**LINGUAGEM DE PROGRAMAÇÃO**

AULA 1

Prof. Wellington Rodrigo Monteiro

**CONVERSA INICIAL**

Olá! Esperamos que esteja tudo bem com você. Uma das frases normalmente comentadas por professores, estudantes e praticantes de TI em qualquer lugar do mundo é a seguinte: “quem domina uma linguagem de programação consegue facilmente dominar outras”. Essa afirmação, naturalmente, não é uma regra: existem linguagens e linguagens de programação. Umas são mais simples, e outras são muito mais complexas. C++ é uma linguagem diferente do Perl, que é diferente do Python, que é diferente do PHP, que é diferente do C#, por exemplo. Por outro lado, poderíamos argumentar que essa frase pode ser compreendida mais ou menos dessa forma: “quando você sabe os conceitos básicos de uma linguagem de programação, como variáveis, estruturas de seleção e/ou repetição, bibliotecas, e sabe ler a documentação, você tem menos dificuldade para entender esses mesmos conceitos em novas linguagens de programação porque agora você já conhece o conceito e já colocou em prática o que aprendeu pelo menos uma vez no passado”.

E é para isso que começamos com Python: é rápido começarmos a criar um algoritmo e, além disso, não há a obrigatoriedade de instalar algo no computador para começar a testar: o Google Colab[[1]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn1" \o ") é uma ótima porta de entrada para isso. Há uma ampla documentação disponível e uma comunidade forte. Além disso, já é utilizada por várias empresas para o desenvolvimento de algoritmos.

Quando aprendemos Python (ou qualquer outra linguagem), geralmente somos apresentados aos conceitos básicos para darmos os primeiros passos. Isso inclui, por exemplo, a entrada e saída de dados (como digitar o seu nome e mostrá-lo logo em seguida com o *input*e *print*), as operações com variáveis (como multiplicar dois valores que um usuário digitou), estruturas de seleção (com o *if, elif*e *else*), estruturas de repetição (como o *for, while*e *while True*), listas e funções. Mas isso é só o começo: serve para você se**ambientar à base** de uma linguagem de programação em específico.

E é muito provável que alguém (ou você mesmo!) tenha feito a seguinte pergunta: “como consigo fazer um site com isso que aprendi? Como consigo mostrar um gráfico? Como consigo tirar uma foto com a minha webcam? Como crio um jogo? Como crio um sistema para gerenciar uma loja? Como crio um sistema para armazenar o usuário e senha de um conjunto de usuários em Python?”.

Note que estes exemplos vão além do que vimos até o momento: é necessário, portanto, que saibamos **como****colocá-los em prática**. E é por esse motivo que estamos aqui. Vamos lá?

**TEMA 1 – EXPANDINDO O NOSSO REPERTÓRIO COM AS BIBLIOTECAS**

Vamos sair um pouco do universo de TI e fazermos juntos um pequeno exercício de pensamento: vamos supor que a torneira da sua cozinha começou a vazar água pela parede. E, para ajudar, você nunca trocou uma torneira na vida e não sabe nada de hidráulica. E agora, o que você faz? Temos aqui algumas alternativas:

1. Você compra uma nova torneira e tenta você mesmo fazer a troca, sem pedir ajuda a ninguém. Nisso, descobre que o problema era mais simples do que pensava e acabou gastando dinheiro à toa.
2. Você mesmo resolve tirar a torneira antiga, sem pedir ajuda a ninguém. Com isso, acaba estourando parte da parede e do encanamento, deixando o problema ainda pior. Agora, precisará chamar um profissional e gastar muito mais dinheiro do que imaginava.
3. Você vai ao YouTube e procura por tutoriais de troca de torneiras que foram bem avaliados. Você tenta reproduzir com cuidado o passo a passo e o adapta a suas próprias necessidades. Não fica perfeito no começo, mas você estará mais bem preparado para as próximas vezes.
4. 4. Contrata alguém para fazer o trabalho por você.

**1.1 PARA QUE SERVEM AS BIBLIOTECAS?**

E se, ao invés de falarmos de torneiras, imaginássemos que você conseguiu um estágio para uma equipe de tecnologia e lhe deram uma tarefa para criar um gráfico em Python logo de cara? Temos também algumas alternativas:

1. Você tenta criar um código para gerar o gráfico usando somente os *if/elif/else*e *for/while/while True*, listas e funções. Depois de ficar toda a semana batendo a cabeça, você percebe que ainda não conseguiu criar gráfico algum.
2. Você tenta criar um código para gerar o gráfico usando os *if/elif/else*e *for/while/while True*, listas e funções. Ao pesquisar na internet, você descobre que existem algumas formas para gerar isso. Os exemplos que você pegou funcionam, mas você não consegue integrar ao código que já possui. Logo, você fica sem saber como adaptar o problema.
3. Você pesquisa na internet para ver como é possível fazer isso em Python. Você descobre que existem **várias**formas e com várias combinações e ajustes possíveis.
4. Você diz que é impossível, e é isso aí.

Essa última alternativa, felizmente, não é uma opção: é possível sim! Para responder a isso, vamos pensar em partes. Você já deve ter se deparado no passado com as **funções**em Python. Elas possuem um objetivo muito legal: o reaproveitamento de código. Com as funções, conseguimos colocar dentro de uma caixa fechada uma lógica que precisa ser **reaproveitada**. Observemos o código a seguir, que reajusta o salário de diferentes cargos se eles estiverem acima de um salário-base.

salario_base = 3000
salario_analista = 3500
salario_estagiario = 1500
salario_lider = 9000

# reajuste salarial
reajuste = 0.035
if salario_analista > salario_base:
  salario_analista *= reajuste
  print(f'Novo salario: {salario_analista}')

if salario_estagiario > salario_base:
  salario_estagiario *= reajuste
  print(f'Novo salario: {salario_estagiario}')

if salario_lider > salario_base:
  salario_lider *= reajuste
  print(f'Novo salario: {salario_lider}')



A lógica utilizada para o reajuste se repete várias vezes, não é mesmo? Logo, podemos **refatorar**o código (ou seja, melhorá-lo sem alterar o resultado esperado) para usar funções (no caso, a função *reajustar\_salario*).

salario_base = 3000
salario_analista = 3500
salario_estagiario = 1500
salario_lider = 9000

# reajuste salarial
reajuste = 0.035
def reajustar_salario(salario, valor_base, valor_reajuste):
  if salario > valor_base:
    salario *= valor_reajuste
    print(f'Novo salario: {salario}')

  return salario

salario_analista = reajustar_salario(salario_analista, salario_base, reajuste)
salario_estagiario = reajustar_salario(salario_estagiario, salario_base, reajuste)
salario_lider = reajustar_salario(salario_lider, salario_base, reajuste)


E se adicionássemos mais 20 cargos? E se precisássemos mudar a lógica para aplicar o reajuste? Com o uso das funções, não é preciso muito esforço: conseguimos mudar a lógica em um lugar só (no caso, dentro da função *reajustar\_salario*) e os ajustes funcionarão para todo o código que vier a chamar essa função. Não precisamos também copiar e colar a mesma lógica de **dentro**da função para usarmos em múltiplos lugares: basta chamar a função em si mais uma vez (ou quantas vezes forem necessárias).

No passado, algumas pessoas desenvolveram funções só para converter uma *string*para um valor inteiro (a função *int***[[2]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn2" \o ")**); para converter uma *string*para um valor decimal (a função *float***[[3]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn3" \o ")**); para mostrar algum texto no terminal ou em um notebook de código (a função *print***[[4]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn4" \o ")**); para ler algum dado que um usuário venha a digitar (a função *input***[[5]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn5" \o ")**), e assim por diante. Note que, no final das contas, trabalhamos até o momento com uma **coleção** de funções que várias outras pessoas desenvolveram e que podemos reaproveitar em vários algoritmos. Essas coleções (ou agrupamentos) possuem outro nome: *módulos*, que, por sua vez, módulos são agrupados em **bibliotecas**(ou *libraries*).

**1.2 BIBLIOTECAS EM PYTHON**

Em Python, existem bibliotecas especializadas para inúmeras aplicações: para mostrar gráficos (como é o caso do problema que comentamos anteriormente); para manipular fotos; para manipular vídeos e músicas; para criar jogos; para comunicar com bancos de dados; para criar algoritmos de inteligência artificial, entre outras.

As bibliotecas possuem algumas vantagens, como:

1. Você consegue rapidamente adicionar funcionalidades no seu código que já foram testadas e desenvolvidas por várias outras pessoas.
2. Você consegue testar diferentes implementações que melhor se adaptem às suas necessidades: em Python, por exemplo, existem dezenas de bibliotecas que tratam somente da geração de gráficos. Uma biblioteca que é preferida para outras pessoas pode não necessariamente ser a sua favorita, e vice-versa.
3. Você consegue instalar e desinstalar rapidamente as bibliotecas de acordo com as suas necessidades.

O Python já possui uma biblioteca padrão[[6]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn6" \o ") e funções como a *input, print, float*e *int* fazem parte dela. As bibliotecas também podem incluir tipos de dados além das listas, matrizes, *strings*, datas e números. Veremos mais sobre isso no decorrer desta aula.

**Saiba mais**

Um bom conselho é explorar a documentação das bibliotecas para conhecer suas possibilidades de uso.

Lembra que comentamos que as bibliotecas são divididas em módulos? A biblioteca padrão do Python possui um módulo especializado[[7]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn7" \o ") somente em manipulação de arquivos compactados do tipo zip; outro[[8]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn8" \o ") especializado somente em listar todos os arquivos ou pastas em determinado local; outro[[9]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn9" \o ") módulo especializado em operações matemáticas; outro[[10]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn10" \o ") módulo especializado em operações estatísticas; e assim por diante. Saber ler essas documentações e encontrar a melhor biblioteca, módulo e função para o que você precisa pode fazer toda a diferença no seu trabalho.

Existem centenas de milhares de bibliotecas disponíveis, que, geralmente são instaladas com o comando *pip*. É possível sabermos quais são as mais utilizadas com base em rankings facilmente encontrados na internet. Uma busca por “pypi download stats” no Google retorna vários desses rankings, por exemplo. Algumas dessas servem para gerar gráficos como o **matplotlib, seaborn**e o **plotly**.

**TEMA 2 – CRIANDO GRÁFICOS COM PYTHON**

Uma das formas de encarar o desenvolvimento de software é pensar que fazemos este trabalho para **resolver problemas**: seja um aplicativo de internet banking, um software que gerencia uma fábrica ou um algoritmo de inteligência artificial que prevê problemas nas vendas. Logo, tão importante quanto desenvolvermos um **código bom** é **mostrar que estamos resolvendo um problema.**

Naturalmente, essa definição muda muito de caso para caso: um desenvolvedor de back-end ou um administrador de banco de dados raramente vão se deparar com um cenário no qual terão de gerar algum gráfico em Python. Por outro lado, um engenheiro de dados, um cientista de dados ou um analista de dados precisarão fazer isso **muito**. Ou, ainda, pessoas que estejam desenvolvendo pesquisas de natureza quantitativa (o que inclui um mestrado ou doutorado, por exemplo). É claro que existe o Excel (ou qualquer software de planilhas) para criar alguns gráficos rapidamente, mas ele não gera todos os tipos de gráficos de que poderíamos necessitar. Além disso, existem casos em que precisamos nos conectar diretamente a bases de dados ou manipular um grande volume de dados, que o Excel simplesmente não suportaria.

**2.1 BIBLIOTECAS MAIS CONHECIDAS**

Em Python, três bibliotecas se destacam: o matplotlib, o seaborn e o plotly. Existem várias outras, como o bokeh, yellowbrick e Altair, mas essas três primeiras são as mais conhecidas:

1. O matplotlib[[11]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn11" \o ") foi criado em 2003 e se tornou a biblioteca de geração de gráficos mais usada em Python. No entanto, ela é bem complexa: imagine um painel de um grande avião, cheio de botões e configurações. Muitas são as possibilidades, mas também há uma curva de aprendizagem considerável.
2. O seaborn[[12]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn12" \o ") foi criado quase dez anos depois. Ele usa o matplotlib como base, mas simplifica várias das operações. Ele é bem mais simples de usar e pode gerar gráficos mais bonitos.
3. O plotly[[13]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn13" \o ") (e em específico o *plotly express*) pode ser simples como o seaborn, mas traz uma diferença: a interatividade. Com o plotly é possível criar animações e interações nos gráficos (como passar o mouse e ver mais informações, por exemplo).

**2.2 EXEMPLOS**

Vamos supor que queiramos criar um gráfico contendo a visualização de diferentes notas e preços de cafés. Temos essas notas e preços armazenados em duas diferentes listas:

preco = [5.29, 6.49, 7.49, 7.99, 8.49, 9.99, 11.99]
nota = [5.5, 7.0, 7.5, 8.5, 9.5, 9.0, 9.5]



Observe o código utilizado para criar os gráficos utilizando os mesmos dados, mas com bibliotecas diferentes.

import plotly.express as px

fig = px.scatter(x=preco, y=nota,
                 title='Comparação entre preço e nota de cafés',
                 labels={"x": "Preço", "y": "Nota"})
fig.show()



Figura 1 – Comparação de preço e nota de cafés (1)



Fonte: Monteiro, 2021.

import matplotlib.pyplot as plt

plt.scatter(x=preco, y=nota)
plt.title('Comparação entre preço e nota de cafés')
plt.xlabel('Preço')
plt.ylabel('Nota')



Figura 2 – Comparação de preço e nota de cafés (2)

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

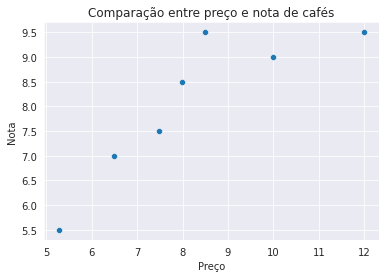
Fonte: Monteiro, 2021.

import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style('darkgrid')
sns.scatterplot(x=preco, y=nota)
plt.title('Comparação entre preço e nota de cafés')
plt.xlabel('Preço')
plt.ylabel('Nota')



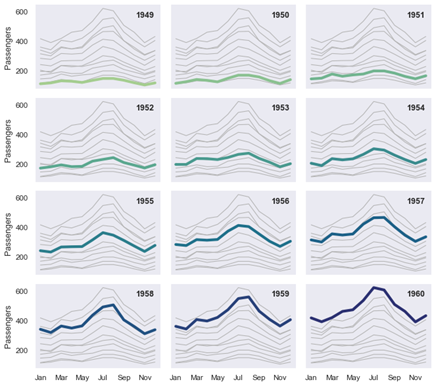
Figura 3 – Comparação de preço e nota de cafés (3)



Fonte: Monteiro, 2021.

Esse tipo de gráfico (dispersão, ou *scatter plot*) é um dos mais simples. Por outro lado, existem outras visualizações mais interessantes como esta, do seaborn:

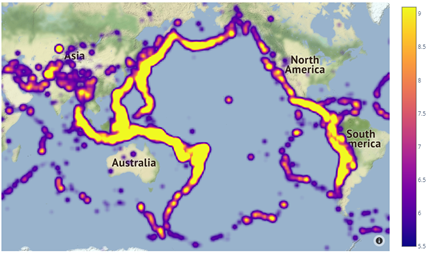
Figura 4 – Seaborn



Fonte: Seaborn, [S.d.].

Ou esta, do Plotly:

Figura 5 – Plotly



Fonte: Plotly Technologies, Inc., 2015.

Naturalmente, para criar gráficos, precisamos de dados. O breve exemplo que criamos anteriormente lida com duas listas: uma chamada *preco* e outra chamada *nota*. Mas será mesmo que o uso de duas listas seria a melhor alternativa? Seria prático permanecermos utilizando listas mesmo se tivéssemos alguns milhares ou milhões de notas? Se você já tentou manipular listas com um tamanho tão alto, deve ter se deparado com problemas de usabilidade e de performance. Felizmente, há uma saída: a biblioteca NumPy.

**TEMA 3 – RESOLVENDO O PROBLEMA DAS LISTAS**

Várias bibliotecas usadas para inteligência artificial (incluindo redes neurais), visualização, análise de mapas, bioinformática, astronomia e demais áreas utilizam NumPy[[14]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn14" \o "). E isso acontece porque essa biblioteca foi pensada para resolver problemas grandes e complexos com o menor esforço possível por parte do desenvolvedor. Para deixar isso mais claro, sugerimos que voltar ao exemplo do café.

preco = [5.29, 6.49, 7.49, 7.99, 8.49, 9.99, 11.99]
nota = [5.5, 7.0, 7.5, 8.5, 9.5, 9.0, 9.5]



Cada uma dessas listas possui sete valores e são listas em Python. Agora, imagine fazer as seguintes operações nelas:

1. Multiplicar todos os elementos de uma lista;
2. Encontrar a média e desvio-padrão dos valores de uma lista;
3. Descobrir se algum dos elementos da lista é um número infinito;
4. Obter o valor absoluto de todos os elementos da lista;
5. Obter todos os elementos que sejam somente acima de 9;
6. Remover todos os valores em duplicata;
7. Inverter a ordem de todos os valores.

É bem provável que consigamos fazer rapidamente e com uma só linha de código algumas dessas operações, mas não todas. O NumPy disponibiliza uma estrutura de dados similar às listas: o *ndarray***[[15]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn15" \o ")**. O uso do *ndarray*em relação às listas possui algumas vantagens, como:

1. **Funcionalidades**: já existem funções prontas para manipular rapidamente todos os elementos de um *ndarray*. Essas aplicações incluem operações algébricas, estatísticas, manipulações de dados e operações matemáticas. Na prática, essas operações são úteis para pesquisadores a nível de mestrado e doutorado; estudantes em cursos de engenharia e demais cursos aplicados na área de exatas; estatísticos e economistas; desenvolvedores de aplicações back-end que necessitam manipular uma grande massa de dados; cientistas e analistas de dados, entre outros. Logo, o *ndarray* é mais prático do que as listas do Python;
2. **Performance**: quando precisamos manipular uma grande massa de dados (na casa das centenas de milhares de entradas ou mais), o *ndarray* geralmente possui uma implementação mais otimizada. Logo, o *ndarray*tende a ser mais rápido do que as listas do Python;
3. **Consumo de espaço**: além disso, o *ndarray*é também otimizado para grandes conjuntos de dados. Uma prova disso é a de que o uso do NumPy foi essencial para gerar a primeira imagem de um buraco negro conhecida pela humanidade[[16]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn16" \o "). Logo, o *ndarray*consome menos espaço em memória do que as listas do Python;
4. **Reaproveitamento do conhecimento**: toda a base de conhecimento que temos sobre listas é usada para trabalharmos com *ndarray*.

Note como as duas listas anteriores seriam implementadas em *ndarray*:

import numpy as np

preco = np.array([5.29, 6.49, 7.49, 7.99, 8.49, 9.99, 11.99])
nota = np.array([5.5, 7.0, 7.5, 8.5, 9.5, 9.0, 9.5])


E é isso! Agora, e se precisássemos ler muito mais dados? E se precisássemos ler um arquivo CSV ou uma planilha do Excel? E se quiséssemos manipular os dados no Python como se fosse o Excel, o que faríamos? É para isso que serve o **pandas**.

**TEMA 4 – TRABALHANDO COM TABELAS EM PYTHON**

A manipulação de dados em formato tabular (isto é, em tabelas) é facilitada pela biblioteca pandas. Em conjunto com ela, outras bibliotecas de cunho estatístico e gráfico podem ser combinadas para analisar os dados, manipular os dados, gerar correlações e relatórios de estatística descritiva.

**4.1 PANDAS**

O pandas[[17]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn17" \o ") é uma biblioteca que foi criada utilizando o NumPy como base. Da mesma forma que uma faca é uma ferramenta que todo cozinheiro possui e um alicate é uma ferramenta que todo eletricista possui, o pandas é uma biblioteca[[18]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn18" \o ") que todo desenvolvedor que atua com dados em Python conhece e trabalha.

O pandas introduz um tipo de dados muito poderoso chamado *DataFrame***[[19]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn19" \o ")**. Para falar do DataFrame, vamos voltar ao exemplo do café. Lembre-se que tínhamos duas listas, *preco*e *nota*:

preco = [5.29, 6.49, 7.49, 7.99, 8.49, 9.99, 11.99]
nota = [5.5, 7.0, 7.5, 8.5, 9.5, 9.0, 9.5]


Nunca deixamos isso explícito, mas **é possível entender**que o primeiro valor do preço corresponde ao primeiro valor da nota; o segundo valor do preço corresponde ao segundo valor da nota; e assim sucessivamente. Agora, e se eu quisesse ordenar de forma decrescente o preço, alterando também a ordem da nota? Da forma que está, isso não seria possível. E se quisesse descobrir rapidamente o preço correspondente à mediana das notas? Também não seria tão simples. Por outro lado, em Python também podemos ter matrizes (isto é, listas de listas).

Então, poderíamos organizar da seguinte forma:

precos_notas = [[5.29, 6.49, 7.49, 7.99, 8.49, 9.99, 11.99],
                [5.5, 7.0, 7.5, 8.5, 9.5, 9.0, 9.5]]



Agora temos somente uma variável, chamada *precos\_notas*. Mas e se quiséssemos incluir a data da coleta das notas? Ou, ainda, se quiséssemos aplicar uma operação para *todas*as notas (um pequeno reajuste de 0.1 pontos, talvez)? Ou, quem sabe, se quiséssemos **dar um nome**para esses dados, como seria?

É para responder a essas questões que existem os DataFrames. Com uma linha (na maioria das vezes), conseguimos ler planilhas e conjuntos de dados inteiros para uma variável, e vice-versa. Quer ler uma tabela em SQL e converter para um DataFrame? É possível. Quer ler uma planilha do Excel? Dá sim! Um JSON, quem sabe? É claro! As possibilidades são múltiplas. Vamos ver como ficariam os mesmos dados dos preços e das notas, mas em DataFrame:

import pandas as pd

precos_notas = pd.DataFrame([[5.29, 5.5], [6.49, 7.0], [7.49, 7.5], [7.99, 8.5], [8.49, 9.5], [9.99, 9.0], [11.99, 9.5]],
                            columns=['Preço', 'Nota'])


Tabela 1 – Preço e nota



E se quiséssemos criar uma data fictícia para a coleta dessas notas?

precos_notas['DataColeta'] = pd.date_range(start='2022-01-01', freq='M',
                                           periods=len(precos_notas))


A função *date\_range***[[20]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn20" \o ")** cria uma sequência de datas. Podemos informar uma data de início e de fim ou, ainda, uma frequência entre as datas. No caso, preferimos escolher somente uma data de início (2022-01-01) e uma frequência mensal, pegando sempre o último dia de cada mês (com o parâmetro ‘M’). A quantidade de datas foi definida pela quantidade de linhas disponíveis no dataframe (pelo *len(precos\_notas)*).

Tabela 2 – Preço, nota e data



E se quiséssemos criar colunas para extrair somente o ano, ou somente o dia, ou somente o dia da semana da data de coleta? Ou, ainda, a proporção da nota em relação ao preço? Vamos lá:

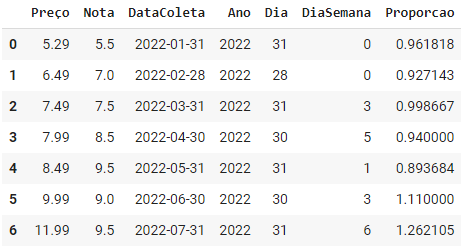
precos_notas['Ano'] = precos_notas['DataColeta'].dt.year
precos_notas['Dia'] = precos_notas['DataColeta'].dt.day
precos_notas['DiaSemana'] = precos_notas['DataColeta'].dt.weekday
precos_notas['Proporcao'] = precos_notas['Preço']/precos_notas['Nota']



Agora, vamos pensar um pouco: já imaginou como seria fazer tudo isso sem o pandas, somente manipulando as listas em Python? Há ainda uma função no pandas para nos fornecer dados estatísticos sobre esse DataFrame: o *describe***[[21]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn21" \o ")**. Observe:

precos_notas.describe()

Tabela 3 – Preço, nota, data e proporção



Aqui, temos dados como a quantidade de valores para cada coluna (*count*), a média (*mean*), o desvio-padrão (*std*), os valores mínimo (*min*) e máximo (*max*), a mediana (50%) e os quartis inferior (25%) e superior (50%). Além disso, veja que na própria documentação[[22]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn22" \o ") existem ainda múltiplos exemplos para operações, como agrupamento, deslocamento de colunas, ordenação, preenchimento de valores nulos, remoção de linhas repetidas, arredondamento, correlação, entre outros. Falando em correlação, vamos testar uma coisa.

**4.2 ESTUDANDO CORRELAÇÃO COM DIFERENTES BIBLIOTECAS**

Quando falamos em **correlação**, queremos, na prática, entender o quanto uma variável afeta outra. Por exemplo, será que, toda vez que o preço aumenta em 10%, a nota também aumenta (ou diminui) na mesma proporção? Com uma única linha, o pandas consegue calcular isso[[23]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn23" \o "). Veja só:

precos_notas.corr(method='spearman')



Tabela 4 – Correlação

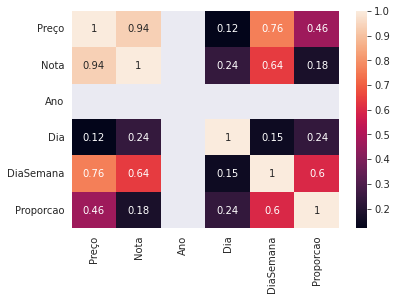


Quanto mais próximo de 1, maior a correlação diretamente proporcional (ou seja, quando um aumenta, o outro aumenta também). Quanto mais próximo de -1, maior a correlação inversamente proporcional (ou seja, quando um aumenta, o outro diminui). Aparentemente, o preço tem uma correlação muito alta com a nota, mas parece que o dia da semana também teria uma correlação interessante. Ficou difícil de enxergar vários números? Vamos ao seaborn[[24]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn24" \o "):

sns.heatmap(precos_notas.corr(method='spearman'), annot=True)



Figura 6 – Seaborn



Perceba que, aos poucos, vamos também **misturando**bibliotecas. A ideia aqui é conhecermos essas diferentes bibliotecas para saber que possibilidades elas nos trazem. E, então, saberemos como aproveitar o melhor de cada uma para atender às nossas necessidades. É como se fôssemos um grande chef de cozinha que mistura diferentes técnicas que aprendeu ao longo da vida para construir um belo prato. Ele não ficará restrito aos instrumentos e temperos que estão à sua frente, mas pensará além, para integrar diferentes técnicas, utensílios e temperos para alcançar algo novo, inovador e que atenda às suas necessidades. Aqui, a lógica é a mesma (ainda que não seja tão saborosa). É por isso que apresentamos diferentes bibliotecas e sua documentação; é importante que você conheça o que existe disponível para o seu uso, que consiga ter a autonomia de encontrar a informação de que precisa e o desejo de ir além, de descobrir coisas novas e experimentar diferentes funções e bibliotecas.

**TEMA 5 – RESOLVENDO PROBLEMAS COM UMA LINHA: USO DE EXPRESSÕES REGULARES EM PYTHON**

Agora que conversamos sobre **outras**bibliotecas, vamos retornar à biblioteca-padrão do Python. E, para isso, vamos conversar sobre uma ferramenta poderosa: as **expressões regulares**.

Para falarmos de expressões regulares (*regular expressions,*ou *regex*), vamos, primeiramente, pensar no seguinte desafio: vamos supor que a empresa na qual você trabalha precisa processar um arquivo contendo o equivalente a 500 páginas de texto, e lhe deram a tarefa de selecionar **somente** as placas de veículos.

Agora, o que seria uma placa válida? Até alguns anos atrás, uma placa válida possuía três letras e quatro dígitos. Logo, as seguintes placas seriam combinações válidas:

1. ABC1234.
2. ABC-1234.

Note que o segundo exemplo possui um hífen: existem pessoas que se referem às placas com o hífen e outras, sem. Agora, as seguintes combinações não são placas válidas:

1. 12345678.
2. ABC12.
3. 1234ABC.
4. A-B-C-1234.
5. ABCDE1234.
6. A1234BC.

Note que essas combinações não respeitam o **padrão**das placas que comentamos anteriormente: três letras seguidas de quatro dígitos. Mas, recentemente, essa regra mudou: também passaram a ser válidas placas cujo quinto caractere é uma letra (conhecido como *padrão Mercosul*), como:

1. ABC1A34.
2. ABC-1A34.

Logo, note que temos quatro **padrões**a que precisaríamos atender, pelo menos:

1. Três primeiros caracteres sendo letras e os quatro caracteres seguintes sendo dígitos (ABC1234).
2. Três primeiros caracteres sendo letras, seguidos de um hífen e, em seguida, os próximos quatro caracteres sendo dígitos (ABC-1234).
3. Três primeiros caracteres sendo letras; o quarto caractere sendo um dígito; o quinto sendo uma letra; e o sexto e sétimo sendo dígitos (ABC1A34);
4. Três primeiros caracteres sendo letras, seguidas de um hífen; o quarto caractere sendo um dígito; o quinto sendo uma letra; e o sexto e sétimo sendo dígitos (ABC-1A34).

Observe que realçamos a palavra ***padrão*** anteriormente: não queremos somente detectar a placa “ABC1234”, mas qualquer outra placa que siga esse exemplo (ZZZ9999, UNI0000, AAA0800 etc.). Isso dá mais de 170 milhões de combinações. Se considerarmos as placas do padrão Mercosul, temos quase 500 milhões de combinações a mais para encontrar.

E para colocar tudo isso em um algoritmo? Já imaginou ter que digitar todas as combinações possíveis? Ainda que tentássemos usar um loop (como o *for*) ou algo do gênero, teríamos que percorrer todas as alternativas possíveis, o que demandaria um considerável esforço nosso e do computador, para dar conta de todo o esforço.

É para isso que serve o regex: criamos uma string contendo **somente**o padrão que desejamos detectar (e, às vezes, o que **não queremos**detectar). Existem listas bem extensas sobre as possibilidades do regex, da Mozilla[[25]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn25" \o "), da Microsoft[[26]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn26" \o ") ou, ainda, do próprio Python[[27]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn27" \o "). Observe alguns exemplos:

Quadro 1 – Regex

|  |  |  |  |
| --- | --- | --- | --- |
| Regex | O que faz? | Exemplo | O que seleciona? |
| \* | Seleciona o elemento anterior nenhuma ou mais vezes | Caf\* | “Ca” em “Caça”  “Caf” em “Café”  “Caffff” em “Caffffé” |
| + | Seleciona o elemento anterior uma ou mais vezes | Caf+ | Nada em “Caça”  “Caf” em “Café”  “Caffff” em “Caffffé” |
| ? | Seleciona o elemento anterior nenhuma ou uma vez | Caf? | “Ca” em “Caça”  “Caf” em “Café”  “Caf” em “Caffffé” |
| {n} | Seleciona o elemento anterior n vezes | Caf{2} | Nada em “Caça”  “Caf” em “Café”  “Caff” em “Caffffé” |
| {n,m} | Seleciona o elemento anterior de n até m vezes | Caf{1,2} | Nada em “Caça”  “Caf” em “Café”  “Cafff” em “Caffffé” |
| [qualquer\_caractere] | Seleciona qualquer caractere que esteja em parênteses. “A-Z” selecionará todos os caracteres de A (maiúsculo) até Z (maiúsculo), por exemplo. | [a-z]+ | As duas letras “a” em “Caça”  “af” em “Café”  “affff” em “Caffffé” |

Imaginemos a frase “*The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually*.”. Veja, no quadro a seguir, quais padrões cada regex detecta na frase. Cada mudança de cor representa uma sequência que foi encontrada.

Quadro 2 – Utilização de regex

|  |  |
| --- | --- |
| Regex | O que seleciona? |
| [0-9] | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [0-9]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [\d]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [a-z]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [A-Z]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [A-Z0-9]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [A-Za-z]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [\w]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [A-Za-z0-9]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [A-Za-z0-9]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [\w\d]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| \s | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [\w\d\s]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| [^\.]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |
| \w[^\.]+ | The quick brown Fox jumps over the lazy dog. 1 or 1000 foxes, actually. |

E aquele caso das placas de automóveis? Observemos a frase “3 placas devem ser selecionadas: UNI0100, NTE-0A00 e RRR9876. A JJ2070 é inválida.”.

Quadro 3 – Placas de automóveis

|  |  |
| --- | --- |
| Regex | O que seleciona? |
| [A-Za-z0-9]{7} | 3 placas devem ser selecionadas: UNI0100, NTE-0a00 e RRR9876. A JJ2070 é inválida. |
| [A-Za-z]{3}[0-9]+ | 3 placas devem ser selecionadas: UNI0100, NTE-0a00 e RRR9876. A JJ2070 é inválida. |
| [A-Za-z]{3}[0-9]+ | 3 placas devem ser selecionadas: UNI0100, NTE-0a00 e RRR9876. A JJ2070 é inválida. |
| [A-Za-z]{3}\d{4} | 3 placas devem ser selecionadas: UNI0100, NTE-0a00 e RRR9876. A JJ2070 é inválida. |
| [A-Za-z]{3}\-?\d[\dA-Za-z]\d{2} | 3 placas devem ser selecionadas: UNI0100, NTE-0a00 e RRR9876. A JJ2070 é inválida. |
| [A-Za-z]{3}\-?\d[A-Ja-j\d]\d{2} | 3 placas devem ser selecionadas: UNI0100, NTE-0a00 e RRR9876. A JJ2070 é inválida. |

Logo, o uso de regex é bem interessante para encontrar rapidamente múltiplos resultados dentro de um conjunto grande de textos. Ele possui algumas aplicações úteis, como:

1. Busca em textos por palavras-chave ou padrões.
2. Validação de dados (o CEP que o usuário digitou é realmente um CEP válido? O e-mail digitado realmente é um email?).
3. Web scraping (como a extração de somente os títulos das notícias em um portal de notícias).
4. Filtros em integrações (como a obtenção de somente os dados que precisamos a partir de uma integração construída com outro servidor ou com um banco de dados).

Em Python, o módulo a cargo do regex é o módulo re[[28]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn28" \o "). O código a seguir é um exemplo de implementação do regex de busca de placas que vimos anteriormente.

import re

# texto que usaremos na busca
texto_busca = '3 placas devem ser selecionadas: UNI0100, NTE-0a00 e RRR9876. A JJ2070 é inválida.'

# regex a ser usado
texto_regex = '[A-Za-z]{3}\-?\d[A-Ja-j\d]\d{2}'

# procurando
resultados = re.findall(texto_regex, texto_busca)
resultados



Caso tenha interesse, sugerimos que dê uma olhada nos links do módulo *re*do Python, bem como nos links sobre regex sobre os quais já comentamos. Saber usar regex pode fazer toda a diferença para resolver rapidamente problemas que seriam complexos no seu dia a dia. Caso também queira testar diferentes tipos de regex, existem sites que podem ajudar você a ver como funciona o regex de forma visual. Alguns exemplos incluem o regex101[[29]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn29" \o "), o RegExr[[30]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn30" \o ") e o Debuggex[[31]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftn31" \o ").

**Saiba mais**

Geralmente testamos um regex em algum desses sites para ver se a lógica funciona. Somente após garantirmos que o regex está selecionando o que gostaríamos (e não selecionando o que não gostaríamos) é que colocamos a lógica no código.

**FINALIZANDO**

Observe que a intenção desta aula foi mostrar a você a relevância do uso de outras bibliotecas em Python: na realidade, este é um dos motivos pelos quais o Python acaba tendo uma preferência por várias empresas, profissionais e estudantes. Isso naturalmente não significa que o Python é a **melhor**linguagem do mundo, cada linguagem de programação possui os seus prós e os seus contras. Em nosso contexto, a oportunidade de conhecermos diferentes bibliotecas é muito útil para sabermos como podemos **rapidamente**aplicar a linguagem de programação em novos desafios, algoritmos e soluções de mercado.

Nesse sentido, observe que comentamos sobre o uso de **expressões regulares**,**gráficos** e dados tabulares com o **NumPy**e**pandas**. Também comentamos sobre outras bibliotecas que podem igualmente ser úteis para o seu conhecimento. Independentemente da biblioteca, observe que também disponibilizamos links para você ter acesso à documentação em sua versão mais atualizada. Saber ler e compreender a documentação é certamente algo que lhe dará vantagem competitiva no mercado de trabalho. Logo, entenda as discussões que tivemos até o momento como direcionadores. Apontamos os diferentes pontos de partida que são do seu interesse, e não um passo a passo – até porque esse passo a passo muda a cada desafio no trabalho. Assim, é importante que você saiba como raciocinar para executar, e não como executar um passo a passo. Novamente: leia, explore e teste as documentações.

**REFERÊNCIAS**

AGUILAR, L. J. **Fundamentos de programação**: algoritmos, estruturas de dados e objetos. 3. ed. Porto Alegre: AMGH, 2011.

GRUS, J. **Data Science do Zero**. Rio de Janeiro: Editora Alta Books, 2021.

PERKOVIC, L. **Introdução à Computação Usando Python**: um Foco no Desenvolvimento de Aplicações. Rio de Janeiro: LTC, 2016.

PLOTLY TECHNOLOGIES, INC. **Collaborative data science**. Montréal, 2015.

SEABORN. **Small multiple time series**. [S.d.]. Disponível em: <https://seaborn.pydata.org/examples/timeseries\_facets.html>. Acesso em: 6 dez. 2021.

WASKOM, M. L. Seaborn: statistical data visualization. **Journal of Open Source Software**, v. 6, n. 60, p. 3021, 2021.

[[1]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref1" \o ") Disponível em: <https://colab.research.google.com/>. Acesso em: 6 dez. 2021.

[[2]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref2" \o ") Disponível em: <https://docs.python.org/3/library/functions.html#int>. Acesso em: 6 dez. 2021.

[[3]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref3" \o ") Disponível em: <https://docs.python.org/3/library/functions.html#float>. Acesso em: 6 dez. 2021.

[[4]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref4" \o ") Disponível em: <https://docs.python.org/3/library/functions.html#print>. Acesso em: 6 dez. 2021.

[[5]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref5" \o ") Disponível em: <https://docs.python.org/3/library/functions.html#input>. Acesso em: 6 dez. 2021.

[[6]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref6" \o ") Disponível em: <https://docs.python.org/3/library/>. Acesso em: 6 dez. 2021.

[[7]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref7" \o ") Disponível em: <https://docs.python.org/3/library/zipfile.html>. Acesso em: 6 dez. 2021.

[[8]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref8" \o ") Disponível em: <https://docs.python.org/3/library/glob.html>. Acesso em: 6 dez. 2021.

[[9]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref9" \o ") Disponível em: <https://docs.python.org/3/library/math.html>. Acesso em: 6 dez. 2021.

[[10]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref10" \o ") Disponível em: <https://docs.python.org/3/library/statistics.html>. Acesso em: 6 dez. 2021.

[[11]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref11" \o ") Disponível em: <https://matplotlib.org/>. Acesso em: 6 dez. 2021.

[[12]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref12" \o ") Disponível em: <https://seaborn.pydata.org/>. Acesso em: 6 dez. 2021.

[[13]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref13" \o ") Disponível em: <https://plotly.com/python/plotly-express/>. Acesso em: 6 dez. 2021.

[[14]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref14" \o ") Disponível em: <https://numpy.org/>. Acesso em: 6 dez. 2021.

[[15]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref15" \o ") Disponível em: <https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html>. Acesso em: 6 dez. 2021.

[[16]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref16" \o ") Disponível em: <https://numpy.org/case-studies/blackhole-image/>. Acesso em: 6 dez. 2021.

[[17]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref17" \o ") Disponível em: <https://pandas.pydata.org/>. Acesso em: 6 dez. 2021.

[[18]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref18" \o ") Disponível em: <https://pandas.pydata.org/docs/index.html>. Acesso em: 6 dez. 2021.

[[19]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref19" \o ") Disponível em: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>. Acesso em: 6 dez. 2021.

[[20]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref20" \o ") Disponível em: <https://pandas.pydata.org/docs/reference/api/pandas.date\_range.html>. Acesso em: 6 dez. 2021.

[[21]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref21" \o ") Disponível em: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html>. Acesso em: 6 dez. 2021.

[[22]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref22" \o ") Disponível em: <https://pandas.pydata.org/pandas-docs/stable/reference/frame.html>. Acesso em: 6 dez. 2021.

[[23]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref23" \o ") Disponível em: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html#pandas.DataFrame.corr>. Acesso em: 6 dez. 2021.

[[24]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref24" \o ") Disponível em: <https://seaborn.pydata.org/generated/seaborn.heatmap.html>. Acesso em: 6 dez. 2021.

[[25]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref25" \o ") Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\_Expressions/Cheatsheet>. Acesso em: 6 dez. 2021.

[[26]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref26" \o ") Disponível em: <https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference>. Acesso em: 6 dez. 2021.

[[27]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref27" \o ") Disponível em: <https://docs.python.org/3/howto/regex.html>. Acesso em: 6 dez. 2021.

[[28]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref28" \o ") Disponível em: <https://docs.python.org/3/library/re.html>. Acesso em: 6 dez. 2021.

[[29]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref29" \o ") Disponível em: <https://regex101.com/>. Acesso em: 6 dez. 2021.

[[30]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref30" \o ") Disponível em: <https://regexr.com/>. Acesso em: 6 dez. 2021.

[[31]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a1&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPeW1oPSfienyKunysb5oZms=&ne=False" \l "_ftnref31" \o ") Disponível em: <https://www.debuggex.com/>. Acesso em: 6 dez. 2021.

**LINGUAGEM DE PROGRAMAÇÃO**

AULA 2

Prof. Wellington Rodrigo Monteiro

**CONVERSA INICIAL**

Olá! Anteriormente, comentamos sobre o uso das bibliotecas e, ainda, a importância de lermos a documentação das bibliotecas. Aqui, vale reforçar: o objetivo é o de mostrar *o***caminho**para você, mas é impossível mostrar tudo que existe pela velocidade de mudança. Apostamos que, nas próximas 24h, pelo menos algumas **centenas**de bibliotecas em Python mudarão alguma coisa. Elas poderão ter novas funcionalidades, mudanças em funcionalidades já existentes, correções na documentação, novos bugs, bugs resolvidos, entre outros. Por outro lado, a**base** não muda: a forma de ler, procurar e interpretar a documentação que aprendemos aqui lhe ajudarão independentemente das possíveis mudanças no futuro.

Bom, e por que relembramos isso agora? Uma das áreas da ciência em maior crescimento e velocidade é a da Inteligência Artificial. E, naturalmente, existem bibliotecas em Python para Inteligência Artificial (IA) – de fato, o Python já é entendido atualmente como a linguagem padrão para o desenvolvimento de algoritmos de IA. Logo, que tal criarmos uma IA juntos?

**TEMA 1 – DEFININDO INTELIGÊNCIA ARTIFICIAL E APRENDIZAGEM DE MÁQUINA**

Quando nos aprofundamos um pouco mais no mundo de IA é comum nos confundirmos com vários termos como: *inteligência artificial, machine learning, aprendizagem de máquina, deep learning, ciência de dados, advanced analytics, redes neurais,*entre outros. Agora, qual é a diferença entre eles? O que é o quê? Vamos pensar nesses exemplos utilizando analogias.

1. Imagine uma **praça de alimentação** de um shopping: existem vários locais para comer e atendendo diversos gostos, como pizzas, massas, comidas típicas e vegetarianas, sorvetes, sanduíches e outros.
2. Agora, imagine que dentro dessa praça de alimentação existe um **restaurante** muito famoso. Pelo fato de ser um restaurante, você pode imaginar que diversos pratos são servidos, e para atender a todos os gostos.
3. Ainda que esse restaurante sirva vários pratos ele é mais conhecido pelos seus **pratos italianos**.
4. E, na cidade, “pratos italianos” é praticamente um sinônimo para **massas**.
5. Quando falamos em massas, uma **macarronada**bem-feita já é digna de muitos elogios e satisfaz o desejo de quase todos os clientes.
6. Por outro lado, quase sempre a **lasanha**vem à mente também. O problema, contudo, é que ela exige mais trabalho para fazer (e limpar) em comparação com a macarronada.
7. Só que existem vários tipos de lasanhas. Lasanhas mais complexas podem **ter mais camadas** de massa e recheio, e existem recheios mais complexos do que outros.



Crédito: Artisticco/ Shutterstock.

Dito isso, vamos explicar as analogias. A praça de alimentação é análoga ao universo de **Advanced Analytics**, uma área que engloba não somente IA, mas também a visualização de dados. Já ouviu falar em *Business Intelligence*, ou BI? Dentro das empresas, BI é praticamente sinônimo de relatórios e painéis visuais (*dashboards*) contendo vários gráficos mostrando o desempenho de indicadores importantes para a área de negócio. Esses indicadores podem ser, por exemplo, o histórico de vendas; a quantidade de problemas nas fábricas; o volume de produção registrado até o momento; entre outros. Note que, nesses exemplos, quase sempre falamos **do passado e do presente**. Esses dashboards, por outro lado, também podem ser combinados com outros algoritmos de IA e outras técnicas para prever o futuro. É como uma praça de alimentação de um shopping no qual podemos comprar as refeições de mais de um lugar e comer em um lugar só.

Já o restaurante é análogo ao mundo da **Ciência de Dados**. Veja que Ciência de Dados fica **dentro**do mundo de Advanced Analytics. Quando trabalhamos com Ciência de Dados não trabalhamos **somente**com IA, mas também usamos a Estatística e os nossos próprios conhecimentos em TI.

Os pratos italianos são análogos à **IA**: é um tópico interessante e conhecido no mundo todo. É também amplo e cheio de possibilidades. Para alguns, pode parecer simples. Para outros, pode parecer algo bem refinado e que exige um bom conhecimento para ser feito com qualidade.

Por outro lado, se você perguntar a uma pessoa qual é a primeira coisa que vem à cabeça dela é provável que tenha respostas bem diferentes. Para alguém novo, isso pode ser sinônimo de lasanha. Para uma pessoa com mais experiência de vida, por outro lado, pode ser sinônimo de um tiramisu artesanal. E, para você, pode ser um gelato. Observe que diferentes públicos podem possuir interpretações completamente diferentes sobre o que é um prato italiano. E, ainda, diferentes públicos possuem diferentes **expectativas**sobre o que seria um bom prato italiano. A mesma coisa é com IA: pessoas que não são da área podem possuir como expectativa um algoritmo superpoderoso e que aprende em um toque de mágica como resolver problemas. Imaginam uma IA próxima daquelas superinteligências que vemos nos filmes – como um assistente ou um robô que conversasse conosco e que resolvesse qualquer problema de qualquer área. Podem, até mesmo, imaginar algo próximo ao J.A.R.V.I.S. do Homem de Ferro (ou, por outro lado, a Skynet do Exterminador do Futuro). Ou, em outro extremo, podem entender que a IA hoje em dia é somente um termo novo para algo mais antigo: Estatística (e/ou Matemática).

Já as massas são análogas ao **machine learning**(ML ou **aprendizagem de máquina**). Apesar dos exemplos anteriores (lasanha, tiramisu e gelato), se entrevistássemos hoje um grande conjunto de pessoas e perguntássemos a eles por um exemplo de “prato italiano servido em restaurante”, certamente a grande maioria das respostas seria massas. E aí vai uma informação importante: **hoje, no contexto da maioria das empresas, ML e IA são sinônimos**.

O que se entende por IA foi mudando ao longo do tempo: até os anos 1990 IA poderia ser sinônimo de força bruta (o Deep Blue, famoso na década de 1990 por vencer o campeão mundial de xadrez Garry Kasparov, usava força bruta para ganhar: isto é, ele avaliava milhões de combinações futuras de jogadas e escolhia a melhor) e, ainda, de **sistemas especialistas** contendo várias regras construídas à mão: um sistema responsável pela segurança dos equipamentos e maquinário de uma usina hidrelétrica entra nessa regra, por exemplo. Por outro lado, esses algoritmos podem ser complexos para serem construídos e geridos – principalmente com o exponencial aumento de regras e necessidade de ajustes nessas regras ao longo do tempo. Já o que veio nos anos 2000 é o ML: algoritmos capazes de reconhecer padrões e aprender, de forma generalizada, a partir de uma base de dados histórica. É esse o tipo de algoritmo utilizado atualmente e entendido como “IA”. Já as superinteligências dos filmes (ainda) não são realidade, mas podem ser em um futuro próximo. Note que Ciência de Dados inclui IA, mas não é somente isto: também inclui Estatística, conhecimento sobre a área na qual se aplicaria o problema (como Vendas, Logística, RH, Jurídico e outros), e de computação.

**1.1 *MACHINE LEARNING***

Continuando: da mesma forma que a macarronada é um tipo de massa, existem técnicas e bibliotecas mais simples em ML, como o **scikit-learn**. Essa é uma das bibliotecas (lembra da nossa última conversa sobre bibliotecas?) mais conhecidas em Python e inclui técnicas simples como uma árvore de decisão ou um algoritmo de agrupamento de dados.

Por outro lado, a lasanha é outro tipo de massa mais complexa. Da mesma forma, também existem algoritmos mais complexos como as **redes neurais** – algoritmos que tiveram suas origens em trabalhos que tentaram traduzir a estrutura dos neurônios de um cérebro em algoritmos e circuitos. Esse tipo de algoritmo possui várias funções de ativação (como neurônios) agrupados em **camadas**. Existem, ainda, diversas conexões entre essas camadas. Sabemos que esse tipo de algoritmo é tratado como se fosse o melhor de todos, mas aqui vale um ponto de atenção: entenda as redes neurais como um “canhão para matar uma formiga”: às vezes, o problema a ser resolvido é tão simples que não precisaria de uma rede neural. Por outro lado, se mesmo assim você quiser usar uma rede neural, você pode acabar tendo um resultado pior do que um algoritmo mais simples – em outras palavras, o canhão também pode “errar” a formiga.

Existem várias arquiteturas de redes neurais como o multi-layer perceptron (MLP), recurrent neural network (RNN), feed-forward (FF), entre outros. Na prática, existem dezenas de arquiteturas: servem tanto para trabalhar com imagens, textos, sons, prever valores, grupos, entre outros: da mesma forma que também existem diferentes tipos de lasanhas. E, ainda, existem lasanhas com várias e várias camadas – da mesma forma, existem redes neurais com várias e várias camadas: essas redes se chamam *deep learning* – ou aprendizagem profunda. *A escolha da melhor arquitetura, biblioteca ou tipo de algoritmo depende muito de cada caso. Existem cientistas de dados que possuem algoritmos e técnicas de “estimação”, enquanto outros cientistas gostam de testar diferentes técnicas para cada problema. A escolha da melhor técnica que se aplica a cada caso é um exemplo de trabalho de Ciência de Dados.*

**TEMA 2 – TRABALHANDO COM PROBLEMAS DE CIÊNCIA DE DADOS**

Um projeto de Ciência de Dados sempre utiliza alguns passos – o nome dos passos e a ordem podem mudar ligeiramente dependendo de cada metodologia. Por outro lado, a execução possui alguns passos típicos. São eles:

1. a definição do problema;
2. a definição dos dados;
3. a preparação dos dados;
4. o desenvolvimento dos modelos;
5. a avaliação dos modelos; e
6. a disponibilização dos modelos.

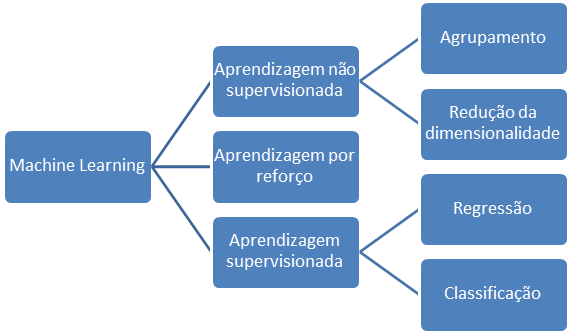
Vamos com calma por cada um desses itens. Primeiro, falaremos aqui sob um olhar de **projeto**, e depois sobre **execução**.

**2.1 DEFINIÇÃO DO PROBLEMA**

Esse passo trata, em sua essência, de responder: “o que você quer fazer com ML?”. Ou, ainda, em que ML pode te ajudar a resolver algum problema. Pensemos do ponto de vista de TI, e não sejamos vagos: “quero diminuir os gastos da empresa” é algo vago. Existem tantas formas de se diminuir os gastos e várias delas não envolvem ML – o consumo consciente de água por parte das pessoas já ajudaria isso, e nada teria a ver com ML.

Agora, um exemplo que poderia envolver ML é: “quero saber quais são os clientes que podem ser inadimplentes no futuro próximo”. Nesse caso, temos um exemplo específico do **escopo** do trabalho (os clientes), qual é a **dor** **da área de negócio** a ser endereçada (as inadimplências) e **o que se deseja** fazer (prever e apontar quem poderiam ser os futuros inadimplentes).

Ao definir o problema, poderemos saber qual é a melhor tipo de técnica a ser utilizada. ML é dividido em alguns subgrupos, conforme ilustrado pela figura a seguir.



A **aprendizagem supervisionada**seaplica quando sabemos **o que**queremos prever e em base **do quê**. Ela é dividida em dois principais tipos.

* **classificação**: quando queremos prever um “grupo”. Pode ser algo como cartão de crédito aprovado/cartão de crédito reprovado; sim/não; foto de cachorro/foto de gato; candidato pertencente ao grupo A/B/C.
* **regressão**: quando queremos prever um valor em uma “faixa numérica”. Pode ser algo como temperatura, cotação do Bitcoin, número de novos acessos no site, valor de venda da casa, nota do estudante.

Quando trabalhamos com problemas de aprendizagem supervisionada partimos da premissa de que precisamos de **uma base de dados**. Se queremos prever a temperatura de uma cidade para os próximos meses precisaremos de uma boa base histórica dela contendo, se possível, alguns anos de dados contendo informações que possuam relação com o que queremos prever (para pegarmos o comportamento dependendo da estação do ano; a relação entre temperatura, vento e chuva; entre outros); e, naturalmente, a coluna que queremos prever (no caso, a temperatura).

Caso queiramos prever se um futuro cliente terá ou não o seu pedido de cartão de crédito aprovado a partir do seu histórico, precisaremos ter também uma base de dados histórica com outros clientes, contendo dados que são relevantes à concessão (ou não) do cartão de crédito e incluindo também a coluna que queremos prever (no caso, se o cartão de crédito foi ou não aprovado). Ou, ainda, se queremos prever se uma nova imagem possui um gato ou cachorro dentro dela, precisaremos obrigatoriamente ter em mãos uma base de dados contendo várias outras fotos com cachorros ou gatos.

Já a **aprendizagem não supervisionada**trata majoritariamente da **exploração**de dados: às vezes, não sabemos exatamente **o que**queremos prever, mas precisamos manipular a base de dados, como a seguir.

* **redução da dimensionalidade**: quando precisamos selecionar quais dados realmente importam/são relevantes. Peguemos uma base de dados da meteorologia com 100 colunas diferentes: será que todaselas são importantes? Quando temos um número muito alto de dados, podemos cair naquilo que se chama de *maldição da dimensionalidade*: como várias técnicas de aprendizagem supervisionada em ML tratam de reconhecer padrões, fica bem difícil de detectar padrões com tantas colunas a serem analisadas. Logo, às vezes, é preferível selecionarmos somente aquelas que são realmente úteis para o nosso problema.
* **agrupamento** (também mencionado, às vezes, como *clusterização*): peguemos as mesmas 100 colunas que comentamos anteriormente – será que não conseguiríamos de alguma forma agrupá-las de acordo com a sua similaridade em novas colunas de um jeito que não percamos muita informação ao fazermos isso?

Por fim, a **aprendizagem por reforço** inclui técnicas que são orientadas a resolver problemas a partir de **recompensas**. Incluem-se aqui algoritmos que buscam otimizartrabalhos complexos como a distribuição de rotas logísticas ou de escalas de trabalho tentando descobrir quais pequenos ajustes levam a um melhor resultado ou, ainda, certos algoritmos usados em jogos de computador e de console.

É possível dizer que, no contexto do mercado de trabalho, grande maioria dos trabalhos de Ciência de Dados se concentram em problemas de aprendizagem supervisionada e não supervisionada.

**2.2 DEFINIÇÃO DOS DADOS**

Um bom projeto de ciência de dados costuma ter também o apoio de quem utilizará essa solução e que entende do problema. Nesse caso, quem entende do problema são as pessoas que lidam com as inadimplências hoje. É imprescindível que elas caminhem junto para resolver o problema. Pensemos aqui: quais fatores impactam? A visão do cientista de dados e do analista da área de negócio que o apoiará são igualmente importantes:

* o **analista da área de negócio**dará várias informações que fazem parte já da **experiência**dele: ele provavelmente pode citar itens como o período do ano (já que podem ter épocas do ano em que a inadimplência é maior); o score de crédito; a data de fundação do comércio; entre outros. Essas informações são valiosas para você por dois motivos.
  + ganho de tempo: você “aproveita” a experiência dessa pessoa ao descobrir com antecedência quais informações **podem ser**importantes. Às vezes, a intuição da pessoa pode estar errada em alguns pontos, mas estará certa em outros: logo, sempre haverá algo bom a se aproveitar.
  + confiança: é o analista (e/ou a equipe na qual essa pessoa faz parte) quem fará uso do algoritmo. Como quase nada na vida é 100% previsível, é improvável que o algoritmo acerte 100% das vezes. Por outro lado, se essa pessoa nunca participou no processo de construção do algoritmo e não sabe quais fatores foram levados em conta, no primeiro momento em que o algoritmo der um resultado diferente do esperado haverá uma desconfiança. E, como você deve imaginar, quando alguém não confia em algo é bem difícil de mudarmos a sua opinião.
* o **cientista de dados** ajudará a trazer um conhecimento fora do contexto do dia a dia do analista: é parte do trabalho do cientista de dados pesquisar em bases científicas quais foram os principais achados sobre um determinado problema: no caso da inadimplência, existem vários estudos apontando quais fatores culminam na inadimplência – pode ser que alguns desses fatores não sejam do conhecimento do analista e poderão ser úteis para o seu trabalho. Logo, um trabalho de ciência de dados vai além de criar somente um algoritmo – é também o de ajudar os humanos em seu trabalho. Além disso, ele poderá trazer a sua experiência em projetos anteriores para fazer perguntas que possam ajudar no desenvolvimento do algoritmo: será que a tendênciado score nos últimos 12 meses é também relevante? Um score baixo pode ser uma coisa, mas e se o score estiver melhorando? Ou, ainda, e se o score é alto, mas está caindo vertiginosamente? Será que isso não é um sinal preocupante? Será que a inflação ao longo do tempo também não poderia ser relevante? Ou a cidade? Cidades menores e em regiões rurais podem ter uma dependência maior com a época da colheita de principais safras, ou cidades mais turísticas podem receber movimentar mais dinheiro nos finais de ano ou feriados estendidos. Veja que é um pensamento “fora da caixa” que pode ser bem-vindo durante o desenvolvimento do projeto.

Com o apoio dessas pessoas, é possível definir **a base de dados**a ser utilizada. Isso ocorre principalmente porque algoritmos de ML não “descobrem” sozinhos novos dados: é importante que um humano informe ao algoritmo que conjunto de dados deve ser utilizado. Esse conjunto pode vir de um banco de dados da própria empresa ou, ainda, de fontes de dados externas como sites governamentais, empresas de consultoria especializadas ou sites de acesso público. A preocupação aqui é a de garantir que a base de dados faça sentido no futuro: ou seja, que todas as informações a serem usadas estejam disponíveis também ao testarmos novos dados futuros.

Peguemos novamente o caso da inadimplência: não adianta treinar um algoritmo para prever casos futuros de inadimplência usando como apoio uma base de dados governamental que foi desativada no mês passado. Novos casos jamais poderão utilizar essa mesma base e, assim, você tem um algoritmo que não funcionará corretamente no futuro.

Algoritmos de ML são capazes de aprender sobre um conjunto de dados predefinido e, a partir dele, cria-se uma **generalização**para que possa funcionar bem com casos futuros. Imaginemos o próprio caso dos nossos estudos: você não quer só criar algoritmos com os conjuntos de dados trabalhados com o que nós aprendemos, não é? Na realidade, você quer aprender diferentes tipos e aplicações de algoritmos, saber o que funciona e o que não funciona para que você possa **aprender com os exemplos** e saber aplicar em **novos cenários**no futuro.

É como se você estivesse aprendendo a dirigir um carro na autoescola pela primeira vez: você não quer aprender a dirigir somente o Volkswagen Up 2016 1.0 branco, placa ABC-1234 da “Autoescola Centro” e somente no bairro “Ribeira”, mas você quer aprender diferentes situações de tempo, ruas, trânsito e outros carros para que você saiba o que fazer com outros carros, outros horários e outras ruas, não é? Por outro lado, para que você saiba o que fazer, você precisa ser exposto a diferentes cenários enquanto está aprendendo. Como você vai saber como estacionar o seu próprio carro se nunca teve contato com isso na vida? Como irá reagir se nunca você nem viu uma aula ou vídeo ensinando isso?

**2.3 PREPARAÇÃO DOS DADOS**

Com os algoritmos de ML é a mesma coisa: precisamos expor a eles as situações nas quais desejamos que ele aprenda para que ele saiba como reagir no futuro. Se vamos criar um algoritmo que vá prever a temperatura nas próximas horas é importante que forneçamos uma base para que o algoritmo aprenda. Agora, que base de dados é essa? Vamos imaginar uma planilha de Excel. Vamos pensar na sua própria cidade: qual é a granularidade? Queremos prever por hora? Por dia? Por semana? Por mês? Bom, comentamos anteriormente que seria por hora – logo, teremos **no mínimo** duas informações, a data com a hora em uma coluna e a temperatura em outra. Vamos usar os dados do Instituto Nacional de Meteorologia (INMET) para a cidade de Curitiba como base:

|  |  |
| --- | --- |
| **DataHora** | **Temperatura** |
| 2021-01-01 00:00 | 20.5 |
| 2021-01-01 01:00 | 19.9 |
| 2021-01-01 02:00 | 19.7 |
| 2021-01-01 03:00 | 19.6 |
| 2021-01-01 04:00 | 19.9 |
| 2021-01-01 05:00 | 18 |
| 2021-01-01 06:00 | 18 |
| 2021-01-01 07:00 | 17.7 |
| 2021-01-01 08:00 | 18.3 |
| 2021-01-01 09:00 | 18.9 |
| 2021-01-01 10:00 | 20.3 |
| 2021-01-01 11:00 | 21 |
| 2021-01-01 12:00 | 21 |
| 2021-01-01 13:00 | 20 |
| 2021-01-01 14:00 | 22.8 |

Veja que essa pequena amostra somente possui dados de algumas horas do dia 1º de janeiro de 2021. Vamos imaginar que queiramos prever a temperatura das 15h do dia 1º de janeiro de 2021 (em que, a partir de agora, sempre deixaremos no padrão *ano-mês-dia hora:minuto*, também convencionado nas documentações das bibliotecas como *yyyy-mm-dd hh:mm*– logo, essa data será escrita como 2021-01-01 15:00). Bom, nós, humanos, esperamos que a temperatura fique por volta dos 22 graus. Sabemos disso porque nas últimas horas a temperatura se manteve nessa faixa e porque é uma tarde de verão. Quer dizer, pode ser que abaixe um pouco (talvez uns 19 graus) ou aumente (talvez 25 graus), talvez? Veja que, para chegar a essa decisão, temos todo um conjunto de conhecimentos prévios: sabemos que a temperatura geralmente não muda muito hora a hora; sabemos que a temperatura à tarde geralmenteé maior do que a temperatura de madrugada; sabemos que uma mudança de 2 ou 3 graus é aceitável, mas 20 ou 30 graus de uma hora para outra não. Veja, também, que esse conhecimento depende de onde vivemos: se você cresceu em uma cidade na qual a temperatura não muda muito de uma hora para outra, você provavelmente chegará à decisão diferente de outra pessoa que está acostumada a grandes variações na temperatura (como é o próprio caso de Curitiba).

Agora, como um algoritmo vai aprender todos esses conceitos? Um algoritmo não sabe o que é “hora”, “sol” ou “Curitiba”. É como se tudo fosse grego para ele – algo incompreensível, algo assim:

|  |  |
| --- | --- |
| **ΗμερομηνίαΩρα** | **Θερμοκρασία** |
| ??? | 20.5 |
| ??? | 19.9 |
| ??? | 19.7 |
| ??? | 19.6 |
| ??? | 19.9 |
| ??? | 18 |
| ??? | 18 |
| ??? | 17.7 |
| ??? | 18.3 |
| ??? | 18.9 |
| ??? | 20.3 |
| ??? | 21 |
| ??? | 21 |
| ??? | 20 |
| ??? | 22.8 |

Mas, e aí? Como resolvemos isso? Uma alternativa é incluirmos as informações que são importantes para que o algoritmo tome uma decisão. O que influencia muitoa temperatura em Curitiba? Será que a quantidade de liquidificadores ligados ao mesmo tempo em São Paulo influencia? Ou seria a umidade, ou a incidência do sol (isto é, se o céu está limpo ou nublado, por exemplo)? O fato de estar chovendo ou não, talvez? Ou o vento? Ou a quantidade de potes de margarina vendidos no maior supermercado da cidade?

Veja que existem informações mais úteis do que outras, e algumas delas realmente mais atrapalham do que ajudam. Não sei se percebeu, mas às 14h a temperatura aumentou quase 3 graus. Se você fosse apostar a temperatura às 15h olhando somente o histórico da temperatura, o que você apostaria? Agora observe como fica a tabela quando incluímos a informação da radiação solar:

|  |  |  |
| --- | --- | --- |
| **DataHora** | **Temperatura** | **RadiacaoSolar** |
| 2021-01-01 00:00 | 20.5 |  |
| 2021-01-01 01:00 | 19.9 |  |
| 2021-01-01 02:00 | 19.7 |  |
| 2021-01-01 03:00 | 19.6 |  |
| 2021-01-01 04:00 | 19.9 |  |
| 2021-01-01 05:00 | 18 |  |
| 2021-01-01 06:00 | 18 |  |
| 2021-01-01 07:00 | 17.7 |  |
| 2021-01-01 08:00 | 18.3 | 33.3 |
| 2021-01-01 09:00 | 18.9 | 268.2 |
| 2021-01-01 10:00 | 20.3 | 583.9 |
| 2021-01-01 11:00 | 21 | 848.6 |
| 2021-01-01 12:00 | 21 | 572.5 |
| 2021-01-01 13:00 | 20 | 621.6 |
| 2021-01-01 14:00 | 22.8 | 2513.2 |

A radiação solar aumentou muito de 13h para 14h: aparentemente o céu ficou com menos nuvens nessa hora. E agora? Você mudaria o valor da sua “aposta” na temperatura às 15h? Veja o que realmente aconteceu:

|  |  |  |
| --- | --- | --- |
| **DataHora** | **Temperatura** | **RadiacaoSolar** |
| 2021-01-01 00:00 | 20.5 |  |
| 2021-01-01 01:00 | 19.9 |  |
| 2021-01-01 02:00 | 19.7 |  |
| 2021-01-01 03:00 | 19.6 |  |
| 2021-01-01 04:00 | 19.9 |  |
| 2021-01-01 05:00 | 18 |  |
| 2021-01-01 06:00 | 18 |  |
| 2021-01-01 07:00 | 17.7 |  |
| 2021-01-01 08:00 | 18.3 | 33.3 |
| 2021-01-01 09:00 | 18.9 | 268.2 |
| 2021-01-01 10:00 | 20.3 | 583.9 |
| 2021-01-01 11:00 | 21 | 848.6 |
| 2021-01-01 12:00 | 21 | 572.5 |
| 2021-01-01 13:00 | 20 | 621.6 |
| 2021-01-01 14:00 | 22.8 | 2513.2 |
| 2021-01-01 15:00 | 23.2 | 2337.6 |

A radiação solar diminuiu um pouco, mas permaneceu em um patamar alto. A temperatura aumentou um pouco também, pulando de 22.8 para 23.2 graus. É nesse sentido que um cientista de dados poderia trabalhar: entendendo e mapeando quais variáveis poderiam ajudar na tomada de decisão. Nesse caso, poderíamos incluir a umidade, vento e pressão atmosférica (mas não os liquidificadores ou os potes de margarina, que nada possuem relação com o nosso caso).

**2.4 DESENVOLVIMENTO DOS MODELOS**

O passo de gerar uma base de dados confiável (o que também inclui limpara base de dados de possíveis erros – se alguém errou um ponto a temperatura de algum dia pode ser apontada como “213” em vez de “21.3” graus) e o passo de desenvolver os modelos pode demandar até 80% de um projeto. É um trabalho que envolve muitos testes e, naturalmente, muito “vai e volta”.

Para os casos de **aprendizagem supervisionada**, retornemos novamente ao caso das inadimplências: após termos uma base de dados em mãos poderemos realizar o treinamento de um modelo preditivo. Para isso, é importante dividirmos a nossa base de dados original em pelo menos duas partes.

* A **base de treino** é uma amostra obtida a partir da base original (entre 60% e 80% da base na maioria das vezes) e é essa base que utilizamos para *treinar*um algoritmo – isto é, é ela que utilizamos para que um algoritmo reconheça e “aprenda” os padrões que estão dentro daquela base e, se possível, algumas exceções.
* A **base de testes** é o restante da base que não foi utilizada pela base de treinamento. Ela é usada para avaliarum algoritmo. Esse passo é importante porque representa como ele funcionaria na vida real e nos dá uma forma de compararum valor previsto com um valor que de fato aconteceu já que nós (humanos) sabemos o resultado que o algoritmo deveria prever, mas ele (o algoritmo) não. Afinal de contas, é fácil o algoritmo ter uma taxa de acerto alta olhando somente para a base de treino, mas não é nada representativo sobre como o algoritmo iria desempenhar no dia a dia: já a métrica de acerto sobre a base de testes possui uma representatividade bem melhor nesse sentido.

Ainda nesse passo, escolheremos o algoritmo a ser utilizado. Existem várias técnicas.

* aprendizagem supervisionada:
  + modelos lineares;
  + algoritmos estatísticos;
  + redes neurais;
  + árvores de decisão;
  + máquinas de vetor de suporte (*support vector machines*, ou SVMS);
  + florestas (múltiplas) de árvores de decisão; e
  + comitês de algoritmos.
* aprendizagem não supervisionada:
  + análise fatorial;
  + análise de componentes principais (*principal component analysis*, ou PCA);
  + K-médias (*k-means*); e
  + eliminação recursiva de atributos.

**2.5 AVALIAÇÃO DOS MODELOS**

Após treinar um modelo, é importante avaliarmos se ele está funcionando como esperado (ou não). Falar de “acurácia” é um termo complicado porque existem dezenas de métricas que possuem como objetivo explicar se um modelo está funcionando como esperado – logo, a forma de se calcular essa acuracidade muda radicalmente dependendo do caso.

Para saber se os resultados estão funcionando de acordo com o esperado, as métricas mais comuns são as que se segue.

* aprendizagem supervisionada
  + regressão:
    - erro médio absoluto (*mean absolute error*, ou MAE);
    - erro quadrático médio (*mean squared error*, ou MSE);
    - erro médio absoluto percentual (*mean absolute percentage error*, ou MAPE); e
    - raiz quadrada do erro quadrático médio (*root* *mean squared error*, ou RMSE).
  + classificação:
    - matriz de confusão;
    - precisão e revocação (*precision*e *recall*); e
    - curva ROC (*receiver operating characteristic*) e a área sob essa curva (*area under curve*, ou AUC).
* aprendizagem não supervisionada
  + testes estatísticos, como os testes de hipótese.

**2.6 DISPONIBILIZAÇÃO DOS MODELOS**

Após os resultados estarem aceitáveis (e a definição do “aceitável” depende muito de cada caso: às vezes, 75% de acerto para as inadimplências pode ser algo muito bom, mas os mesmos 75% para um algoritmo da área médica pode ser completamente inaceitável), o modelo pode ser disponibilizado para o uso diário.

Essa disponibilização (*deployment*) do modelo trata de deixá-lo disponível para um uso futuro. Isso inclui a sua integração com uma arquitetura técnica em uso por uma empresa e a forma que os dados serão alimentados ao algoritmo e seus resultados propriamente utilizados. É nesse ponto que se determina quem visualizará os valores e de qual forma.

**TEMA 3 – TRABALHANDO COM DADOS**

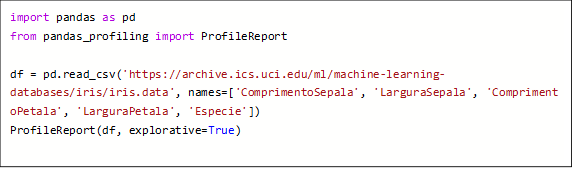
Agora que conversamos sobre o passo a passo de um projeto típico de Ciência de Dados, poderemos falar sobre o trabalho em si. Existem trabalhos nessa área que são executados em várias linguagens de programação, como Scala, R, Java e Python. Do ponto de vista corporativo, o Python recentemente se tornou a mais utilizada dentre essas.

As bases de dados (*datasets*) trabalhadas em ML possuem alguns tipos – os principais são os que se seguem.

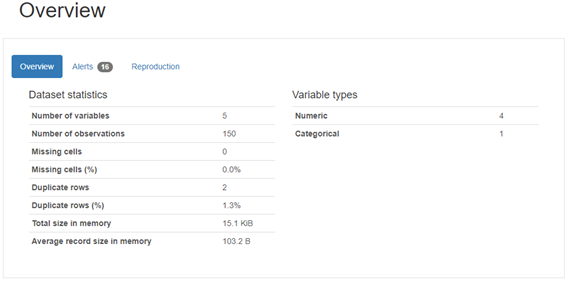
1. Dados **tabulares**: são aqueles que se parecem com planilhas de Excel, e são os mais comuns. São caracterizados por **atributos** (ou *features*): características (colunas) que fazem parte da base; e **instâncias**, isto é, indivíduos (linhas) que fazem parte da base de dados.
   1. Para problemas de aprendizagem supervisionada podemos ter um **label** ou **target** – um atributo que é aquele que queremos prever.
   2. Os dados podem ser numéricos (como valores referentes à temperatura, idade, preços e afins); categóricos (como aprovado/reprovado; as categorias da Carteira Nacional de Habilitação; tipos sanguíneos; entre outros); dados temporais (que possuem relação com data e hora, por exemplo); e textos (como esta descrição que você está lendo agora).
2. Bases de **imagens**: são fotos ou outras imagens as quais queremos detectar o seu contexto (por exemplo: ao tirarmos uma foto de dentro de um comércio estamos interessados em saber se ele é uma loja de autopeças ou um supermercado “ao olhar o todo”, mas não que estejamos interessados em reconhecer cada item das gôndolas) ou seus objetos (ou seja, quais produtosestão dentro das gôndolas sendo mostradas na foto).
3. Bases de **texto**: são conjuntos de texto nos quais queremos entender o seu contexto; encontrar entidades (pessoas ou lugares); detectar similaridades entre documentos; traduções; entre outros.

Independentemente da base sendo trabalhada é importante relembrar de um conceito chamado *garbage in, garbage out*(entra lixo, sai lixo): algoritmos de ML aprendem em cima de uma base em que os humanos fornecem. Logo, é importante que nós (humanos) repassemos uma base confiável e previamente curada por nós. Ainda que a ideia possa soar interessante, por enquanto, os algoritmos ainda não conseguem adivinhar o que nós pensamos e, ainda, procurar novos dados da internet sem supervisão alguma.

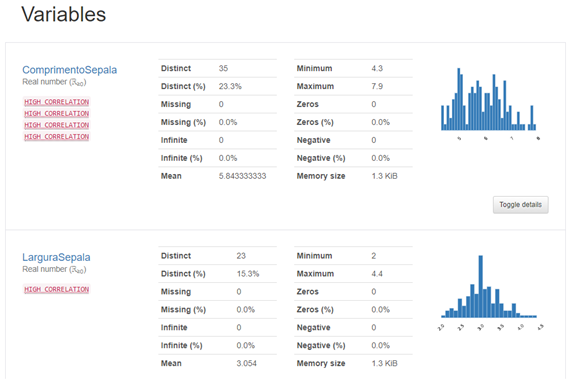
Nesse passo, nós podemos realizar uma **análise de dados exploratória** para ver como está o relacionamento entre os dados. Uma das bases de dados mais simples é a *Iris*, a qual possui dados sobre o comprimento e a largura das sépalas e pétalas de três espécies de flor do gênero Iris. O código a seguir combina duas bibliotecas para realizar a análise exploratória: o pandasspan >, responsável por gerar o DataFrame a partir dos dados de um arquivo CSV, e o span >pandas-profilingspan >, o qual nos gera um poderoso relatório sobre um DataFrame.



O relatório resultante nos fornece várias informações, como: o número de atributos (*variables*) e instâncias (*observations*), bem como os dados nulos e o espaço em memória gasto:



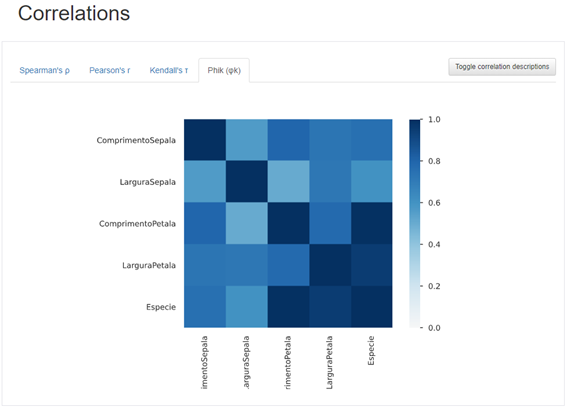
A distribuição dos dados entre as variáveis:



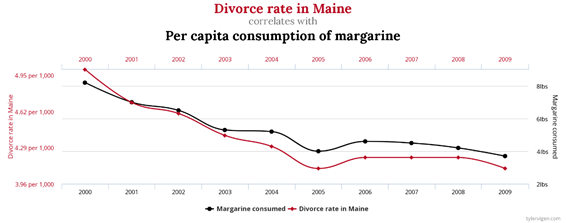
Dados duplicados:



As correlações com diferentes técnicas:



Entre outros. Isso é útil para detectarmos possíveis relacionamentos importantes entre as variáveis e que podem ser úteis para validar a base antes de avançarmos com os modelos. Sobre as correlações, vale também comentar sobre as “correlações espúrias”: não é porque temos uma alta correlação entre duas variáveis que isso significa alguma coisa útil. Um exemplo clássico é a alta correlação entre a porcentagem de divórcios no estado do Maine, nos EUA, com o consumo per capita de margarina:



Fonte: Vigen, [S.d.].

Observe que ambos possuem uma correlação altíssima: quando um aumenta, o outro também aumenta; quando um abaixa, o outro também abaixa. Por outro lado, um não influencia no outro de forma alguma. Tome o mesmo cuidado ao trabalhar com bases de dados, pois certamente não queremos detectar correlações espúrias nela!

Nesse momento, também podemos criar atributos adicionais ou manipular os que já temos. Os tamanhos máximo e mínimo de todas as dimensões da sépala e pétala são diferentes, e existem algoritmos (como as próprias redes neurais) que se beneficiam se todos estiverem na mesma escala. Logo, o código a seguir já poderia endereçar isso usando a MinMaxScaler:

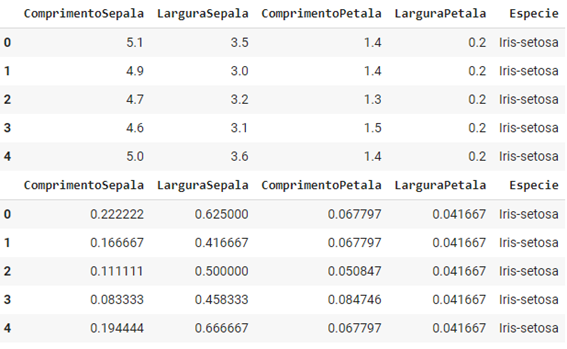
from sklearn.preprocessing import MinMaxScaler

display(df.head())

# colocando todas as colunas na mesma escala numérica
df_scaled = df.copy()
df_scaled[['ComprimentoSepala', 'LarguraSepala', 'ComprimentoPetala', 'LarguraPetala']] = MinMaxScaler().fit_transform(df_scaled[['ComprimentoSepala', 'LarguraSepala', 'ComprimentoPetala', 'LarguraPetala']])
display(df_scaled.head())



Agora, veja como eram as primeiras cinco linhas da tabela **antes**de colocarmos todos na mesma escala e como ficou depois:



Existem ainda outras técnicas, como:

* a transformação de categorias em múltiplas colunas numéricas;
* a extração de números como a semana, o dia da semana ou o mês a partir de uma coluna de data e hora; e
* o preenchimento de valores nulos com zeros ou outros valores.

**TEMA 4 – TRABALHANDO COM MODELOS**

Antes de qualquer coisa, vale um lembrete: **um modelo de Ciência de Dados não é necessariamente um modelo de ML!**Dependendo do caso, podemos ter modelos de áreas como Estatística e Econometria, por exemplo. Por outro lado, ML possui várias aplicações úteis no dia a dia.

As bibliotecas mais conhecidas em Python para a criação de modelos de ML são as que se seguem.

* Scikit-learn
  + O scikit-learn é potencialmente a biblioteca de ML mais versátil. Ela possui implementações de técnicas de aprendizagem supervisionada e não supervisionada.
  + Seus modelos vão desde a regressão linear até modelos de redes neurais, passando por SVMs, comitês de algoritmos, árvores de decisão e outros.
  + Possui técnicas de pré-processamento de dados e métricas de acuracidade dos algoritmos.
* Tensorflow, Keras e PyTorch
  + Os três são bibliotecas para o desenvolvimento de algoritmos de redes neurais (e *deep learning*). Pela complexidade desse tipo de algoritmo, recomenda-se que sejam usados computadores com mais poder de processamento.
  + O Tensorflow também possui um playground disponível para se testar diferentes configurações de redes com problemas simples e, assim, ver como o processo de treinamento de uma rede neural funciona na prática.
* LightGBM e XGBoost
  + Ambas as bibliotecas trabalham com *gradient boosting*, uma técnica de ML a qual é composta por várias árvores de decisão. Nessa técnica, as árvores vão sendo adicionadas no modelo para se obter um bom resultado de forma que uma possa reduzir o erro das anteriores.
* Statsmodels
  + O statsmodels é uma biblioteca com foco estatístico, e não exatamente em ML. De qualquer modo, ela permanece sendo relevante ao incluir funções para séries temporais (como o ARIMA) e outras técnicas de estatística descritiva.

Vejamos um exemplo do uso do scikit-learn. Testaremos dois modelos diferentes (RandomForest e SVC – um tipo de SVM específico para problemas de classificação) para a base de dados Iris, a qual comentamos anteriormente. Na prática, seria algo como: “aprenda os padrões entre os tamanhos da pétala e sépala de uma flor que determinam qual é a espécie dela. Validaremos em seguida com ‘novos’ exemplares de flor”.

Para melhor ilustrar como um algoritmo aprendeu utilizaremos também a biblioteca mlxtend junto com o matplotlib, que já vimos antes. Dito isso, observe o seguinte código. Note também os comentários (sempre iniciados com um “#”) para que saiba o que cada seção do código está fazendo. Nesse caso, estamos considerando somente o comprimento (e não a largura) da pétala e da sépala para tomar uma decisão.

Observe também que introduzimos duas novas funções que não vimos antes: a OrdinalEncoder, a qual transforma somente as categorias da coluna “Especie” que hoje estão como texto (“Iris-setosa”, “Iris-virginica” e “Iris-versicolor”) para números (0, 1 e 2). Essa transformação é necessária para utilizar a função plot\_decision\_regions do mlxtend. Além disso, usamos o train\_test\_split para dividir a nossa base de dados em duas partes: uma de treinamento e outra de teste. Veja que o parâmetro “test\_size” está com o valor 0.4. A explicação sobre esse parâmetro se encontra na própria documentação do scikit-learn e indica a porcentagem de distribuição aleatória para a base de treino e teste: no caso, 40% para a base de teste e os 60% restantes para a base de treinamento.

Ainda, veja que em alguns momentos existe também o parâmetro “random\_state”. Ele também é útil ao permitir a **reprodutibilidade** dos nossos testes. Vários algoritmos inicializam de forma aleatória e, para fins de teste e confiabilidade do algoritmo, em alguns casos, é importante garantir que conseguimos ter o mesmo resultado se rodarmos o mesmo código hoje ou amanhã, ou em computadores diferentes considerando a mesma base de dados. Dessa forma, ao indicarmos um valor para o “random\_state” (que chamamos de *seed*), garantiremos que o número aleatório utilizado para gerar a lógica do algoritmo será sempre o mesmo.

Veja, também, que no *train\_test\_split*não estamos só informando o dataframe *df*, mas sim dois parâmetros: df[['ComprimentoSepala', 'ComprimentoPetala']] (isto é, os atributos que queremos considerar no treinamento do modelo) e df['Especie'] (ou seja, o atributo que desejamos prever). O resultado, então, são quatro variáveis: os atributos utilizados para treinar e testar o modelo (X\_train e X\_test, respectivamente) – isto é, as variáveis que devem ser usadas pelo modelo aprender e reconhecer padrões para prever o nosso *target* (no caso, o atributo “Especie”); e o *target*em si, também correspondente à base de treinamento e teste e exatamente na mesma ordem do X\_train e X\_test, e que convenientemente chamamos de y\_train e y\_test.

# importação de libraries
import matplotlib.pyplot as plt

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split

from mlxtend.plotting import plot_decision_regions

# transformando o texto da espécie em categorias numéricas
df['Especie'] = OrdinalEncoder().fit_transform(df[['Especie']]).astype(int)

# dividindo a base de dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(df[['ComprimentoSepala', 'ComprimentoPetala']], df['Especie'], test_size=0.4, random_state=0)

# treinando o modelo de RandomForest (RF)
modelo_rf = RandomForestClassifier(random_state=0)
modelo_rf.fit(X_train, y_train)

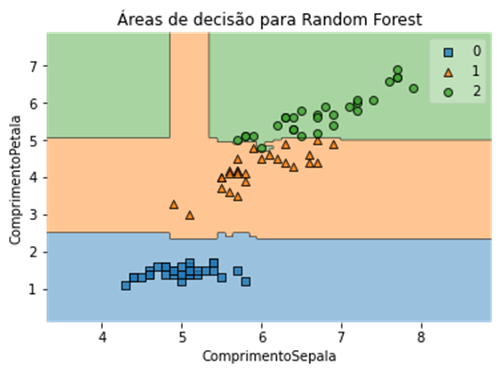
# mostrando como ele tomou decisões
fig = plot_decision_regions(X=X_train.values, y=y_train.values, clf=modelo_rf)
plt.xlabel('ComprimentoSepala')
plt.ylabel('ComprimentoPetala')
plt.title('Áreas de decisão para Random Forest')
plt.show()

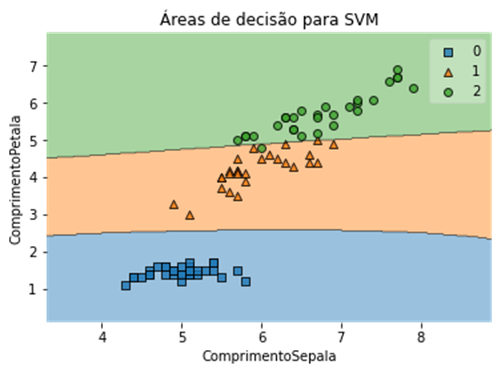
# treinando o modelo de GradientBoostingClassifier (RF)
modelo_svm = SVC(random_state=0)
modelo_svm.fit(X_train, y_train)

# mostrando como ele tomou decisões
fig = plot_decision_regions(X=X_train.values, y=y_train.values, clf=modelo_svm)
plt.xlabel('ComprimentoSepala')
plt.ylabel('ComprimentoPetala')
plt.title('Áreas de decisão para SVM')
plt.show()



Agora, veja como cada um dos modelos aprendeu:





**TEMA 5 – TRABALHANDO COM RESULTADOS**

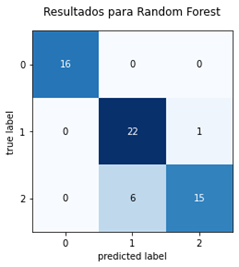
Veja que, no exemplo anterior, dividimos a base em uma de treinamento e de teste. Por outro lado, somente mostramos como o algoritmo aprendeu a partir da base de treinamento (no caso, X\_train e y\_train). Lembra quando comentamos que um dos passos necessários é a **avaliação**dos resultados? Pois bem – vejamos quais foram os resultados via matriz de confusão para as duas técnicas. Veja que agora estamos utilizando a base de testes para validar os resultados:

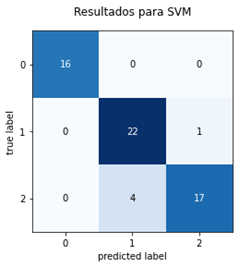
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix

plot_confusion_matrix(confusion_matrix(y_test, modelo_rf.predict(X_test)))
plt.title('Resultados para Random Forest')

plot_confusion_matrix(confusion_matrix(y_test, modelo_svm.predict(X_test)))
plt.title('Resultados para SVM')







Veja que cada linha indica a classe real e cada coluna indica qual foi o valor predito. Queremos o máximo de acertos para todas as classes. Logo, gostaríamos de ter o máximo de valores na diagonal principal da matriz: o máximo de valores para a primeira coluna da primeira linha; da segunda coluna da segunda linha e da terceira coluna da terceira linha. Veja que ambos acertaram todos os valores da classe 0 e erraram somente 1 caso da classe 1 (no caso, houve um 1 caso em que uma flor era da classe 1 e os algoritmos incorretamente classificaram como sendo da classe 2). O Random Forest classificou corretamente 15 flores como sendo da classe 2, mas 6 flores da mesma classe foram incorretamente classificadas como sendo da classe 1. O SVM foi um pouco melhor: acertou 17 flores e errou 4 da mesma classe.

**FINALIZANDO**

Aqui, nesta nossa conversa, tivemos a oportunidade de prover a você uma breve introdução ao mundo de Ciência de Dados em Python. O termo “Ciência” pressupõe o uso da Metodologia Científica para a análise e modelagem de dados e, com isso, também requer um rigor científico igualmente importante. Por essa razão, naturalmente, há uma fundamentação teórica forte por trás de um trabalho dessa área a qual não se restringe somente a ML: também é perfeitamente possível usarmos técnicas estatísticas para resolver esse problema.

Isso posto, mostramos o que se entende por ML, IA, Ciência de Dados, Advanced Analytics e demais termos utilizando uma analogia de uma praça de alimentação. Esse tipo de comparação é importante, uma vez que os termos podem ser confusos e serem trocados entre si ainda que representem coisas completamente diferentes. Também mostramos como funciona um projeto de Ciência de Dados – tanto do ponto de vista de gestão como do ponto de vista de implementação com Python.

Também ilustramos quais são as principais bibliotecas em Python para trabalharmos com ML com exemplos e mostrando algumas possibilidades da construção de algoritmos. Veja que estamos trabalhando com bibliotecas da mesma forma que vimos anteriormente e que aos poucos vamos também expandindo o nosso “repertório” de bibliotecas como o Tensorflow, scikit-learn, LightGBM, mlxtend e outros.

**REFERÊNCIAS**

VIGEN, T. **Spurious correlations**. Hachette UK, 2015.

**LINGUAGEM DE PROGRAMAÇÃO**

AULA 3

Prof. Wellington Rodrigo Monteiro

**CONVERSA INICIAL**

Olá! Nas nossas conversas anteriores, comentamos sobre o uso das bibliotecas e, ainda, sobre o trabalho com algoritmos de *machine learning*(ML). Por outro lado, o Python oferece também outras possibilidades de desenvolvimento: a sua versatilidade é uma das razões pelas quais estamos trabalhando com esta linguagem dentro destes estudos.

Dito isso, vamos conversar aqui sobre programação orientada a objetos (POO). Até este momento, é provável que você não tenha um contato profundo com este tema em Python, por outro lado, o uso de POO pode ser bem útil em sistemas corporativos de larga escala, servidores, sistemas de simulação, entre outros. Vamos lá?

**TEMA 1 – UM PROBLEMA PARA SER RESOLVIDO PELA PROGRAMAÇÃO ORIENTADA A OBJETOS**

Quando ingressamos em um curso de Tecnologia da Informação (TI) e aprendemos a desenvolver algoritmos utilizando linguagens de programação, quase sempre aprendemos da mesma forma. Independentemente da universidade, do país ou do professor, (quase) sempre aprendemos na seguinte linha de pensamento.

* **Algoritmos são como receitas de bolo**: possuem começo, meio e fim bem definidos.
* Também como receitas de bolo, foram feitos para **sempre funcionarem de uma mesma forma***:* não importa se você está fazendo a receita de noite ou de dia; se quem faz a receita é um adolescente ou um idoso; se a receita está sendo feita em uma fazenda ou em uma cozinha em um arranha-céu. **Sempre esperamos o mesmo resultado como consequência das mesmas entradas.**
  + Da mesma forma, um algoritmo pode ser executado em um *notebook* ou um *desktop*; um computador antigo ou um que acabou de ser comprado: sempre esperamos que ele produza o mesmo comportamento. Peguemos um exemplo de um aplicativo de *internet banking* de um dado banco. É esperado que uma transferência sempre funcione da mesma forma, independentemente do modelo de celular, da cor do celular ou da versão do sistema em uso.
* E, como receitas de bolo, é relativamente fácil mudarmos alguma coisa no **final** do processo. Por outro lado, se precisarmos mudar algo no **meio do caminho**,teremos problemas. Ou você nunca se deparou com alguma atividade em certa disciplina em que você arrumou algo no meio do seu código só para se deparar com um conjunto de novos erros em outras partes do código?

Este paradigma chama-se ***programação estruturada***. Essa forma de criar algoritmos possui suas vantagens: ao usar esse paradigma, por exemplo, é relativamente fácil começarmos a criar algoritmos simples e de entendermos o que ele faz, na prática. Isso inclui casos como:

* leitura e escrita de dados com base no que um usuário pode digitar;
* estruturas de seleção (como o *se/se-então/senão*);
* estruturas de repetição (como o *para/enquanto*);
* estruturas de dados (como vetores, listas e matrizes);
* estruturas de divisão de código como as funções; e
* também permite que você encontre os erros de forma mais rápida porque ela possui menos *overhead –*isto é, necessita que você produza menos código que serviriam somente para **servir como fundação** para a sua lógica.

Durante esta aula, voltaremos a usar várias analogias (igual a receita de bolo). Logo, se estamos falando em receitas culinárias, a programação estruturada serve como uma maneira mais simples de ensinar a cozinhar para quem nunca cozinhou na vida. Observe o quadro a seguir.

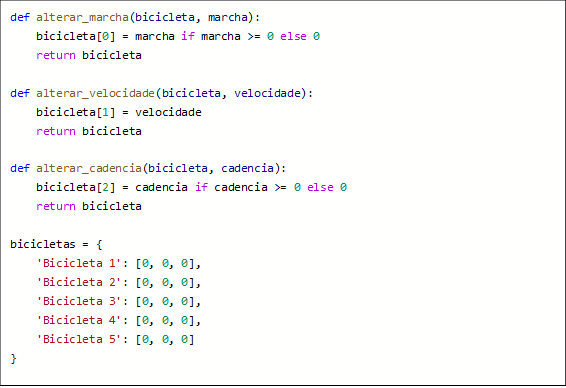
Quadro 1 – Programação estruturada

|  |  |
| --- | --- |
| **Cozinha** | **Programação** |
| Fazer salada | Mostrar um texto na tela |
| Ferver água | Guardar em uma variável o que um usuário digitou |
| Cozinhar um ovo | Usar estruturas de seleção (se/se-então/senão) |
| Fazer omelete | Usar estruturas de seleção com variáveis que o usuário digitou |
| Cozinhar miojo | Usar estruturas de repetição (para/enquanto) |
| Cozinhar arroz | Usar listas, vetores e/ou matrizes |
| Cozinhar carne moída | Usar funções |
| Preparar um almoço com arroz, carne moída, ovo e salada | Criar um algoritmo com várias funções, estruturas de seleção, repetição, variáveis e listas |

Entenda o quadro anterior como apenas formas de **comparar**o esforço entre ambos e não compreenda isto de forma literal. Dito isso, provavelmente, preparar um almoço com arroz, carne moída, ovo e salada não faria de você um *chef* de cozinha, muito menos o prepararia para todos os tipos de pratos possíveis. Por outro lado, poder-se-ia dizer que é a base necessária para preparar pratos mais complexos: cozinhar arroz envolve coisas como gerenciar a temperatura do fogão, saber lidar com os utensílios da cozinha, saber controlar o tempo e os ingredientes necessários, entre outros. Logo, quando fôssemos aprender sobre algo mais complexo, todos saberiam que você já dominaria o básico (e que alguns poderiam até mesmo dizer que seria o “óbvio”).

É exatamente a mesma coisa conosco: primeiro, você aprendeu os fundamentos da programação para então chegar ao momento de trabalhar com paradigmas mais complexos.

Dito isso, vamos pensar no seguinte desafio: você foi encarregado de desenvolver o sistema de software de uma empresa de aluguel de bicicletas. Utilizando sensores e tecnologia recente, a missão é construir um software em Python que registre dados das bicicletas, por exemplo, a troca de marchas, a velocidade e a cadência (a quantidade de pedaladas por minuto). Utilizando todas as técnicas que listamos e com o que já vimos anteriormente, poderíamos fazer algo assim:



Veja que temos um dicionário chamado *bicicletas* e que possui cinco bicicletas. As chaves são referentes ao nome das bicicletas (como *Bicicleta 1* e *Bicicleta 4*), e os valores são uma lista contendo os dados referentes às bibliotecas. Por exemplo:

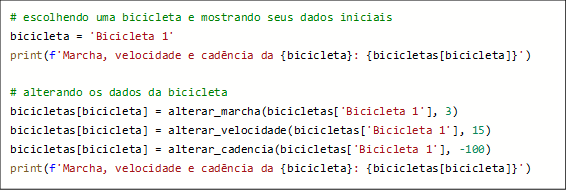
'Bicicleta 1': [0, 0, 0]



Armazena três valores para a *Bicicleta 1*. Sem olhar o código das funções, fica bem difícil de sabermos a que se refere o primeiro valor (ou o segundo, ou o terceiro). Ao analisarmos novamente as funções apresentadas, sabemos que o primeiro valor equivale à marcha, o segundo valor equivale à velocidade e o terceiro valor equivale à cadência.

Aqui vai a primeira dor de cabeça: e se, em vez de três características, estivéssemos trabalhando com 300 características? Em grandes sistemas corporativos, isto pode ser bem comum: imagine que não estamos mais trabalhando com bicicletas, mas sim com notas fiscais ou com dados cadastrais de empresas – já imaginou a quantidade de características que existem? Agora, pense se em algum dia um determinado algoritmo está dando um erro, e você precisa analisar o código. Pense só: se estivéssemos trabalhando com uma lista com 300 itens, como poderíamos encontrar rapidamente **qual é o elemento**que está com problemas, principalmente se nunca vimos aquele código antes em nossa vida?

Para falar da segunda dor de cabeça, vamos pensar em um código no qual alteraríamos marcha, velocidade e cadência da Bicicleta 1:



Ao rodar esse código, temos os seguintes resultados:



Veja que a marcha original (primeiro valor da lista) era 0 e foi alterada para 3; a velocidade (segundo valor da lista) era 0 e foi alterada para 15; a cadência (terceiro valor da lista) era 0 **e permaneceu em 0**. Por outro lado, observe novamente o código para a alteração da cadência: tentamos mudar a cadência de 0 para -100, mas sem sucesso:

bicicletas[bicicleta] = alterar_cadencia(bicicletas['Bicicleta 1'], -100)

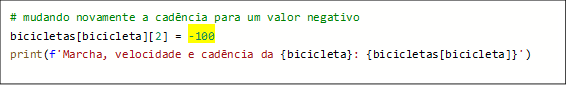


Ao analisarmos novamente a função *“alterar\_cadencia*”, vemos que isto, na realidade, é esperado: dentro da função existe um **if** que altera quaisquer cadências negativas para zero. Poderíamos dizer que é uma forma de evitarmos que tenhamos cadências abaixo de zero na nossa lista:

def alterar_cadencia(bicicleta, cadencia):
    bicicleta[2] = cadencia if cadencia >= 0 else 0
    return bicicleta



Mas é justamente aqui que a dor de cabeça se apresenta: se realmente quisermos, há a possibilidade de burlarmos essa regra – basta alterar diretamente o valor da lista:



E, assim, a cadência estará negativa:



É claro que o exemplo aqui é meramente ilustrativo, mas pensemos nos problemas de **segurança** (ou **confiança**) que isso poderia causar. E se, em vez da cadência de uma bicicleta, estivéssemos falando das notas do sistema da universidade? Ou do valor da nota fiscal em um sistema contábil? Ou do saldo na conta em um *internet banking*?

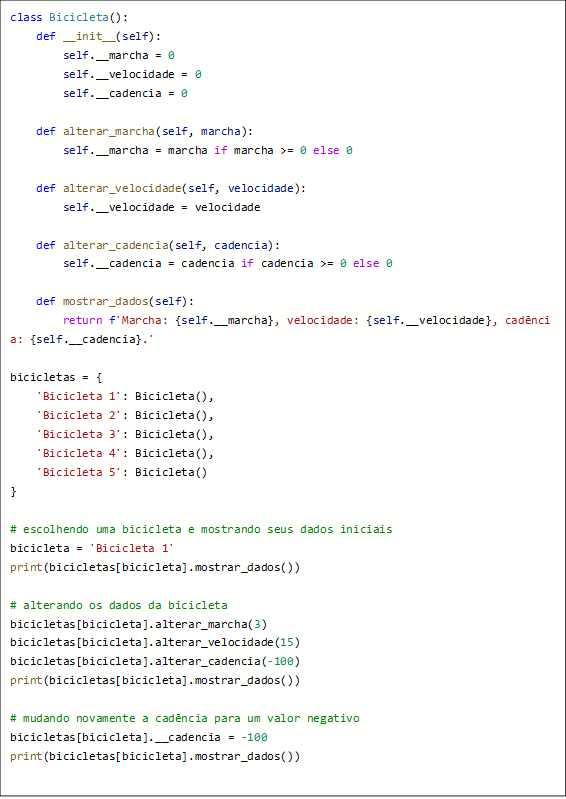
Como garantir que as **mesmas regras de negócio**sempre serão aplicadas para todos os casos sendo analisados, sem espaço para possíveis alterações indesejadas e tendo uma boa confiabilidade?

É aí que entra o paradigma da programação orientada a objetos.

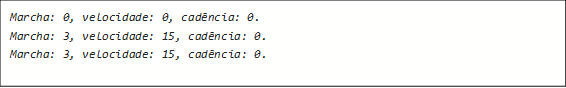
**TEMA 2 – NOÇÕES DE PROGRAMAÇÃO ORIENTADA A OBJETOS**

A programação orientada a objetos (POO) e os bancos de dados relacionais que usam o SQL possuem uma origem muito similar: ambos basicamente procuram representar o mundo real em um código. Da mesma forma que dois objetos não ocupam jamais o mesmo lugar, duas bicicletas também são coisas completamente diferentes. Esta lógica também se aplica a duas notas fiscais ou duas pessoas. Em SQL, temos tabelas e, dentro das tabelas, temos registros e colunas. Em POO, algo similar também é possível.

Veja como seria o mesmo código apresentado, mas implementado usando técnicas de POO:



Vamos explicar cada um dos conceitos. Antes disso, observe como ficaram os resultados – veja que, apesar de tentarmos “forçar” a cadência para um número negativo (-100), o algoritmo não permite:



**2.1 OBJETOS**

O principal conceito da POO são, como o próprio nome já diz, os objetos. Para falarmos dos objetos e de POO, deixemos um pouco de lado todos os conceitos aplicáveis ao mundo do desenvolvimento de algoritmos e pensemos no mundo real. Olhe ao seu redor. O que você vê agora? Provavelmente deve estar vendo uma tela, um teclado, um celular, uma janela, uma porta, quem sabe até mesmo um copo de água (a propósito, você já bebeu água recentemente?).

Agora, perceba que “uma tela” ou “uma porta” ou “um celular” são únicos em si: alguém não teria “um celular do modelo Apple iPhone 12” em mãos, mas sim “um celular do modelo Apple iPhone 12 com o número de série A1234567890”. A porta é única em sua estrutura: ainda que várias outras portas sejam bem parecidas, a porta mais próxima de você é única, e só ela existe exatamente do jeito que ela é (talvez com alguns riscos ou com a pintura um pouco descascada).

Observe que todas as coisas do mundo real são únicas em si: até mesmo se nós estivermos segurando uma folha de papel em branco cada, teremos em mãos *folhas diferentes*. Elas podem até ser bem parecidas, mas perceba que a sua folha de papel **não é**a minha folha de papel.

Ainda que tenhamos uma bicicleta azul cada um, fabricadas pela mesma fábrica e pertencentes ao mesmo modelo, teremos também bicicletas totalmente diferentes: a nossa bicicleta pertence a nós e, a sua, a você. Pode ser que você cuide melhor da sua bicicleta do que nós ou, independentemente do caso, ainda que estejam juntas uma da outra, elas **nunca ocuparão o mesmo lugar ao mesmo tempo** – lembra quando mencionam na física que dois objetos jamais ocupam o mesmo lugar? Pois bem.

Os objetos da vida real possuem **estados**e **comportamento**. Estados são aquelas coisas que **definem**um objeto. Para um carro, pode ser placa, número de chassi, cor, ano e modelo. Para um cachorro, pode ser cor, raça, idade e nome. Logo, entendemos estados como sendo as suas **características**. Já os comportamentos são **ações** que um objeto pode fazer. Um carro pode virar o volante, pode estar em movimento ou parado, pode estar ligado ou desligado. Um cachorro pode correr, abanar o seu rabo ou latir, por exemplo.

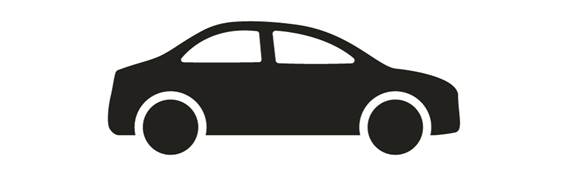
Em POO, mudamos um pouco os nomes:

* **estados**(características) – são armazenados em variáveis e constantes (que agora serão chamadas de **atributos**); e
* **comportamentos**(ações) – são geridos por funções (que agora serão chamadas de **métodos**).

**2.2 CLASSES**

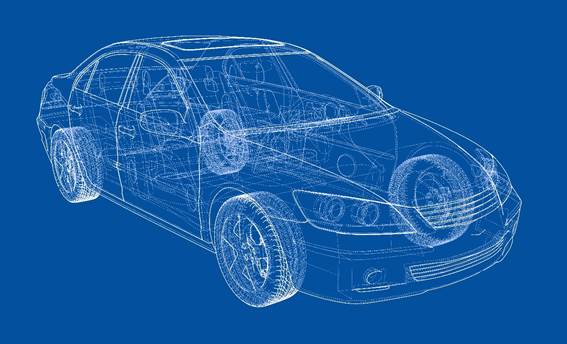
Como você deve imaginar, na realidade os objetos não possuem os mesmos estados e comportamentos ao mesmo tempo. Neste exato momento, todos os cachorros do mundo **não**estão latindo, nem todos os carros estão em movimento. Logo, cada carro e cada cachorro terá os seus estados e comportamentos de forma independente uns dos outros **ainda que possam compartilhar**os tipos de estados que são possíveis: todos os carros do mundo possuem alguma cor e foram fabricados em algum ano. Todos os cachorros do mundo também possuem alguma cor e alguma raça (ainda que a raça seja “sem raça definida”). É como se todos os objetos de um mesmo “tipo” seguissem sempre um molde mínimo contendo todos os estados e comportamentos esperados.

Figura 1 – Interface



Crédito: premium design/Shutterstock.

Figura 2 – Classe



Crédito: cherezoff/Shutterstock.

Figura 3 – Objetos da classe



Crédito: Paisan579/Shutterstock.

O nome que damos a este molde é ***classe***: uma classe chamada *Carro* terá um conjunto predefinido de atributos e métodos, e isso também vale para uma classe chamada *Cachorro*. Imagine uma classe como um manual de como gerar um novo objeto ou um cardápio – a classe *Carro* não é um carro em si, mas é tão somente uma lista de definições (como um manual de instruções, mesmo) contendo tudo o que um novo objeto (um novo “Carro”) deveria ter.

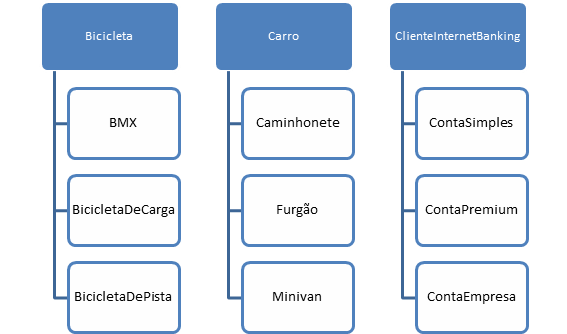
Ao criarmos um objeto de uma classe, estamos **instanciando-a**,ou seja, estamos criando um objeto seguindo o “manual” preestabelecido. Se a classe chamada *Carro* possui cinco atributos (placa, número de chassi, ano, cor e modelo) e três métodos (virar o volante, movimentar/parar, ligar/desligar), sabemos que todos os carros que instanciarmos deverão sempre seguir estes comportamentos: todos os carros virarão um volante seguindo o mesmo comportamento (para a direita e para a esquerda, e não como para cima e para baixo); vão se movimentar do mesmo jeito (ou seja, para frente ou para trás, mas não começarão a flutuar do nada); e terão um conjunto de atributos predeterminado. Isto nos dá muita segurança e previsibilidade ao trabalharmos com aplicações mais complexas: já pensou o que aconteceria se um método chamado *mostrar saldo* funcionasse de formas completamente diferentes a cada dia para um mesmo conjunto de pessoas? Ou já pensou o que aconteceria se cada “Carro” não tivesse nenhum **padrão**de direção?

**2.3 HERANÇAS**

Heranças são outro conceito bem importante em POO. Ao ler as descrições anteriores, você deve ter pensado algo como: “beleza, mas nem todos os carros são iguais – existem carros que são à combustão que funcionam de forma diferente de carros elétricos. Já vi carros que nem possuem volante para virar! E aí?”. Ou, ainda: “existem cachorros que fazem mais coisas do que abanar o rabo e latir. Existem cachorros que gostam de correr, e outros que gostam de caçar”. Ou, voltando ao nosso exemplo das bicicletas: “Existem bicicletas que são especializadas para cada caso. Existem bicicletas para correr em velódromos, e outras, para transportar cargas. Também existem bicicletas para manobras, como a BMX. Nem todas são iguais!”. Ou, no caso do *internet banking*: “nem todos os clientes são iguais! Alguns são pessoa física, e outros, jurídica. Existem também clientes com mais privilégios do que outros”. E todos esses argumentos são justificáveis.

É para isso que serve a **herança**: podemos criar classes com base em outras classes, mas que respeitam as especificidades do seu caso de uso. No caso do carro.

Figura 4 – Exemplo de classes



Veja que uma classe herdada possui esse nome porque ela **herda**os métodos e atributos da classe principal: BMX herdou tudo de bicicleta, mas podemos modificar algumas coisas (as BMX possuem marchas diferentes) e podemos incluir novos métodos (como a execução de manobras).

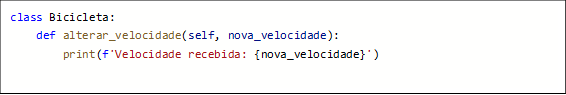
**2.4 INTERFACES**

Além disso, também podemos implementar **interfaces**: existem casos nos quais queremos somente definir quais métodos uma classe deve ter, mas não queremos implementar a **lógica**dentro dos métodos. No caso das bicicletas ou dos clientes de banco, pode ser que cada implementação de um método seja completamente diferente da de outro, mas, mesmo assim, precisamos garantir que todas as classes utilizem um conjunto predefinido de métodos: o nome disso é *interface*.

**TEMA 3 – CLASSES E OBJETOS EM PYTHON**

Da mesma forma que a biblioteca-padrão do Python oferece listas, dicionários de dados e funções, ela também possibilita a criação de algoritmos utilizando POO[[1]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a3&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPUebEn4KPWvx1BrnMfd7u90=&ne=False#_ftn1).

Vamos começar com o mesmo caso das bicicletas que vimos anteriormente, mas agora seguindo um passo a passo:

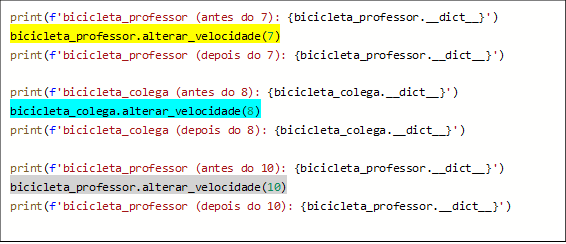


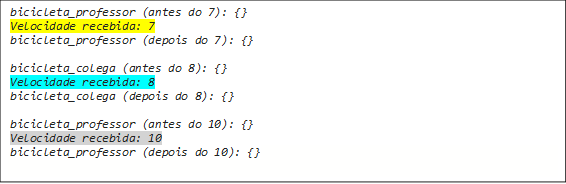
Veja que a classe foi **definida**ao usarmos a palavra-chave *class* seguida pelo nome (no caso, *Bicicleta*). Isto é bem parecido com a forma como criamos funções em Python, em que usamos o *def*seguido pelo nome da função. Após o nome da classe, é **opcional** usarmos parênteses, ao contrário das funções.

**Dentro**da classe, podemos criar uma função. Neste primeiro momento, só queremos testar para saber se estamos fazendo tudo certo. Para tal, só mostramos em tela a velocidade que recebemos no método *alterar\_velocidade*. Este método também possui o parâmetro *self*. Isto significa que o método pode ser utilizado pelos objetos, e os seus resultados se aplicam somente ao escopo do objeto em si. Lembre-se: agora só temos a *classe*, mas nenhum *objeto –*ou seja, não temos nenhuma bicicleta instanciada no nosso código. Para instanciar, basta fazer isto:

bicicleta_professor = Bicicleta()
bicicleta_colega = Bicicleta()

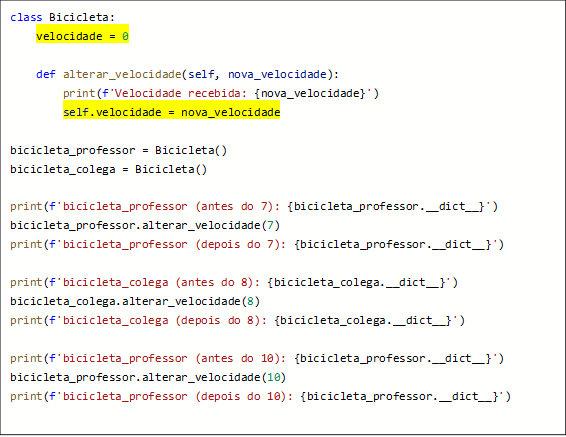

Agora, sim, temos duas instâncias de bicicletas: uma chamada *bicicleta\_professor*, e outra, *bicicleta\_colega.*Espera-se que a lógica de ambos seja igual, mas um é independente do outro em sua essência. Veja como chamamos os métodos para cada um desses objetos e os seus resultados:

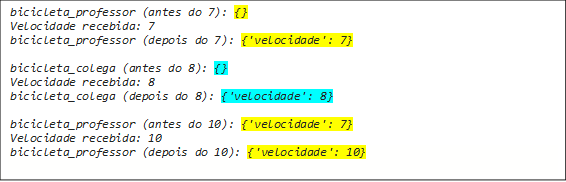




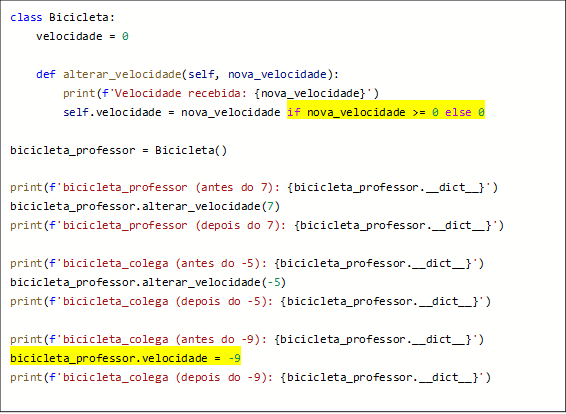
Os valores que informamos (7, 8 e 10) são as velocidades. Logo, aquele *self*não é considerado como o primeiro parâmetro, mas, sim, o parâmetro “*nova\_velocidade*”. Veja que usamos o ponto seguido pelo nome do método para chamar um método para um objeto em específico.

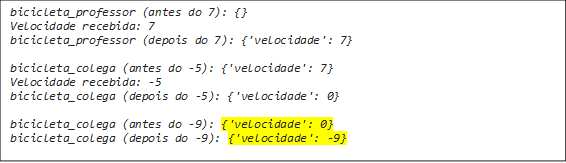
Isto posto, a forma como está agora não é das melhores: como saber se a velocidade realmente mudou para um objeto? Ao mostrarmos o antes e o depois de chamarmos cada um dos métodos, o resultado é aparentemente a mesma coisa: um dicionário vazio de atributos (o *\_\_dict\_\_*é uma funcionalidade-padrão do Python que mostra o que existe dentro de uma classe). Logo, a pergunta então é outra: como **armazenar**esses valores dentro de um objeto? Vamos inserir um novo parâmetro chamado *velocidade*fora do método, mas dentro da classe.



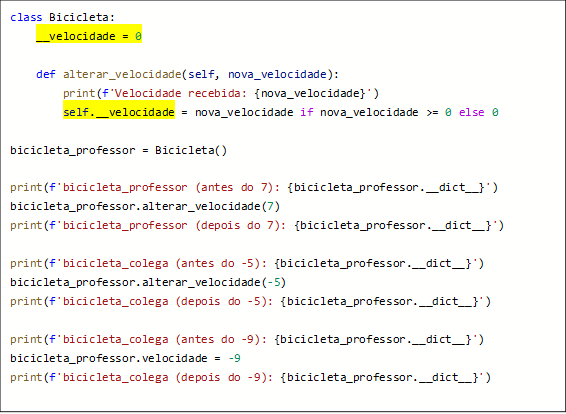


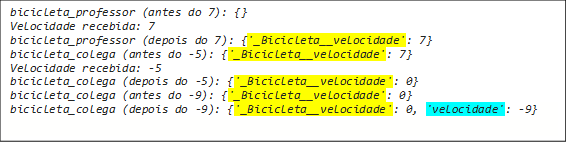
Vejamos isso! Agora, a variável *velocidade*está sendo armazenada dentro de cada objeto, e de forma individual. A velocidade da *bicicleta\_professor* não interfere na velocidade da *bicicleta\_colega*. Agora, vamos aumentar um pequeno grau de complexidade: vamos modificar o método *alterar\_velocidade*para não permitirmos velocidades negativas e tentar passar uma velocidade negativa de duas formas para a *bicicleta\_professor*: a primeira, via método; a segunda, forçando diretamente a variável.





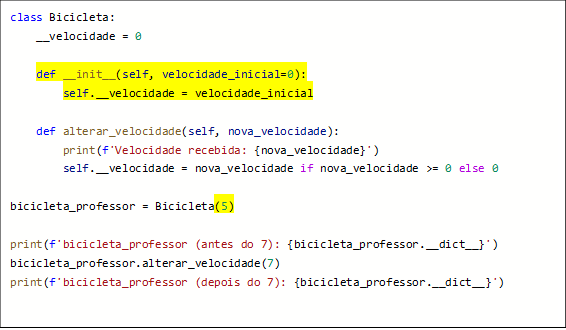
Veja que a classe permitiu que forçássemos o –9 sem maiores problemas. E agora, como endereçaríamos isso? É para isso que existem os **atributos privados**: atributos que somente deveriam ser acessados e modificados de dentro da classe e por seus respectivos métodos. Em Python, os atributos privados possuem dois *underscores* (“\_”) em seu início. Observe:

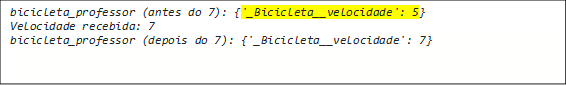




Note que ainda conseguimos visualizar a variável (agora chamada de *\_Bicicleta\_\_velocidade*), mas o risco de mudarmos manualmente é menor (quando tentamos mudar para –9, acabamos não afetando a *\_Bicicleta\_\_velocidade*, mas, sim, só criando uma variável chamada *velocidade*que o objeto acaba não utilizando). A própria documentação do Python[[2]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a3&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPUebEn4KPWvx1BrnMfd7u90=&ne=False#_ftn2) diz que é possível modificarmos os valores de variáveis privadas, mas que, ao seguirmos essa convenção, diminuímos o risco de modificá-las acidentalmente.

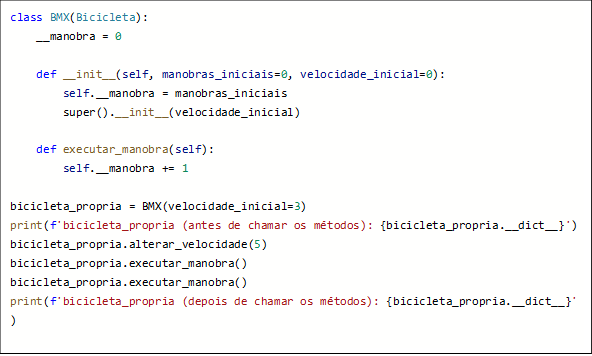
E se quiséssemos instanciar o *bicicleta\_professor*com um valor inicial para a velocidade? Há uma função-padrão em Python chamada *\_\_init\_\_* na qual podemos informar valores que devem ser usados logo ao inicializar a classe. Lembra do *random\_state* que usamos no *scikit-learn* anteriormente? Este é um exemplo de parâmetro que é usado na inicialização da classe daquela biblioteca. No nosso caso, vamos utilizar o *\_\_init\_\_* para definir uma velocidade inicial da bicicleta. Se nenhuma velocidade for informada, usaremos o zero como padrão.

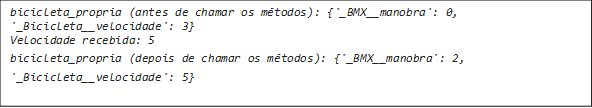




Seguindo na mesma linha, o que precisaríamos fazer se quiséssemos usar a herança para criar uma classe chamada BMX, a qual usaria a classe Biblioteca como base? Ao analisarmos o código a seguir, perceba algumas coisas.

* Na definição da classe BMX, estamos colocando entre parênteses a classe Bicicleta. Isso significa que BMX herda de Bicicleta.
* O \_\_init\_\_ é próprio da classe BMX, mas ela também logo em seguida chama o método \_\_init\_\_ próprio da classe Bicicleta por meio do *super()*, um termo-padrão do Python para chamar a classe herdada.
* Veja que em momento algum estamos implementando o método *alterar\_velocidade.*Logo, estamos reaproveitando exatamente a mesma lógica da *alterar\_velocidade*. Se quiséssemos, também poderíamos reimplementar este mesmo método com uma lógica específica à classe BMX.
* Também estamos criando um método específico para a classe BMX chamado *executar\_manobra* – este método é próprio da BMX, mas não da Bicicleta.





**TEMA 4 – POLIMORFISMO**

Outro conceito-chave em POO é o **polimorfismo**: a possibilidade de diferentes classes implementarem de formas diferentes um mesmo método e/ou de uma mesma função operar de formas diferentes com tipos de dados distintos.

Um exemplo básico de polimorfismo é a função a seguir: ela aceita dois ou três parâmetros e consegue gerar um resultado para ambas as combinações.

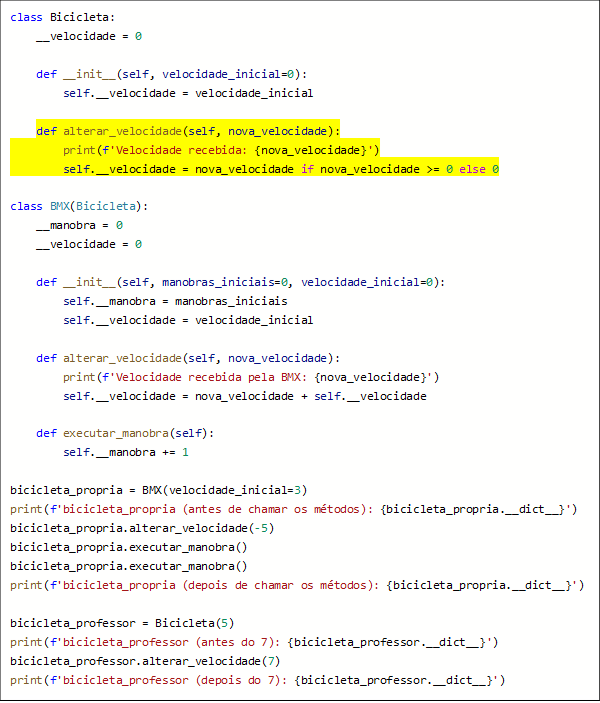
def mostrar_nome(primeiro_nome, sobrenome, nome_meio=None):
    if nome_meio is None:
        print(f'{primeiro_nome} {sobrenome}')
    else:
        print(f'{primeiro_nome} {nome_meio} {sobrenome}')

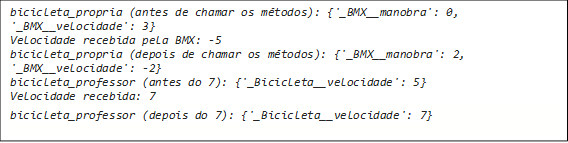
mostrar_nome('Jéssica', 'Silva')
mostrar_nome('Rodrigo', 'dos Santos', 'André')



Jéssica Silva
Rodrigo André dos Santos


Para classes, algo similar também pode ocorrer – comentamos anteriormente que a classe BMX poderia reimplementar a função *alterar\_velocidade*: este é um exemplo de polimorfismo também. Ambas as classes podem ter o mesmo nome da função, mas as lógicas podem ser completamente diferentes uma da outra.





Os resultados (e a lógica) de ambas as classes são diferentes. A BMX permite velocidades negativas – a Bicicleta, não.

**TEMA 5 – CLOSURES E DECORATORS**

Em Python também é possível termos **funções encapsuladas**, uma função que fica dentro de outra função. Essas funções que ficam dentro de outras funções e que usam variáveis atribuídas a elas são chamadas de ***closures***. O uso delas possui algumas vantagens, por exemplo, a redução de casos em que precisamos usar variáveis globais e, além disso, a oportunidade de encapsular a lógica (e protegê-la) do resto do algoritmo. O código a seguir implementa uma *closure* para transformar a escala de um valor – no caso, estamos convertendo números que estejam entre 0.0 e 1.0 para 0% a 100% e arredondando em 3 casas decimais quando necessário:

def converte_porcentagem(proporcao):
    # inicio closure
    def multiplicador(valor):
        return round(proporcao * valor, 3)
    # fim closure
    
    return multiplicador

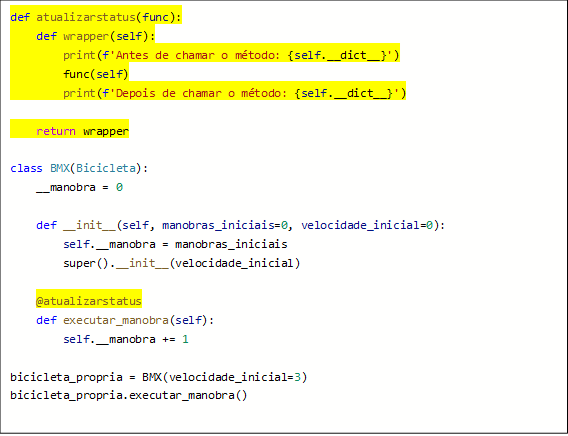
conversor = converte_porcentagem(100)
conversor(0.45)



45.0

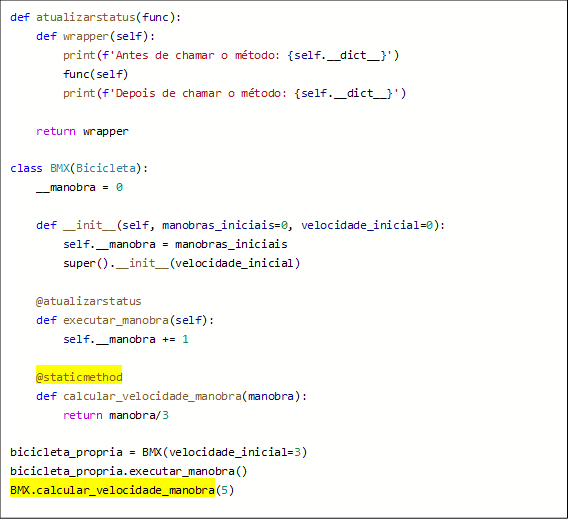
O ***decorator***é outra funcionalidade bem interessante em Python. Você deve ter percebido que em vários dos exemplos anteriores usamos muito a função *print*para mostrar como estavam as coisas antes e depois de aplicar determinada função. Felizmente, há uma forma de simplificarmos esse trabalho. Os *decorators* são *closures* que podem ser incluídas no código sem grandes alterações deste com o uso do “*@*”.

Vamos criar um *decorator* chamado “*atualizarstatus*”, o qual deve conter as mensagens que desejamos mostrar.





Veja que o *decorator* é uma função a qual encapsula outra função qualquer dentro dela – logo, da mesma forma como é utilizada para a *executar\_manobra*,ela também pode ser reaproveitada para a *atualizar\_velocidade* e quaisquer outras funções. O Python também possui alguns *decorators* já prontos: um deles é o @staticmethod. O @staticmethod serve para métodos estáticos (isto é, quando a lógica do método **não depende**do objeto, mas sim é algo específico da classe). Veja, por exemplo, o caso a seguir, em que calculamos a velocidade com base no número de manobras. O resultado desse método não possui relação com os objetos, mas é algo que possui relação com a **classe**.





**FINALIZANDO**

Esta aula teve como foco apresentar a você o tópico de programação orientada a objetos. Esse paradigma, como você deve ter percebido, possui vantagens e desvantagens. Como vantagens, poderíamos citar a sua facilidade em transcrever requisitos de negócio e conceitos da vida real em algoritmos funcionais (afinal de contas, é fácil entender o que é um “objeto” na realidade). Também é mais fácil de dar manutenção, pois é mais modular – logo, também é possível modificarmos pequenas partes do código com o risco menor de comprometer o todo. Além disso, pode ter vários trechos de código reaproveitados e tem ainda como vantagens a abstração da sua lógica e a segurança.

Por outro lado, também traz uma camada de complexidade bem maior, que se traduz em requisitos computacionais mais pesados, uma curva de aprendizagem bem maior e mais linhas de código. É por esta razão que aqui procuramos tomar um cuidado em traduzir vários conceitos para você de uma forma palatável, uma vez que existe um risco bem grande em tão somente repassar conceitos e não necessariamente **compreendê-los**.

[[1]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a3&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPUebEn4KPWvx1BrnMfd7u90=&ne=False#_ftnref1) Caso tenha interesse, sugerimos muito que leia a documentação oficial do Python sobre o assunto. Disponível em: <https://docs.python.org/3/tutorial/classes.html>. Acesso em: 30 nov. 2021.

[[2]](https://conteudosdigitais.uninter.com/libraries/newrota/?c=/gradNova/2022/bachareladoEngSoftware/linguagemProgramacao/a3&hash=3u0Jf1zuk93zpxVYIcnLAmHwQtRvF/KOUxjnpHAo9IsItQ+AbiwLM2ZnNOiiviwQHH+3y1pMqbowFtMsQEvs92pYTR5Z4p6srhj1b/3cQh7R0wvnfjmTJjC5jKw6Y4rSZlvFclqPIl1SldTtl5gKPUebEn4KPWvx1BrnMfd7u90=&ne=False#_ftnref2) Disponível em: <https://docs.python.org/3/tutorial/classes.html#private-variables>. Acesso em: 30 nov. 2021.