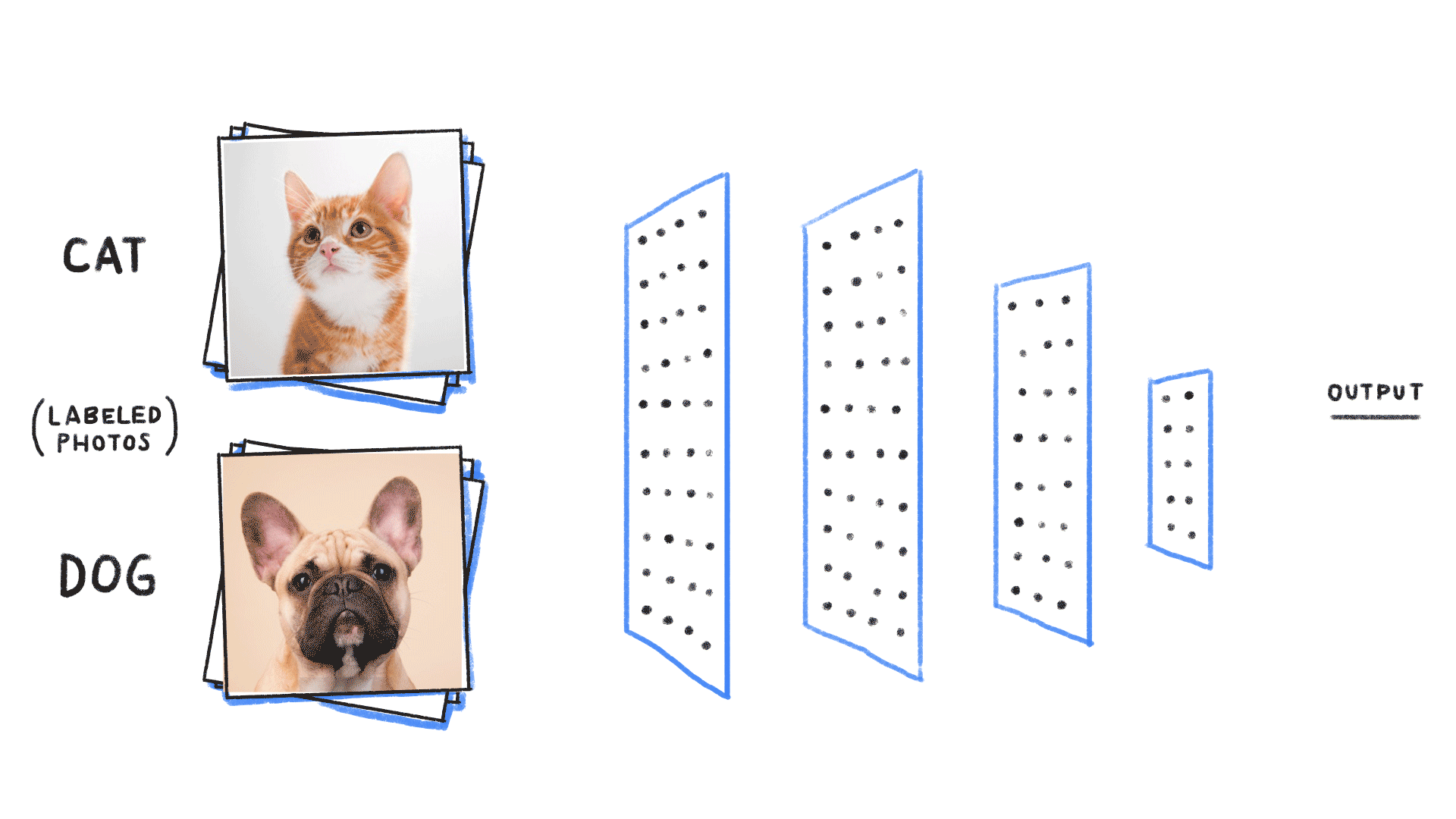


Imagem 7 - Rótulos de Gato e Cachorro

Fonte: https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Na Figura 7 é apresentado um exemplo de aprendizado supervisionado. Antes de iniciar o treinamento, as imagens de gatos e cachorros foram devidamente rotuladas como tal, ou seja, cada imagem foi associada à categoria correta, seja "gato" ou "cachorro". Isso permite que o aprendizado se baseie na informação prévia de que esses dados pertencem a uma dessas duas categorias, o que, por sua vez, facilita a realização de generalizações precisas durante o processo de classificação.

Por outro lado, modelos descritivos têm como principal objetivo a análise e descrição das características de um conjunto de dados. Seu foco principal reside em identificar padrões ou tendências nos dados, sem a necessidade de utilizar rótulos ou informações prévias de categorização.

Essas abordagens, aprendizado supervisionado e modelos descritivos, desempenham papéis distintos no campo do aprendizado de máquina, com o primeiro sendo orientado para a classificação com base em rótulos conhecidos, enquanto o último se concentra na análise das características intrínsecas dos dados.

Em problemas de sumarização, o objetivo é criar automaticamente resumos a partir de um ou mais textos. Existem duas estratégias principais para a sumarização: a extrativa e a gerativa. Na abordagem extrativa, o sumário é produzido selecionando partes relevantes do próprio texto original. Por outro lado, na abordagem gerativa, é criada uma síntese completamente nova do(s) texto(s).

No contexto da associação, as técnicas visam identificar relações entre dados com base em suas ocorrências. Um exemplo prático é a análise da cesta de compras de clientes, onde o objetivo é identificar quais itens são frequentemente adquiridos em conjunto. Por exemplo, um supermercado pode analisar as transações dos clientes e descobrir que maionese e pão são frequentemente comprados juntos. Com essa informação, o supermercado pode decidir posicionar esses produtos próximos nas prateleiras, aumentando a probabilidade de compra conjunta.

Para identificar essas relações, existem diversos algoritmos de mineração de dados disponíveis. Além disso, métricas de probabilidade e estatística, como a correlação de Pearson, podem ser úteis na descrição dessas relações. Um exemplo de algoritmo popular é o *APRIORI*, que opera da seguinte maneira:

1. Identifica todos os itens que atendem a uma frequência mínima definida pelo usuário.

2. Em seguida, identifica todos os pares de itens que atendem a uma frequência mínima pré-definida.

3. Repete o passo 2, identificando trios, quartetos e assim por diante.

4. Utiliza o conceito de suporte para determinar a frequência dos conjuntos de itens. O suporte é a porcentagem de transações que contêm esse conjunto específico de itens.

Por exemplo, se em um supermercado de 100 transações, o suporte do conjunto {pão, maionese} é de 76%, significa que 76 das 100 transações incluem ambos os produtos. Isso permite aos supermercados recomendar produtos ou organizar suas prateleiras de maneira mais eficaz.

Essa abordagem pode ser aplicada em vários setores, incluindo comércio presencial e online, para recomendação de produtos, segmentação de clientes com hábitos de compra semelhantes e promoções direcionadas.

Outro tipo de problema envolvendo modelos descritivos é o agrupamento, conhecido como clustering. Essa técnica visa agrupar instâncias de uma base de dados em clusters, onde instâncias semelhantes pertencem ao mesmo grupo, como ilustrado na Figura 8. Isso permite a identificação de padrões e relações entre os dados, o que pode ser valioso em diversos contextos.

.

Imagem 8 - Grupos formados a partir de alguma base de dados

Fonte: Autoria Própria

Na Figura 8, podemos observar um exemplo de agrupamento em três clusters, onde os membros de cada grupo têm uma relação mais próxima entre si do que com os membros dos outros clusters.

Por fim, os modelos de aprendizado por reforço buscam simular o processo de aprendizado com base em experiências passadas, semelhante ao que ocorre no aprendizado humano. Nesse modelo, o agente de aprendizagem recebe um feedback na forma de reforços, permitindo assim ajustar seus parâmetros de forma a otimizar o desempenho. O agente deverá executar esse processo de aprendizagem até que não ocorra mais alguma melhoria em seu aprendizado. O algoritmo mais conhecido desse modelo de aprendizado é o *Q-Learning* (Sutton; Barto, 2018, p. 3)

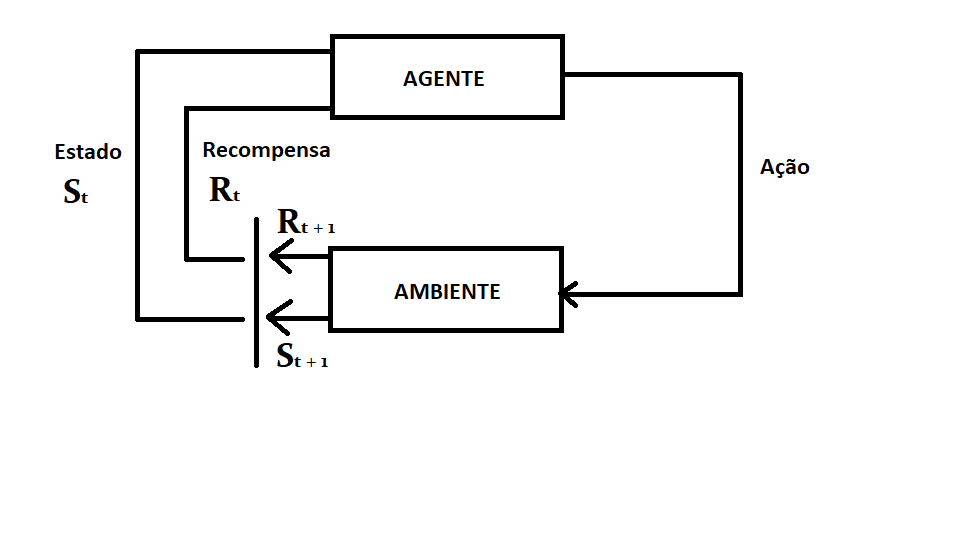


Imagem 9 - Aprendizado por Reforço

Fonte: Autoria Própria

Na figura 9 vemos como funciona o processo de aprendizado, onde o ambiente simula o problema, o estado é a forma que o ambiente e o agente se encontram, e o agente é aquele que executa as ações disponíveis. O sinal de reforço é o que o agente recebe após tomar uma determinada ação.

Um exemplo disso seria uma IA que joga xadrez, ela executaria ações no tabuleiro, que seria um ambiente, e o resultado disso seria a vitória (reforço positivo) ou derrota (reforço negativo). Nota-se a possibilidade da IA jogar diversas partidas até aprender.

## Classificação e Regressão

Como visto nas seções anteriores, o aprendizado de máquina ou *machine learning* pode resolver problemas ou realizar tarefas que possui um nível alto de trabalho para uma pessoa resolver. Essas tarefas geralmente são descritas no processo de um exemplo, e esse exemplo é a coleção de características que podem ser mensuradas de um objeto ou evento. Onde um exemplo pode ser representado como um vetor onde cada entrada seria uma característica do objeto.

Um exemplo de regressão poderia ser o seguinte: para determinar o valor de uma casa são necessários diversos fatores como quantidade de cada cômodo, tamanho, localização e entre outros, um algoritmo de regressão conheceria os valores e os atributos de diversas casas. Dessa forma, quando lhe fosse dado informações acerca de uma casa esse algoritmo poderia inferir um valor para ela, a figura 10 mostra um exemplo dessa situação.

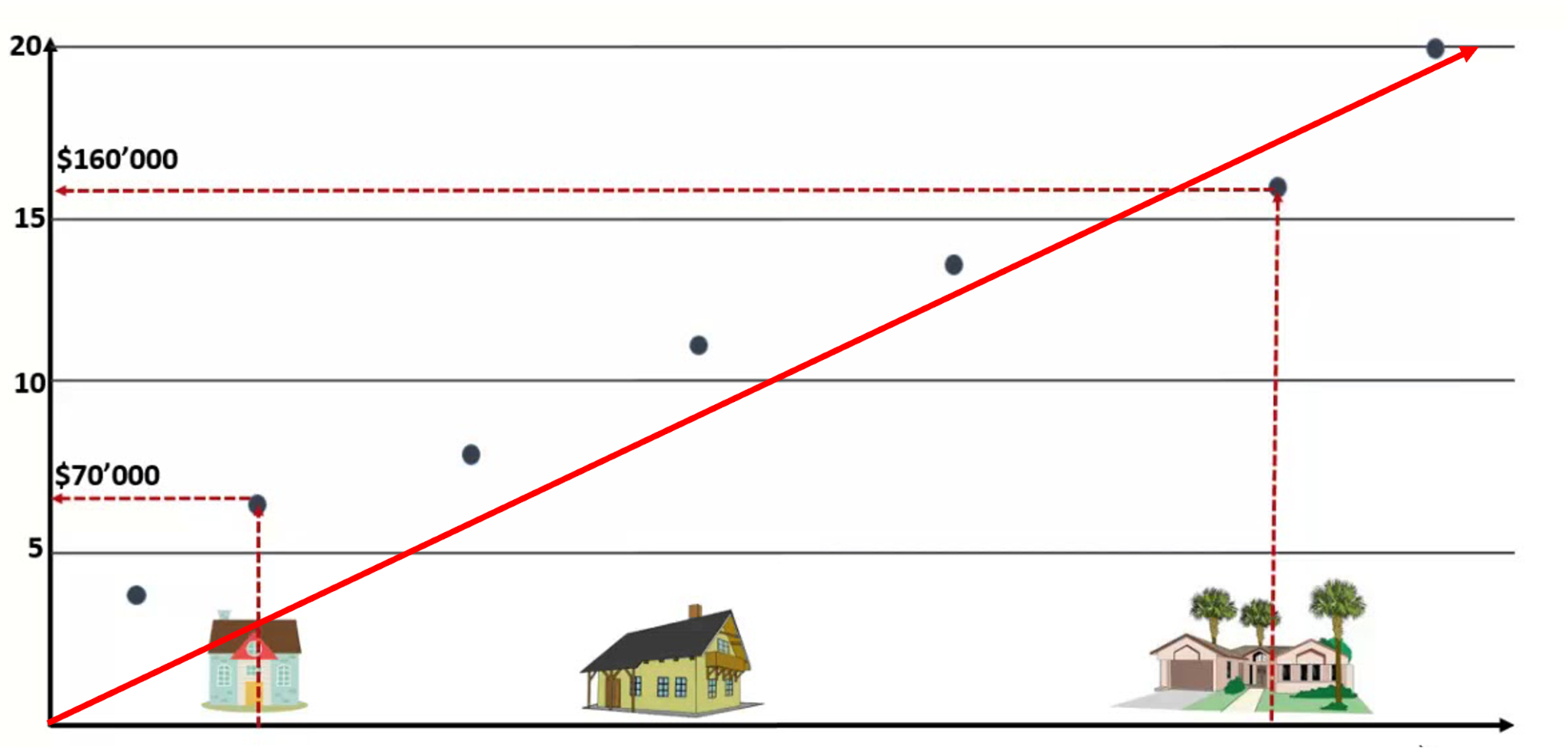


Imagem 10 - Aprendizado por Reforço

Fonte: Autoria Própria

A linha vermelha representa o modelo treinado pelo algoritmo para sugerir preços, para o valores de casas. Os pontos pretos foram pontos de um conjunto de treinamento, os quais foram utilizados como base para que o algoritmo pudesse inferir outros valores a partir do seu treinamento.

Já um exemplo de classificação seria um radar eletrônico que possui uma base de dados sobre motos, carros e caminhões. Ao aprender as características dessas três classes de veículos assim que lhe for fornecida uma foto de uma bicicleta, tentará encaixá-la em uma das classes existentes, a qual mais se assemelha a bicicleta, uma vez que não saberá diferenciá-la de uma moto, carro ou caminhão.

## Conjunto de dados

Um conjunto de dados é uma coleção de dados organizados de forma estruturada, que pode ser utilizado para fins de análise e aprendizado. Um conjunto de dados flat é um tipo de conjunto de dados que é composto por um conjunto de objetos, cada um dos quais é representado por um vetor de características (Witten et al., 2016, p. 14).

Os objetos de um conjunto de dados flat podem representar qualquer coisa, desde objetos físicos, como pessoas ou produtos, até objetos abstratos, como textos ou imagens. As características de um objeto são atributos que descrevem o objeto, como idade, peso, cor ou tamanho.

Um conjunto de dados flat é formalmente representado por uma matriz , onde **n** é o número de objetos e **d** é a dimensionalidade do espaço de objetos. A dimensionalidade do espaço de objetos é o número de características que descrevem cada objeto. Como mostra o exemplo a seguir.

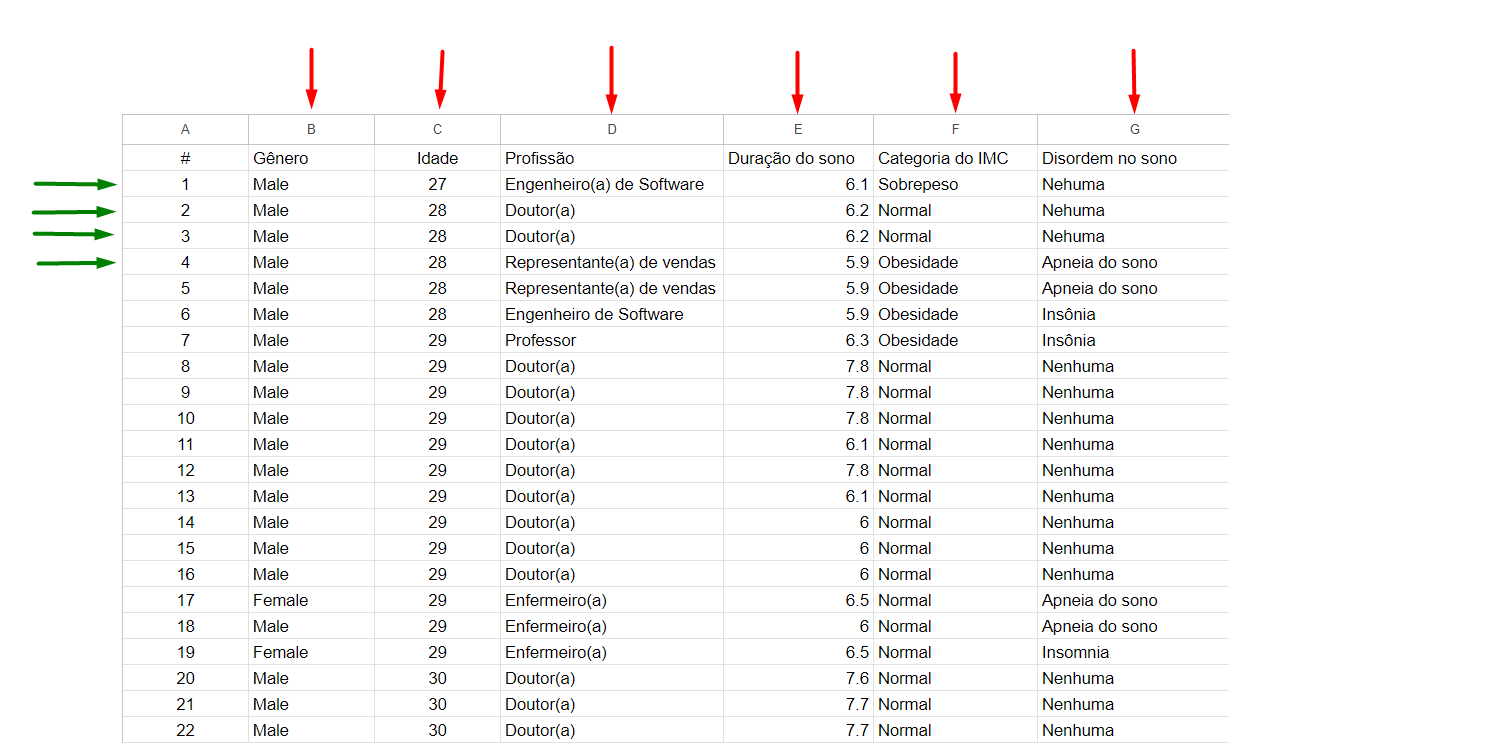


Imagem 11 - Exemplo de um conjunto de dados

Fonte da base de dados: <https://www.kaggle.com/datasets/uom190346a/sleep-health-and-lifestyle-dataset>

Na figura 11 podemos observar um conjunto de objetos dispostos em uma tabela, ou seja, temos um conjunto de dados. As setas vermelhas representam os atributos do conjunto de dados, totalizando 7 atributos ou 7 dimensões. As setas verdes indicam cada instância de um objeto, ou seja, temos 22 instâncias. Formando uma matriz .

Atributos são características que descrevem um objeto, como idade, peso, cor ou tamanho. Eles são uma parte essencial de qualquer conjunto de dados, pois fornecem informações sobre os dados que estão sendo analisados.

Os atributos podem ser classificados em dois tipos principais: qualitativos e quantitativos.

**Atributos qualitativos:** são aqueles que descrevem as características de um objeto de forma qualitativa, ou seja, eles não podem ser medidos e comparados de forma numérica. Os atributos qualitativos podem ser divididos em dois subtipos: Nominais ou Categóricos e Ordinais ou Discretos.

**Atributos Categóricos:** são atributos que não possuem uma ordem natural. Por exemplo, o atributo "*cor*" pode ter os valores "*azul*", "*verde*" e "*vermelho*", e eles não possuem uma ordem entre si.

**Atributos Ordinais:** são atributos que têm uma ordem natural. Por exemplo, o atributo "*tamanho*" pode ter os valores "*pequeno*", "*médio*" e "*grande*".

**Atributos Quantitativos:** são aqueles que descrevem as características de um objeto de forma quantitativa, ou seja, eles podem ser medidos e comparados de forma numérica. Os atributos quantitativos podem ser divididos em dois subtipos: Numéricos e Contínuos

**Atributos numéricos:** são atributos que podem assumir apenas um número limitado de valores. Por exemplo, o atributo "*número de filhos*" pode ter os valores 0, 1, 2, 3, etc, mas não pode ter os valores 2,5; 1,2; 3,6; etc.

**Atributos contínuos:** são atributos que podem assumir qualquer valor dentro de um intervalo. Por exemplo, o atributo "*peso*" pode ter qualquer valor entre 0 e 100 kg ou o atributo de “*altura*” que pode ter qualquer valor entre 1,40m e 2,00m.

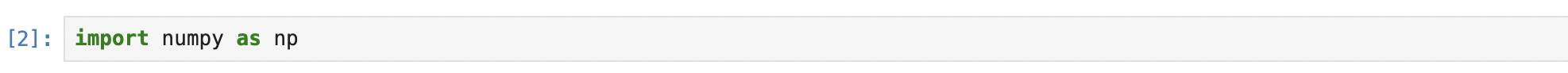
A compreensão dos diferentes tipos de atributos é importante para o processamento de dados. As técnicas de aprendizado de máquina podem ser mais ou menos eficazes a depender dos atributos. Com isso, vamos aprender a manusear os atributos e nossos conjuntos de dados em Python, utilizando algumas das bibliotecas mais conhecidas como: *Numpy*, *Pandas* e *Polars*. *Polars* não é tão conhecido, mas é uma biblioteca de manipulação de conjunto de dados, assim como Pandas. Contudo, há testes indicando que ele é mais rápido, podendo ser um grande competidor contra o Pandas em um futuro próximo.

### NumPy

*NumPy* é um pacote *Python* amplamente utilizado na computação científica. Esta biblioteca fornece funcionalidades e classes para se trabalhar com *arrays* multidimensionais, como rotinas de operação em *arrays*, matemática lógica, manipulação de forma, simulações aleatórias e operações de álgebra e cálculo.

A biblioteca possui uma otimização em C, o que garante uma boa performance com a facilidade do *Python*, e suas *arrays* possuem algumas diferenças comparadas com as da linguagem de programação, como por exemplo o tamanho fixo das *arrays*, seus elementos têm os mesmos tipos e por padrão possui diversas funções prontas para manipular esses *arrays*.

Pode ser instalada facilmente com o comando *"pip install numpy",* e logo após importá-lo no projeto*.*



Para seguir essa explicação utilizando o código, pode acessar o Notebook [Numpy](https://colab.research.google.com/drive/1dRVoquhOuyvnX3coQMbaq9a7Y2zGppmF?usp=sharing). A seguir, tem-se um exemplo mostrando a como se instancia vetores e matrizes em python e utilizando numpy.

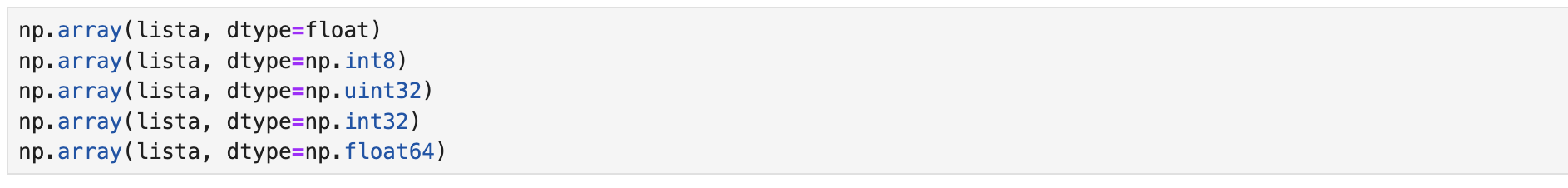






#### Tipos

O Numpy possui os tipos numéricos do C, o uso desses tipos deve ser informado no código dependendo da necessidade do projeto ou do uso. Por padrão todas as *arrays* são do tipo *float32*, mas se por exemplo for necessário utilizar valores menores pode-se configurar o padrão float16. Abaixo vemos exemplos de como definir esses tipos em um array.

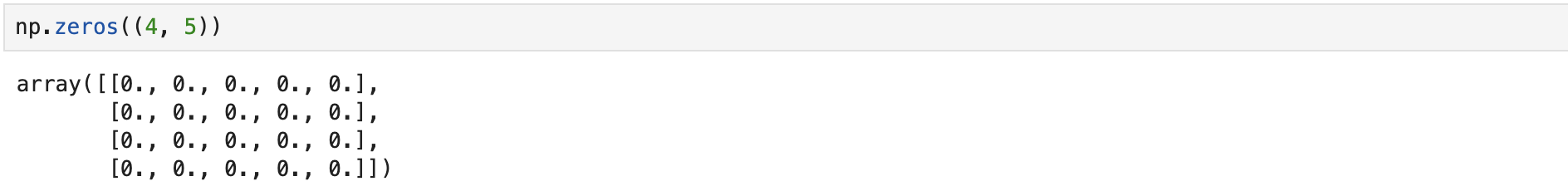


#### Criando Arrays

Existem algumas funções especiais para a criação de *arrays*, alguns exemplos são: *zeros*, *ones*, *arange*, *linspace* e *copy*.

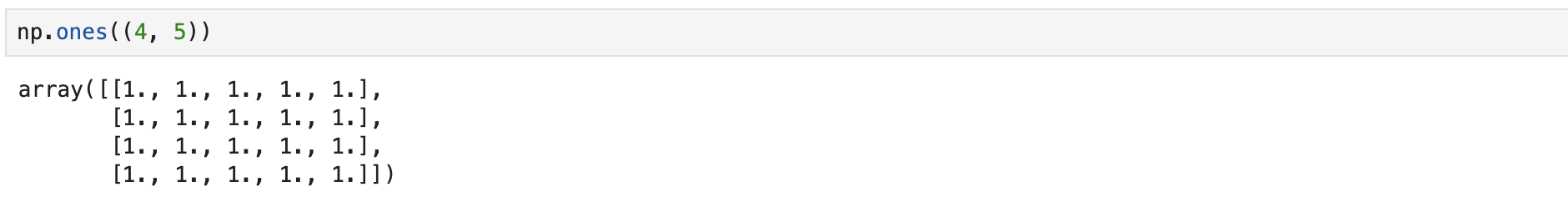
* Função *zeros()*

Esta função cria um *arrays* populado por valores 0, sendo necessário passar uma tupla com as dimensões desse *array*, como vemos no exemplo a seguir.

****

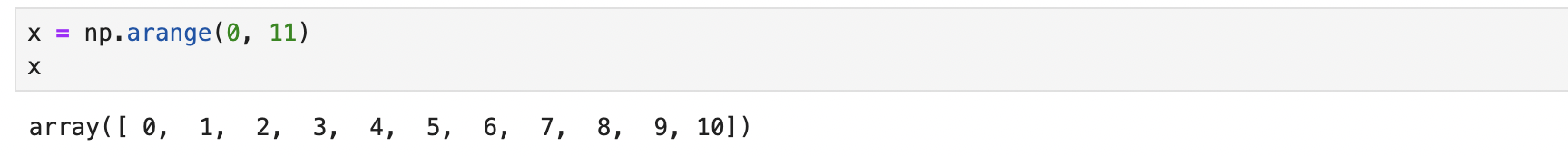
* Função *ones()*

Esta função por sua vez cria um array populado por valores 1, também sendo necessário passar como atributo uma tupla com as dimensões do array desejado, como pode-se ver no exemplo a seguir.

****

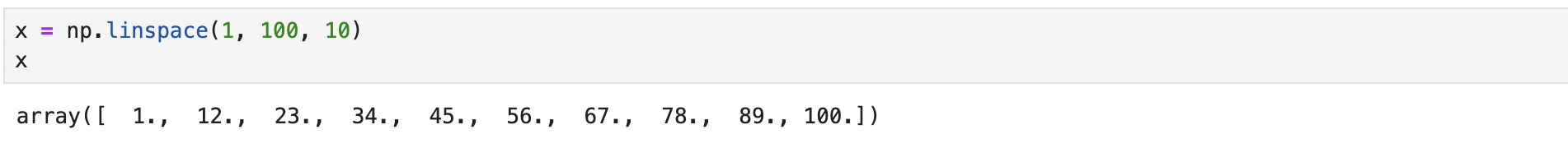
* Função *arange()*

Nessa função é possível criar um array de valores a partir de um intervalo. Nela são passados como atributo o início e o fim do intervalo, assim como o incremento desse intervalo, mas sendo o incremento opcional, como no exemplo a seguir.

****

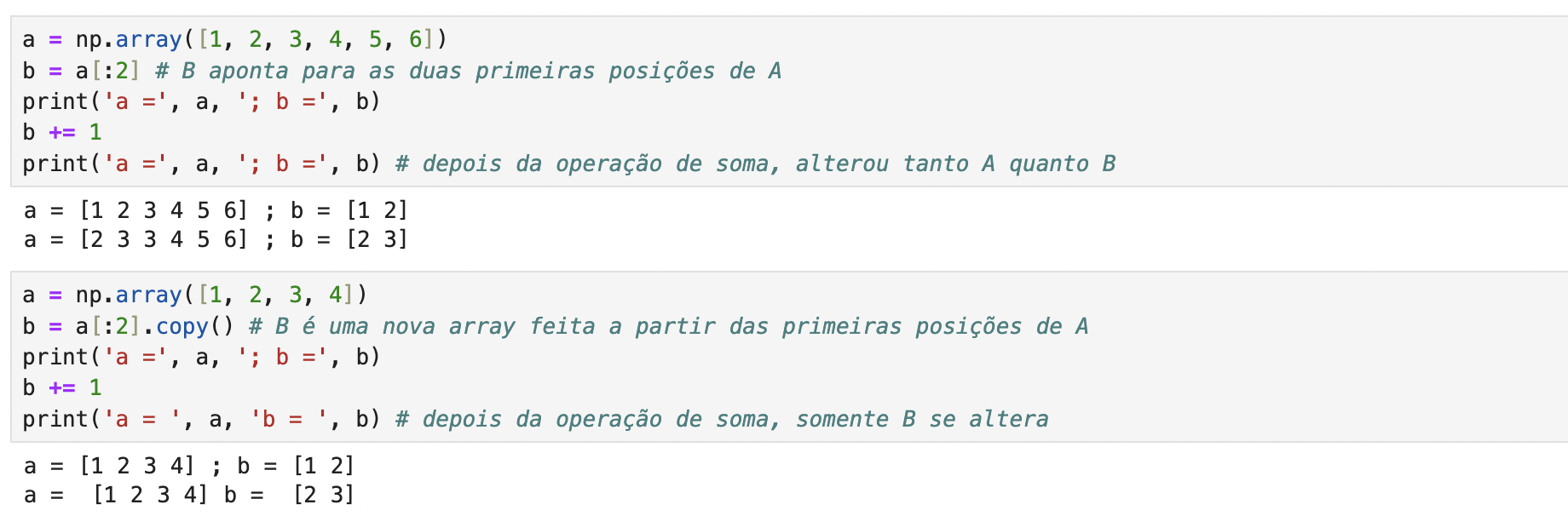
* Função *linspace()*

Na função *linspace()* gera um certo número de elementos em um certo intervalo de forma perfeitamente espaçada, recebe como atributos o menor e o maior valor do intervalo e também o número de elementos da array. Como exemplo tem-se a seguir um de seus usos.

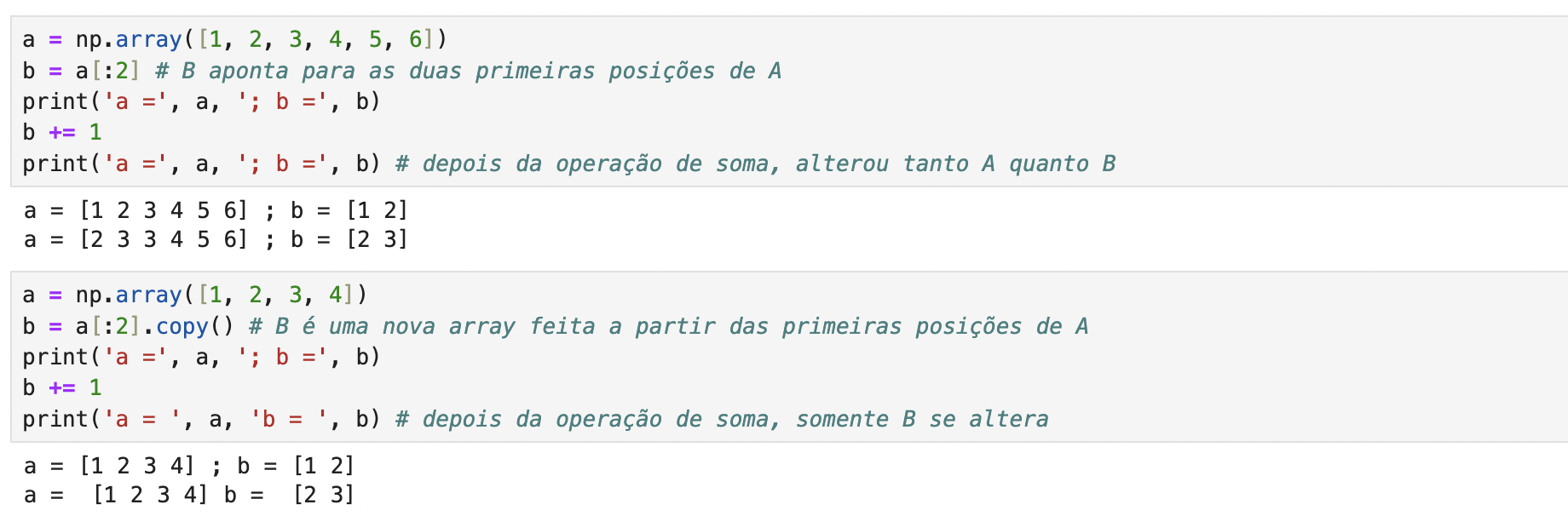
****

* Função *copy()*

Como podemos ver no exemplo a seguir, usar *slices* (ou até mesmo outros métodos) não gera uma cópia de uma *array* e sim armazena um ponteiro para aquela parte da *array* original.

****

Portanto, para se criar uma nova *array* a partir da original é necessário utilizar o método *copy()* para gerar uma nova *array* idêntica à primeira como vemos a seguir.

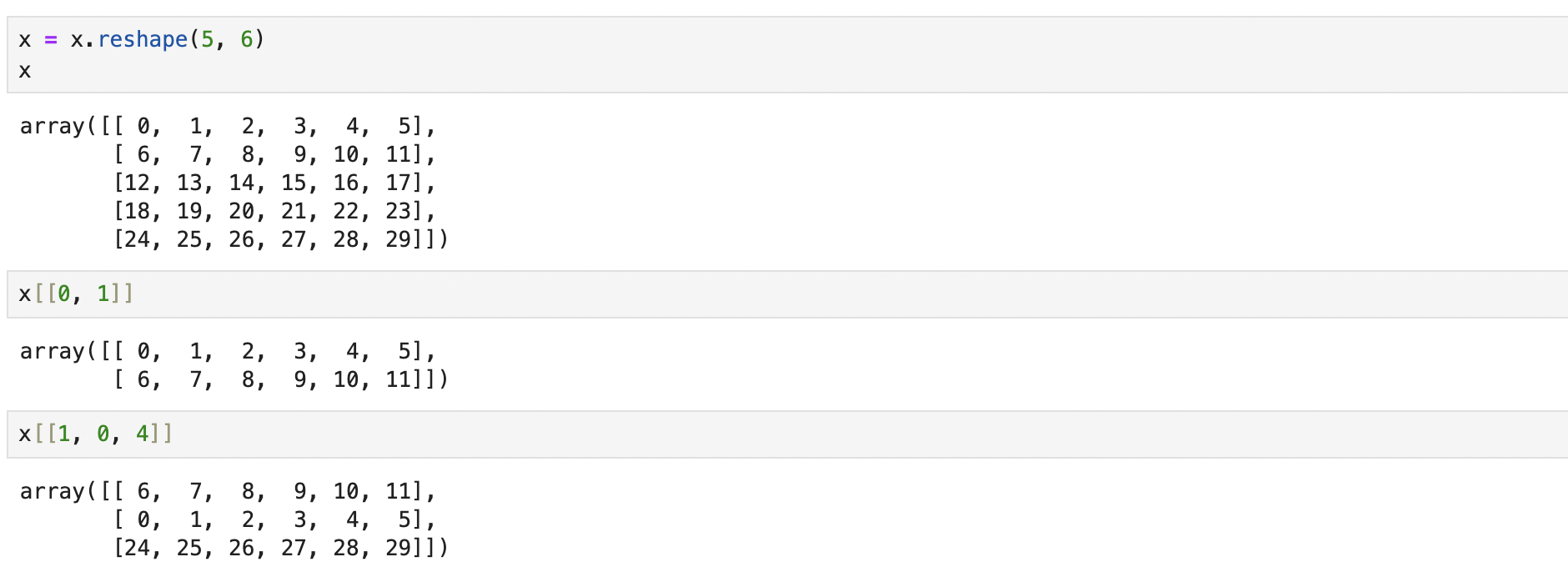
****

#### Indexação

Para indexar *arrays* no *numpy* é bem simples, o acesso dos valores é feito assim como nos *arrays* do *python*, como vemos a seguir.

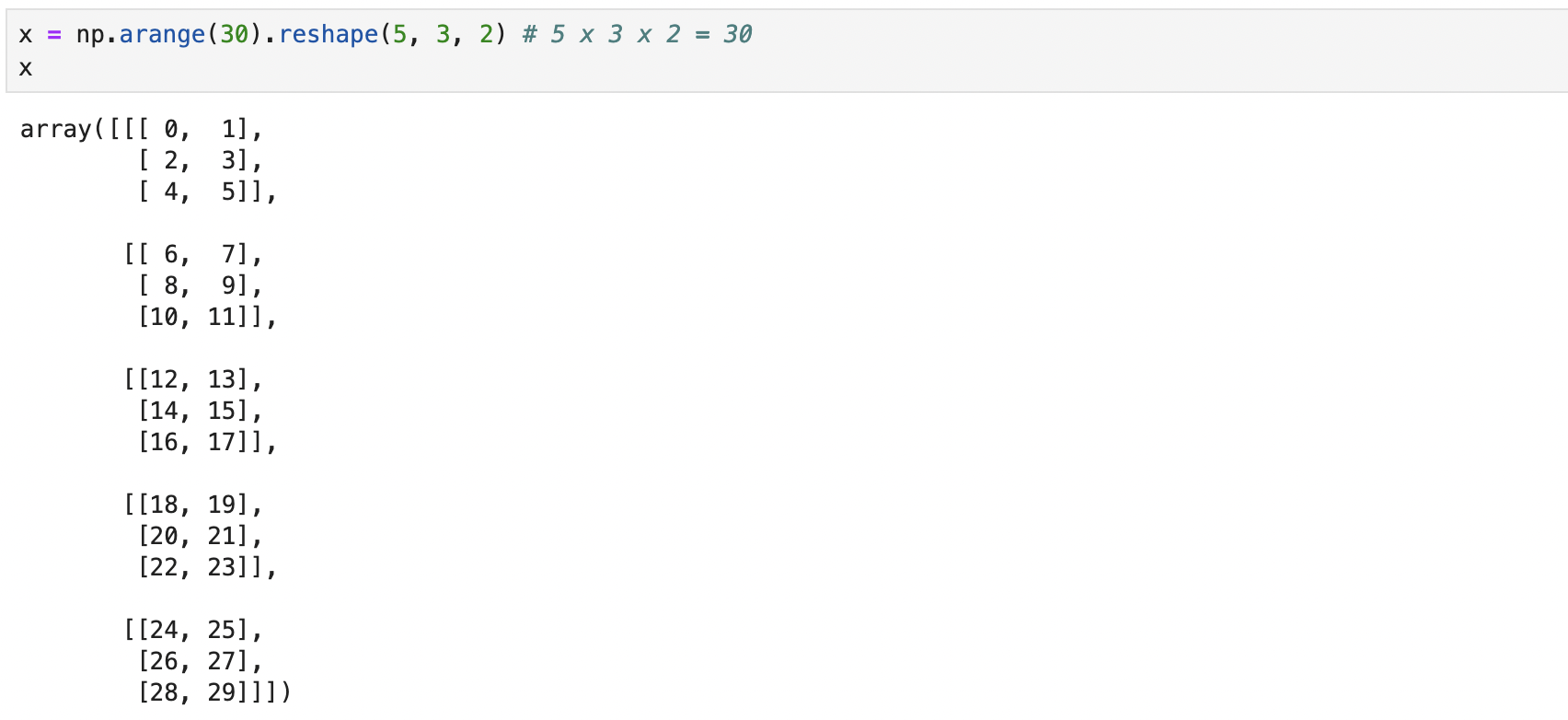


Também é possível indexar vários valores de uma vez em um conjunto de *arrays*, ou *arrays* com várias dimensões, como nos exemplos a seguir.



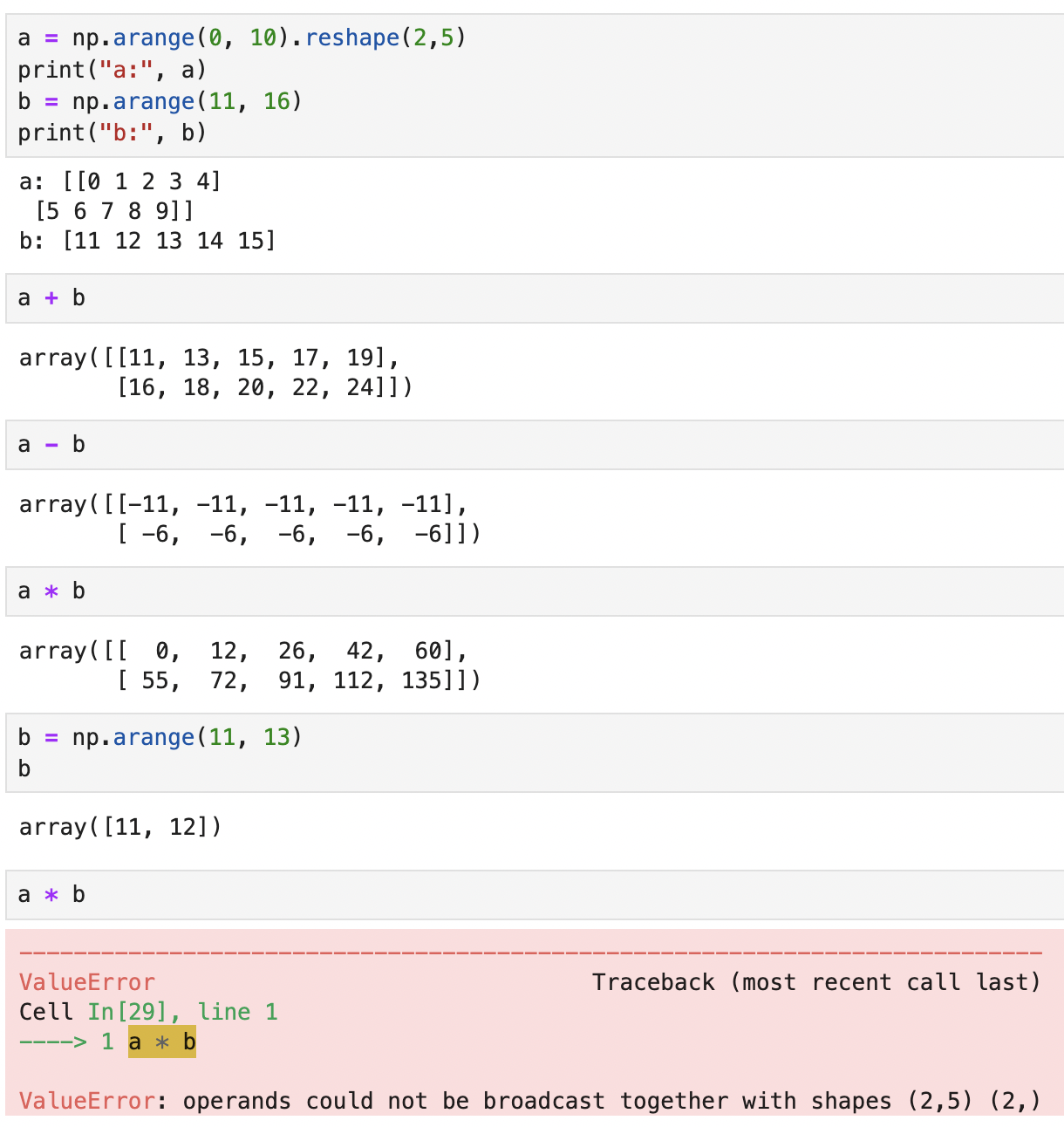
#### Reshape

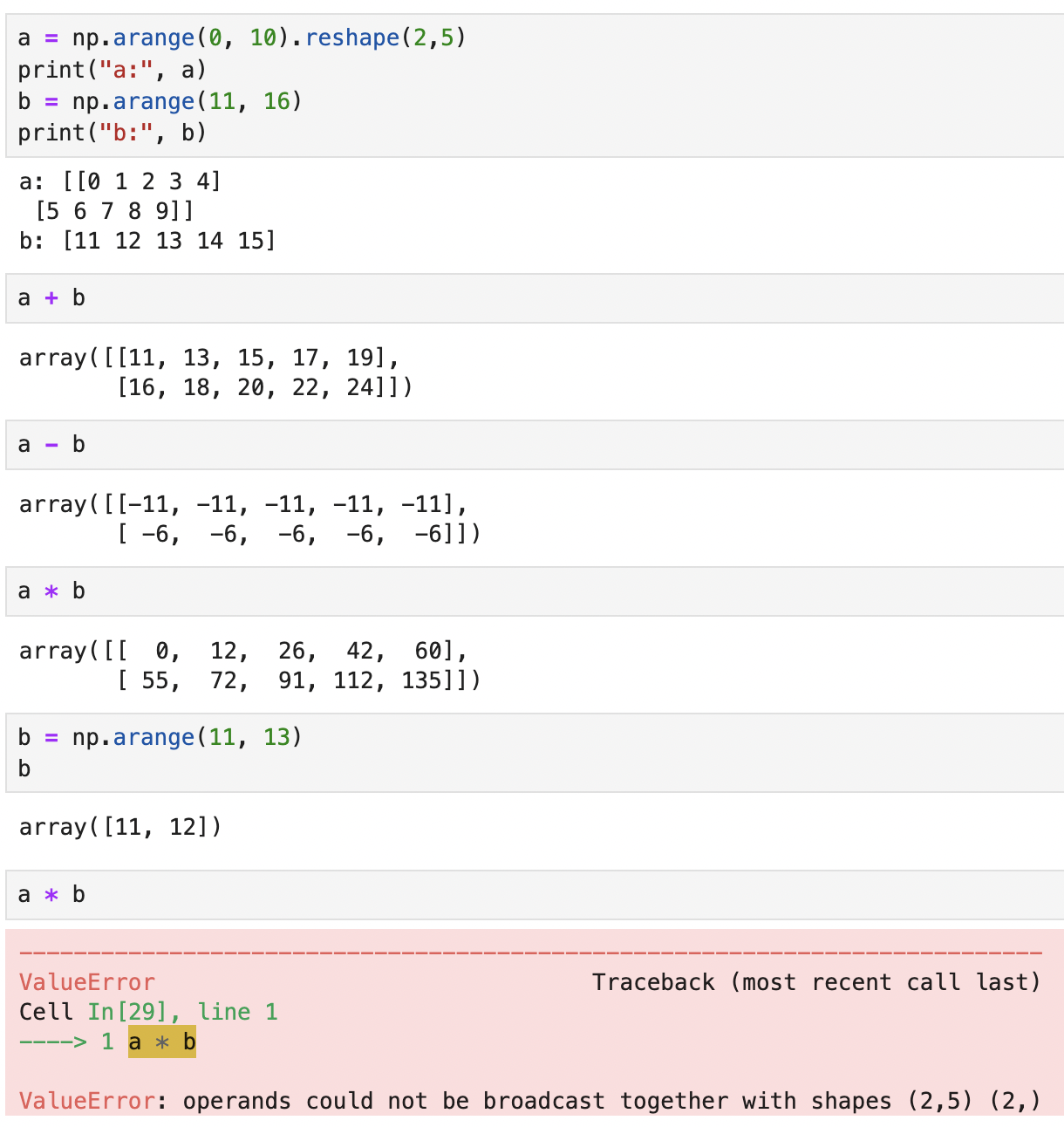
No treinamento de redes neurais é normal que haja a necessidade de mudar o formato do dado ou conjunto de dados para se adequar ao modelo (ou o contrário), e para isso a função *reshape*() funciona. Dado um *array* pode-se passar uma tupla com as novas dimensões da *array*, desde que se respeite a forma inicial dos dados. Um *array*  pode ser remodelado para por exemplo. Um exemplo prático disso está a seguir.

****

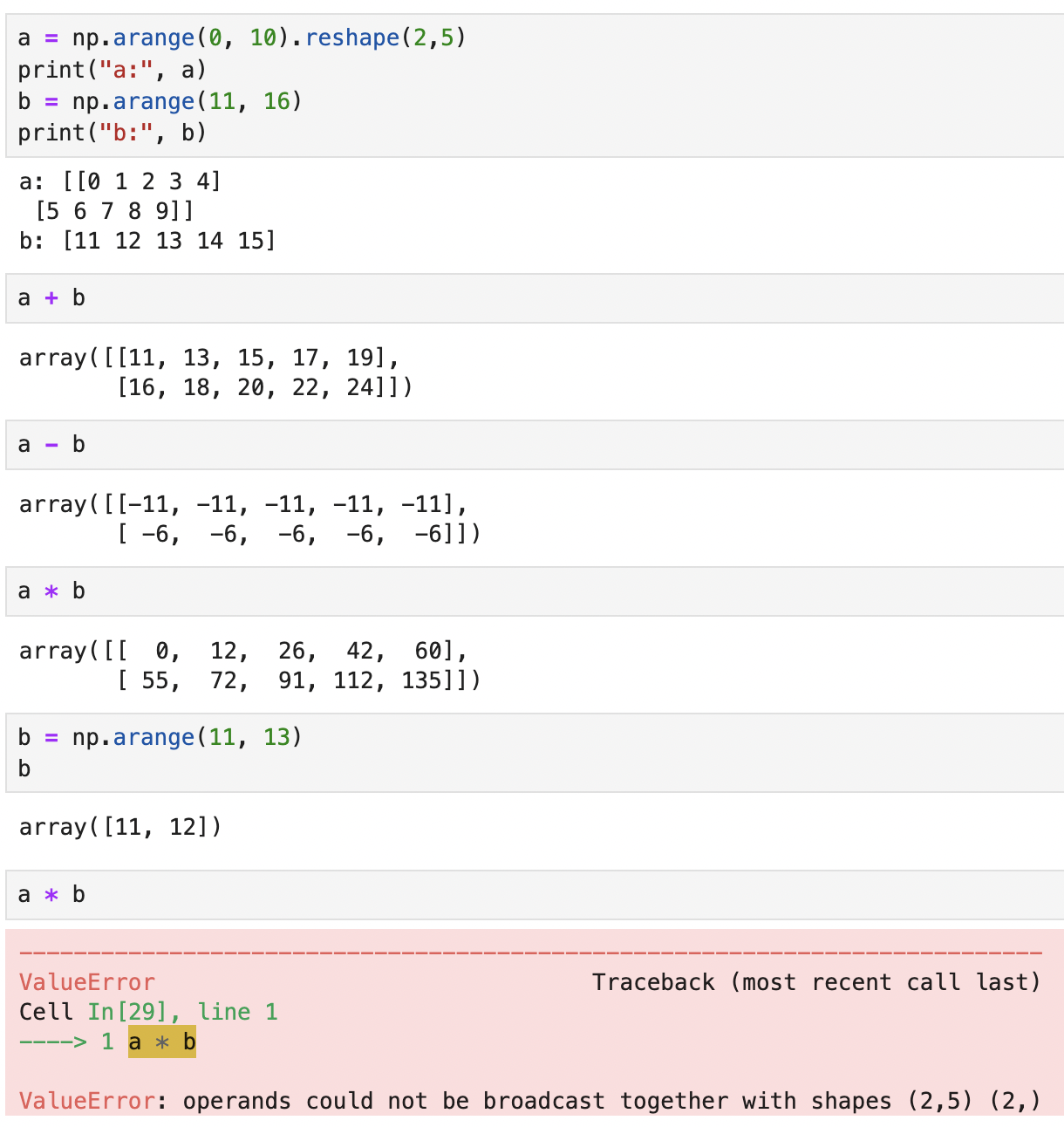
#### Operações

O *Numpy* também permite operações entre matrizes, no exemplo abaixo vemos dois *arrays* sendo criados e em seguida somados e subtraídos.

****

****

Porém, como no exemplo abaixo, é importante notar que a multiplicação de *arrays* deve seguir os princípios matemáticos como na multiplicação de matrizes, onde o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz.

****

E também é possível fazer operações com todos os elementos de um *array* (ou uma determinada parte) de uma só vez usando *Broadcasting*, como mostrado no exemplo a seguir.



* + - 1. **Retornando informações**

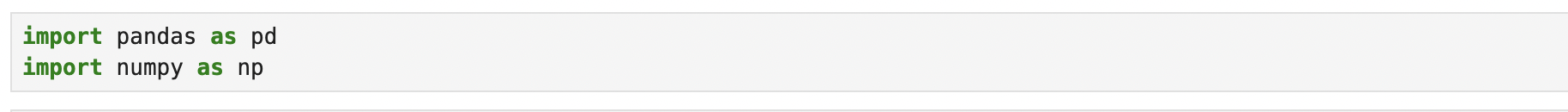
No N*umpy* também é possível retornar algumas informações da *array* como por exemplo, o número de dimensões (*ndim*) ou o formato da *array (shape),* entre outros. Como pode-se ver abaixo.



### Pandas

*Pandas* é uma ferramenta de análise e manipulação de dados de código aberto rápida, poderosa, flexível e fácil de usar, construída sobre a linguagem de programação *Python*. Com ela é possível ler conjuntos de dados em arquivos .csv, .xls, JSON e entre outros, além de possuir diversas funções para trabalhar e tratar esses dados, sejam numéricos ou não.

Pode ser instalado facilmente com o comando "*pip install pandas",* e logo após importá-lo no projeto*.*



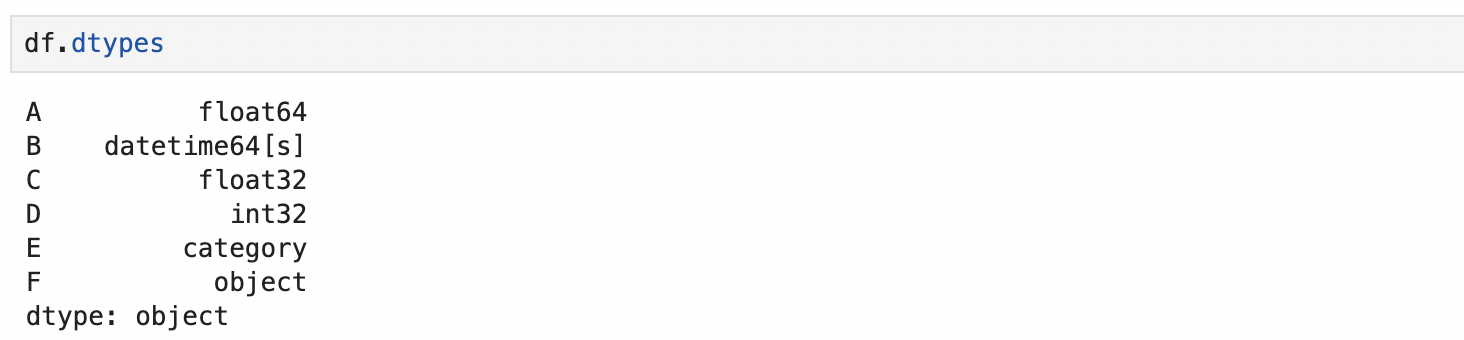
#### Criando DataFrames

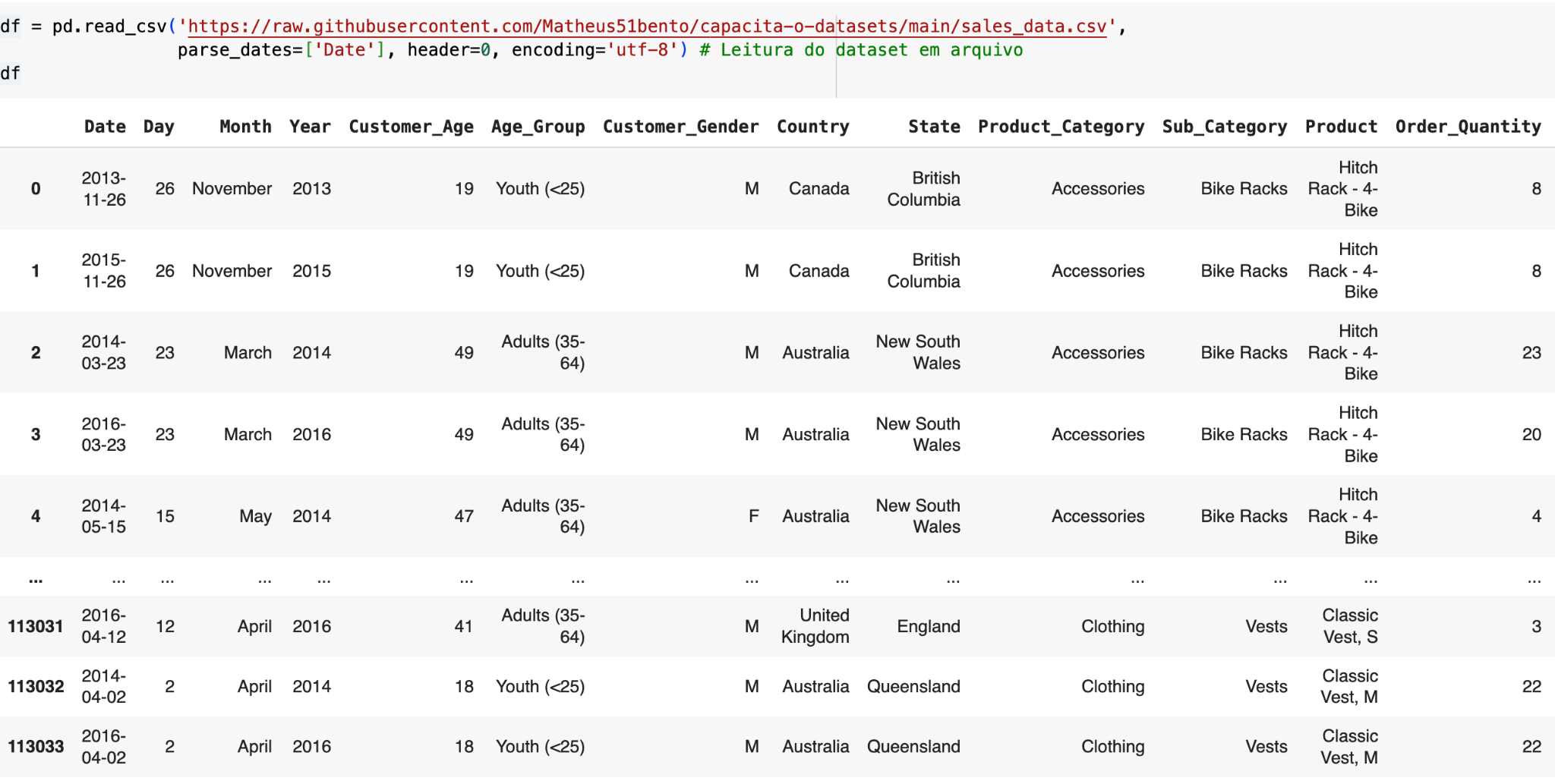
Lembrando que essa explicação pode ser acompanhada pelo Notebook [Pandas](https://colab.research.google.com/drive/1oMFs3TxKfIt1WC9eBrm4EN4GyGfkzxrJ?usp=sharing).

No *Pandas*, os conjuntos de dados são denominados *dataframes*, e além das matrizes de dados podem ou não possuir cabeçalhos, e para criá-los de forma simples pode-se passar um dicionário onde as *keys* são o cabeçalho da coluna, e seus valores são os elementos da tabela como vemos no exemplo a seguir:



No exemplo acima, pode-se observar uma grande variedade de tipos de dados aceitos ou instanciados pelo próprio pandas, que vão desde números racionais, até datas ou strings. Os tipos apresentados em cada coluna pode ser vistos de forma prática com o comando a seguir.

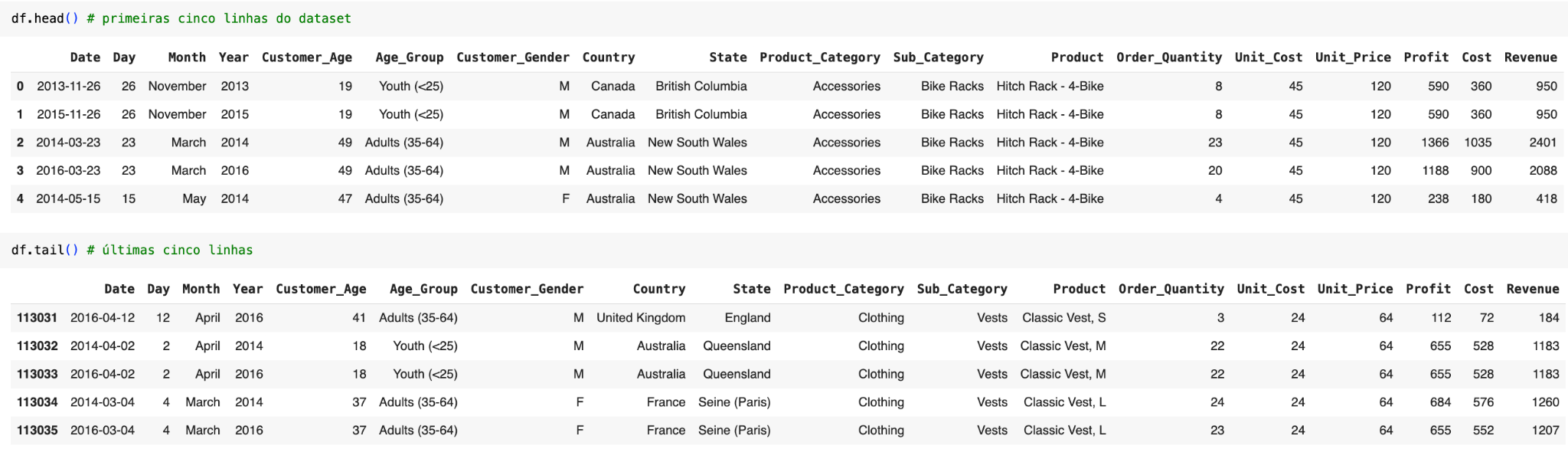


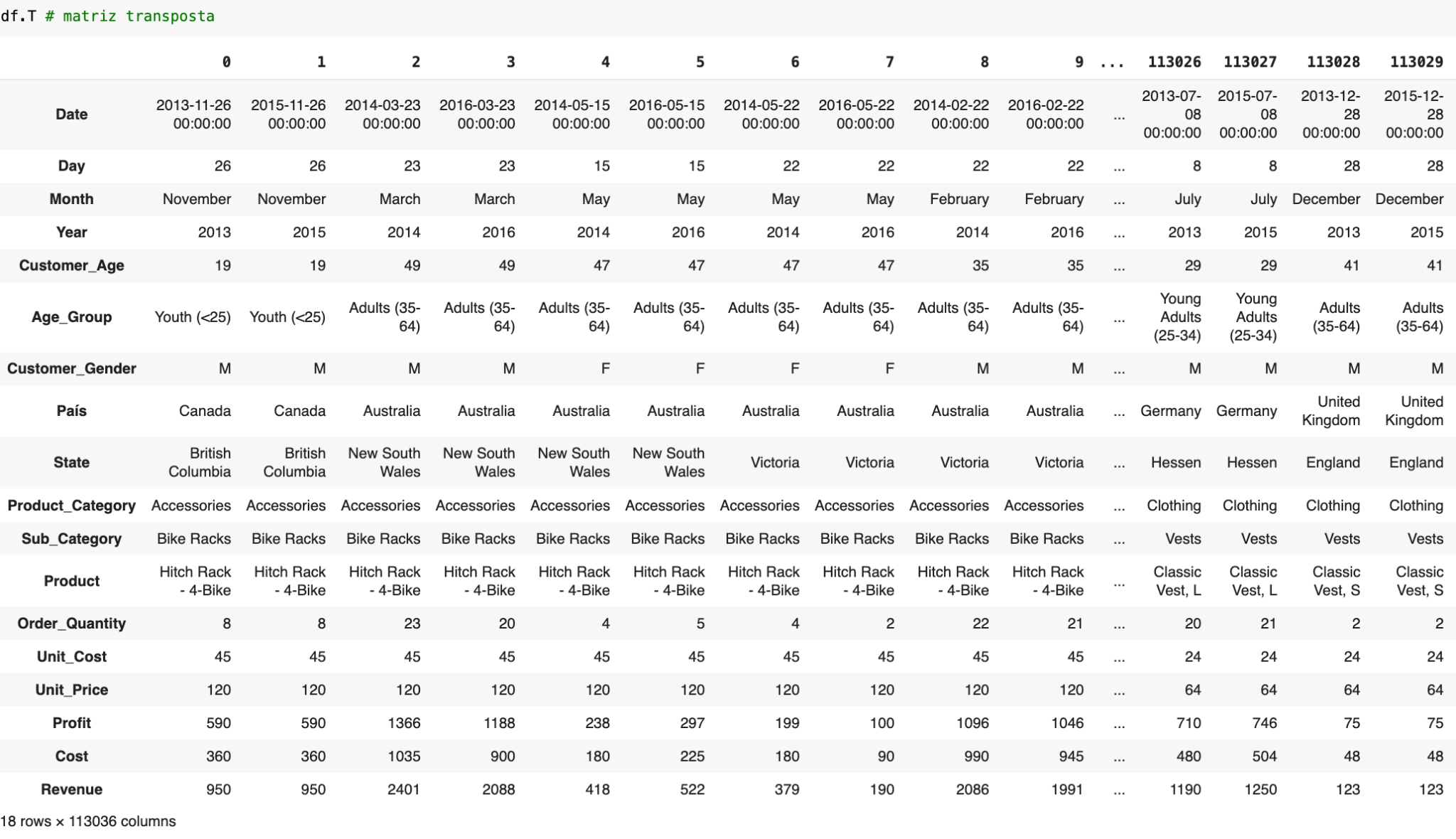
Outro ponto é que esses *DataFrames* podem ser importados de arquivos como no exemplo a seguir.  


Pode-se observar que a função *read\_csv*() recebe como parâmetros o caminho até o arquivo, a linha da tabela onde se encontra o cabeçalho, e nesse caso a forma de codificação dos caracteres. No exemplo acima foi importado um *dataset* com a informação de todos os países que disputaram jogos olímpicos de verão ou inverno, bem como as medalhas conquistadas.

#### Visualizando Dados

O Pandas possui funções built-in para mostrar os dados, para ajudar no processo de trabalho com eles. Exemplos dessas funções são a *head()* que mostra as primeiras 5 linhas do *dataframe* e a função *tail()* que mostra as últimas 5 linhas. Como pode-se observar no exemplo a seguir.

****

Outro método que pode ser utilizado de forma visual ou até mesmo prática é o método T, que transpõe o *dataframe*.  


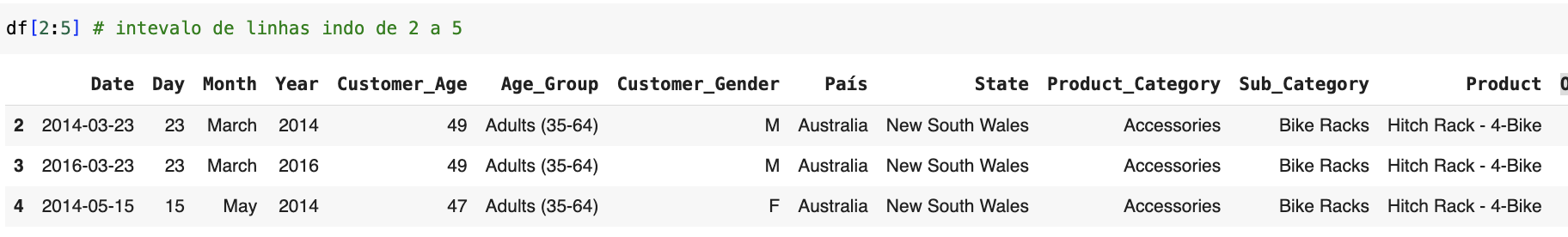
A biblioteca permite também mostrar os dados separadamente ou em conjunto de cada linha ou coluna como no exemplo a seguir.

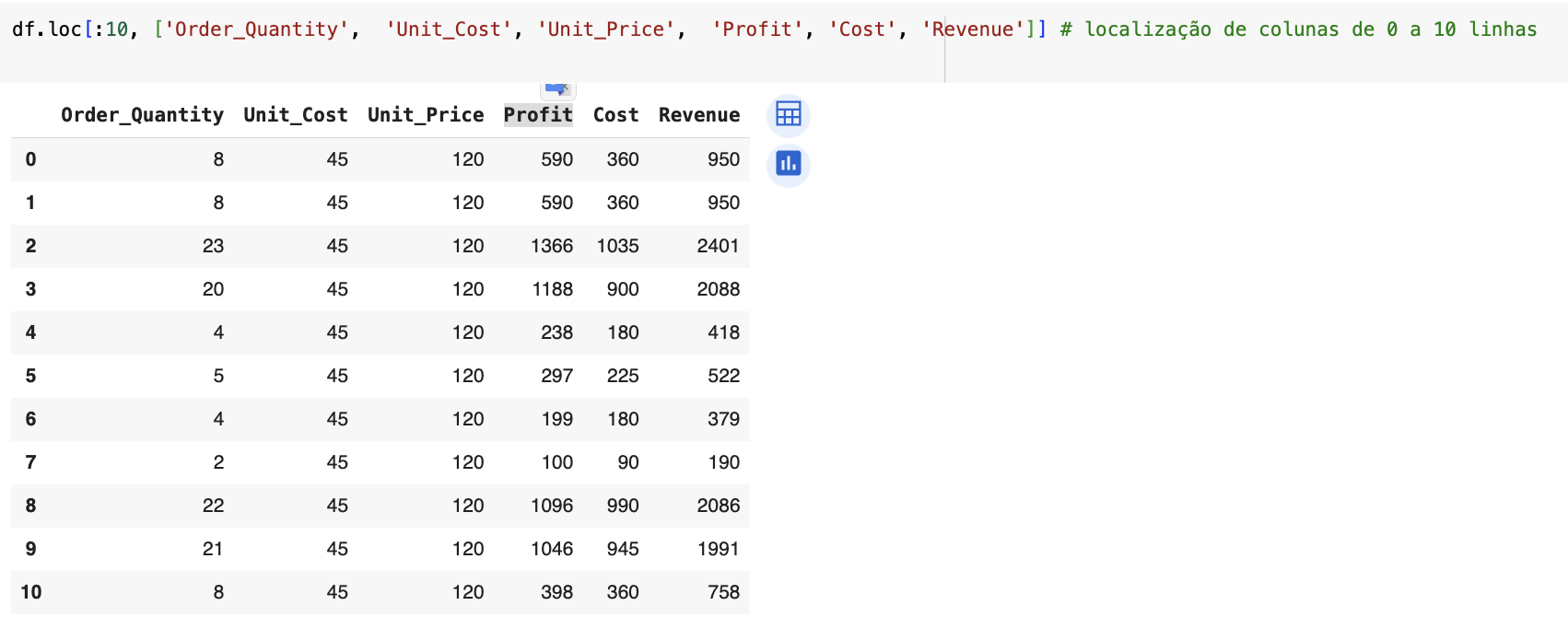


Assim como permite renomear as colunas, nesse caso como não havia nenhum valor no cabeçalho para a primeira coluna, o pandas nomeou automaticamente para "*Unnamed: 0*", mas pode-se alterar isso como mostrado abaixo.

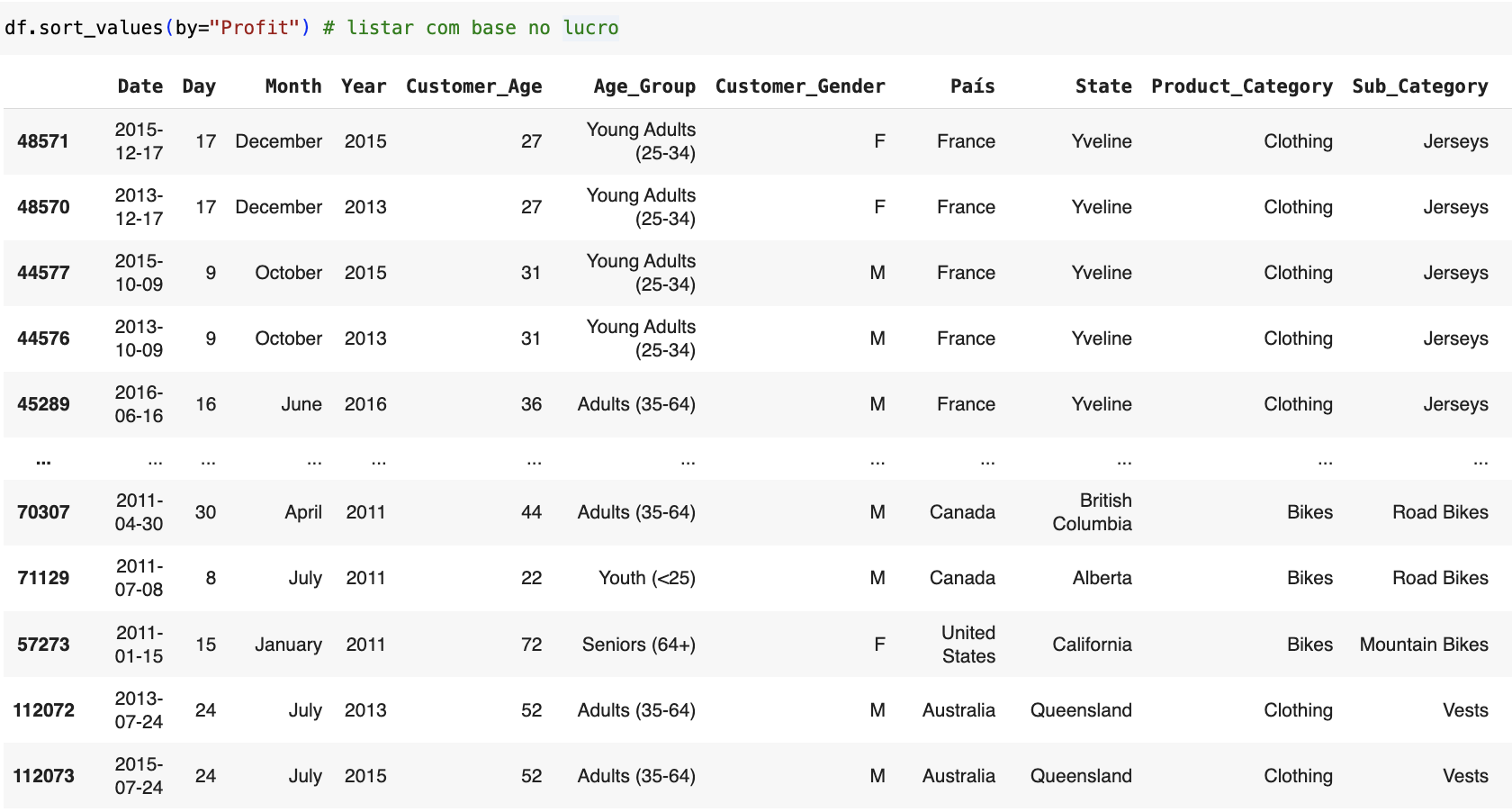


Os valores também podem ser mostrados de forma indexada, como no exemplo abaixo que utiliza slices.

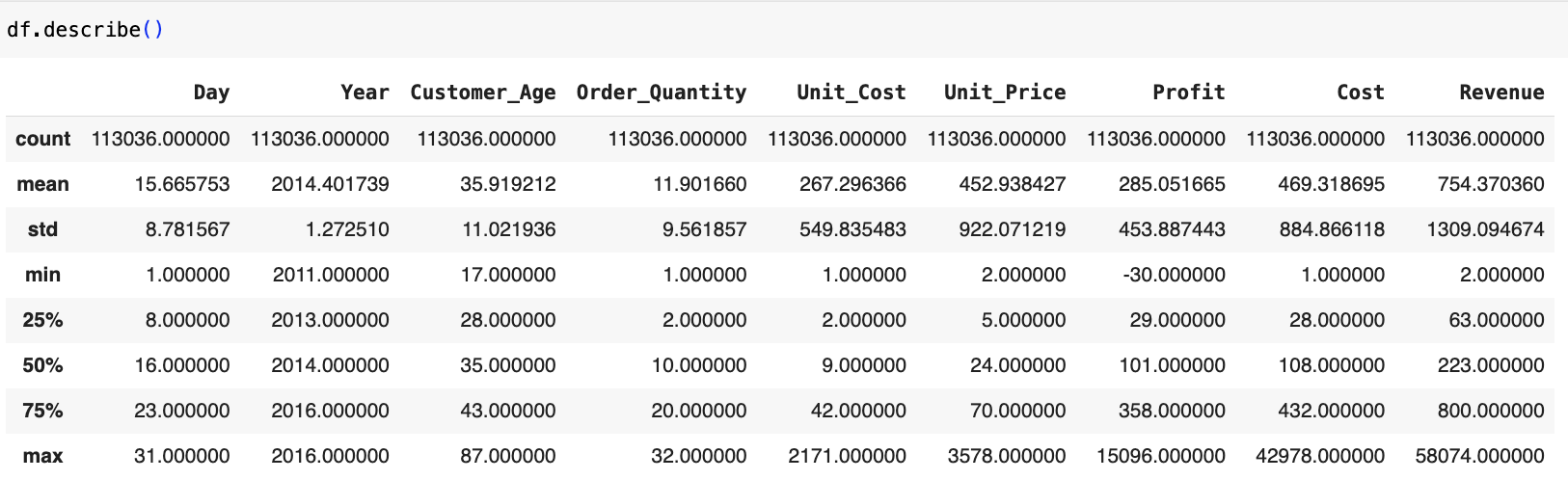
 Outra forma de fazer isso seria com a função *loc()*, pode não só indexar as linhas com um *slice* mas também pode indexar as colunas passando uma lista com seus nomes, como no exemplo a seguir.



Outro método visual e prático é o *sort\_values()* onde é possível selecionar uma coluna e a ordem, crescente ou decrescente. Como no exemplo abaixo onde foi reordenado pelos países com mais medalhas.



E por fim temos o *describe()* que mostra valores ou algumas métricas de cada coluna do *dataframe* como podemos ver a seguir.



### Polars

*Polars* é uma biblioteca *Python* e *Rust* projetada para trabalhar com estruturas de dados tabulares na forma de *dataframes*, semelhantes aos *dataframes* do *Pandas*. Distingue-se por ser implementado em *Rust*, o que permite processamento e manipulação de dados de alto desempenho. Além disso, *Polars* usa o formato colunar *Apache Arrow* como modelo de memória, permitindo o processamento eficiente de dados em formato colunar.

Podendo ser instalado facilmente com o comando "*pip install polars",* e logo após importá-lo no projeto*.*



Para acompanhar essa prática pode-se utilizar o Notebook [Polars](https://colab.research.google.com/drive/1y5k7AxJhXMfslgMoo06gRG_9o4WE4Z07?usp=sharing).

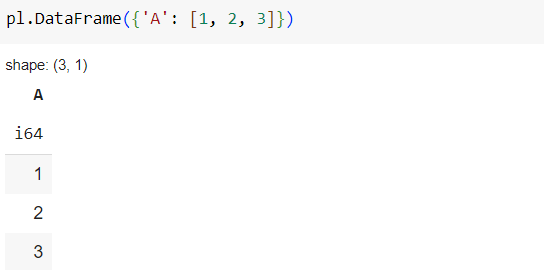
#### Criando Dataframes

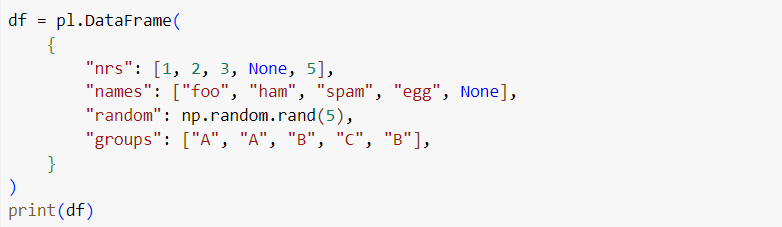
Funções para criação de DataFrmes:

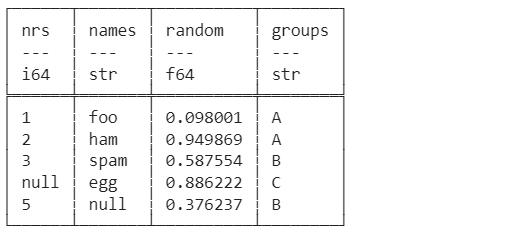
* *DataFrame()*
* *read\_csv()*

* **Função *DataFrame()***

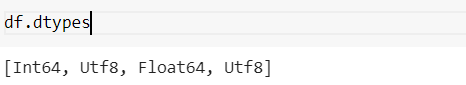
No *Polars*, os conjuntos de dados são chamados de *DataFrames*, e eles compartilham semelhanças com os *DataFrames* do Pandas. Assim como no *Pandas*, os *DataFrames* no Polars podem ou não ter cabeçalhos, e você pode criar *DataFrames* de maneira simples, usando um dicionário onde as chaves representam os nomes das colunas e os valores correspondem aos elementos da tabela, como ilustrado no exemplo a seguir:



******

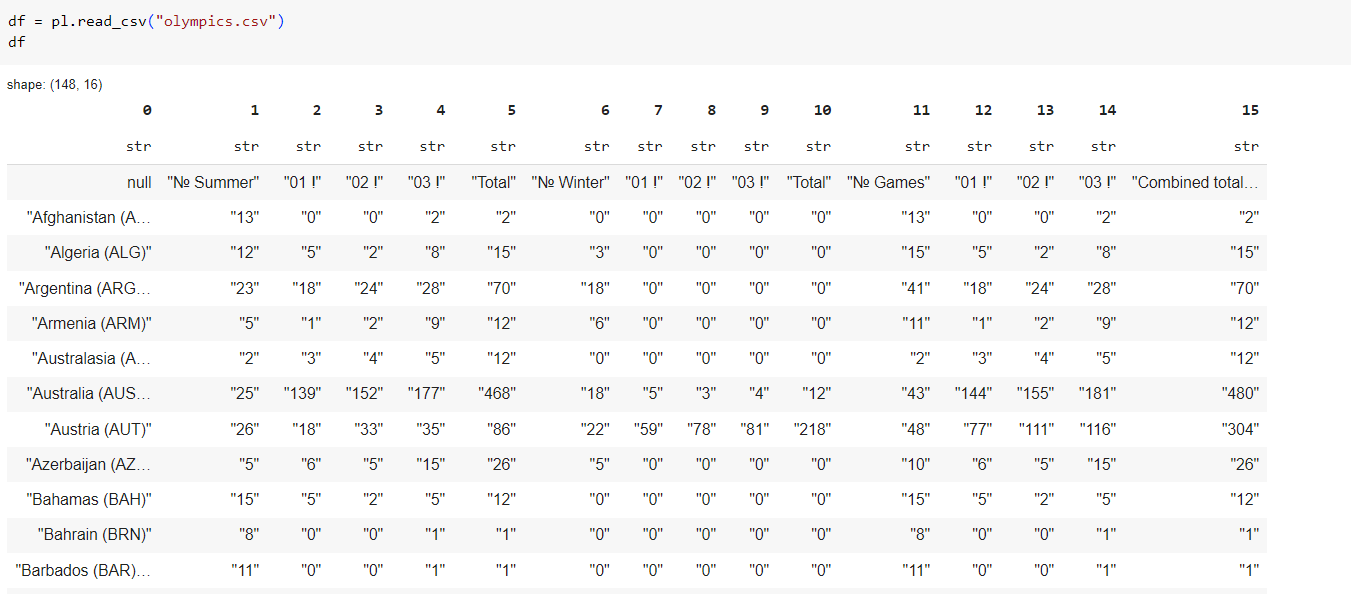
******

Acima, podemos ver vários tipos de dados aceitos pelo *Polars*, tanto quanto o pandas. Contudo, diferentemente do pandas, o Polars mostra o tipo de cada atributo abaixo do nome de cada um. Ainda assim, podemos verificar o tipo de cada atributo com o seguinte comando:



* **Função *read\_csv()***

Outro jeito de criar *DataFrames* no Polars, é importante eles de arquivos tais como: *Csv, Parquet e Json*.



Aqui, o método “*read\_csv*” é parecido com o do pandas, contudo, o único parâmetro necessário é o caminho até o arquivo, os outros são definidos por *default*. Aqui, diferentemente do pandas, o Polars não identificou o nome das colunas no csv, assim, temos que chamar cada coluna pelos índices.

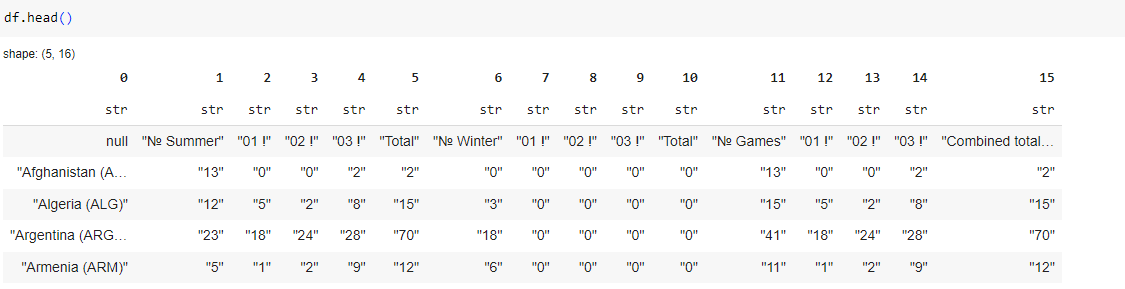
#### Visualizando Dados

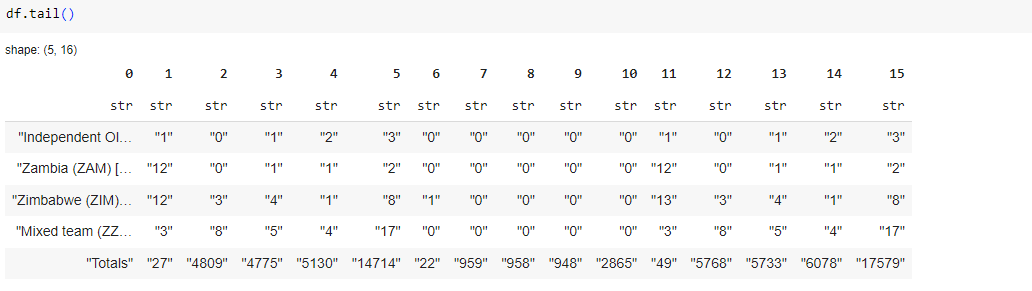
Funções para visualização de dados:

* *head()*
* *tail()*
* *transpose()*
* *select()*
* *sample()*
* *describe()*
* *concat()*

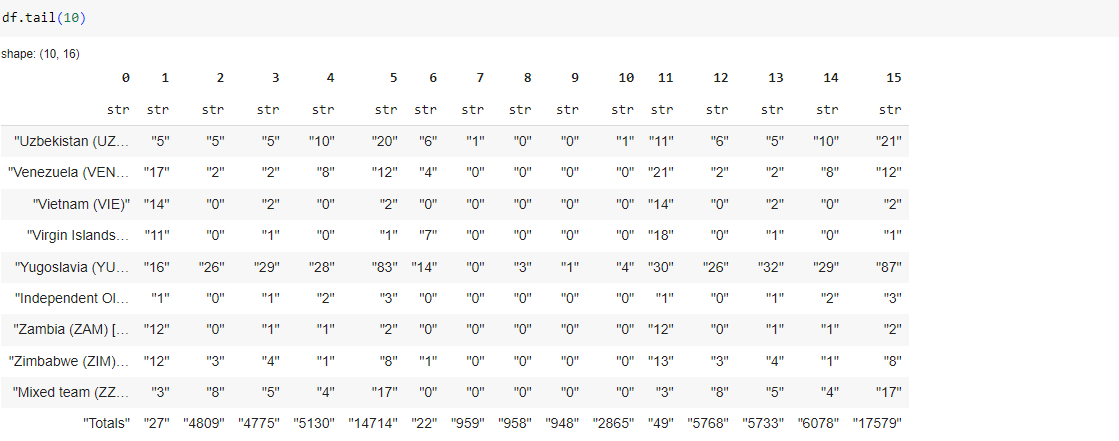
* **Funções *head()*, *tail()* e *transpose()***

No *Polars*, também existem funções built-in parecidas com o Pandas. As funções *head e tail* têm funcionamento idêntico ao *Pandas*. Como mostrado a seguir:

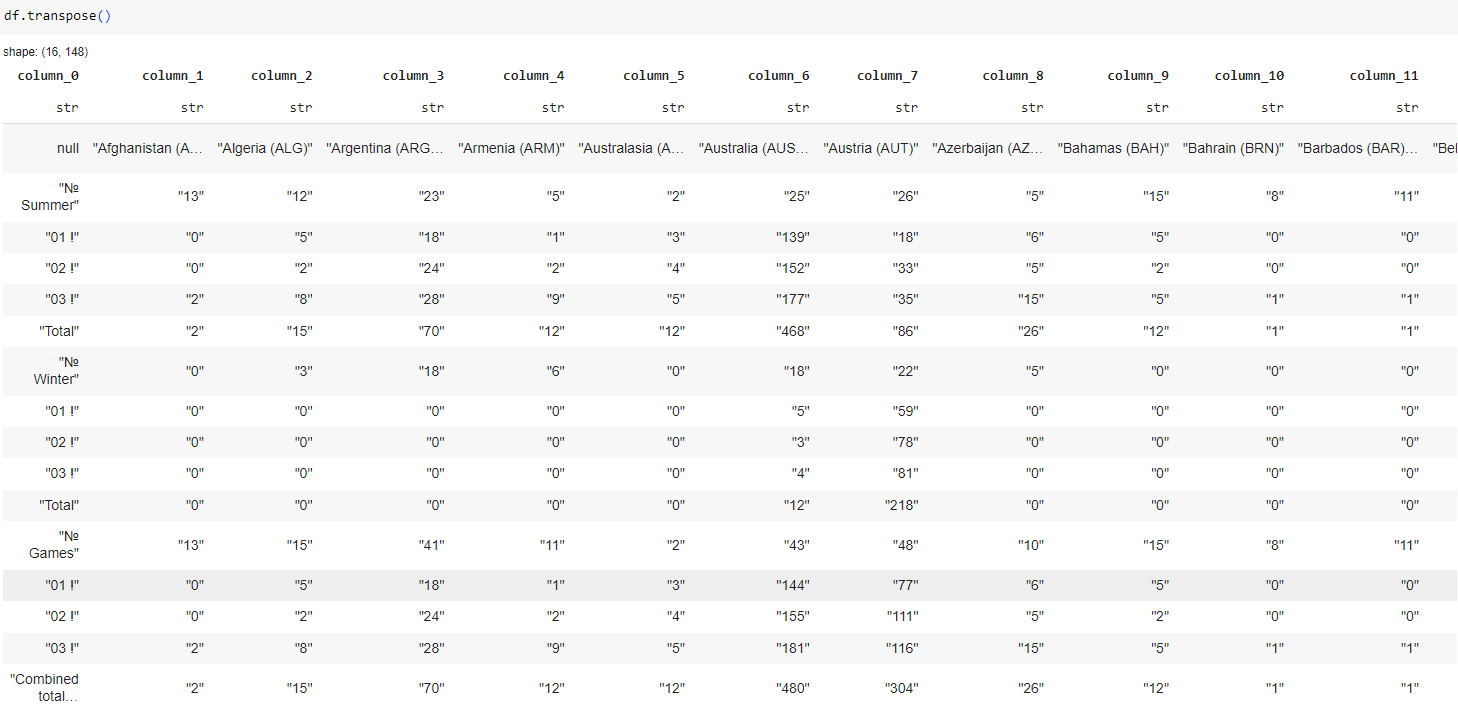




Em ambas funções *head() e tail()* caso você queira mais do que as 5 primeiras instâncias, você pode especificar nos parâmetros. Segue abaixo um exemplo do tail com 10 instâncias da tabela.



Para transpor a tabela, diferentemente do *Pandas*, aqui utilizamos o método “*transpose*”. Como mostrado abaixo:



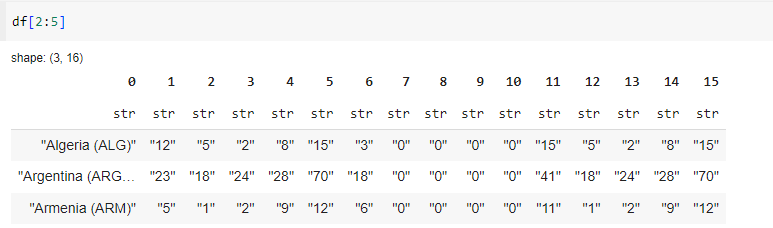
* **Função *select()***

Para pegar uma coluna em específico, como não há nome nas colunas, assim, precisamos recuperar elas pelo índice de cada uma, o que pode ser feita da seguinte forma no Polars:



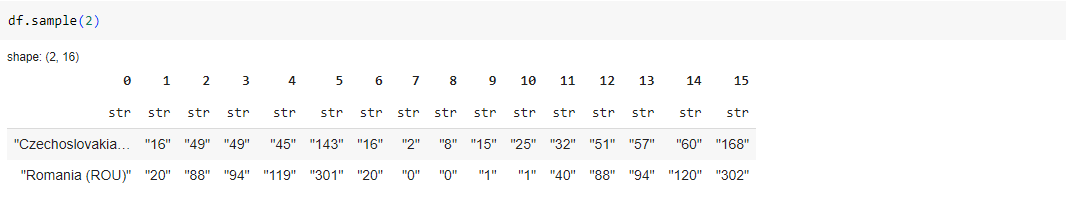
Como não havia nome da coluna nos países, a biblioteca auto completou com “*null*” no momento em que leu o csv.

Para buscar instâncias específicas, o Polars permite buscar por slices, assim como o *Pandas*. Como mostrado abaixo:



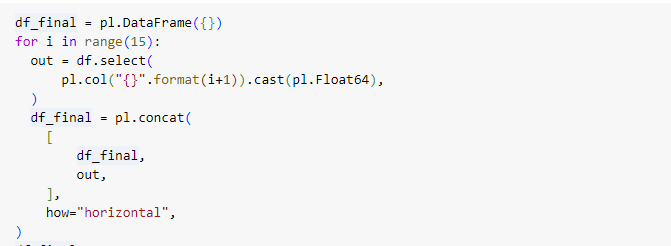
* **Função *sample()***

Outro método interessante no Polars é que, caso você queira 2 instâncias aleatórias, pode utilizar o método “*sample*” que tem como parâmetro a quantidade de instâncias que você deseja. Retornando x instâncias aleatórias do *dataframe*. Veja:

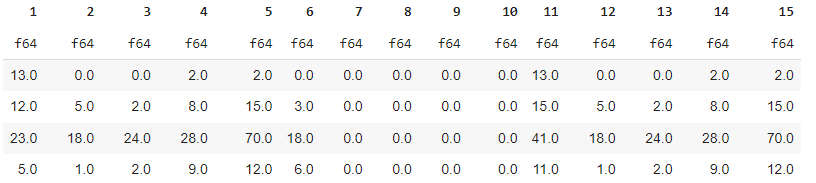


* **Função *describe()***

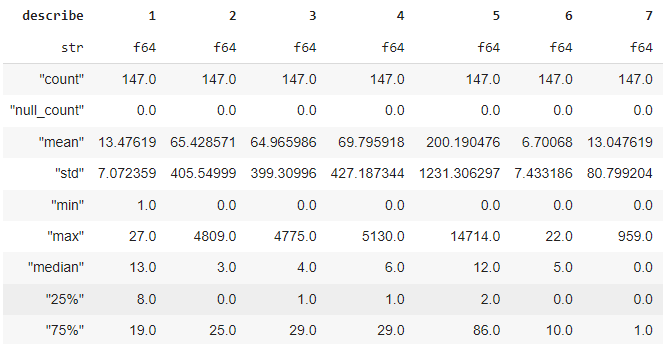
O “*describe*” é, com certeza, um dos métodos mais úteis quando estamos analisando uma base de dados, com ele é possível ter estatísticas resumidas do seu *DataFrame*. Ele fornece várias estatísticas rápidas, se possível. Para aplicar a função “*describe*” em nosso modelo, precisamos transformar os nossos dados para *float* ou *int*, pois na leitura do csv, o Polars leu os valores como *string*. Para isso, foi utilizado o seguinte código:



Após essas linhas de código, o *dataset* final ficou:

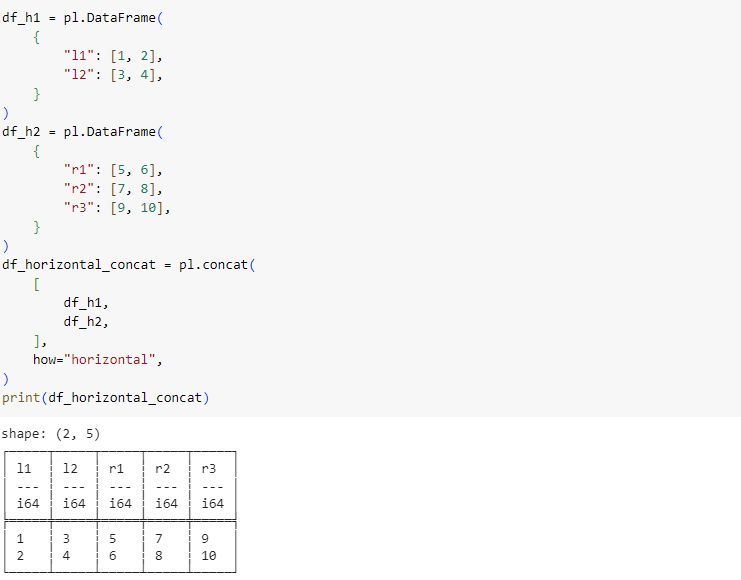


Assim, é possível utilizar o “*describe*” para conseguir as métricas estatísticas de cada coluna, veja:



* **Função *concat()***

A função “*concat()*” faz a concatenação de um ou mais datasets, se tiverem o mesmo número de instâncias, se usa a concatenação horizontal, se tiver o mesmo número de atributos, utiliza-se a concatenação vertical, como mostra o exemplo a seguir:



## Projeto prático I

| **Exercício Comentado 1** |
| --- |
| Algoritmos de ordenação são algoritmos utilizados para organizar conjunto de  dados em ordem específica, como crescente ou decrescente. Existem diversos algoritmos de ordenação, e eles variam em eficiência, facilidade de implementação e uso de recursos da máquina.  O Quicksort é um dos algoritmos de ordenação mais eficientes e amplamente utilizados. Ele segue a abordagem "dividir e conquistar" para organizar um conjunto de dados, bem parecido com a lógica por trás da busca binária.  Basicamente, o quicksort escolhe um dos elementos do vetor como pivô e a partir desse pivô o vetor é dividido em dois vetores menores, um com elementos menores que o pivô e outro com elementos maiores que o pivô. É aqui que ocorre a parte da “divisão” do dividir e conquistar.  Podemos perceber que a alma do algoritmo se dá na escolha do pivô e que caso seja escolhido um pivô ruim (ser o maior elemento do vetor), o nosso algoritmo demora mais para ordenar. Assim, existem várias versões diferentes do algoritmo do quicksort que escolhem o pivô de diferente maneiras:   * Sempre escolher o primeiro elemento como pivô * Sempre escolher o último elemento como pivô * Escolher um elemento aleatório como pivô * Escolher a mediana de três elementos do vetor como pivô   Caso escolhemos a medina de três elementos do vetor como pivô evitamos que o algoritmo caia no seu pior caso.  O segundo processo chave da ordenação usando quicksort é a parte de particionar o vetor. O objetivo de particionar o vetor é que dado um valor x sendo o pivô, colocar o x no seu lugar correto do vetor já ordenado, ou seja, colocando todos os elementos menores que x do lado esquerdo do vetor e todos os elementos maiores que x no lado direito do vetor. Assim, tendo essas duas partições do vetor, basta usar o mesmo método nelas. Podemos ter um exemplo de quicksort com a escolha do pivô sendo a mediana pode ser visto na imagem a seguir:    Você recebe um array NumPy de números inteiros e precisa implementar o algoritmo QuickSort para classificar o array em ordem crescente. Implemente a função quicksort\_numpy(arr) que aceita o array NumPy como entrada e retorna o array classificado. Certifique-se de que sua função seja eficiente e utilize as funcionalidades do NumPy para otimizar o processo de classificação. |
| Resposta: |

| **Exercício Comentado 2** |
| --- |
| Muitas vezes no mundo da ciência de dados e da inteligência artificial, nos deparamos com dados que precisam ser tratados, o que vai ser mais visto e com mais detalhes no próximo módulo. Grande parte dos algoritmos de inteligência artificial não lidam bem com dados categóricos, assim, surge a necessidade de transformar variáveis categóricas em variáveis discretas.  Muitas bases de dados possuem variáveis dicotômicas, ou seja, só existem dois valores, essas variáveis, podem ser facilmente transformadas em valores de 0 e 1. Além disso, caso haja variáveis com mais de dois valores, podemos indicar valores discretos para cada um deles, essa técnica é chamada como Label encoding.  Dada uma base de dados, disponível em: <https://www.kaggle.com/datasets/uom190346a/sleep-health-and-lifestyle-dataset> , ou, pode ser baixada da pasta do drive disponibilizada por nós.  Sleep Health and Lifestyle Dataset é um conjunto de dados composto por 400 linhas e 13 colunas, abrangendo uma ampla gama de variáveis relacionadas ao sono e hábitos diários. Ele inclui detalhes como gênero, idade, ocupação, duração do sono, qualidade do sono, nível de atividade física, níveis de estresse, categoria de IMC (Índice de Massa Corporal), pressão sanguínea, frequência cardíaca, passos diários e a presença ou ausência de distúrbios do sono. Transforme o atributo “gender” em variáveis numéricas usando o Label encoding. Para maior facilidade na hora da visualização dos dados, utilize o google colab notebooks. |
| Resposta:     * Utilizando pd.Categorical   Aqui, temos um exemplo utilizando o método categorical do pandas, onde ele recebe o atributo que desejamos codificar e cria um código numérico automaticamente.       * Utilizando a função de mapping customizável  Aqui temos outra forma de codificar as categorias, sendo um trabalho mais manual e dando mais liberdade ao programador. Com ela, conseguimos dizer exatamente quais valores serão associados a cada categoria. O mais usual seria utilizar 0 e 1 para variáveis dicotômicas, utilizei outros números para mostrar que temos essa liberdade, desde que os valores sejam distintos. |

| **Desafio I - Calculando Determinante** |
| --- |
| O determinante de uma matriz é uma medida importante na álgebra linear. Sendo o determinante de uma matriz quadrada AxB onde A = B podendo ser calculado da seguinte forma:    Onde nessa matriz 3x3 são as duas primeiras colunas são copiadas para se tornarem as colunas 4 e 5, e em seguida, na diagonal são somados os valores de forma espelhada, de forma que as somas da direita são multiplicadas por -1 e em seguida soma se o resultado de todas as diagonais, esse é o determinante. Seu desafio é implementar um código utilizando a biblioteca numpy que calcule o determinante de qualquer matriz do formato:    É importante ressaltar que não se pode fazer uso das função *built in* linalg.det() do Numpy, pois esta função já soluciona o problema. |

**Considerações finais**

Dessa forma terminamos o conteúdo do curso de Inteligência Artificial. Nele foi possível se aprofundar nessa área, vendo desde a preparação dos dados, as formas e estratégias de treinamento, o funcionamento e criação de redes neurais, as tecnologias utilizadas na área e os principais algoritmos utilizados atualmente.

Esperamos que a partir dos desafios e projetos desenvolvidos possa se levar uma experiência real de como se desenvolve a Inteligência Artificial atualmente. Assim como entender a importância dessas tecnologias atualmente, automatizando tarefas repetitivas ou solucionando problemas práticos da realidade.

Ter esses conhecimentos pode abrir um futuro novo, pois é uma área com muitas possibilidades e que avança constantemente, cada vez mais presente no nosso cotidiano. Logo, dominar seu desenvolvimento pode abrir novas oportunidades profissionais e pessoais.

**Referências**

FAYYAD, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). **From data mining to knowledge discovery in databases.** AI Magazine, 17(3), 37–54.

GÉRON, A. (2022). **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow**. Porto Alegre: Editora Artmed.

HAYKIN, Simon S. **Neural Networks and Learning Machines**. 3rd ed. Pearson Education, 2009.

KOLB, D. A. **The Nature of Learning**. Educational Psychologist, Vol. 11, No. 2, 1976, pp. 251-284. DOI: 10.1002/1099-1046(197609)11:2<251::AID-EP2280110203>3.0.CO;2-X

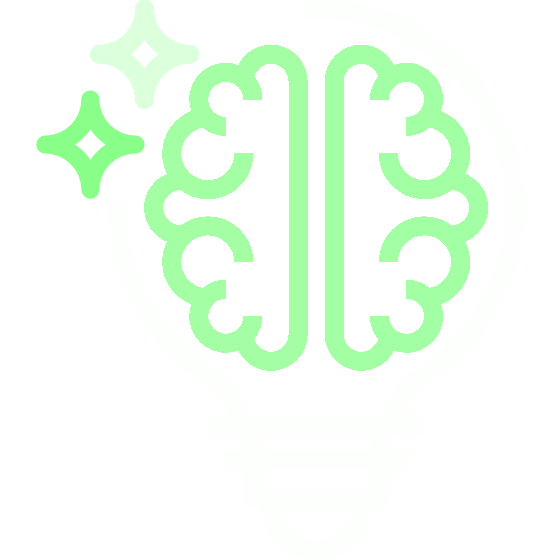
META. (2023, 7 de novembro). **Meta Pixel**. Recuperado de <https://www.facebook.com/business/tools/meta-pixel>.

SAMUEL, A. (1959). **Aprendizado de Máquina é o campo de estudo que dá aos computadores a habilidade de aprender sem ser explicitamente** programado. In: IBM Journal of Research and Development, 3(3), 363-373.

SUTTON, R. S.; BARTO, A. G. (2018). **Reinforcement learning: an introduction** (2nd ed.). MIT Press.

WITTEN, I. H.; FRANK, E.; HALL, M. A.; PAL, C. J. **Data Mining: Practical Machine Learning Tools and Techniques.** 4. ed., Morgan Kaufmann, 2016.





**BOM CURSO!**