

# Análise de dados com Python

Parceria:



## Créditos

---

Todos os direitos autorais reservados. Este material não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da Catarino Soluções, estando o contrafator sujeito a responder por crime de Violação de Direito Autoral, conforme o art. 184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos os logotipos usados neste material pertencem à sua respectiva empresa.

# SUMÁRIO

---

## Conteúdo Referência

|                               |     |
|-------------------------------|-----|
| Introdução ao Python .....    | 06  |
| Introdução ao Numpy .....     | 32  |
| Introdução ao Pandas I .....  | 50  |
| Introdução ao Pandas II ..... | 67  |
| Data Visualization .....      | 84  |
| Scikit Learning .....         | 105 |

## Mãos à Obra

|                   |     |
|-------------------|-----|
| Mãos à obra ..... | 154 |
|-------------------|-----|

# Conceito de Referência

# Introdução ao Python

Parceria:



# Análise de dados com Python

## Sumário

01

- » O que é Python e onde é utilizado
- » Anaconda
- » Variáveis
- » Operadores
- » Listas
- » Tuplas
- » Dicionários
- » Condicionais
- » Loops
- » Funções

## O que é?

02

- » Python é uma linguagem de alto nível, de código aberto, tipagem dinâmica e forte.
- » Foi criada por Guido Van Rossum, em 1991.
- » O nome é inspirado na série britânica Monty Python.



# Análise de dados com Python

03

## O que é utilizada?

- » Aplicações web, desktop e mobile
- » Cálculos científicos
- » Computação gráfica
- » Automação de sistema
- » Mineração de dados
- » Big Data
- » Machine learning
- » Processamento de textos
- » Tratamento e reconhecimento de imagens
- » Animações 3D



04

## Zen do Python

Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one—and preferably only one—obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than right now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea — let's do more of those!

Bonito é melhor que feio.  
Explícito é melhor que implícito.  
Simples é melhor que complexo.  
Complexo é melhor que complicado.  
Linear é melhor do que aninhado.  
Esparsos é melhor que denso.  
Legibilidade conta.  
Casos especiais não são especiais o bastante para quebrar as regras.  
Ainda que praticidade vença a pureza.  
Erros nunca devem passar silenciosamente.  
A menos que sejam explicitamente silenciados.  
Diante da ambigüidade, recuse a tentação de adivinhar.  
Deveria haver um — e preferencialmente só um — modo óbvio para fazer algo.  
Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.  
Agora é melhor que nunca.  
Embora nunca freqüentemente seja melhor que já.  
Se a implementação é difícil de explicar, é uma má idéia.  
Se a implementação é fácil de explicar, pode ser uma boa idéia.  
Namespaces são uma grande idéia — vamos ter mais dessas!

# Análise de dados com Python

05

## Instalando o Python



- » Direto Ir até o [site oficial](#), fazer o download da versão escolhida e instalar de acordo com o sistema operacional de seu computador. A Python Brasil tem um tutorial bem explicado sobre como instalar Python no windows: [Clique aqui](#). Nessa opção, todas as bibliotecas adicionais precisam ser instaladas uma a uma.
- » Outra opção é instalar através da plataforma [Anaconda](#), uma distribuição gratuita de Python (e R!) que já vem com os principais pacotes instalados. (Trabalharemos mais sobre essa plataforma na próxima aula)



06

## idle Python

- » Existem diversas maneiras de utilizar o Python. Uma delas é através do prompt de comando (terminal). Para isso, basta digitar python em um terminal.

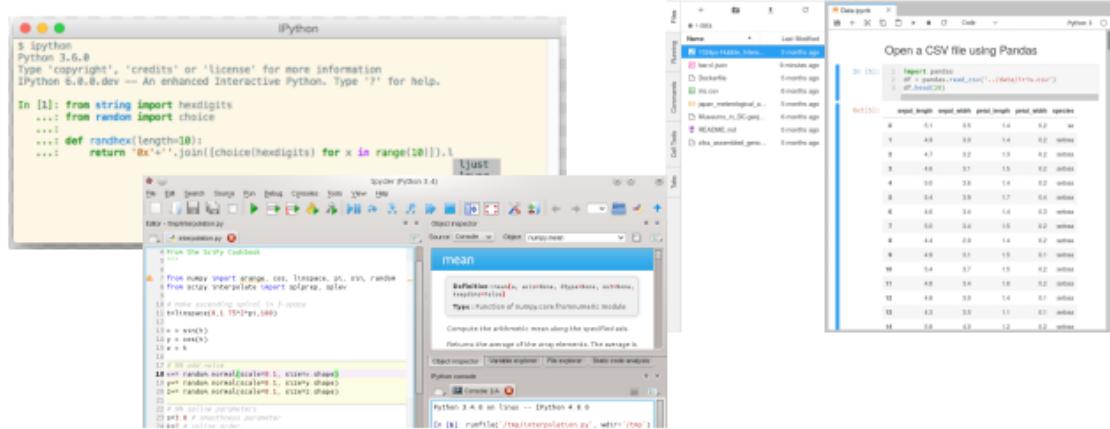
A screenshot of a Windows terminal window titled "Anaconda Prompt (anaconda3) - python". The window displays the Python version information: "Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32". It also shows the standard Python >>> prompt.

# Análise de dados com Python

07

## Outros idles

- » Há outras formas mais dinâmicas e visuais de utilizar o Python que serão abordadas nas próximas aulas.



08

## Versões

### » Python 2.\*

- » Muitas aplicações foram criadas com essa versão
- » Muitas pessoas ainda utilizam essa versão
- » As atualizações foram descontinuadas desde o dia 01/01/2020.

### » Python 3.\*

- » Python 3 não é retrocompatível: aplicações em Python 2 podem não funcionar em Python 3
- » Veio para arrumar algumas falhas de design do Python 2
- » Já estamos na versão 3.8.\*

```
(base) C:\Users\p135894>python --version
Python 3.7.3
(base) C:\Users\p135894>
```

Para verificar qual a versão de Python instalada.

# Análise de dados com Python

09

## Variáveis

- » Variável, em programação, é o nome que damos a um valor ou expressão. Em Python, usamos o sinal de igual, '=', para atribuir um valor à uma variável.

```
a = 10  
b = 'banana'  
c = 0.5
```

- » É sempre importante tentar utilizar nomes de variáveis que nos lembre qual a informação estamos armazenando nela.

```
fruta = 'banana'  
xpto = 'banana'
```

- » Qual das variáveis acima são mais autoexplicativas?

10

## Variáveis - Regras

- » Em Python, existem algumas regras para declararmos variáveis:

- » podem ser usados algarismos, letras ou \_
- » nunca devem começar com um algarismo
- » não podemos usar palavras-chave naturais
- » no Python, por exemplo if, while, etc.

- » Para saber quais as palavras-chave reservadas:

```
import keyword  
print(keyword.kwlist)
```

```
(base) C:\Users\p135894>python  
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: AnaCo-  
nda, Inc. on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import keyword  
>>> print(keyword.kwlist)  
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',  
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global',  
'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',  
'return', 'try', 'while', 'with', 'yield']  
>>>
```

# Análise de dados com Python

11

## Variáveis - Tipos

### » Numéricos:

#### » inteiros (int)

» i = 2

» j = 5

#### » ponto flutuante (float)

» x = 0.5

» y = 2.3

» avogadro = 6E23

#### » complexos(complex)

» z = 2 + 1j

» z2 = 3 - 4j

Observe que o decimal é dado pelo ponto, não pela vírgula.

### » Literais:

#### » caracteres (string)

» fruta = 'manga'

» nome = 'Patricia'

### » Lógicos:

#### » valores verdadeiros ou falsos (bool)

» hoje\_chove = True

» hoje\_chove = False

Python é case-sensitive!

## Variáveis - Tipos

12

### » E como saber qual o tipo de uma variável?

Usamos a função type!

type(a)

type(x)

type(z)

```
>>> a = 5
>>> b = 3.2
>>> z = 'banana'
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(z)
<class 'str'>
>>>
```

# Análise de dados com Python

## Variáveis – Atribuição Múltipla

13

- » Podemos atribuir diversos valores à diversas variáveis simultaneamente, utilizando a atribuições múltiplas!

- » `a, b, c = 2, 3, 4`
- » `a, x, z = 2, 0.5, 1+1j`
- » `hoje_chove, a, y = True, 3, 3.14`

```
>>> a, b, c = 2,3,4
>>> a
2
>>> b
3
>>> c
4
>>>
```

## Variáveis – Conversões

14

- » É possível converter uma variável de um tipo em outro, usando as funções `int()`, `float()`

- » `a = 2`
- » `b = float(a)`
- » `b = int(b)`
- » Para converter para variável complexa, é necessário dois valores
  - » `z = complex(x,y)`

```
>>> a = 2
>>> type(a)
<class 'int'>
>>> a = float(a)
>>> a
2.0
>>> type(a)
<class 'float'>
>>>
```

# Análise de dados com Python

15

## Exercícios – Variáveis

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Atribua os seguintes valores à variáveis, um a um:
  - a. 347
  - b. 2.71
  - c. "347"
  - d. 2+3j
3. Quais os tipos das variáveis acima?
4. Faça uma atribuição múltipla das variáveis do exercício 2.
5. Declare a seguinte variável e verifique o que acontece:
  - a. teste = true
6. Transforme as variáveis do exercício 2 conforme segue:
  - a. para float
  - b. para inteiro
  - c. para float
  - d. para string
7. Crie uma variável complexa com os valores de 2.a e 2.b.

16

## Operadores

Temos dois tipos de operadores, os numéricos e os lógicos.

### » Operadores numéricos

- » Adição: +
- » Subtração: -
- » Divisão: /
- » Multiplicação: \*
- » Potenciação: \*\*
- » Resto de uma divisão: %

Cuidado para não confundir o operador '==' com a atribuição '='!

### » Operadores lógicos

- » Maior que: >
- » Menor que: <
- » Maior ou igual a: >=
- » Menor ou igual a: <=
- » Idêntico: ==
- » Diferente de: !=
- » Não: not
- » E: and
- » Ou: or

# Análise de dados com Python

## Operadores numéricos

17

» Sejam  $a = 5$  e  $b = 3$ :

- » Adição:  $a + b$
- » Subtração:  $a - b$
- » Divisão:  $a/3$
- » Multiplicação:  $a * b$
- » Potenciação:  $a ** b$
- » Resto da divisão:  $a \% b$

```
>>> a = 5
>>> b = 3
>>> a + b
8
>>> a - b
2
>>> a * b
15
>>> a ** b
125
```

## Operadores numéricos – Ordem de execução

18

Como em qualquer linguagem de programação, a ordem de execução das operações é

Parênteses > Exponencial > Multiplicação e/ou Divisão > Adição e/ou Subtração



## Operadores lógicos

» Sejam  $a = 5$  e  $b = 3$ :

- » Maior que:  $a > b$
- » Menor que:  $a < b$
- » Maior ou igual a:  $a \geq 3$
- » Menor ou igual a:  $a \leq b$
- » Idêntico:  $a == b$
- » Diferente de:  $a != b$
- » Não:  $a \text{ not } b$
- » E:  $a \text{ and } b$
- » Ou:  $a \text{ or } b$

```
>>> a, b = 5, 3
>>> a > b
True
>>> a < b
False
>>> a!=b
True
>>> a==b
False
```

## Operadores lógicos – Tabela Verdade

ESTUAR

Tabela Verdade

| A     | B     | A and B | A or B | not(A) | not(B) |
|-------|-------|---------|--------|--------|--------|
| True  | True  | True    | True   | False  | False  |
| True  | False | False   | True   | False  | True   |
| False | True  | False   | True   | True   | False  |
| False | False | False   | False  | True   | True   |

# Análise de dados com Python

## Exercícios – Operadores

21

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Defina as seguintes variáveis  $x = 3$ ,  $y = 4.0$ ,  $z = 12$  e  $t = \text{'banana'}$
3. Calcule:
  - a.  $x + y$
  - b.  $z - y$
  - c.  $y ** x$
  - d.  $x * z$
  - e.  $z / x$
  - f.  $y / x$
  - g.  $y \% x$
4. Teste as seguintes relações  
 $\geq$     $<$     $\sim$     $\approx$

PERGUNTA

## Operações com Strings - Algumas Funções

22

» Existem diversas operações que podemos aplicar sobre as strings. Vamos definir a variável  $fruta01 = \text{'BaNaNa'}$  e  $fruta02 = \text{'AbaCaXi'}$  e testar algumas delas.

- » Lowercase:  $fruta01.lower()$
- » Uppercase:  $fruta02.upper()$
- » Capitalize:  $fruta01.capitalize()$
- » Title:  $fruta02.title()$
- » Concatenação:  $fruta01 + fruta02$
- » Multiplicação de strings:  $fruta01 * 3$
- » Começa com:  $fruta01.startswith('B')$ ,  $fruta02.startswith('B')$
- » Termina com:  $fruta01.endswith('I')$ ,  $fruta02.endsswith('I')$
- » Tamanho da string:  $\text{len}(fruta01)$

```
>>> fruta01 = 'Banana'  
>>> fruta01.lower()  
'banana'  
>>> fruta01.upper()  
'BANANA'  
>>>
```

# Análise de dados com Python

## Operações com Strings - Índice e Fatiamento

23

» As strings em Python podem ser acessadas caracter a caracter, de acordo com seu índice.

Seja salada = 'alface com tomate', temos:

» salada[0]      string[i]  
» salada[5]

```
>>> salada = 'alface com tomate'  
>>> salada[0]  
'a'  
>>>
```

» Também é possível usar os índices para fatiar a string:

» salada[0:4]      string[i1:i2]  
» salada[7:10]  
» salada[11:17]

```
>>> salada[7:10]  
'com'  
>>>
```

» Ainda, é possível determinar qual o 'passo' utilizado no Fatiamento da string:

» salada[0:10:2]      string[i:  
» salada[::-3]

```
>>> salada[0:10:2]  
'afc o'  
>>>
```

» Por último, é possível acessar os índices de trás pra frente!

» salada[-1]  
» salada[-10:-7]

Hello

|    |    |    |    |    |
|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  |
| -5 | -4 | -3 | -2 | -1 |

## Operações com Strings - Exercícios

24

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie uma variável com seu nome completo.
3. Escreva a variável em lowercase.
4. Escreva em uppercase.
5. Verifique se o nome começa com a letra P.
6. Verifique se o nome termina com a letra J.
7. Fatie a string para apenas o 1º nome.
8. Fatie a string de 2 em 2.
9. Considerando os espaços, qual o tamanho do seu nome?

# Análise de dados com Python

## Listas

25

» Umas das principais estruturas do Python, a lista permite armazenar diversas variáveis de diversos tipos em apenas uma variável só.

» Para declarar uma lista, as variáveis precisam estar entre colchetes, [].

```
» lista_compra = ['banana', 'pera', 'laranja']
» lista_coisas = ['Patricia', 47, 'abobrinha', 3.14, 1+1j]
```

» Os elementos de uma lista podem ser acessados através dos índices:

```
» lista_compra[0]
» lista2 = lista_coisas[0:3]
»> lista_compras = ['banana', 'pera', 'laranja']
»> lista_compras[0]
'banana'
»>
```

» É possível, inclusive, criar uma lista de listas!

```
» lista_geral = [lista_compra, lista_coisas, lista2]
```

## Listas

26

» Para adicionar elementos à uma lista já existente, usamos o comando append():

```
» lista_compra.append('batata')
```

» Já para criar uma string com todos os elementos de uma lista, podemos utilizar o comando join():

```
» supermercado = ' e '.join(lista_compra)
```

» Ainda com strings, podemos criar uma lista a partir de uma string com o comando split():

```
» feira = 'batata, cebola, tomate, pepino'
» lista_feira = feira.split(', ')
```

```
>>> lista_compra = ['banana', 'pera', 'laranja']
>>> lista_compra
['banana', 'pera', 'laranja']
>>> lista_compra.append('batata')
>>> lista_compra
['banana', 'pera', 'laranja', 'batata']
>>>
```

## Listas - Exercícios

27

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie uma lista com nomes de 4 times de futebol.
3. Acesse o time que está na 3a posição.
4. Crie uma nova lista com duas listas de 3 times de futebol, cada uma de uma divisão diferente.
5. Crie uma lista com 3 diferentes moedas. Acrescente mais 2 outras moedas à essa mesma lista.
6. Crie uma string com a lista do exercício anterior.
7. Agora utilize a string do exercício 6 para recriar uma lista.

## Tuplas

28

» Diferente das listas, as Tuplas são objetos imutáveis, objetos que não permitem adição ou exclusão de elementos.

» Uma tupla é declarada com parênteses ou por elementos separados apenas por vírgulas

» `tupla1 = (1, 3.14, 'abacate')`  
» `tupla2 = 1, 3.14, 'abacate'`

```
>>> tupla1 = (1, 3.14, 'abacate')
>>> tupla1
(1, 3.14, 'abacate')
>>> tupla2 = 1, 3.14, 'abacate'
>>> tupla1 == tupla2
True
>>>
```

# Análise de dados com Python

29

## Dicionários

- » Dicionário é uma coleção não ordenada em que há uma mapeamento de chave:valor.
- » Cada elemento de um dicionário é chamado por uma chave imutável.
- » Para se declarar um Dicionário, usamos os {} e cada elemento chave:valor é separado por vírgula.

```
>>> cadastro = {'nome':'Tania','idade':35,'fruta':'uva','cor':'roxa','musica':'pop'}
>>> cadastro['nome']
'Tania'
>>> cadastro['cor']
'roxa'
>>>
```

30

## Dicionários

- » Podemos criar um dicionário a partir de uma lista de tuplas usando a função dict().

```
>>> tupla_1 = ('nome','Tania')
>>> tupla_2 = ('idade',35)
>>> tupla_3 = ('fruta','uva')
>>> tupla_4 = ('cor','roxa')
>>> tupla_5 = ('musica','pop')
>>> lista_tuplas = [tupla_1, tupla_2, tupla_3, tupla_4, tupla_5]
>>> cadastro = dict(lista_tuplas)
>>> cadastro
{'nome': 'Tania', 'idade': 35, 'fruta': 'uva', 'cor': 'roxa', 'musica': 'pop'}
>>>
```

# Análise de dados com Python

31

## Dicionários

» Podemos adicionar ou editar um valor em um dicionário:

Editando um valor

```
>>> cadastro['fruta'] = 'pera'  
>>> cadastro  
{'nome': 'Tania', 'idade': 35, 'fruta': 'pera', 'cor': 'roxa', 'musica': 'pop'}  
>>>
```

Adicionando novo chave:valor

```
>>> cadastro['sobrenome'] = 'Silva'  
>>> cadastro  
{'nome': 'Tania', 'idade': 35, 'fruta': 'pera', 'cor': 'roxa', 'musica': 'pop', 'sobrenome': 'Silva'}  
>>>
```

## Dicionários

32

» Podemos deletar um elemento:

```
>>> del cadastro['musica']  
>>> cadastro  
{'nome': 'Tania', 'idade': 35, 'fruta': 'pera', 'cor': 'roxa', 'sobrenome': 'Silva'}  
>>>
```

» Podemos atualizar mais de um valor usando a função update():

```
>>> cadastro.update({'nome': 'Ana', 'idade': 27, 'fruta': 'pera'})  
>>> cadastro  
{'nome': 'Ana', 'idade': 27, 'fruta': 'pera', 'cor': 'roxa', 'sobrenome': 'Silva'}  
>>>
```

# Análise de dados com Python

## Exercícios – Dicionário

33

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie um dicionário chamado cardapio em que as chaves são os dias da semana e os respectivos valores sejam os pratos do dia.
3. Crie um dicionário chamado hemograma que contenha as seguintes chave:valor:  
hemacias: 4.71  
hemoglobina: 14.1  
hematocrito: 41.2  
linfocitos: 38  
monocitos: 7  
resultado: saudável
4. Corrija o dicionário acima com monocitos:12.

## Leitura e Escrita

35

» O Python permite entrada de dados tanto por arquivo quanto direto pelo IDLE. A função input() é a função utilizada para leitura de informação digitadas fora do código.

```
>>> nome = input('Escreva seu nome: ')
Escreva seu nome:
```

Abre-se um campo para digitar a informação

» A função input() sempre gera uma string. Quando necessário, devemos usar uma função para transformar em outro tipo de dado.

```
>>> idade = int(input('Qual sua idade: '))
Qual sua idade: 33
>>> type(idade)
<class 'int'>
>>> altura = float(input('Qual sua altura: '))
Qual sua altura: 1.55
>>> type(altura)
<class 'float'>
>>>
```

Transformando em valor inteiro

Transformação em valor float

# Análise de dados com Python

## Leitura e Escrita

36

- » Já para imprimir valores ao longo do código, podemos usar a função print() ou ainda a função display() em alguns idles.

```
>>> a = 3  
>>> b = 5  
>>> c = a * b  
>>> print(c)  
15
```

- » Existem algumas maneiras de referenciar uma variável dentro da função print(), dentre elas é usar o %.

```
>>> nome = input('Escreva seu nome: ')  
Escreva seu nome: Patricia  
>>> print('Seu nome é %s.' %nome)  
Seu nome é Patricia.  
>>>
```

## Leitura e Escrita

37

- » Para referenciar mais de uma variável, usamos como argumento da função print() uma tupla com as variáveis.

```
>>> a = 3  
>>> b = 5  
>>> c = a * b  
>>> print('A multiplicação de %d por %d é: %f' %(a,b,c))  
A multiplicação de 3 por 5 é: 15.000000  
>>>
```

- » Os tipos utilizados para referenciar corretamente as variáveis são:

|    |          |
|----|----------|
| %s | strings  |
| %d | inteiros |
| %f | float    |
| %% | %        |

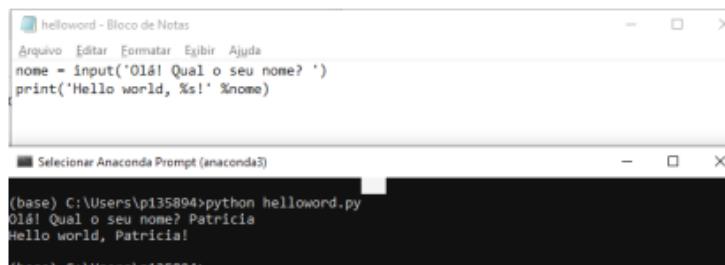
- » Para mais formatos de escrita: <https://realpython.com/python-print/>

# Análise de dados com Python

38

## Criado arquivos .py

- » Quando temos um programa com muitos comandos é inviável ter que escrevê-los direto no terminal.
- » Para essas situações, podemos escrever todos os nossos códigos em um arquivo e salvá-lo com o formato .py.
- » Para rodar o código basta, no terminal mas fora do idle Python, digitar *python nome\_do\_arquivo.py*



39

## Exercícios – Leitura e Escrita

1. Abra um bloco de notas e crie um código chamado imc.py que:
  - Pergunte o nome da pessoa e atribua na variável nome.
  - Pergunte a altura (em metros) e atribua na variável alt. Não esqueça de que a variável deve ser do tipo float.
  - Pergunte o peso (em quilos) e atribua na variável kg. Não esqueça de que a variável deve ser do tipo float.
  - Calcule o IMC através da fórmula  $IMC = \text{peso}/(\text{alt}^{\star}\text{alt})$
  - Escreva o resultado do cálculo do IMC como "Olá <Fulano>, seu IMC é <xx>", em que <Fulano> seja o nome da pessoa e <xx> seja o valor do IMC.

# Análise de dados com Python

40

## Condicionais

» Como qualquer linguagem de programação, as estruturas condicionais são partes importantes de um algoritmo.

» if:

```
>>> temperatura = 35
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
...
Hoje está muito quente!
>>>
```

Atenção à indentação!

```
>>> temperatura = 25
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
...
>>> print('Fim')
Fim
>>>
```

A condição fica entre os parênteses e os `:` indicam o início dos comandos a serem realizados caso a condição seja satisfeita.

## Condicionais

41

» if-else:

```
>>> temperatura = 28
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
... else:
...     print('A temperatura hoje está agradável!')
...
A temperatura hoje está agradável!
>>>
```

Uma vez que a 1a condição não foi satisfeita, passou-se para a 2a condição.

# Análise de dados com Python

## Condicionais

42

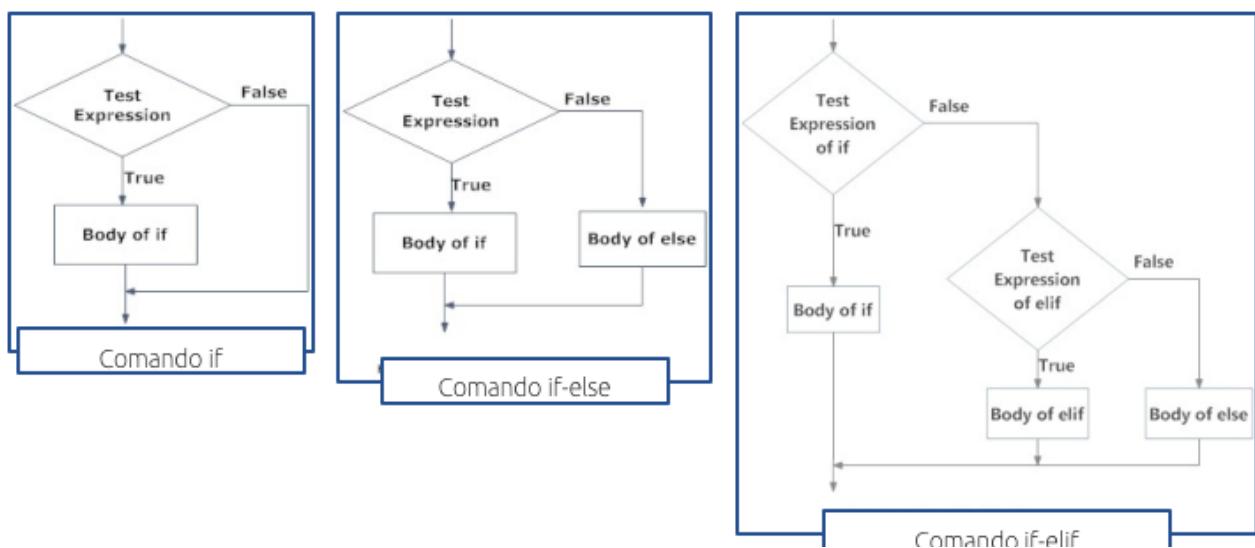
» if-elif: em Python, para evitar o aninhamento de diversos else's, existem a condicional elif, que seria uma espécie de else if.

```
>>> temperatura = 18
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
... elif(temperatura > 25):
...     print('A temperatura hoje está agradável!')
... elif(temperatura > 15):
...     print('O tempo está mudando!')
... else:
...     print('Que friaca!')
...
O tempo está mudando!
>>>
```

Usa-se quantos elif's forem necessários.

## Condicionais

43



# Análise de dados com Python

## Exercícios – Condicionais

44

1. Atualize o programa imc.py feito anteriormente para que o resultado exibido seja

"Olá <Fulano>, seu IMC é <xx>, logo você está <situação>."

em que a <situação> segue as condições abaixo:

| Resultado           | Situação                |
|---------------------|-------------------------|
| Abaixo de 17        | Muito abaixo do peso    |
| Entre 17 e 18,49    | Abaixo do peso          |
| Entre 18,50 e 24,99 | Peso normal             |
| Entre 25 e 29,99    | Acima do peso           |
| Entre 30 e 34,99    | Obesidade I             |
| Entre 35 e 39,99    | Obesidade II (severa)   |
| Acima de 40         | Obesidade III (mórbida) |

## Repetições – Loops

45

- » Outra estrutura importante de um algoritmo são as operações em laço (loop), onde uma determinada ação é feita repetidas vezes.
- » Em Python, temos os loops for e while.

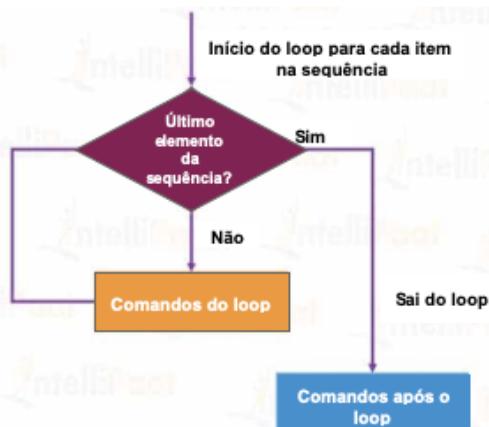
- for: opera sobre os itens de qualquer tipo de sequência (lista ou string) na ordem em que aparecem.
- while: usado quando precisamos repetir uma ação algumas vezes ou fazer uma iteração até que uma condição seja satisfeita.

# Análise de dados com Python

46

## Repetições: for

- » for: opera sobre os itens de qualquer tipo de sequência (lista ou string) na ordem em que aparecem.



## Repetições: for

47

- » Exemplos:

```
>>> frutas = ['abacate','pera','abacaxi']
>>> for fruta in frutas:
...     print(fruta)
...
abacate
pera
abacaxi
```

Repetição e  
Condicionall

```
>>> frutas = ['abacate','pera','abacaxi']
>>> for fruta in frutas:
...     if fruta.startswith('a'):
...         print(fruta)
...
abacate
abacaxi
```

Acumulador.

```
>>> lista_num = [1,2,3,4,5]
>>> soma = 0
>>> for numero in lista_num:
...     soma = soma + numero
...
>>> print(soma)
```

# Análise de dados com Python

## Exercícios – for

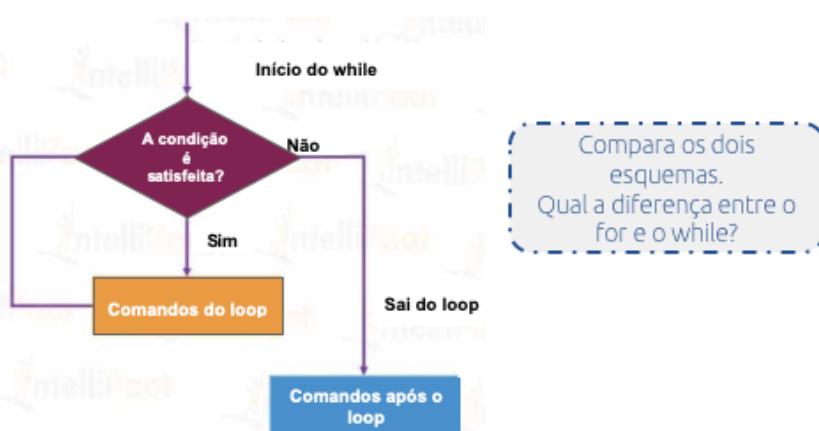
48

1. Abra um bloco de notas e crie um programa chamado tabuada.py que faça:
  - a. Declare uma lista multiplos = [1,2,3,4,5,6,7,8,9,10]
  - b. Peça ao usuário um número inteiro de 1 a 10 e atribua na variável number.
  - c. Faça um laço for que imprima os valores da tabuada de number.

## Loops - While

49

» while: usado quando precisamos repetir uma ação algumas vezes ou fazer uma iteração até que uma condição seja satisfeita.



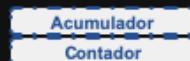
# Análise de dados com Python

## Loops - While

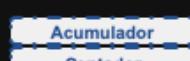
50

- Exemplos:

```
>>> condicao = 1  
>>> soma = 0  
>>> while condicao <= 5:  
...     soma = soma + condicao  
...     condicao = condicao + 1  
...  
>>> print(soma)  
15  
>>>
```



```
>>> condicao = 5  
>>> soma = 0  
>>> while condicao >= 0:  
...     soma = soma + condicao  
...     condicao = condicao - 1  
...  
>>> print(soma)  
15  
>>>
```



## Exercícios – While

51

1. Abra um bloco de notas e crie um programa chamado factorial.py que:
  - a. Peça um número inteiro entre 2 e 15 ao usuário e atribua na variável valor.
  - b. Crie uma variável fat e atribua um valor inicial igual a zero.
  - c. Crie um contador cont e atribua um valor inicial igual a zero.
  - d. Usando um loop while, enquanto cont for menor que valor, atualize fat como fat = fat x valor
  - e. Quando o loop terminar, imprima "O factorial de <valor> é <fat>", em que <valor> é o número dado pelo usuário e <fat> seja o resultado do loop.

# Introdução ao Numpy

Parceria:



# Análise de dados com Python

## Sumário

01

- » O que é Numpy e onde é utilizado
- » Arrays
- » Índices
- » Fatiamentos e seleções
- » Operações com arrays
- » Broadcasting
- » Funções universais
- » Métodos matemáticos

## O que é Numpy?

02

- » Numpy (*Numerical Python*) é uma biblioteca Python que permite trabalhar computação científica, com o uso de arrays e matrizes multidimensionais.
- » Inspirada no Matlab
- » Cálculos computacionais muito mais eficientes
- » **Funções nativas que facilitam e agilizam a manipulação de dados.**
- » O Numpy é *core* de diversas outras bibliotecas dentro do Python, como as bibliotecas Pandas e Scipy.



Como  
o Numpy  
funciona  
nativos?

# Análise de dados com Python

## Numpy: importação e alias

03

- » Numpy não é uma função built-in do Python, o que significa que ela não é nativa da linguagem e precisa ser carregada/importada antes de utilizarmos.
- » Em Python é muito comum o uso de *alias* para a chamada de funções de bibliotecas não nativas.
- » O Numpy, em geral, é chamado com o *alias np*.

```
import numpy as np
```

## Arrays: o que são?

04

- » As principais estruturas do Numpy são os arrays, uma estrutura que armazena diversos dados em N dimensões.
- » É similar ao conceito de listas.
- » Todos os dados precisam ser do mesmo tipo.
- » Existem diversas maneiras de criar arrays no Numpy. Dentre elas, podemos citar as funções `np.array()`, `np.ones()` e `np.full()`.

# Análise de dados com Python

05

## Arrays: o princípio

» A maneira mais simples e comum de declarar um array é usando a função `np.array()` onde usamos uma lista de valores como argumento.

```
np.array([1,2,3])                                array unidimensional  
array([1, 2, 3])  
  
np.array([[1,2,3],[4,5,6]])                      array bidimensional  
array([[1, 2, 3],  
       [4, 5, 6]])  
  
np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]]) array tridimensional  
array([[[1, 2, 3],  
       [4, 5, 6]],  
       [[1, 2, 3],  
       [4, 5, 6]]])
```

06

## Arrays: outros tipos

» Podemos ainda criar arrays específicos como, por exemplo, com 1 ou 0 com as funções `np.ones()` e `np.zeros()` em que os argumentos são as dimensões.

```
one = np.ones([2,2])                            array bidimensional, com valores  
one                                         1s.  
  
array([[1., 1.],  
       [1., 1.]])  
  
zeros = np.zeros([2,2,2])                       array tridimensional, com valores  
zeros                                       0s.  
  
array([[[0., 0.],  
       [0., 0.]],  
       [[0., 0.],  
       [0., 0.]]])
```

# Análise de dados com Python

07

## Arrays: outros tipos

Table 4-1. Array creation functions

| Function             | Description   |
|----------------------|---|
| array                | Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype; copies the input data by default          |
| asarray              | Convert input to ndarray, but do not copy if the input is already an ndarray  |
| arange               | Like the built-in range but returns an ndarray instead of a list  |
| ones,<br>ones_like   | Produce an array of all 1s with the given shape and dtype; ones_like takes another array and produces a ones array of the same shape and dtype  |
| zeros,<br>zeros_like | Like ones and ones_like but producing arrays of 0s instead  |
| empty,<br>empty_like | Create new arrays by allocating new memory, but do not populate with any values like ones and zeros   |
| full,<br>full_like   | Produce an array of the given shape and dtype with all values set to the indicated “fill value” full_like takes another array and produces a filled array of the same shape and dtype |
| eye, identity        | Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere)   |

Extraído de Mckinney, Wes (2012).

08

## Arrays: índices

» Assim como as listas, podemos acessar informações dos arrays através dos índices dos elementos.

» Array unidimensional:

```
data = np.array([0,1,2,3,4,5,6,7,8,9])  
  
data[2]  
2  
  
data[2:4]  
array([2, 3])
```

# Análise de dados com Python

09

## Arrays: Índices

» Array com mais de uma dimensão:

```
vet = np.array([[1,2,3],[4,5,6],[7,8,9]])
vet

array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

vet[1,:]

array([4, 5, 6])

vet[:,2]

array([3, 6, 9])

vet[1,1]

5
```

```
arr = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
arr
```

```
array([[[1, 2],
       [3, 4]],
      [[5, 6],
       [7, 8]]])
```

```
arr.shape
```

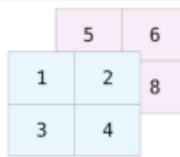
```
(2, 2, 2)
```

```
arr[0,0,1]
```

```
2
```

```
arr[1,1,0]
```

```
7
```



## Arrays: fatiamento

10

\* É preciso muito cuidado ao fatiar um array do Numpy.

» Qualquer alteração no fatiamento irá refletir no próprio array original.

```
data = np.arange(10)

data

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

data[0:5]

array([0, 1, 2, 3, 4])

data[0:5] = -99

data
```

```
data_1 = data[0:5]

data_1

array([-99, -99, -99, -99, -99])

data_1[2:4] = 5555

data

array([-99, -99, 5555, 5555, -99, 5, 6, 7, 8, 9])
```

# Análise de dados com Python

## Arrays: fatiamento e cópia

11

- » O fatiamento do Numpy gera uma visualização do array original.
- » Isso porque o Numpy Foi criado para trabalhar com muitos dados e fazer diversas cópias de um array pode sair computacionalmente caro.
- » Sempre que precisar copiar uma Fatia de um array, use a função `copy()`.

```
data_c = data[5:].copy()  
data_c  
array([5, 6, 7, 8, 9])  
  
data_c[0:3] = -99  
data_c
```

```
array([-99, -99, -99, 8, 9])  
  
data  
array([-99, -99, 5555, 5555, -99, 5, 6, 7, 8, 9])
```

## Arrays: fatiamento condicional

12

- » Podemos fatiar um array de acordo com uma condição.

```
data = np.arange(10)  
data[data > 5]  
array([6, 7, 8, 9])  
  
data[data == 4]  
array([4])  
  
data[data != 4]  
array([0, 1, 2, 3, 5, 6, 7, 8, 9])
```

# Análise de dados com Python

13

## Exercícios: fatiamento

1. Abra o Jupyter lab
2. Crie um array `data` com a seguinte lista: [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
3. Selecione apenas os dados diferentes de 4,6,8,10. Aloque em um array chamado `data_2`.
4. Crie um array `dim2` que tenha 2 dimensões e a lista [1,2,3,4,5] na primeira linha e a lista [6,7,8,9,10] na 2a linha.
5. Crie um array chamado `dim2_2` que seja uma fatia de `dim2` e que contenha apenas os valores 3,4,8,9.

14

## Arrays: funções úteis

- » Há várias funções que podem ser aplicadas aos arrays e que são muito úteis.
- » `shape`: informa as dimensões de um array.

```
data1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
data2 = np.array([[1,2,3,4],[4,5,6,7],[7,8,9,10]])
data3 = np.array([[1],[2],[3],[4],[5]])
data4 = np.array([1,2,3,4,5])

data1.shape
(3, 3)

data2.shape
(3, 4)

data3.shape
(5, 1)

data4.shape
(5,)
```

# Análise de dados com Python

15

## Arrays: funções úteis

» len: determina o comprimento da 1a dimensão de um array.

```
len(data1)
```

3

```
len(data2)
```

3

```
len(data3)
```

5

```
len(data4)
```

5

Ambos possuem apenas 1 dimensão e de mesmo comprimento!

16

## Arrays: funções úteis

» ndim: determina o número de dimensões de um array.

```
data1.ndim
```

2

```
data2.ndim
```

2

```
data3.ndim
```

2

```
data4.ndim
```

1

```
zeros.ndim
```

3

data3 possui uma coluna e 5 linhas, por isso 2 dimensões!

# Análise de dados com Python

17

## Arrays: Funções úteis

» size: retorna a quantidade de elementos presente em um array.

```
data1.size
```

```
9
```

```
data2.size
```

```
12
```

```
data3.size
```

```
5
```

```
data4.size
```

```
5
```

18

## Arrays: reshape

» A função reshape permite remodelar um array sem alterar os elementos que o compõe.

```
data2
```

```
array([[ 1,  2,  3,  4],
       [ 4,  5,  6,  7],
       [ 7,  8,  9, 10]])
```

```
data2.reshape(4,3)
```

```
array([[ 1,  2,  3],
       [ 4,  4,  5],
       [ 6,  7,  7],
       [ 8,  9, 10]])
```

```
data3
```

```
array([[[1],
        [2],
        [3],
        [4],
        [5]]])
```

```
data3.reshape(1,5)
```

```
array([[1, 2, 3, 4, 5]])
```

# Análise de dados com Python

19

## Arrays: flatten

» A função `flatten` transforma um array multidimensional em um unidimensional.

```
data1.flatten()  
array([1, 2, 3, 4, 5, 6, 7, 8, 9])  
  
data2.flatten()  
array([ 1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10])  
  
data3.flatten()  
array([1, 2, 3, 4, 5])  
  
data4.flatten()  
array([1, 2, 3, 4, 5])
```

20

## Arrays: transpose

ESTUDAR

» Para transpor um array, podemos usar a função `np.transpose()`.

|   |   |
|---|---|
| data1   | data2   |
| array([[1, 2, 3],<br>[4, 5, 6],<br>[7, 8, 9]])                      | array([[ 1,  2,  3,  4],<br>[ 4,  5,  6,  7],<br>[ 7,  8,  9, 10]])                           |
| data1.transpose()<br>array([[1, 4, 7],<br>[2, 5, 8],<br>[3, 6, 9]]) | data2.transpose()<br>array([[ 1,  4,  7],<br>[ 2,  5,  8],<br>[ 3,  6,  9],<br>[ 4,  7, 10]]) |

# Análise de dados com Python

## Exercícios: funções úteis

21

1. Abra o Jupyter lab
2. Crie um array *data* com a seguinte lista: [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
3. Verifique a dimensão, a quantidade de elementos e a forma desse array.
4. Crie um novo array chamado *data2*, com 4x4 elementos, aplicando a função *reshape* no array *data*.
5. Verifique a dimensão, a quantidade de elementos e a forma desse array.
6. Crie um novo array chamado *data\_t*, que seja a transposição do array *data2*.
7. Use a função *np.flatten()* e crie o array *data\_flat*.
8. Compare o array *data* com o array *data\_flat*.

## Operações com arrays

22

» Podemos calcular as operações aritméticas básicas dentro de um array (soma, subtração, multiplicação, divisão...)

```
a = np.array([[1,2],[3,4]])
a
array([[1, 2],
       [3, 4]])

a*2
array([[2, 4],
       [6, 8]])

a + 3
array([[4, 5],
       [6, 7]])
```

```
a - 2
array([[-1,  0],
       [ 1,  2]])

a/5
array([[0.2,  0.4],
       [0.6,  0.8]])

a ** 2
array([[ 1,  4],
       [ 9, 16]], dtype=int32)
```

# Análise de dados com Python

## Operações com arrays

23

- » Quando temos dois arrays de mesma forma (*shape*), também podemos realizar algumas operações e comparações.

```
a = np.array([[1,2],[3,4]])
```

```
a
```

```
array([[1, 2],  
       [3, 4]])
```

```
b = np.array([[1,2],[3,4]])
```

```
b
```

```
array([[1, 2],  
       [3, 4]])
```

```
a + b
```

```
array([[2, 4],  
       [6, 8]])
```

```
a - b
```

```
array([[0, 0],  
       [0, 0]])
```

```
a > b
```

```
array([[False, False],  
       [False, False]])
```

```
a == b
```

```
array([[ True,  True],  
       [ True,  True]])
```

```
a * b
```

```
array([[ 1,  4],  
       [ 9, 16]])
```

## Operações com Arrays: broadcasting

24

- » *Broadcasting* é um método usado pelo Numpy para realizar operações entre arrays de tamanhos diferentes.

```
b = np.array([2,3])
```

```
b
```

```
array([2, 3])
```

```
b * 3
```

escalar e 1 dimensional

```
array([6, 9])
```

```
a = np.array([[1,2],[3,4]])
```

```
a
```

```
array([[1, 2],  
       [3, 4]])
```

```
a * 3
```

escalar e 2 dimensões

```
array([[ 3,  6],  
       [ 9, 12]])
```

```
a * b
```

1 dimensão e 2 dimensões

```
array([[ 2,  6],  
       [ 6, 12]])
```

# Análise de dados com Python

25

## Exercícios: operações com arrays

1. Abra o Jupyter lab
2. Crie 5 arrays a, b, c, d e e com as seguintes listas.
  - a. [7]
  - b. [[4,6,8],[3,5,7]]
  - c. [3,3,3]
  - d. [[3,2,1],[1,2,3]]
  - e. [5,10]
3. Faça as seguintes operações:
  - a. a \* b
  - b. d - b
  - c. c + b
  - d. c + e
4. O que aconteceu no item d) do último exercício? Explique.

26

## Funções universais

- » Funções universais são funções matemáticas que são aplicadas a todos os elementos de um array.
- » Consulte as funções universais na documentação do Numpy:  
<https://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs>

```
a = np.array([[1, 2], [3, 4]])
a
array([[1, 2],
       [3, 4]])

np.sqrt(a)
array([[ 1.        ,  1.41421356],
       [ 1.73205081,  2.        ]])

np.exp(a)
array([[ 2.71828183,  7.3890561 ],
       [20.08553692, 54.59815003]])
```

```
np.sin(a)
array([[ 0.84147098,  0.90929743],
       [ 0.14112001, -0.7568025 ]])

np.negative(a)
array([[-1, -2],
       [-3, -4]])
```

# Análise de dados com Python

## Métodos matemáticos e estatísticos

27

- » O Numpy possibilita ainda extrair informações matemáticas e estatísticas dos arrays.
- » Podemos calcular a média, mediana, soma ou ainda extraír os máximos e mínimos de arrays.

```
a = np.array([[15, 22], [7, 12]])  
a  
  
array([[15, 22],  
       [ 7, 12]])  
  
a.mean()  
14.0  
  
a.min()  
7
```

```
a.max()  
22  
  
a.sum()  
56  
  
a.cumsum()  
array([15, 37, 44, 56], dtype=int32)
```

máximo

soma

soma cumulativa

onde  
junto?

## Exercícios: Funções e métodos matemáticos

28

1. Abra o Jupyter lab
2. Crie 2 arrays a e b com as seguintes listas:
  - a. [[4,6,8],[3,5,7]]
  - b. [[11,13,17],[23,29,31]]
3. Use a função `cbrt()` e calcule a raiz cúbica dos arrays a e b.
4. Calcule a soma cumulativa de a.
5. Calcule a média de a e b.
6. Aplique a função `np.negative()` no array b e some com b. O que aconteceu?

# Análise de dados com Python

29

## Concatenação

» Para concatenar arrays, usamos a função `np.concatenate()`.

» Para arrays unidimensionais:

```
a = np.array([1,2])
a
array([1., 2.])

b = np.array([3,4,5,6])
b
array([3., 4., 5., 6.])

c = np.array([7,8,9])
c
array([7., 8., 9.])

np.concatenate((a, b, c))
array([1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

30

## Concatenação

» Arrays com mais dimensões:

Por padrão, a concatenação acontece na 1a dimensão.

```
a = np.array([[1, 2], [3, 4]])
a
array([[1., 2.],
       [3., 4.]])

b = np.array([[5, 6], [7,8]])
b
array([[5., 6.],
       [7., 8.]])

np.concatenate((a,b))
array([[1., 2.],
       [3., 4.],
       [5., 6.],
       [7., 8.]])
```

# Análise de dados com Python

31

## Concatenação

» Para especificar o eixo da concatenação, usamos o argumento *axis*.

```
np.concatenate((a,b), axis=0)
```

```
array([[1., 2.],
       [3., 4.],
       [5., 6.],
       [7., 8.]])
```

```
np.concatenate((a,b), axis=1)
```

```
array([[1., 2., 5., 6.],
       [3., 4., 7., 8.]])
```

32

## Concatenação

» Para especificar o eixo da concatenação, usamos o argumento *axis*

```
a = np.array([[5, 6], [7,8]], [[5, 6], [7,8]]], float)
```

```
array([[[5., 6.],
       [7., 8.]],
      [[5., 6.],
       [7., 8.]]])
```

```
b = np.array([[1, 1], [3,4]], [[1, 2], [3,4]]], float)
```

```
array([[[1., 1.],
       [3., 4.]],
      [[1., 2.],
       [3., 4.]]])
```

```
np.concatenate((b,c), axis=0)
```

```
array([[[5., 6., 1., 1.],
       [7., 8., 3., 4.]],
      [[5., 6., 1., 2.],
       [7., 8., 3., 4.]]])
```

```
np.concatenate((b,c), axis=1)
```

```
array([[[5., 6.],
       [7., 8.],
       [1., 1.],
       [3., 4.]],
      [[5., 6.],
       [7., 8.],
       [1., 2.],
       [3., 4.]]])
```

```
np.concatenate((b,c), axis=2)
```

```
array([[[5., 6., 1., 1.],
       [7., 8., 3., 4.]],
      [[5., 6., 1., 2.],
       [7., 8., 3., 4.]]])
```

# Análise de dados com Python

## Exercícios: Concatenação

33

✓ Abra o Jupyter lab

✗ Crie 2 arrays arrays com as seguintes listas:

- a. `[[1,2,3],[4,5,6]]`
  - b. `[[2,4,6,8],[10,12,14,16]]`
  - c. `[[11,13,17],[23,29,31]]`
  - d. `[[13,14,5],[19,21,23]]`
3. Concatene a e b. O que aconteceu?
  4. Concatene b e c, tanto usando o eixo 0 (linha) quanto o eixo 1 (coluna).
  5. Concatene a e d, tanto usando o eixo 0 (linha) quanto o eixo 1 (coluna).

# Introdução ao Pandas I

Parceria:



# Análise de dados com Python

## Sumário

01

- » Pandas?
- » Leitura de arquivos
- » Dataframes e series
- » Manipulação de dataframes
- » Cópias, seleções e fatiamentos

## Pandas?

02

Pandas é uma biblioteca do Python muito utilizada em programação científica. Pandas trouxe um plus ao Python, pois possibilita trabalhar com análise de dados sem ter que recorrer a outras linguagens (como o R).

**Pandas = PANel DAtaS**

**PANel DAta S**



# Análise de dados com Python

---

Pandas?

03

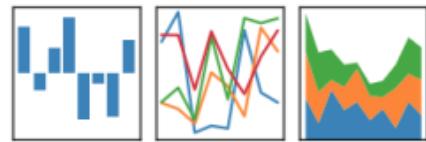
Mas o que o Pandas pode fazer por mim (por nós)?

O que podemos fazer com Pandas

04

- » Manipulação de dados: de forma rápida, ágil e com indexação integrada.
- » Análise de dados: Leitura, escrita, alinhamento, reshaping, slicing, agrupamentos, fusão, concatenação...
- » Variedade de usos: Mercado financeiro, Neurociência, Economia, Estatística, Publicidade(!) e muito mais...

**pandas**  
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



# Análise de dados com Python

## Vamos começar?

05

- » Importando a biblioteca

```
[1]: import pandas as pd
```

→ Pandas não é uma biblioteca built-in, então é preciso instalá-la, caso a instalação do Python tenha sido feita sem o Anaconda.

## Tipos de estruturas

06

- » No Pandas podemos trabalhar com dois tipos de estruturas: *Series* e *Dataframe*.
- » *Series*: um objeto unidimensional (*array*) que contém uma sequência de valores e seus respectivos índices. Podemos usar a função `pd.Series()` para criar uma *series*:

```
idades = pd.Series([15,21,30,12,24,52])  
idades  
0    15  
1    21  
2    30  
3    12  
4    24  
5    52  
dtype: int64
```

Como parâmetro, colocamos uma lista de valores

# Análise de dados com Python

ESTUDAR

## Tipos de estruturas - Series

07

- » Podemos também dizer qual o índice queremos para a Series:

```
idades2
```

```
Joao    15
Maria   21
Ana     30
Pedro   12
dtype: int64
```

- » Para confirmar os índices:

```
idades2.index
```

```
Index(['Joao', 'Maria', 'Ana', 'Pedro'], dtype='object')
```

- » Para obter apenas os valores:

```
idades2.values
```

```
array([15, 21, 30, 12], dtype=int64)
```

## Tipos de estruturas - Dataframe

08

- » Outra importante estrutura, o *Dataframe* é uma tabela de dados com diversas colunas, que podem ser de diferentes tipos (numérico, categórico, booleano...)

- » Podemos criar dataframes a partir da função pd.DataFrame(), utilizando como argumento tanto um array quanto usando dicionários.

```
» df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), columns=['a', 'b', 'c'])

df

   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

# Análise de dados com Python

09

## Tipos de estruturas - Dataframe

### » Usando dicionários:

```
dados = {'valor1': [1, 2], 'valor2': [3, 4], 'valor3':[5,6]}\ndf2 = pd.DataFrame(data=dados)
```

```
df2
```

|   | valor1 | valor2 | valor3 |
|---|--------|--------|--------|
| 0 | 1      | 3      | 5      |
| 1 | 2      | 4      | 6      |

ESTUDAR

### » Usando dicionários e explicitando os índices:

```
dados = {'valor1': [1, 2], 'valor2': [3, 4], 'valor3':[5,6]}\n df3 = pd.DataFrame(data=dados, index=['a','b'])
```

```
df3
```

|   | valor1 | valor2 | valor3 |
|---|--------|--------|--------|
| a | 1      | 3      | 5      |
| b | 2      | 4      | 6      |

## Tipos de estruturas - Dataframe

10

- » Similar ao caso da *Series*, podemos acessar tanto os valores quanto os índices do Dataframe com os métodos *pd.values* e *pd.index*:

```
df3.values\narray([[1, 3, 5],\n       [2, 4, 6]], dtype=int64)\n\ndf3.index\nIndex(['a', 'b'], dtype='object')
```

- » Ainda podemos obter os nomes das colunas com o *pd.columns*:

```
df3.columns\nIndex(['valor1', 'valor2', 'valor3'], dtype='object')
```

## Lendo um arquivo

- » Outra maneira de se criar um dataframe é através da leitura de um arquivo.
- » Pandas é capaz de ler diversos tipos de arquivos, com uma sintaxe muito simples. Dentre os tipos de arquivos que podemos ler com Pandas, temos:

```
» read_table
» read_csv
» read_excel
» read_hdf
» read_sql
» read_json
» read_html
» read_stata
» read_sas
```

## Exemplos de leitura

- » Pandas é capaz de ler diversos tipos de arquivos, com uma sintaxe muito simples. Veja alguns exemplos:

```
[2]: df = pd.read_csv('bank-full.csv')
      df = pd.read_table('bank-full.txt', delim_whitespace=True)
      df = pd.read_excel('bank-full.xlsx', sheet_name='Dados')
```

Há diversos parâmetros que podem ser adicionados ao comando de leitura...  
Veja a documentação de cada um sempre que precisar!

# Análise de dados com Python

13

## Manipulando o Dataframe Head e Tail

» Podemos verificar o cabeçalho do dataframe com o comando `df.head()`:

```
[3]: df.head()
```

|   | age | job          | marital | education | default | balance | housing | loan | contact | day | month | duration | campaign |
|---|-----|--------------|---------|-----------|---------|---------|---------|------|---------|-----|-------|----------|----------|
| 0 | 58  | management   | married | tertiary  | no      | 2143    | yes     | no   | unknown | 5   | may   | 261      |          |
| 1 | 44  | technician   | single  | secondary | no      | 29      | yes     | no   | unknown | 5   | may   | 151      |          |
| 2 | 33  | entrepreneur | married | secondary | no      | 2       | yes     | yes  | unknown | 5   | may   | 76       |          |
| 3 | 47  | blue-collar  | married | unknown   | no      | 1506    | yes     | no   | unknown | 5   | may   | 92       |          |
| 4 | 33  | unknown      | single  | unknown   | no      | 1       | no      | no   | unknown | 5   | may   | 198      |          |

14

## Manipulando o Dataframe Head e Tail

» Podemos verificar o final do dataframe com o comando `df.tail()`:

```
[4]: df.tail()
```

|       | age | job          | marital  | education | default | balance | housing | loan | contact   | day | month | duration | campaign |
|-------|-----|--------------|----------|-----------|---------|---------|---------|------|-----------|-----|-------|----------|----------|
| 45206 | 51  | technician   | married  | tertiary  | no      | 825     | no      | no   | cellular  | 17  | nov   | 977      |          |
| 45207 | 71  | retired      | divorced | primary   | no      | 1729    | no      | no   | cellular  | 17  | nov   | 456      |          |
| 45208 | 72  | retired      | married  | secondary | no      | 5715    | no      | no   | cellular  | 17  | nov   | 1127     |          |
| 45209 | 57  | blue-collar  | married  | secondary | no      | 668     | no      | no   | telephone | 17  | nov   | 508      |          |
| 45210 | 37  | entrepreneur | married  | secondary | no      | 2971    | no      | no   | cellular  | 17  | nov   | 361      |          |

# Análise de dados com Python

## Manipulando o Dataframe Nomeando as colunas

15

» Podemos usar o `df.columns` para modificar todos os nomes das colunas:

```
[4]: df.columns = ['idade', 'job', 'estado civil', 'educacao', 'default', 'balance', 'housing',  
    'emprestimo', 'contato', 'dia', 'mes', 'duracao', 'campanha', 'pdays', 'previous',  
    'poutcome']
```

» Ou modificar o nome de colunas específicas usando a função `df.rename()`:

```
[6]: df.rename(columns={"idade": "Idade", "job": "Profissao"}, inplace=True)
```

## Manipulando o Dataframe Info

16

```
df.info()  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 45211 entries, 0 to 45210  
Data columns (total 16 columns):  
 Idade      45211 non-null int64  
 Profissao   45211 non-null object  
 estado civil 45211 non-null object  
 educacao   45211 non-null object  
 default    45211 non-null object  
 balance    45211 non-null int64  
 housing    45211 non-null object  
 emprestimo 45211 non-null object  
 contato    45160 non-null object  
 dia        45211 non-null int64  
 mes        45211 non-null object  
 duracao    45200 non-null float64  
 campanha   45211 non-null int64  
 pdays      45211 non-null int64  
 previous   45211 non-null int64  
 poutcome   45211 non-null object  
 dtypes: float64(1), int64(6), object(9)  
memory usage: 5.5+ MB
```

» É essencial entendermos o tipo de dado que temos em mãos. O comando `df.info()` nos ajuda a verificar os tipos das nossas variáveis e, inclusive, se há valores faltantes/nulos.

As variáveis `contato` e `duracao`  
possuem dados faltantes/nulos

# Análise de dados com Python

## Manipulando o Dataframe

### Describe: estatísticas básicas

17

» A função `df.describe()` exibe as estatísticas básicas dos dados numéricos.

| [10]: | df.describe() |               |              |              |              |              |              |  |
|-------|---------------|---------------|--------------|--------------|--------------|--------------|--------------|--|
|       | Idade         | balance       | dia          | duracao      | campanha     | pdays        | previous     |  |
| count | 45211.000000  | 45211.000000  | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 |  |
| mean  | 40.936210     | 1362.272058   | 15.806419    | 258.163080   | 2.763841     | 40.197828    | 0.580323     |  |
| std   | 10.618762     | 3044.765829   | 8.322476     | 257.527812   | 3.098021     | 100.128746   | 2.303441     |  |
| min   | 18.000000     | -8019.000000  | 1.000000     | 0.000000     | 1.000000     | -1.000000    | 0.000000     |  |
| 25%   | 33.000000     | 72.000000     | 8.000000     | 103.000000   | 1.000000     | -1.000000    | 0.000000     |  |
| 50%   | 39.000000     | 448.000000    | 16.000000    | 180.000000   | 2.000000     | -1.000000    | 0.000000     |  |
| 75%   | 48.000000     | 1428.000000   | 21.000000    | 319.000000   | 3.000000     | -1.000000    | 0.000000     |  |
| max   | 95.000000     | 102127.000000 | 31.000000    | 4918.000000  | 63.000000    | 871.000000   | 275.000000   |  |

## Manipulando o Dataframe

### Describe: estatísticas básicas

18

» Para estatísticas de variáveis categóricas, precisamos adicionar o argumento `include=['O']`.

| [11]:  | df.describe(include=['O']) |              |           |         |         |            |          |       |          |  |
|--------|----------------------------|--------------|-----------|---------|---------|------------|----------|-------|----------|--|
|        | Profissao                  | estado civil | educacao  | default | housing | emprestimo | contato  | mes   | poutcome |  |
| count  | 45211                      | 45211        | 45211     | 45211   | 45211   | 45211      | 45211    | 45211 | 45211    |  |
| unique | 12                         | 3            | 4         | 2       | 2       | 2          | 3        | 12    | 4        |  |
| top    | blue-collar                | married      | secondary | no      | yes     | no         | cellular | may   | unknown  |  |
| freq   | 9732                       | 27214        | 23202     | 44396   | 25130   | 37967      | 29285    | 13766 | 36959    |  |

top: o valor que mais aparece  
freq: frequência do valor mais comum (top)

# Análise de dados com Python

19

## Manipulando o Dataframe

Describe: estatísticas básicas

» Ainda, é possível aplicar o `pd.describe` a todos os dados:

|        | idade        | Profissao   | estado_civil | educacao  | default | balance       | housing | emprestimo | contato |         |
|--------|--------------|-------------|--------------|-----------|---------|---------------|---------|------------|---------|---------|
| count  | 45211.000000 | 45211       | 45211        | 45211     | 45211   | 45211.000000  | 45211   | 45211      | 45160   | 45211.0 |
| unique | NaN          | 12          | 3            | 4         | 2       | NaN           | 2       | 2          | 3       |         |
| top    | NaN          | blue-collar | married      | secondary | no      | NaN           | yes     | no         | celular |         |
| freq   | NaN          | 9732        | 27214        | 23202     | 44396   | NaN           | 25130   | 37967      | 29285   |         |
| mean   | 40.936210    | NaN         | NaN          | NaN       | NaN     | 1362.272058   | NaN     | NaN        | NaN     | 15.81   |
| std    | 10.618762    | NaN         | NaN          | NaN       | NaN     | 3044.765829   | NaN     | NaN        | NaN     | 8.31    |
| min    | 18.000000    | NaN         | NaN          | NaN       | NaN     | -8019.000000  | NaN     | NaN        | NaN     | 1.0     |
| 25%    | 33.000000    | NaN         | NaN          | NaN       | NaN     | 72.000000     | NaN     | NaN        | NaN     | 8.0     |
| 50%    | 39.000000    | NaN         | NaN          | NaN       | NaN     | 448.000000    | NaN     | NaN        | NaN     | 16.0    |
| 75%    | 48.000000    | NaN         | NaN          | NaN       | NaN     | 1428.000000   | NaN     | NaN        | NaN     | 21.0    |
| max    | 95.000000    | NaN         | NaN          | NaN       | NaN     | 102127.000000 | NaN     | NaN        | NaN     | 31.0    |

20

## Manipulando o Dataframe

Exercícios

1. Abra o arquivo `world_happiness_report_2015.csv` e aloque em um dataframe.
2. Verifique o cabeçalho e o final do dataframe.
3. Quais as colunas desse dataframe?
4. Quais os tipos de dados temos no dataframe?
5. Há valores faltantes ou nulos? Em quais colunas?
6. Renomeie as variáveis como segue:

|                                      |    |                  |
|--------------------------------------|----|------------------|
| <b>happiness rank</b>                | => | rank_felicidade  |
| <b>happiness score</b>               | => | score_felicidade |
| <b>standard error</b>                | => | stand_error      |
| <b>economy (GDP per Capita)</b>      | => | PIB              |
| <b>health (Life Expectancy)</b>      | => | expect_vida      |
| <b>trust (Government Corruption)</b> | => | corrupcao        |

7. Quais os valores médios de `expect_vida`? E o valor mediano? E o máximo da variável `PIB`?
8. Crie uma `series` que contenha a altura de 5 colegas e deixe seus nomes como índice.

# Análise de dados com Python

21

## Manipulando o Dataframe Selecionando dados

» Podemos acessar as informações dos dataframes e series de diversas maneiras.

» Quando queremos apenas 1 coluna:

`df['Idade']`

|   |    |
|---|----|
| 0 | 58 |
| 1 | 44 |
| 2 | 33 |
| 3 | 47 |

ou

`df.Idade`

|   |    |
|---|----|
| 0 | 58 |
| 1 | 44 |
| 2 | 33 |
| 3 | 47 |

» Quando queremos mais de 1 coluna, podemos usar uma lista de variáveis

`df[['Idade','dia','Profissao']]`

|   | Idade | dia | Profissao    |
|---|-------|-----|--------------|
| 0 | 58    | 5   | management   |
| 1 | 44    | 5   | technician   |
| 2 | 33    | 5   | entrepreneur |

22

## Manipulando o Dataframe Seleção: iloc e loc

» Os métodos `iloc` e `loc` são utilizados para selecionar dados de um dataframe, mas possuem diferenças importantes.

» **iloc:** seleção baseadas nas posições dos índices das linhas e colunas (inteiros)

» **loc:** seleção baseadas nos nomes das variáveis

» Em ambos os casos, os argumentos do método são as linhas e as colunas de interesse.

`df.loc[<linhas>, <colunas>]`

`df.iloc[<linhas>, <colunas>]`

# Análise de dados com Python

23

## Seleção: iloc

» O iloc faz a seleção através dos valores inteiros dos índices, por um array ou ainda por fatias dos dados.

```
df.iloc[1]  
Idade          44  
Profissao      technician  
estado civil   single  
educacao       secondary  
default         no  
balance        29  
housing         yes  
emprestimo     no  
contato        NaN  
dia             5  
mes             may  
duracao        151  
campanha       1  
pdays          -1  
previous        0  
outcome         unknown  
Name: 1, dtype: object
```

```
df.iloc[[1]]  
Idade  Profissao  estado civil  educacao  default  balance  housing  emprestimo  contato  dia  mes  duracao  campa  
1      technician  single    secondary    no       29      yes      no      no      NaN      5    may    151.0
```

Note a diferença entre os resultados. Embora os valores sejam os mesmos, a apresentação é diferente

## Seleção: iloc

24

» Seleção de linhas

```
df.iloc[[1]]
```

Apenas uma linha

```
Idade  Profissao  estado civil  educacao  default  balance  housing  emprestimo  contato  dia  mes  duracao  campa  
1      technician  single    secondary    no       29      yes      no      no      NaN      5    may    151.0
```

```
df.iloc[[-3]]
```

Apenas uma linha

```
Idade  Profissao  estado civil  educacao  default  balance  housing  emprestimo  contato  dia  mes  duracao  campa  
45208    72      retired  married  secondary    no      5715      no      no      no      celular    17    nov    1127.0
```

```
df.iloc[1:3]
```

Um fatiamento de linhas

```
Idade  Profissao  estado civil  educacao  default  balance  housing  emprestimo  contato  dia  mes  duracao  campa  
1      technician  single    secondary    no       29      yes      no      no      NaN      5    may    151.0  
2      entrepreneur  married  secondary    no        2      yes      yes      yes      NaN      5    may    76.0
```

# Análise de dados com Python

25

## Seleção: iloc

### » Seleção de colunas

`df.iloc[:,1]`

```
0      management
1      technician
2    entrepreneur
3     blue-collar
4        unknown
5      management
6      management
7    entrepreneur
8       retired
9      technician
10     admin.
11     admin.
12    technician
13    technician
```

Apenas uma coluna

`df.iloc[:,1:3]`

|   | Profissao    | estado civil |
|---|--------------|--------------|
| 0 | management   | married      |
| 1 | technician   | single       |
| 2 | entrepreneur | married      |
| 3 | blue-collar  | married      |
| 4 | unknown      | single       |
| 5 | management   | married      |
| 6 | management   | single       |
| 7 | entrepreneur | divorced     |

Um fatiamento de colunas

## Seleção: iloc

26

### » Seleção de linhas e colunas

`df.iloc[1,0:3]`

```
Idade            44
Profissao      technician
estado civil    single
Name: 1, dtype: object
```

Uma linha e um fatiamento de colunas

`df.iloc[0:2,0:3]`

|   | Idade | Profissao  | estado civil |
|---|-------|------------|--------------|
| 0 | 58    | management | married      |
| 1 | 44    | technician | single       |

Uma fatiamento de linhas e colunas

`df.iloc[[0,5,8],0:3]`

|   | Idade | Profissao  | estado civil |
|---|-------|------------|--------------|
| 0 | 58    | management | married      |
| 5 | 35    | management | married      |
| 8 | 58    | retired    | married      |

Um array e um fatiamento de colunas

# Análise de dados com Python

27

## Seleção: loc

» Selecionando apenas os valores de uma linha:

| df.loc[[1]] |       |            |              |           |         |         |         |            |         |     |     |         |     |
|-------------|-------|------------|--------------|-----------|---------|---------|---------|------------|---------|-----|-----|---------|-----|
|             | Idade | Profissao  | estado civil | educacao  | default | balance | housing | emprestimo | contato | dia | mes | duracao | car |
| 1           | 44    | technician | single       | secondary | no      | 29      | yes     | no         | NaN     | 5   | may | 151.0   |     |

Similar ao que observamos com o iloc

| df.loc[1]    |                  |
|--------------|------------------|
| Idade        | 44               |
| Profissao    | technician       |
| estado civil | single           |
| educacao     | secondary        |
| default      | no               |
| balance      | 29               |
| housing      | yes              |
| emprestimo   | no               |
| contato      | NaN              |
| dia          | 5                |
| mes          | may              |
| duracao      | 151              |
| campanha     | 1                |
| pdays        | -1               |
| previous     | 0                |
| outcome      | unknown          |
| Name:        | 1, dtype: object |

28

## Seleção: loc

» Selecionando uma lista de linhas:

| df.loc[[0,1,2]] |       |              |              |           |         |         |         |            |         |     |     |         |     |
|-----------------|-------|--------------|--------------|-----------|---------|---------|---------|------------|---------|-----|-----|---------|-----|
|                 | Idade | Profissao    | estado civil | educacao  | default | balance | housing | emprestimo | contato | dia | mes | duracao | car |
| 0               | 58    | management   | married      | tertiary  | no      | 2143    | yes     | no         | NaN     | 5   | may | 261.0   |     |
| 1               | 44    | technician   | single       | secondary | no      | 29      | yes     | no         | NaN     | 5   | may | 151.0   |     |
| 2               | 33    | entrepreneur | married      | secondary | no      | 2       | yes     | yes        | NaN     | 5   | may | 76.0    |     |

Um array de linhas

| df.loc[0:2] |       |              |              |           |         |         |         |            |         |     |     |         |     |
|-------------|-------|--------------|--------------|-----------|---------|---------|---------|------------|---------|-----|-----|---------|-----|
|             | Idade | Profissao    | estado civil | educacao  | default | balance | housing | emprestimo | contato | dia | mes | duracao | car |
| 0           | 58    | management   | married      | tertiary  | no      | 2143    | yes     | no         | NaN     | 5   | may | 261.0   |     |
| 1           | 44    | technician   | single       | secondary | no      | 29      | yes     | no         | NaN     | 5   | may | 151.0   |     |
| 2           | 33    | entrepreneur | married      | secondary | no      | 2       | yes     | yes        | NaN     | 5   | may | 76.0    |     |

Uma fatia de linhas

# Análise de dados com Python

29

## Seleção: loc

» Selecionando uma lista de linhas e colunas:

```
df.loc[[0,1],['Idade','Profissao','default']]
```

Uma lista de linhas e uma lista de colunas

|   | Idade | Profissao  | default |
|---|-------|------------|---------|
| 0 | 58    | management | no      |
| 1 | 44    | technician | no      |

» Selecionando uma fatia de linhas e uma lista de colunas:

```
df.loc[0:2,['Idade','Profissao','default']]
```

Uma fatia de linhas e uma lista de colunas

|   | Idade | Profissao    | default |
|---|-------|--------------|---------|
| 0 | 58    | management   | no      |
| 1 | 44    | technician   | no      |
| 2 | 33    | entrepreneur | no      |

## Seleção: loc

30

» Selecionando com base em *uma condição*:

```
df.loc[df.Idade > 55]
```

Uma condição

|    | Idade | Profissao  | estado civil | educacao | default | balance | housing | emprestimo | contato | dia | mes | durac |
|----|-------|------------|--------------|----------|---------|---------|---------|------------|---------|-----|-----|-------|
| 0  | 58    | management | married      | tertiary | no      | 2143    | yes     | no         | NaN     | 5   | may | 26    |
| 8  | 58    | retired    | married      | primary  | no      | 121     | yes     | no         | NaN     | 5   | may | 3     |
| 13 | 58    | technician | married      | unknown  | no      | 71      | yes     | no         | NaN     | 5   | may | 7     |

» Selecionando com base em *mais de uma condição*:

```
df.loc[(df.Idade > 55) & (df.Profissao == 'technician')]
```

Duas ou mais condições

|    | Idade | Profissao  | estado civil | educacao  | default | balance | housing | emprestimo | contato | dia | mes | duracao |
|----|-------|------------|--------------|-----------|---------|---------|---------|------------|---------|-----|-----|---------|
| 13 | 58    | technician | married      | unknown   | no      | 71      | yes     | no         | NaN     | 5   | may | 71.0    |
| 30 | 57    | technician | married      | secondary | no      | 839     | no      | yes        | unknown | 5   | may | 225.0   |
| 35 | 57    | technician | divorced     | secondary | no      | 63      | yes     | no         | unknown | 5   | may | 242.0   |

# Análise de dados com Python

31

## Exercícios: iloc e loc

1. Selecione apenas os dados de country, region, family e freedom ([usando loc](#))
2. Selecione apenas os dados de country, region, family e freedom ([usando iloc](#))
3. Selecione apenas as primeiras 15 linhas de country e PIB ([usando loc](#))
4. Selecione apenas as primeiras 15 linhas de country e PIB ([usando iloc](#))
5. Selecione apenas os dados cujo score\_felicidade seja maior que 5.
6. Selecione apenas os dados que sejam da *Southern Asia*.

# Introdução ao Pandas II

Parceria:



# Análise de dados com Python

## Sumário

01

- » Operações com Dataframes
- » União de dataframes: merge e concat
- » Agrupamentos: groupby
- » Agregações: agg

## Operações com Dataframes

02

- » Os dataframes do Pandas são estruturas muito versáteis pois permitem uma série de operações que ajudam nas análises dos dados. Veremos agora as seguintes operações:
  - » contagem de valores
  - » média
  - » soma
  - » valores únicos
  - » limpeza de dados duplicados
  - » limpeza de dados faltantes

# Análise de dados com Python

03

## Value\_counts

» Não poucas vezes precisaremos saber a frequência com que uma dada informação se repete. Para obter tal informação de uma dada variável no Pandas podemos utilizar a função pd.value\_counts().

```
df.educacao.value_counts()
```

```
secondary    23202  
tertiary     13301  
primary      6851  
unknown      1857  
Name: educacao, dtype: int64
```

```
df['estado_civil'].value_counts()  
  
married    27214  
single     12790  
divorced   5207  
Name: estado_civil, dtype: int64
```

Quantidade de dados com  
estado civil casado, solteiro e  
divorciado.

04

## Média

» Veremos em mais detalhes nas próximas aulas, mas podemos obter a média simples dos dados utilizando a função pd.mean().

```
df.balance.mean()
```

```
1362.2720576850766
```

```
df.duracao.mean()
```

```
258.16935840707964
```

```
df[['balance','duracao']].mean()
```

```
balance    1362.272058  
duracao    258.169358  
dtype: float64
```

Média de uma lista de variáveis

# Análise de dados com Python

## Soma

05

- » Quando precisamos saber a soma dos valores de uma determinada variável podemos usar a função pd.sum().

```
df.balance.sum()
```

```
61600855
```

```
df.duration.sum()
```

```
11672773.0
```

```
df[['pdays', 'duration', 'balance']].sum()
```

```
pdays      1818535.0  
duration   11672773.0  
balance    61600855.0  
dtype: float64
```

Soma de uma lista de variáveis

## Unique e nunique

06

- » Outra importante função é a pd.unique(), que nos mostra todos os valores únicos em uma dada variável.

```
df.Profissao.unique()
```

```
array(['management', 'technician', 'entrepreneur', 'blue-collar',  
       'unknown', 'retired', 'admin.', 'services', 'self-employed',  
       'unemployed', 'housemaid', 'student'], dtype=object)
```

- » Já a função pd.nunique() nos mostra a quantidade de valores únicos em uma variável.

```
df.Profissao.nunique()
```

```
12
```

# Análise de dados com Python

07

## Dados duplicados

» Outra importação ação aos analisar dados é verificar se há **duplicidade** nas **informações**, evitando assim que tiremos conclusões errôneas sobre os dados. Para verificar a existência de duplicidade, podemos usar a função pd.duplicated() e pd.sum(). **NAO ENTENDEI**

```
df.duplicated().sum()
```

10

» Para saber quais os valores duplicados podemos utilizamos o loc.

```
df.loc[df.duplicated() == True]
```

|       | age | job          | marital | education | default | balance | housing | loan | contact  | day | month | duration | campaign |
|-------|-----|--------------|---------|-----------|---------|---------|---------|------|----------|-----|-------|----------|----------|
| 45211 | 44  | technician   | single  | secondary | no      | 29      | yes     | no   | NaN      | 5   | may   | 151.0    |          |
| 45212 | 33  | entrepreneur | married | secondary | no      | 2       | yes     | yes  | NaN      | 5   | may   | 76.0     |          |
| 45213 | 38  | technician   | married | secondary | no      | 557     | yes     | no   | cellular | 16  | nov   | 1556.0   |          |

08

## Dados duplicados

» Para criar um novo dataframe sem os dados duplicados utilizamos a função pd.drop\_duplicates()

```
df2 = df.drop_duplicates()
```

```
df2.head()
```

|   | age | job          | marital | education | default | balance | housing | loan | contact | day | month | duration | campaign |
|---|-----|--------------|---------|-----------|---------|---------|---------|------|---------|-----|-------|----------|----------|
| 0 | 58  | management   | married | tertiary  | no      | 2143    | yes     | no   | NaN     | 5   | may   | 261.0    |          |
| 1 | 44  | technician   | single  | secondary | no      | 29      | yes     | no   | NaN     | 5   | may   | 151.0    |          |
| 2 | 33  | entrepreneur | married | secondary | no      | 2       | yes     | yes  | NaN     | 5   | may   | 76.0     |          |
| 3 | 47  | blue-collar  | married | unknown   | no      | 1506    | yes     | no   | NaN     | 5   | may   | 92.0     |          |
| 4 | 33  | unknown      | single  | unknown   | no      | 1       | no      | no   | NaN     | 5   | may   | 198.0    |          |

```
df2.duplicated().sum()
```

Verificando que não há dados duplicados

# Análise de dados com Python

## Dados faltantes ou nulos

09

- » Dados faltantes ou nulos podem enviesar as análises, por isso é sempre importante eliminá-los ou substituí-los (de acordo com o contexto de análise).
- » Uma maneira de saber a quantidade de dados faltantes é utilizando a função pd.isna ou pd.isnull com a função pd.sum().

```
df.isna().sum()           df.isnull().sum()  
age      0                age      0  
job      0                job      0  
marital  0                marital  0  
education 0               education 0  
default   0               default   0  
balance   0               balance   0  
housing   0               housing   0  
loan      0               loan      0  
contact   78              contact   78  
day       0               day       0  
month     0               month     0  
duration  11              duration  11  
campaign  0               campaign  0  
pdays    0                pdays    0  
previous  0               previous  0  
poutcome  0               poutcome  0  
dtype: int64
```

## Dados faltantes ou nulos

10

- » Para criar um novo dataframe sem os dados faltantes ou nulos utilizamos a função pd.dropna()

```
df3 = df.dropna()  
  
df3.isna().sum()
```

Verificando que não há dados faltantes

Existem várias maneiras de retirar os valores faltantes. Para conhecê-los, consulte a documentação da função.

# Análise de dados com Python

## Dados faltantes ou nulos

11

- » Quando for necessário substituir os dados faltantes por um novo valor, podemos usar a função pd.fillna()

|   | age | job          | marital | education | default | balance | housing | loan | contact      | day | month | duration | campaign |
|---|-----|--------------|---------|-----------|---------|---------|---------|------|--------------|-----|-------|----------|----------|
| 0 | 58  | management   | married | tertiary  | no      | 2143    | yes     | no   | desconhecido | 5   | may   | 261      |          |
| 1 | 44  | technician   | single  | secondary | no      | 29      | yes     | no   | desconhecido | 5   | may   | 151      |          |
| 2 | 33  | entrepreneur | married | secondary | no      | 2       | yes     | yes  | desconhecido | 5   | may   | 76       |          |
| 3 | 47  | blue-collar  | married | unknown   | no      | 1506    | yes     | no   | desconhecido | 5   | may   | 92       |          |
| 4 | 33  | unknown      | single  | unknown   | no      | 1       | no      | no   | desconhecido | 5   | may   | 198      |          |

Existem várias maneiras de preencher os valores nulos. Para conhecê-los, consulte a documentação da função.

## Exercícios – Operações

12

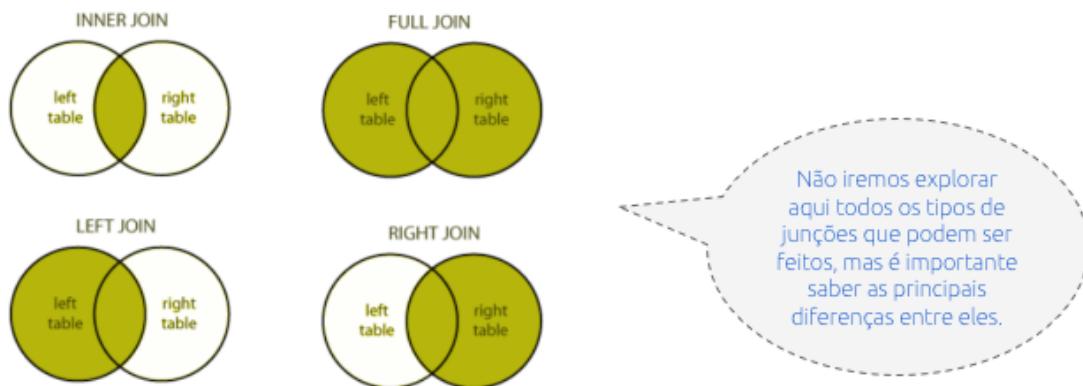
- » Qual a média do score\_felicidade?
- » Qual a soma do PIB?
- » Qual a soma do freedom e corrupcao?
- » Há dados duplicados? Quantos? Verifique quais são eles.
- » Verifique a quantidade de dados faltantes.
- » Crie um novo dataframe onde os valores faltantes de score\_felicidade sejam substituídos por -9999.
- » Quantas e quais são regions existentes nos dados?
- » Verifique a frequência dos dados segundo suas regiões. Qual a região com maior quantidade de dados? E a região com a menor quantidade?

# Análise de dados com Python

## Unindo amostras de dados

13

» É muito comum precisarmos unir dados de diferentes conjuntos de dados. O esquema abaixo exemplifica bem alguns tipos de união.



## Unindo amostras de dados

14

» Com o Pandas temos alguns métodos para unir dados de datasets distintos, dentre os quais podemos citar:

- concat: função que une os datasets ao longo de um eixo.
- merge: método que combina um ou mais datasets baseado em uma coluna em comum.

Veremos o básico de ambos a seguir.

# Análise de dados com Python

## Dataframes

15

» Vamos antes criar os dataframes que iremos utilizar:

```
dados1 = {
    'ID': ['1', '2', '3', '4', '5'],
    'nome': ['Paula', 'Claudia', 'Joao', 'Carlos', 'Ana'],
    'sobrenome': ['Pereira', 'Silva', 'Silveira', 'Bezerra', 'Souza']}
df1 = pd.DataFrame(dados1, columns = ['ID', 'nome', 'sobrenome'])
display('df1')
display(df1)

dados2 = {
    'ID': ['4', '5', '6', '7', '8'],
    'nome': ['Eder', 'Joana', 'Paulo', 'Pedro', 'Bete'],
    'sobrenome': ['Silve', 'Bezerra', 'Fernandes', 'Brito', 'Oliveira']}
df2 = pd.DataFrame(dados2, columns = ['ID', 'nome', 'sobrenome'])
display('df2')
display(df2)

notas = {
    'ID': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
    'notas_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}
df_notas = pd.DataFrame(notas, columns = ['ID','notas_id'])
display('df_notas')
display(df_notas)
```

## Dataframes

15

» Vamos antes criar os dataframes que iremos utilizar:

| 'df1' |    |         | 'df2'    |    |      | 'df_notas' |    |          |
|-------|----|---------|----------|----|------|------------|----|----------|
|       | ID | nome    |          | ID | nome |            | ID | notas_id |
| 0     | 1  | Paula   | Pereira  | 0  | 4    | Eder       | 1  | 51       |
| 1     | 2  | Claudia | Silva    | 1  | 5    | Joana      | 2  | 15       |
| 2     | 3  | Joao    | Silveira | 2  | 6    | Paulo      | 3  | 15       |
| 3     | 4  | Carlos  | Bezerra  | 3  | 7    | Pedro      | 4  | 61       |
| 4     | 5  | Ana     | Souza    | 4  | 8    | Bete       | 5  | 16       |

# Análise de dados com Python

## concat

16

» A função pd.concat() irá unir os dados ao longo de um eixo (o default é pelo índice).

```
df_1_2 = pd.concat([df1, df2])  
df_1_2
```

| ID | nome | sobrenome       |
|----|------|-----------------|
| 0  | 1    | Paula Pereira   |
| 1  | 2    | Claudia Silva   |
| 2  | 3    | Joao Silveira   |
| 3  | 4    | Carlos Bezerra  |
| 4  | 5    | Ana Souza       |
| 0  | 4    | Eder Silva      |
| 1  | 5    | Joana Bezerra   |
| 2  | 6    | Paulo Fernandes |
| 3  | 7    | Pedro Brito     |
| 4  | 8    | Bete Oliveira   |

## concat

16

» Unindo pelas colunas, utilizando o parâmetro axis=1:

```
df_1_2 = pd.concat([df1, df2], axis=1)  
df_1_2
```

| ID | nome | sobrenome      | ID | nome  | sobrenome |
|----|------|----------------|----|-------|-----------|
| 0  | 1    | Paula Pereira  | 4  | Eder  | Silva     |
| 1  | 2    | Claudia Silva  | 5  | Joana | Bezerra   |
| 2  | 3    | Joao Silveira  | 6  | Paulo | Fernandes |
| 3  | 4    | Carlos Bezerra | 7  | Pedro | Brito     |
| 4  | 5    | Ana Souza      | 8  | Bete  | Oliveira  |

# Análise de dados com Python

## concat

16

» Unindo pelas colunas, utilizando o parâmetro axis=1:

```
df_1_notas = pd.concat([df1, df_notas], axis=1)
df_1_notas
```

|   | ID  | nome    | sobrenome | ID | notas_id |
|---|-----|---------|-----------|----|----------|
| 0 | 1   | Paula   | Pereira   | 1  | 51       |
| 1 | 2   | Claudia | Silva     | 2  | 15       |
| 2 | 3   | Joao    | Silveira  | 3  | 15       |
| 3 | 4   | Carlos  | Bezerra   | 4  | 61       |
| 4 | 5   | Ana     | Souza     | 5  | 16       |
| 5 | NaN | NaN     | NaN       | 7  | 14       |
| 6 | NaN | NaN     | NaN       | 8  | 15       |
| 7 | NaN | NaN     | NaN       | 9  | 1        |
| 8 | NaN | NaN     | NaN       | 10 | 61       |
| 9 | NaN | NaN     | NaN       | 11 | 16       |

## merge

17

» A função pd.merge também une datasets, porém utilizando explicitamente a coluna de interesse.

```
df_1_2_notas = pd.merge(df_1_2, df_notas, on='ID')
df_1_2_notas
```

|   | ID | nome    | sobrenome | notas_id |
|---|----|---------|-----------|----------|
| 0 | 1  | Paula   | Pereira   | 51       |
| 1 | 2  | Claudia | Silva     | 15       |
| 2 | 3  | Joao    | Silveira  | 15       |
| 3 | 4  | Carlos  | Bezerra   | 61       |
| 4 | 4  | Eder    | Silva     | 61       |
| 5 | 5  | Ana     | Souza     | 16       |
| 6 | 5  | Joana   | Bezerra   | 16       |
| 7 | 7  | Pedro   | Brito     | 14       |
| 8 | 8  | Bete    | Oliveira  | 15       |

Os dataframes foram unidos com base na coluna ID.

Repare na quantidade de registros. O que há de diferente?

# Análise de dados com Python

## merge

18

- » A função pd.merge permite fazer diversos tipos de união, como os mostrado no diagrama anterior, bastando ajustar o parâmetro how:

| Merge method | SQL Join Name    | Description                               |
|--------------|------------------|---|
| left         | LEFT OUTER JOIN  | Use keys from left frame only             |
| right        | RIGHT OUTER JOIN | Use keys from right frame only            |
| outer        | FULL OUTER JOIN  | Use union of keys from both frames        |
| inner        | INNER JOIN       | Use intersection of keys from both frames |

```
pd.merge(df_esq, df_dir, on='coluna_em_comum', how='metodo_de_junção')
pd.merge(df_esq, df_dir, on='coluna_em_comum', how='left')
pd.merge(df_esq, df_dir, on='coluna_em_comum', how='right')
pd.merge(df_esq, df_dir, on='coluna_em_comum', how='outer')
pd.merge(df_esq, df_dir, on='coluna_em_comum', how='inner')
```

## Documentações

19

- » Todas os tipos de possibilidades de união de dados, por si só, já renderia um curso.
- » Existe uma extensa documentação sobre os usos e especificidades das funções pd.concat() e pd.merge(). Estudar essas documentações te fará um melhor analista de dados.
- » Material do Pydata: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/merging.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html)

# Análise de dados com Python

---

## *Exercícios –* Unindo dataframes

20

1. Crie um dataframe para cada um dos arquivos nba\_2015\_a.csv, nba\_2015\_b.csv, nba\_2015\_c.csv, e bust\_nba\_2015.csv. Chame esses dataframes de df\_a, df\_b, df\_c, bust, respectivamente.
2. Visualize o head() de cada um dos dataframes.
3. Concatene os arquivos df\_a, df\_b e df\_c usando a função concat() usando os índices. Salve um dataframe chamado df\_total.
4. Faça a concatenação do dataframe df\_total com o dataframe bust utilizando a função merge() e a variável ID.
5. Busque a documentação das funções concat() e merge() e veja que outros parâmetros podem ser utilizados.

---

## Agrupando os dados: groupby

21

- » Diversas vezes precisamos analisar os dados agrupados, ao invés de um a um, para entender o comportamento do todo.
- » Podemos querer saber a soma de valores, a frequência com que eles ocorrem, as médias agrupadas, dentre outras operações.
- » No Pandas temos uma função excelente para isso, a função pd.groupby(), que agrupa os dados, calcula algumas propriedades dos grupos formados e sumariza os resultados.

# Análise de dados com Python

## Agrupando os dados: groupby

22

» Para apenas separarmos os dados de acordo com uma variável:

```
df.groupby('emprestimo')
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000018DC4F17E80>

df.groupby('emprestimo').groups
{'no': Int64Index([    0,     1,     3,     4,     5,     7,     8,     9,     1
0,
11,
...
45209, 45210, 45211, 45213, 45214, 45215, 45216, 45217, 45218,
45219],
dtype='int64', length=37975),
'yes': Int64Index([    2,     6,    20,    22,    24,    27,    29,    30,
32,
54,
...
45074, 45103, 45108, 45122, 45151, 45153, 45194, 45205, 45212,
45220],
dtype='int64', length=7246)}
```

Agrupando pela variável emprestimo

Visualizando os grupos

## Agrupando os dados: groupby

23

» Para apenas separarmos os dados de acordo com múltiplas variáveis, usando uma lista de variáveis:

```
df.groupby(['emprestimo','profissao'])
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000018DC00388D0>

df.groupby(['emprestimo','profissao']).groups
{('no',
 'admin'): Int64Index([    0,     1,     16,     25,     38,     39,     45,
3,
60,
80,
...
45142, 45144, 45147, 45162, 45167, 45171, 45173, 45176, 45177,
45202],
dtype='int64', length=4180),
('no',
 'blue-collar'): Int64Index([    3,     17,     33,     36,     42,     50,
58,
62,
64,
...
45100, 45124, 45127, 45135, 45174, 45178, 45181, 45190, 45199,
45209],
dtype='int64', length=8048),
('no',
 'entrepreneur'): Int64Index([    7,     94,    172,    222,    232,    241,
264,
273,
357,
...
])}
```

Agrupando pelas variáveis  
emprestimo e profissao.

# Análise de dados com Python

## Agrupando os dados: groupby

24

» Podemos ainda aplicar funções sobre os agrupamentos

| df.groupby('emprestimo').mean() |           |             |           |            |          |           |          | Média dos valores agrupados |  |
|---------------------------------|-----------|-------------|-----------|------------|----------|-----------|----------|-----------------------------|--|
|                                 | idade     | balance     | dia       | duracao    | campanha | pdays     | previous |                             |  |
| <b>emprestimo</b>               |           |             |           |            |          |           |          |                             |  |
| no                              | 41.011244 | 1474.428993 | 15.764450 | 259.587868 | 2.750125 | 41.204687 | 0.592153 |                             |  |
| yes                             | 40.555479 | 774.139387  | 16.020563 | 250.864163 | 2.834391 | 35.024427 | 0.522219 |                             |  |

| df.groupby('emprestimo').count() |       |           |              |          |         |         |         |         |       | Contagem dos valores agrupados |  |
|----------------------------------|-------|-----------|--------------|----------|---------|---------|---------|---------|-------|--------------------------------|--|
|                                  | idade | profissao | estado civil | educacao | default | balance | housing | contato | dia   |                                |  |
| <b>emprestimo</b>                |       |           |              |          |         |         |         |         |       |                                |  |
| no                               | 37975 | 37975     | 37975        | 37975    | 37975   | 37975   | 37975   | 37910   | 37975 |                                |  |
| yes                              | 7246  | 7246      | 7246         | 7246     | 7246    | 7246    | 7246    | 7235    | 7246  |                                |  |

## Agrupando os dados: groupby

25

» Podemos ainda aplicar funções sobre os agrupamentos

| df.groupby(['emprestimo','estado civil']).mean() |          |           |             |           |            |          |           | Média dos valores agrupados |  |
|--|----------|-----------|-------------|-----------|------------|----------|-----------|-----------------------------|--|
|  | idade    | balance   | dia         | duracao   | campanha   | pdays    | previous  |                             |  |
| <b>emprestimo</b>                                |          |           |             |           |            |          |           |                             |  |
| no   | divorced | 46.012133 | 1278.097993 | 15.688521 | 262.818585 | 2.602193 | 43.178721 | 0                           |  |
|  | married  | 43.663446 | 1548.183990 | 15.810647 | 255.551408 | 2.838305 | 38.417136 | 0                           |  |
|  | single   | 33.708034 | 1400.515277 | 15.700036 | 266.525791 | 2.628325 | 46.095884 | 0                           |  |
| yes  | divorced | 44.716612 | 717.111835  | 16.298588 | 261.144408 | 2.764387 | 30.870793 | 0                           |  |
|  | married  | 42.185368 | 834.025531  | 16.061360 | 243.327039 | 2.863120 | 35.792534 | 0                           |  |
|  | single   | 33.686899 | 637.957332  | 15.752404 | 266.291040 | 2.792668 | 35.171875 | 0                           |  |

# Análise de dados com Python

## Agrupando os dados: Groupby e aggregation

26

» Podemos ainda querer mais do que apenas uma informação dos grupos, então podemos usar a função agg().

|            |     | df.groupby(['emprestimo','housing']).agg(['mean','sum']) |        |             |          |           |        |          |     |
|------------|-----|--|--------|-------------|----------|-----------|--------|----------|-----|
|            |     | idade  |        |             |          | balance   |        |          |     |
|            |     | mean   | sum    | mean        | sum      | mean      | sum    | mean     | sum |
| emprestimo | no  | 43.264280  | 744535 | 1737.451566 | 29899804 | 16.061538 | 276403 | 237.9042 |     |
|            | yes | 39.144130  | 812867 | 1256.459453 | 26091637 | 15.510251 | 322252 | 260.9834 |     |
| housing    | no  | 42.417449  | 122035 | 752.713243  | 2165556  | 16.093848 | 46302  | 249.4928 |     |
|            | yes | 39.329366  | 171830 | 788.248569  | 3443858  | 15.972305 | 69783  | 251.7665 |     |

|            |     | df.groupby(['emprestimo','housing']).agg(['count','mean']) |           |       |             |         |           |       |          |
|------------|-----|--|-----------|-------|-------------|---------|-----------|-------|----------|
|            |     | idade  |           |       |             | balance |           |       |          |
|            |     | count  | mean      | count | mean        | count   | mean      | count | mean     |
| emprestimo | no  | 17209  | 43.264280 | 17209 | 1737.451566 | 17209   | 16.061538 | 17207 | 237.90   |
|            | yes | 20766  | 39.144130 | 20766 | 1256.459453 | 20766   | 15.510251 | 20759 | 260.98   |
| housing    | no  | 2877   | 42.417449 | 2877  | 752.713243  | 2877    | 16.093848 | 2875  | 249.49   |
|            | yes | 14369  | 39.329366 | 14369 | 788.248569  | 14369   | 15.972305 | 14367 | 251.7665 |

## Agrupando os dados: Groupby e aggregation

27

|          |       | df.groupby(['emprestimo','housing']).agg(['count','mean']).T |              |             |             |     |
|----------|-------|--|--------------|-------------|-------------|-----|
|          |       | emprestimo   |              | no          |             | yes |
|          |       | housing  | no           | yes         | no          | yes |
| idade    | count | 17209.000000   | 20766.000000 | 2877.000000 | 4369.000000 |     |
|          | mean  | 43.264280  | 39.144130    | 42.417449   | 39.329366   |     |
| balance  | count | 17209.000000   | 20766.000000 | 2877.000000 | 4369.000000 |     |
|          | mean  | 1737.451566  | 1256.459453  | 752.713243  | 788.248569  |     |
| dia      | count | 17209.000000   | 20766.000000 | 2877.000000 | 4369.000000 |     |
|          | mean  | 16.061538  | 15.510251    | 16.093848   | 15.972305   |     |
| duracao  | count | 17207.000000   | 20759.000000 | 2875.000000 | 4369.000000 |     |
|          | mean  | 257.904225   | 260.983429   | 249.492870  | 251.766537  |     |
| campanha | count | 17209.000000   | 20766.000000 | 2877.000000 | 4369.000000 |     |
|          | mean  | 2.822767   | 2.689926     | 2.980188    | 2.738384    |     |
| pdays    | count | 17209.000000   | 20766.000000 | 2877.000000 | 4369.000000 |     |
|          | mean  | 27.902144  | 52.228643    | 16.828989   | 47.006180   |     |
| previous | count | 17209.000000   | 20766.000000 | 2877.000000 | 4369.000000 |     |

» Para facilitar a visualização, podemos usar a função transposta T.

# Análise de dados com Python

## Exercícios - Agrupando os dados

28

1. Abra o arquivo preferencias.csv como um dataframe chamado pref.
2. Visualize os 5 primeiras linhas do arquivo.
3. Agrupe os dados pela variável Gender (gênero).
4. Verifique a contagem de itens por cada gênero.
5. Agrupe os dados pelas variáveis Gender e Favorite Color (cor favorita).
6. Quantos itens de gênero E também possuem cor favorita Cool?
7. Agrupe os dados pelas variáveis Gender e Favorite Color e Favorite Beverage (bebida favorita).
8. Verifique a quantidade de itens de gênero M que têm cor preferida Warm e que preferem Beer.

# Data Mining

Parceria:



## Sumário

01

- » O que é Mineração de dados?
- » Principais frameworks de Data Mining
- » Passos básicos da exploração de dados
  - » Conhecendo seus dados
  - » Limpeza e tratamento de dados
  - » Distribuição dos dados
  - » Extração de informações
  - » Respondendo perguntas
  - » Criando conclusões

### O que é Mineração de Dados?

02

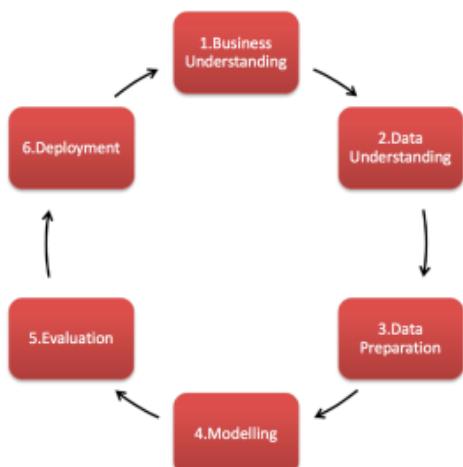
- » Explorar e extrair informações dos dados ~~não~~ é um processo simples. Para ajudar nessa tarefa, existe o processo de mineração de dados.
- » Minerar de dados consiste em analisar uma massa de dados utilizando técnicas e algoritmos para extrair informações, padrões, associações e correlações.
- » Existem diversas metodologias de mineração de dados. Dentre os frameworks de mineração de dados, podemos citar os três mais famosos: CRISP-DM, KDD e SEMMA.

- CRISP-DM: Cross Industry Standard Process for Data Mining
- KDD: Knowledge Discovery in Databases
- SEMMA: Sample, Explore, Modify, Model e Assess

# Análise de dados com Python

## Mineração de Dados – CRISP DM

02

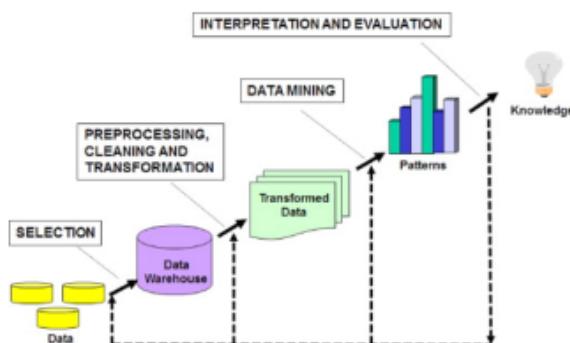


Cross Industry Standard Process for Data Mining

- » Entendimento do negócio
- » Entendimento dos dados
- » Preparação dos dados
- » Modelagem
- » Avaliação
- » Produção

## Mineração de Dados – KDD

03



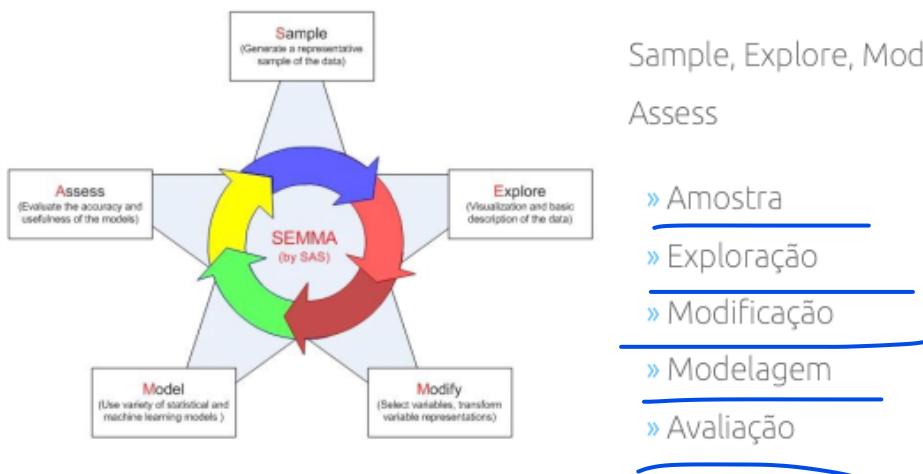
Knowledge Discovery in Databases

- » Seleção
- » Pré-processamento, limpeza e transformação
- » Mineração dos dados
- » Interpretação e avaliação

# Análise de dados com Python

## Mineração de Dados – SEMMA

04



Sample, Explore, Modify, Model e Assess

- » Amostra
- » Exploração
- » Modificação
- » Modelagem
- » Avaliação

## Exploração de dados

05

- » Independente de qual seja o framework escolhido, um dos passos mais importantes é a exploração dos dados (EDA, Exploration Data analysis).
- » Iremos explorar os principais passos a serem realizados para uma boa exploração de dados.



# Análise de dados com Python

## EDA – Primeiros passos

06

» Sempre que temos novos dados, precisamos conhecê-los.

» Sempre que temos novos dados, precisamos conhecê-los.

» Mesmo que haja um metadados, crie uma descrição sua para os dados. Ao Fazê-lo ganhamos mais familiaridade com os dados.

» O Jupyter possibilita a criação de notebooks com seções em Markdown, use-

```
# Abrindo o arquivo .csv  
df = pd.read_csv('carros.csv')
```

### Breve reconhecimento do dataset

Abaixo, descrevemos as informações contidas no dataset **Carros**.

#### Informações contidas no dataset

As variáveis contidas e seus significados (quando obtidos) estão abaixo:

- riskiness: Classificação do risco associado à subscrição de uma nova apólice e o prêmio que deve ser cobrado pela cobertura
- losses: Taxa de desvalorização
- make: Fabricante
- fuel type: tipo de combustível

## EDA

07

Podemos usar as funções head() e describe() para ter uma visualização rápida dos dados e de suas principais estatísticas.

```
# Overview dos dados  
df.head()
```

|   | riskiness | losses | make | fuel type | aspiration | doors | body  | drive | engine location | wheel base | ... | engine size | fue system |
|---|-----------|--------|------|-----------|------------|-------|-------|-------|-----------------|------------|-----|-------------|------------|
| 0 | 2         | 164    | audi | gas       | std        | four  | sedan | fwd   | front           | 99.8       | ... | 109         | mpf        |
| 1 | 2         | 164    | audi | gas       | std        | four  | sedan | 4wd   | front           | 99.4       | ... | 136         | mpf        |
| 2 | 1         | 158    | audi | gas       | std        | four  | sedan | fwd   | front           | 105.8      | ... | 136         | mpf        |
| 3 | 1         | 158    | audi | gas       | turbo      | four  | sedan | fwd   | front           | 105.8      | ... | 131         | mpf        |
| 4 | 2         | 192    | bmw  | gas       | std        | two   | sedan | rwd   | front           | 101.2      | ... | 108         | mpf        |

5 rows × 26 columns

# Análise de dados com Python

## EDA

08

- » Podemos usar as funções head() e describe() para ter uma visualização rápida dos dados e de suas principais estatísticas.

df.describe().T # Descrição básica das variáveis numéricas

|            | count | mean       | std       | min   | 25%    | 50%    | 75%    | max    |
|------------|-------|------------|-----------|-------|--------|--------|--------|--------|
| riskiness  | 159.0 | 0.735849   | 1.193086  | -2.00 | 0.000  | 1.00   | 2.00   | 3.00   |
| losses     | 159.0 | 121.132075 | 35.651285 | 65.00 | 94.000 | 113.00 | 148.00 | 256.00 |
| wheel base | 159.0 | 98.264151  | 5.167416  | 86.60 | 94.500 | 96.90  | 100.80 | 115.60 |

df.describe(include=['O']).T # Descrição básica das variáveis categóricas

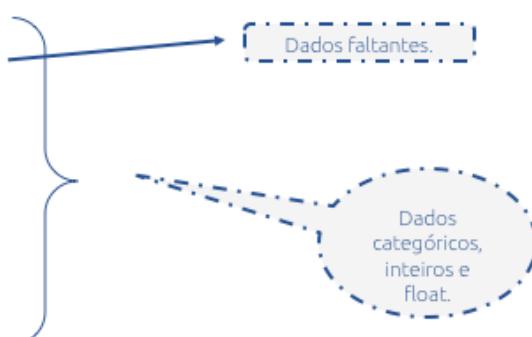
|            | count | unique | top    | freq |
|------------|-------|--------|--------|------|
| make       | 159   | 18     | toyota | 31   |
| fuel type  | 159   | 2      | gas    | 144  |
| aspiration | 159   | 2      | std    | 132  |
| doors      | 159   | 2      | four   | 95   |

## EDA

09

- » A função info() nos informa os tipos de variáveis que temos e já é possível saber se há valores faltantes.

```
df.info()  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 165 entries, 0 to 164  
Data columns (total 26 columns):  
riskiness          165 non-null int64  
losses             165 non-null float64  
make               165 non-null object  
fuel type          165 non-null object  
aspiration         165 non-null object  
doors              165 non-null object  
body               165 non-null object  
drive              165 non-null object  
engine location    165 non-null object  
wheel base         165 non-null float64  
length             165 non-null float64  
width              165 non-null float64  
height             165 non-null float64  
weight              165 non-null int64
```



# Análise de dados com Python

## EDA – Missing Data

10

- » Para visualizar um resumo das variáveis que possui dados faltantes, podemos usar a função `isnull()` (ou a função `isna()`) e a função `sum()`.

```
df.isnull().sum()
```

|                 |   |
|-----------------|---|
| riskiness       | 0 |
| losses          | 6 |
| make            | 0 |
| fuel type       | 0 |
| city mpg        | 1 |
| highway mpg     | 6 |
| aspiration      | 0 |
| doors           | 0 |
| body            | 0 |
| drive           | 0 |
| engine location | 0 |
| wheel base      | 0 |
| length          | 0 |
| width           | 0 |
| height          | 0 |
| wheelbase       | 0 |

```
df.isna().sum()
```

|                 |   |
|-----------------|---|
| riskiness       | 0 |
| losses          | 6 |
| make            | 0 |
| fuel type       | 0 |
| city mpg        | 1 |
| highway mpg     | 6 |
| aspiration      | 0 |
| doors           | 0 |
| body            | 0 |
| drive           | 0 |
| engine location | 0 |
| wheel base      | 0 |
| length          | 0 |
| width           | 0 |
| height          | 0 |
| wheelbase       | 0 |

## EDA – Missing Data

11

- » Caso sejam poucos dados e eles não afetem os demais dados, podemos fazer a exclusão das linhas com dados faltantes com a função `dropna()` e o subconjunto com as variáveis que queremos excluir.

```
df.dropna(subset=['losses','city mpg','highway mpg'], inplace=True)  
df.isna().sum()
```

|                 |   |
|-----------------|---|
| riskiness       | 0 |
| losses          | 0 |
| make            | 0 |
| fuel type       | 0 |
| city mpg        | 0 |
| highway mpg     | 0 |
| aspiration      | 0 |
| doors           | 0 |
| body            | 0 |
| drive           | 0 |
| engine location | 0 |
| wheel base      | 0 |
| length          | 0 |

Agora sem  
dados faltantes.

# Análise de dados com Python

## EDA – Missing Data

12

- » Caso seja necessário, há também a possibilidade substituí-los por outros dados utilizando a função fillna().

```
df['losses'].fillna(0, inplace=True) Substituídos por 0
df['city mpg'].fillna(0, inplace=True) Substituídos por 0
df['highway mpg'].fillna(df['highway mpg'].median(), inplace=True) Substituídos pela mediana

df.isna().sum()

riskiness      0
losses         0
make           0
fuel type     0
city mpg      0
highway mpg   0
aspiration    0
doors          0
body           0
drive          0
...
```

## EDA – Duplicidade

13

- » Outro problema recorrente é a duplicidade dos dados. Para verificar a existência de duplicidade, usamos a função duplicated().

```
df[df.duplicated()]
```

|     | riskiness | losses | make      | fuel type | city mpg | highway mpg | aspiration | doors | body      | drive | ... | engine type |
|-----|-----------|--------|-----------|-----------|----------|-------------|------------|-------|-----------|-------|-----|-------------|
| 159 | 1         | 158.0  | audi      | gas       | 17.0     | 20.0        | turbo      | four  | sedan     | fwd   | ... | ohc         |
| 160 | 2         | 192.0  | bmw       | gas       | 23.0     | 29.0        | std        | two   | sedan     | rwd   | ... | ohc         |
| 162 | 0         | 188.0  | bmw       | gas       | 21.0     | 28.0        | std        | two   | sedan     | rwd   | ... | ohc         |
| 163 | 0         | 188.0  | bmw       | gas       | 21.0     | 28.0        | std        | four  | sedan     | rwd   | ... | ohc         |
| 164 | 2         | 121.0  | chevrolet | gas       | 47.0     | 53.0        | std        | two   | hatchback | fwd   | ... | i           |

5 rows × 26 columns

# Análise de dados com Python

14

## EDA – Duplicidade

» Para eliminar os dados duplicados podemos utilizar a função `drop_duplicated()`.

```
df.drop_duplicates(inplace=True)
```

```
df[df.duplicated()]
```

```
riskiness losses make fuel city highway aspiration doors body drive ... engine cylinder
type mpg mpg
0 rows × 26 columns
```

15

## Exercícios – Limpeza de Dados

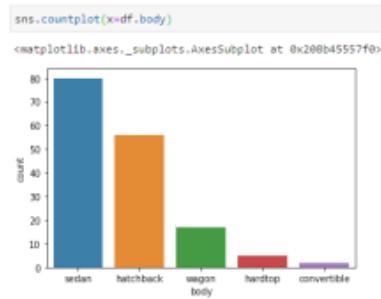
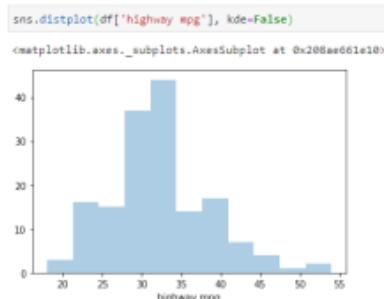
1. Crie um dataframe com o arquivo carros.csv
2. Faça uma primeira visualização dos dados.
3. Faça o describe das variáveis numéricas.
4. Faça o describe das variáveis categóricas.
5. Verifique se há dados nulos ou faltantes. Caso haja, substitua pela média das variáveis.
6. Verifique se há dados duplicados. Caso haja, elimine esses dados duplicados do dataframe.

# Análise de dados com Python

16

## EDA – Entendendo as variáveis

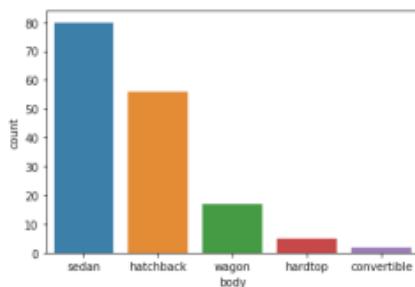
- » Uma vez feita a limpeza dos dados, precisamos agora conhecê-los.
- » Uma das primeiras coisas que podemos fazer é analisar a distribuição de suas variáveis. Para dados numéricos podemos utilizar o histograma, já para dados categóricos podemos usar o gráfico de countplot() do Seaborn.



## EDA – Entendendo as variáveis

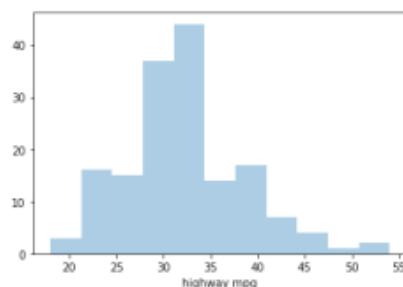
17

- » Sempre descreva quais as principais informações extraídas dos gráficos, mesmo que pareça simples. Isso vai te ajudar nas conclusões.



Distribuição da variável body

Nota-se que a maior parte dos veículos são do tipo sedan e hatchback, com uma proporção pequena de veículos conversíveis.



Distribuição da variável highway mpg

A distribuição da variável "highway mpg" se estende desde ~ 17 até ~ 54, com pico em torno de 32 (condizente com a média de 32, obtida anteriormente).

# Análise de dados com Python

## Exercícios –Entendendo as variáveis

17

1. Use o dataframe anterior.
2. Faça histogramas das variáveis numéricas.
3. Escreva pequenas conclusões sobre os histogramas.
4. Faça countplots das variáveis categóricas.
5. Escreva pequenas conclusões sobre os gráficos.

## EDA – Algumas estatísticas

18

- » A função `describe()` exibe algumas estatísticas básicas, mas iremos analisá-las melhor.
- » A média amostral e a média populacional de uma variável são dadas pelas equações

$$\bar{x} = \sum_i^n \frac{x_i}{n-1} \quad \bar{x} = \sum_i^n \frac{x_i}{n}$$

- » Podemos usar a função `df.mean()` para calcular a média de um conjunto de dados.

```
df['highway_mpg'].mean()
```

```
32.1
```

```
df['horsepower'].mean()
```

```
95.86875
```

População: Um conjunto de pessoas, itens ou eventos sobre os quais podemos fazer inferências  
Amostra: Um subconjunto de uma população, sobre o qual podemos fazer inferências.

# Análise de dados com Python

## EDA – Algumas estatísticas

19

- » Por outro lado, a mediana é o valor do meio de um conjunto de dados ordenados.
- » A mediana é o valor que está na posição  $\frac{n+1}{2}$ .
- » Podemos usar a função `df.median()` para obter os valores medianos de uma variável.

```
df['highway mpg'].median()  
32.0  
  
df['horsepower'].median()  
88.0
```

## EDA – Algumas estatísticas

20

- » Ao contrário da média, a mediana é menos sensível a valores discrepantes pois é uma medida de posição.
- » Vejamos um exemplo simples:

```
dt  
0  
0 2  
1 3  
2 2  
3 4  
4 3  
5 2  
6 15  
  
dt.mean()  
0 4.428571  
dtype: float64  
  
dt.median()  
0 3.0  
dtype: float64
```

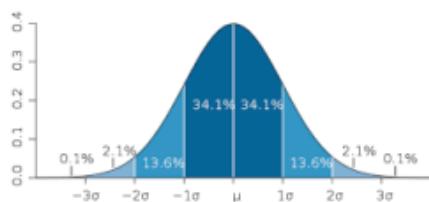
» Vemos que, devido a um valor discrepante, o valor médio é menos fiel ao conjunto de dados do que o valor mediano.

# Análise de dados com Python

## EDA – Algumas estatísticas

21

- » O Desvio Padrão é uma medida muito importante, visto que nos diz o quanto um certo valor está distante da média do conjunto de dados, sob a seguinte equação:



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

- » Podemos usar a função `std()` para calcular o desvio padrão de uma variável.

```
df['horsepower'].std()
```

30.62455172307006

```
df['wheel base'].std()
```

## EDA – Algumas estatísticas

22

- » A moda também é uma medida de posição, em que buscamos o valor que possui maior ocorrência entre os dados.

- » Podemos usar a função `mode()` para calcular o desvio padrão de uma variável.

```
df['doors'].mode()
```

0 four  
dtype: object

```
df['body'].mode()
```

0 sedan  
dtype: object

```
df['stroke'].mode()
```

0 3.03  
1 3.15  
dtype: float64

Perceba que nesse caso temos mais de uma moda.

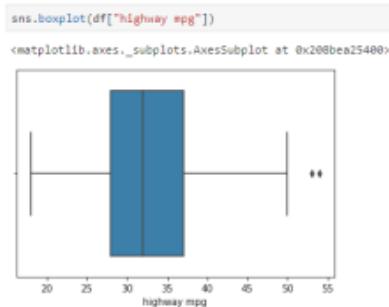
QUE?

# Análise de dados com Python

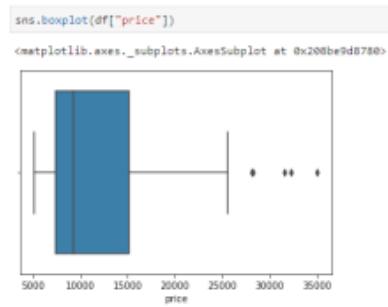
## EDA – Entendendo as variáveis

23

- » O gráfico de boxplot é uma boa ferramenta para analisarmos como as medidas centrais estão distribuídas nos dados.



A variável *highway mpg* possui mediana também próxima à média de 32 e bem centralizada. Apesar de ter uns poucos outliers, a variável aparente ter uma distribuição normal.



Já a variável *price* mostra que os valores dessa variável não estão normalmente distribuídos, exibindo ainda outliers que podem enviesar os valores médios da variável.

## Exercícios – Algumas estatísticas

24

1. Qual a média da variável *losses*? Qual a mediana da variável *highway mpg*?
2. Qual a média da variável *price*? Qual a mediana da variável *price*? Discuta os valores encontrados.
3. Qual a moda da variável *fuel type*? Qual a moda da variável *make*?
4. Calcule a moda das variáveis *horsepower* e *price*.
5. Escolha 3 variáveis numéricas e faça seus boxplots. Que informações é possível obter?

# Análise de dados com Python

25

## EDA – Correlações

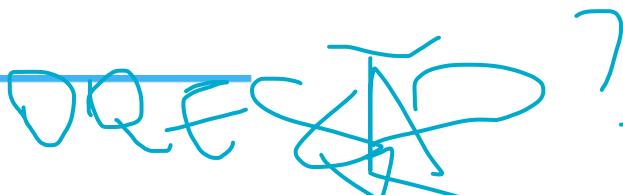
» Agora que já temos mais conhecimento sobre as variáveis isoladamente, podemos tentar descobrir se existe algum tipo de correlação entre elas.

» Para isso, podemos usar a função corr() e obter os coeficientes de correlação entre as variáveis.

|             | riskiness | losses    | city mpg  | highway mpg | wheel base | length    |
|-------------|-----------|-----------|-----------|-------------|------------|-----------|
| riskiness   | 1.000000  | 0.470646  | 0.129707  | 0.183753    | -0.480394  | -0.371223 |
| losses      | 0.470646  | 1.000000  | -0.215598 | -0.201539   | -0.064884  | 0.051280  |
| city mpg    | 0.129707  | -0.215598 | 1.000000  | 0.925434    | -0.628569  | -0.743179 |
| highway mpg | 0.183753  | -0.201539 | 0.925434  | 1.000000    | -0.646068  | -0.758012 |
| wheel base  | -0.480394 | -0.064884 | -0.628569 | -0.646068   | 1.000000   | 0.918351  |
| length      | -0.371223 | 0.051280  | -0.743179 | -0.758012   | 0.918351   | 1.000000  |
| width       | -0.227257 | 0.111442  | -0.711178 | -0.714865   | 0.808158   | 0.870616  |
| height      | -0.446075 | -0.304963 | -0.232409 | -0.261901   | 0.619922   | 0.532515  |
| weight      | -0.301082 | 0.114322  | -0.834483 | -0.844491   | 0.821019   | 0.905314  |
| engine size | -0.218965 | 0.139953  | -0.796852 | -0.788991   | 0.764331   | 0.843950  |
| bore        | -0.282872 | -0.028567 | -0.634570 | -0.636331   | 0.636052   | 0.696238  |
| stroke      | -0.000251 | 0.063143  | -0.042835 | -0.026834   | 0.192567   | 0.147062  |

## EDA – Correlações

26



» A função corr() permite escolher entre três métodos de correlação: Pearson, Spearman e Kendall.

» Busque sempre informações sobre qual o método melhor se adequa aos seus dados.

» Ainda que muito útil, quanto maior a quantidade de variáveis, mais inviável é enxergar quais as variáveis que possuem maior ou menor correlação.

» Para isso podemos usar gráficos do tipo heatmap para visualizar melhor as variáveis que aparentam possuir dependência.

# Análise de dados com Python

→ NÃO  
ENTENDI

EDA – Correlações

27

- » Com o heatmap já é possível, por exemplo, notar que a variável horsepower possui uma anticorrelação com a variável highway mpg.
  - » Também podemos notar que as variáveis length e losses praticamente não possuem qualquer correlação.



## EDA – Correlações

28

1. Use a função `corr()` e faça a correlação entre as variáveis do dataframe utilizado anteriormente.
  2. Quais as variáveis com maior correlação?
  3. Quais as variáveis com maior anticorrelação?
  4. Escolha 2 correlações e 2 anticorrelações e tentem argumentar o porquê dessa relação.

# Análise de dados com Python

29

## EDA – Estratificações

» Estratificar significa dividir seus dados em camadas, subgrupos de acordo com alguma estratégia relevante.

» A estratificação busca separar os dados homogêneos em grupos que sejam os mais distintos possíveis.

» Por exemplo, podemos estratificar dados conforme:

- » Sexo;
- » Idade;
- » Região;
- » Perfil Socioeconômico;
- » Qualquer outra característica inerente ao problema a ser resolvido.

30

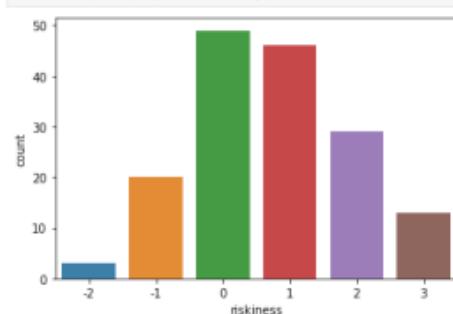
## EDA – Estratificações

» Em nosso caso, por exemplo, podemos estratificar os carros por algumas variáveis:

- » body
- » riskiness
- » doors
- » etc.

» Iremos estratificar nossos dados de acordo com o riskiness, que apresenta 6 categorias diferentes de risco.

`sns.countplot(df["riskiness"])`

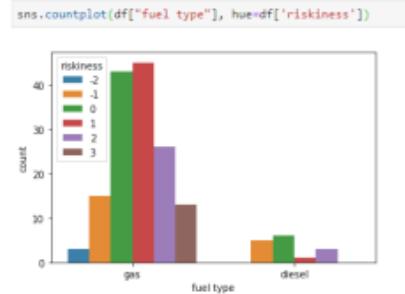
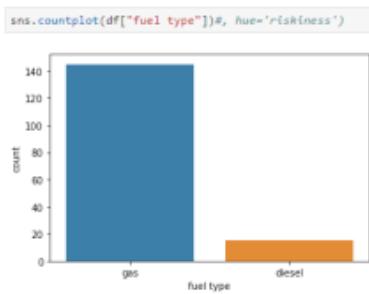


# Análise de dados com Python

31

## EDA – Estratificações

» Observando as demais variáveis estratificadas pela variável riskiness, podemos notar diversas características interessantes. Vejamos o caso da variável fuel type:

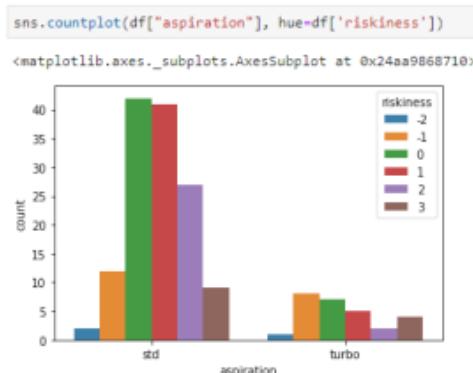
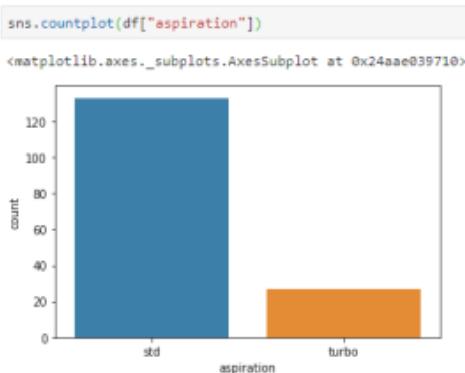


» Sem estratificar não poderíamos saber, por exemplo, que os dados possuem uma distribuição aproximadamente gaussiana para carros com combustível do tipo gas, enquanto que os carros com combustível do tipo diesel tendem a ter mais dados em risco 0 e -1.

32

## EDA – Estratificações

» O mesmo pode ser feito com relação à variável aspiration.  
» Carros com aspiration do tipo turbo tendem a ter riscos mais negativos, evidenciando comportamento distinto dos carros com aspiration padrão (std).



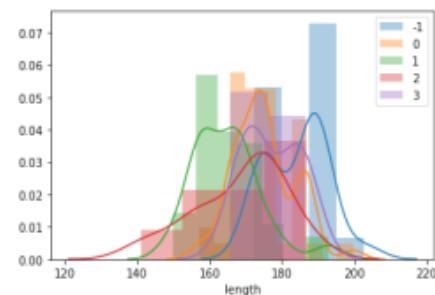
# Análise de dados com Python

33

## EDA – Estratificações

- » Analisando agora uma variável contínua, lenght, também notamos que existem diferenças nas distribuições da variável dependendo da classe de riskiness que o carro possuir.

```
#sns.distplot(df[df.riskiness == -2].length)
sns.distplot(df[df.riskiness == -1].length,label='-1')
sns.distplot(df[df.riskiness == 0].length,label='0')
sns.distplot(df[df.riskiness == 1].length,label='1')
sns.distplot(df[df.riskiness == 2].length,label='2')
sns.distplot(df[df.riskiness == 3].length,label='3')
plt.legend()
```



- » Carros com classe de riskiness igual a -1 possui distribuição bimodal com picos em lenght em torno de 175 e 190, enquanto que a classe de riskiness igual a 2 possui maior quantidade de carros com lenght em torno de 175.

34

## EDA – Estratificações

1. Vamos analisar os dados sob a perspectiva da quantidade de portas.
2. Faça gráficos de countplot simples para as variáveis riskiness e body.
3. Agora refaça os gráficos anteriores, mas estratificando pela variável doors. (Dica: coloque a variável estratificadora no parâmetro hue.)
4. O que podemos concluir a partir desses gráficos?
5. Faça o countplot da variável aspiration estratificado pela variável doors.
6. O que podemos concluir?
7. Comparando os resultados, qual deles foi mais fácil de obter?

# Análise de dados com Python

## EDA – Hipóteses

35

- » Após conhecer os dados e entender como eles se correlacionam, temos base para criar e testar algumas hipóteses.
- » Podemos criar diversas hipóteses para um mesmo conjunto de dados.
- » Para facilitar, comece sempre com as hipóteses mais simples.
- » Conforme as hipóteses mais simples forem testadas, passe a fazer hipóteses mais complexas.
- » Criar e testar hipóteses é valioso e quanto mais treinarmos nosso pensamento científico, melhores hipóteses faremos.

## EDA – Hipóteses

36

- » Exemplos de hipóteses que podemos testar com nossos dados:
  - » Carros com aspiration do tipo turbo tendem a ter riskiness igual a -2 pois são carros que possuem carroceria do tipo hatch.
  - » Carros com maior consumo de combustível na cidade tendem a ter menores riskiness.
  - » Carros com 4 portas são mais compridos do que aqueles com 2 portas.
- » Exemplos de hipóteses que poderíamos fazer mas que não podemos responder com os nossos dados:
  - » Carros dirigidos por mulheres têm menores riskiness do que aqueles dirigidos por homens.
  - » Carros mais velhos tendem a consumir mais combustível.

# Análise de dados com Python

## Exercícios – Hipóteses

37

1. Crie mais 2 outras hipóteses que poderiam ser feitas com base nos dados sobre carros.
2. Explique em que situações essas hipóteses poderiam ser utilizadas.
3. Cite 2 hipóteses que não poderiam ser testadas com os dados sobre carros.
4. Que outros dados seriam necessários para testá-las?

## EDA – Conclusões

38

» Ainda que você tenha feito diversas conclusões parciais ao longo da análise exploratória, é sempre bom fazer um apanhado geral, suas conclusões finais.

- » Retome suas hipóteses e discuta os resultados.
- » Não deixe de apresentar uma hipótese que não se confirmou, pois também é um resultado relevante.
- » Apresente possibilidades de uso para os resultados encontrados.

# Data Visualization

Parceria:



# Análise de dados com Python

## Sumário

01

- » Quarteto de Anscombe e a importância da visualização de dados
- » Um pouco sobre Information Design
- » Matplotlib
- » Seaborn
- » Principais tipos de gráficos

## Quarteto de Anscombe

02

- » 4 conjuntos de dados: o que podemos inferir?

|      | I     | II   | III  | IV   |       |      |       |
|------|-------|------|------|------|-------|------|-------|
| x    | y     | x    | y    | x    | y     | x    | y     |
| 10.0 | 8.04  | 10.0 | 9.14 | 10.0 | 7.46  | 8.0  | 6.58  |
| 8.0  | 6.95  | 8.0  | 8.14 | 8.0  | 6.77  | 8.0  | 5.76  |
| 13.0 | 7.58  | 13.0 | 8.74 | 13.0 | 12.74 | 8.0  | 7.71  |
| 9.0  | 8.81  | 9.0  | 8.77 | 9.0  | 7.11  | 8.0  | 8.84  |
| 11.0 | 8.33  | 11.0 | 9.26 | 11.0 | 7.81  | 8.0  | 8.47  |
| 14.0 | 9.96  | 14.0 | 8.10 | 14.0 | 8.84  | 8.0  | 7.04  |
| 6.0  | 7.24  | 6.0  | 6.13 | 6.0  | 6.08  | 8.0  | 5.25  |
| 4.0  | 4.26  | 4.0  | 3.10 | 4.0  | 5.39  | 19.0 | 12.50 |
| 12.0 | 10.84 | 12.0 | 9.13 | 12.0 | 8.15  | 8.0  | 5.56  |
| 7.0  | 4.82  | 7.0  | 7.26 | 7.0  | 6.42  | 8.0  | 7.91  |
| 5.0  | 5.68  | 5.0  | 4.74 | 5.0  | 5.73  | 8.0  | 6.89  |

# Análise de dados com Python

## Quarteto de Anscombe

03

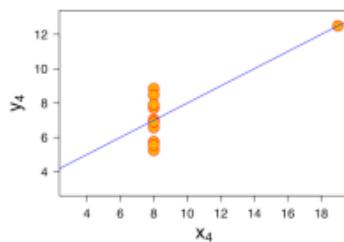
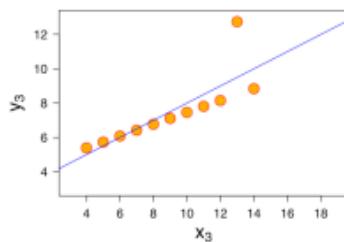
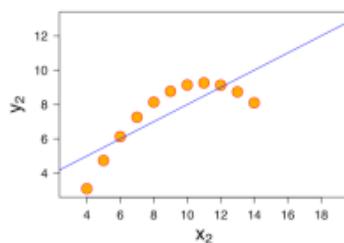
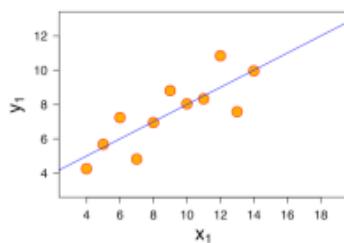
» 4 conjuntos de dados que possuem as mesmas médias, medianas, desvio padrão e correlação.

| I    |       | II   |      | III  |       | IV   |       |
|------|-------|------|------|------|-------|------|-------|
| x    | y     | x    | y    | x    | y     | x    | y     |
| 10.0 | 8.04  | 10.0 | 9.14 | 10.0 | 7.46  | 8.0  | 6.58  |
| 8.0  | 6.95  | 8.0  | 8.14 | 8.0  | 6.77  | 8.0  | 5.76  |
| 13.0 | 7.58  | 13.0 | 8.74 | 13.0 | 12.74 | 8.0  | 7.71  |
| 9.0  | 8.81  | 9.0  | 8.77 | 9.0  | 7.11  | 8.0  | 8.84  |
| 11.0 | 8.33  | 11.0 | 9.26 | 11.0 | 7.81  | 8.0  | 8.47  |
| 14.0 | 9.96  | 14.0 | 8.10 | 14.0 | 8.84  | 8.0  | 7.04  |
| 6.0  | 7.24  | 6.0  | 6.13 | 6.0  | 6.08  | 8.0  | 5.25  |
| 4.0  | 4.26  | 4.0  | 3.10 | 4.0  | 5.39  | 19.0 | 12.50 |
| 12.0 | 10.84 | 12.0 | 9.13 | 12.0 | 8.15  | 8.0  | 5.56  |
| 7.0  | 4.82  | 7.0  | 7.26 | 7.0  | 6.42  | 8.0  | 7.91  |
| 5.0  | 5.68  | 5.0  | 4.74 | 5.0  | 5.73  | 8.0  | 6.89  |

|             |       |
|-------------|-------|
| média de x  | 9.0   |
| média de y  | 7.5   |
| var x       | 11    |
| var y       | 4.12  |
| correlation | 0.816 |

## Quarteto de Anscombe

04



» O quarteto de Anscombe nos mostra que apenas exibir os dados em um tabela pode não trazer todas as informações sobre os dados.

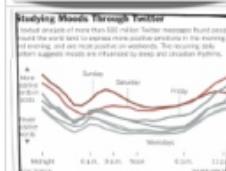
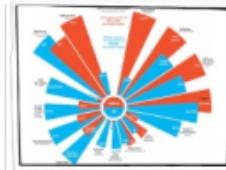
» Visualizar como os dados estão distribuídos e relacionados nos ajudar a extrair ainda mais informações.

# Análise de dados com Python

## Information Design

05

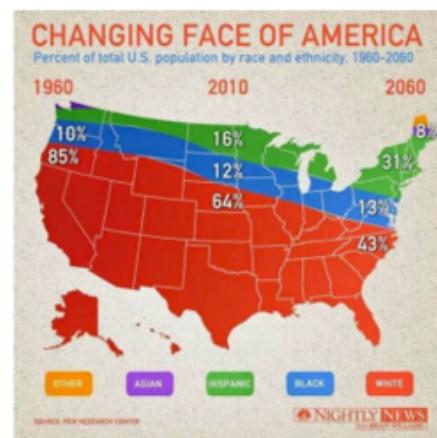
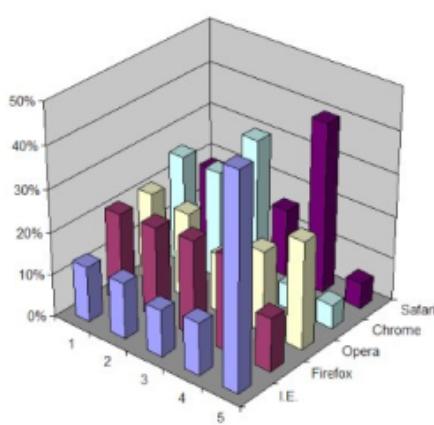
- » Consiste em apresentar dados e informações de uma maneira visual, promovendo um entendimento eficiente e eficaz do que se pretende comunicar.
- » É uma atividade relacionada à seleção, organização e apresentação de informação para uma determinada audiência (Wildbur e Burke, 1998).



## Information Design

06

- » O que podemos extrair dos gráficos?

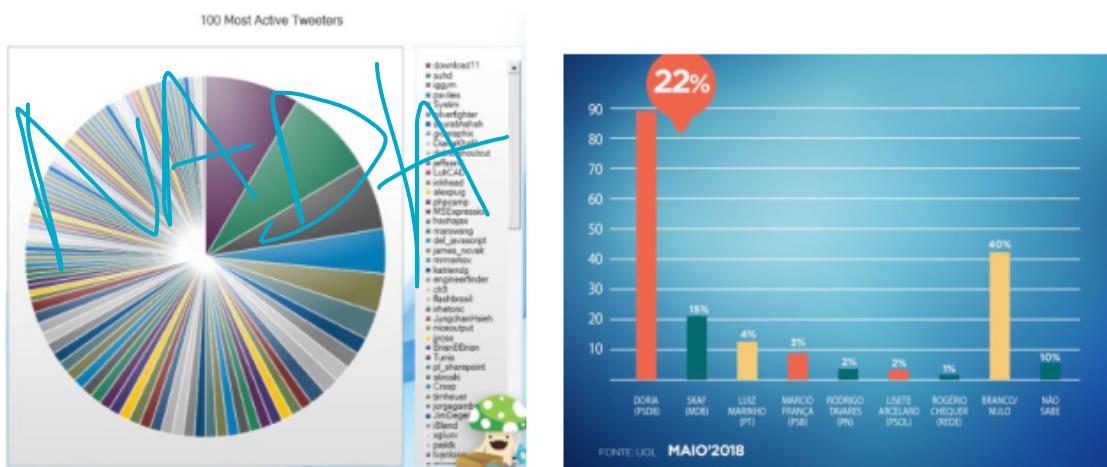


# Análise de dados com Python

## Information Design

07

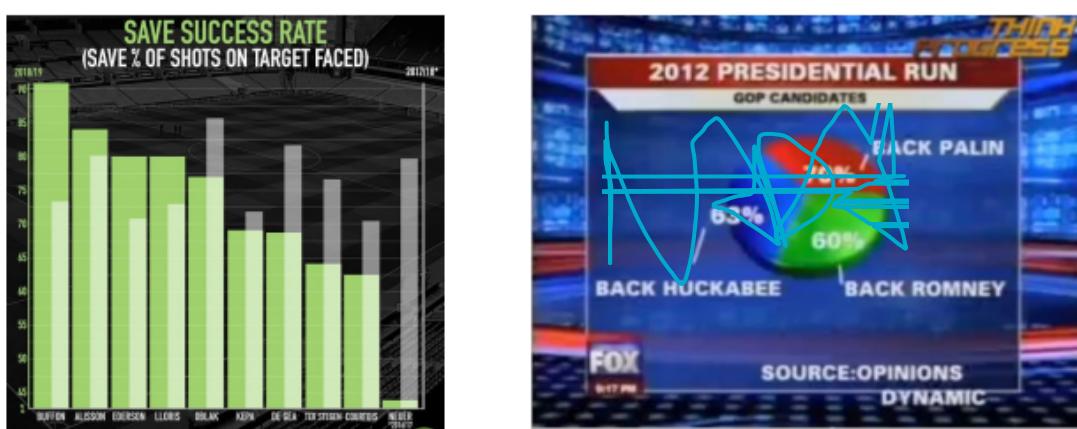
» O que podemos extrair dos gráficos?



## Information Design

08

» O que podemos extrair dos gráficos?

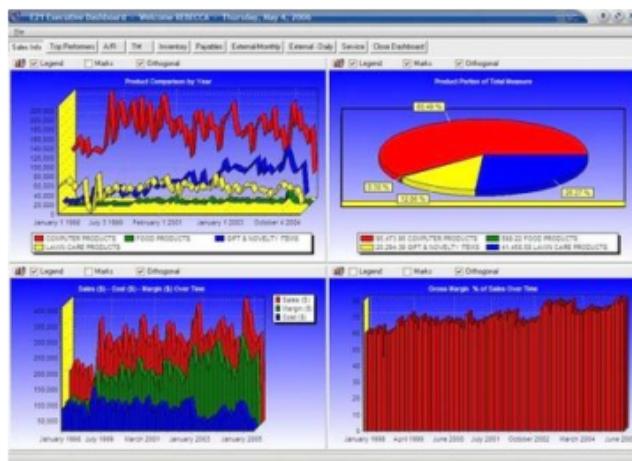


# Análise de dados com Python

## Information Design

09

» O que podemos extrair dos gráficos?



## Information Design

10

» O que podemos extrair dos gráficos?

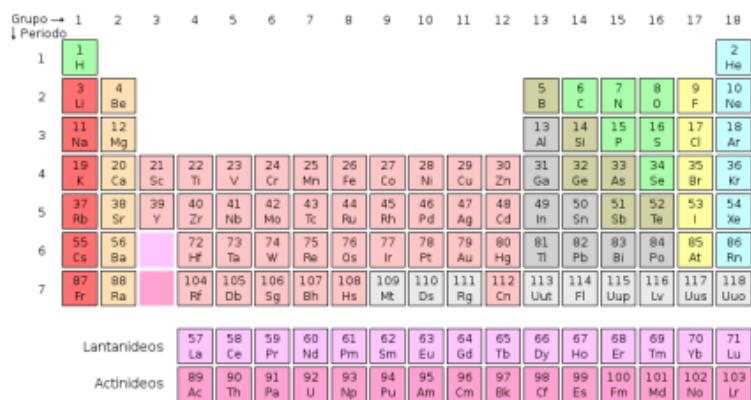


# Análise de dados com Python

11

## Information Design – bons exemplos

» O que podemos extrair dos gráficos?



12

## Information Design – bons exemplos

» O que podemos extrair dos gráficos?

| Month of Year | Sales Amount     | Total Product Ma... | Gross Profit Ma... | Gross Profit      |
|---------------|------------------|---------------------|--------------------|-------------------|
| January       | 1309863.2511     | 1046855.0401        | 0.20079058694...   | 263008.211        |
| February      | 2451605.6244     | 2161789.71439...    | 0.11821473532...   | 289815.910000...  |
| March         | 2099415.6158     | 1781531.84109...    | 0.15141536164...   | 317883.774700...  |
| April         | 1546592.2292     | 1250946.0643        | 0.19115973772...   | 295646.164900...  |
| May           | 2942672.90960... | 2583467.20809...    | 0.12206783170...   | 359205.701500...  |
| June          | 1678567.4193     | 2010739.61289...    | -0.19789029012...  | -332172.193599... |
| July          | 962716.741700... | 754715.7636         | 0.21605623942...   | 208000.978100...  |
| August        | 2044600.0034     | 1771778.75389...    | 0.13343502349...   | 272821.249500...  |
| September     | 1639840.109      | 1393936.67389...    | 0.14995573882...   | 245903.43510001   |
| October       | 1358050.4703     | 1124337.2647        | 0.17209463912...   | 233713.205600...  |
| November      | 2068129.20330... | 2561131.77409...    | 0.10703751729...   | 306997.42920002   |
| December      | 2458472.4342     | 2085375.78659...    | 0.15179954076...   | 373096.647600...  |

2002 Revenue and Profits (in US\$ Thousands)

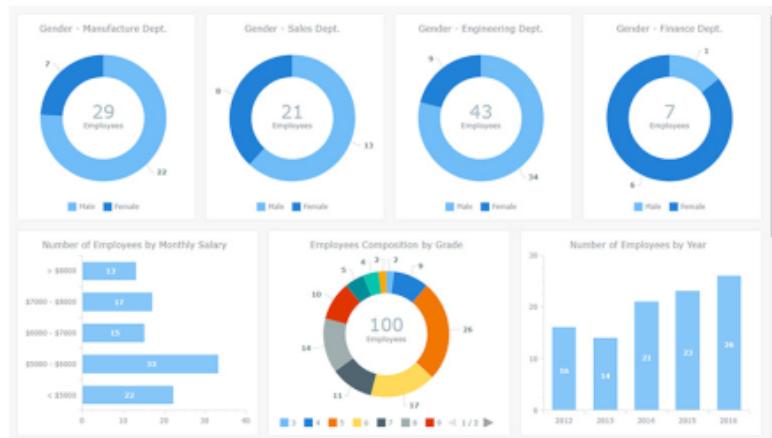


# Análise de dados com Python

13

## Information Design – bons exemplos

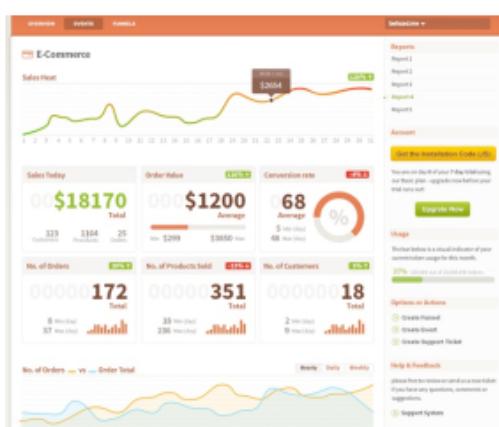
» O que podemos extrair dos gráficos?



14

## Information Design – bons exemplos

» O que podemos extrair dos gráficos?



# Análise de dados com Python

15

## Visualização de dados

Há basicamente 2 tipos de visualização de dados: exploratória e explanatória.

- » Exploratória: quando usamos a visualização para encontrar relações e informações importantes dos dados
- » Explanatória: quando usamos a visualização para explicar/contar uma estória para algum usuário.

16

## Dicas para visualização de dados

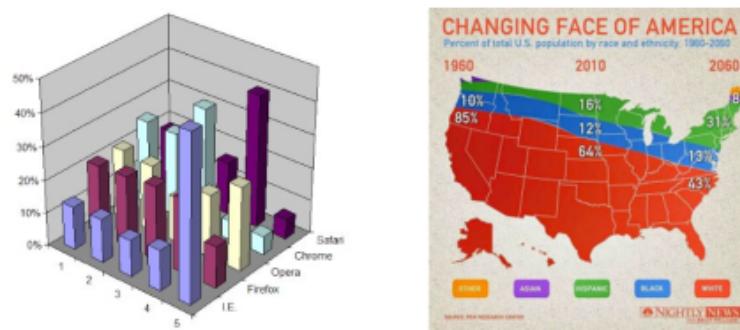
- » Qual seu objetivo?
  - » Qual a informação você quer passar?
  - » Qual o nível de detalhamento?
- » Por que apresentar essa informação?
  - » Quem vai usar essa informação?
  - » Coloque-se no lugar do usuário
- » O que você usará na visualização?
  - » Quais os símbolos?
  - » Quais as cores? *(Cuidado com os daltônicos!)*
  - » Quais os melhores padrões?

# Análise de dados com Python

## Exercícios: information Design

17

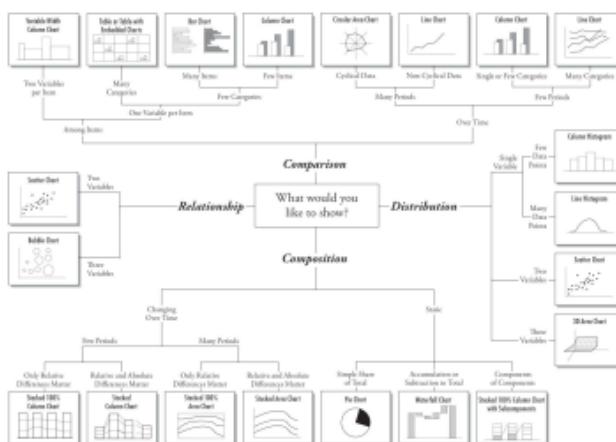
- Como exibir as informações dos gráficos abaixo de maneira mais eficiente? Faça, usando apenas papel e caneta, um esboço de como os dados poderiam ser melhor apresentados.



## Melhor gráfico para o melhor dado

18

PEDIR ~



Fonte: A. Abela, 2006.

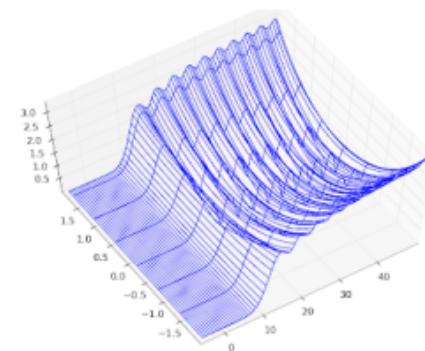
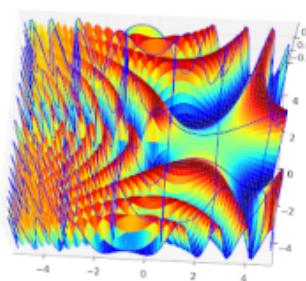
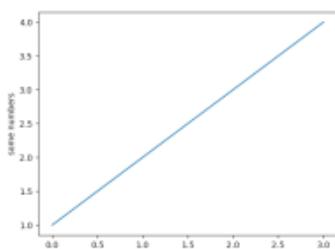
## Gráficos em Python?

- » Existem hoje diversas bibliotecas em Python para criação de gráficos. Dentre as várias possibilidades, exploraremos aqui as duas mais comuns: Matplotlib e Seaborn.
- » Em cada biblioteca, considerando os principais tipos de dados, iremos apresentar alguns dos tipos de gráficos mais comuns.



## Matplotlib – Primeiros passos

- » Matplotlib é a principal biblioteca para geração de gráficos em Python.
- » A biblioteca possui uma infinidade de possibilidades, sendo possível criarmos desde gráficos simples até outros mais rebuscados.



# Análise de dados com Python

21

## Matplotlib – Primeiros passos

» Importando a biblioteca

```
import matplotlib.pyplot as plt
```

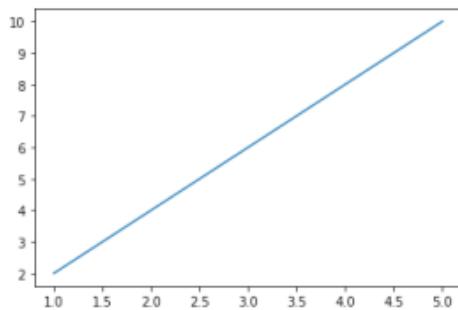
» Criando o primeiro gráfico.

A função `plot()` é uma das mais simples, que permite plotar 2 conjuntos de dados ( $x, y$ )

```
x = np.array([1,2,3,4,5])  
y = np.array([2,4,6,8,10])
```

```
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x190bb1c1588>]
```



## Matplotlib – Primeiros passos

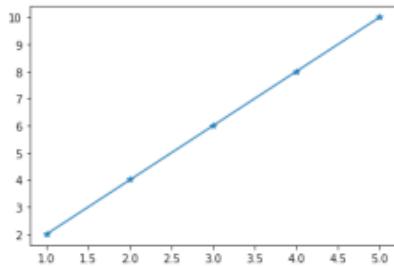
22

» Mudando alguns parâmetros

```
plt.plot(x,y, marker="*")
```

Mudando o tipo de marcador

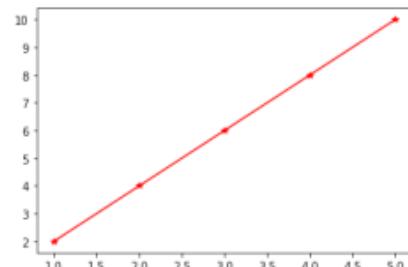
```
[<matplotlib.lines.Line2D at 0x190bc9d4518>]
```



```
plt.plot(x,y, marker="*", c="red")
```

Mudando a cor da linha

```
[<matplotlib.lines.Line2D at 0x190bca555c0>]
```

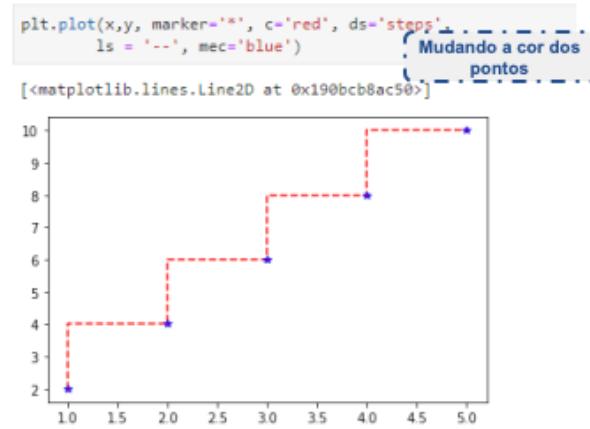
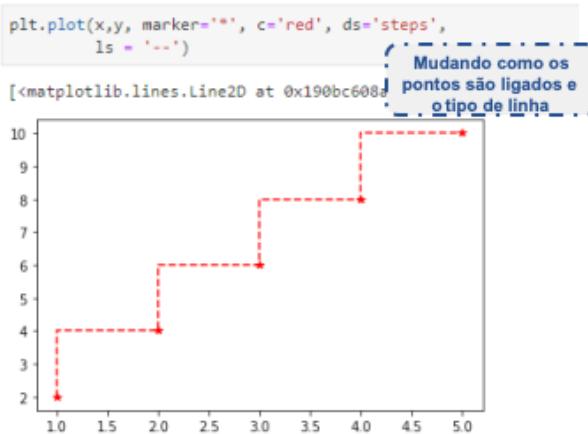


# Análise de dados com Python

23

## Matplotlib – Primeiros passos

- » Mudando alguns parâmetros



## Matplotlib – Primeiros passos

24

- » Existem diversos outros parâmetros (argumentos) que podem ser utilizados para personalizar o gráfico.
- » Pode-se encontrar todos os argumentos nas documentações do Matplotlib.

| Property                                    | Description   |
|---|---|
| <code>agg_filter</code>                     | a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array |
| <code>alpha</code>                          | float   |
| <code>animated</code>                       | bool  |
| <code>antialiased</code> or <code>aa</code> | bool  |
| <code>clip_box</code>                       | Bbox  |
| <code>clip_on</code>                        | bool  |
| <code>clip_path</code>                      | [(Path, Transform)   Patch   None]  |
| <code>color</code> or <code>c</code>        | color   |
| <code>contains</code>                       | callable  |
| <code>dash_capstyle</code>                  | {'butt', 'round', 'projecting'}   |
| <code>dash_joinstyle</code>                 | {'miter', 'round', 'bevel'}   |
| <code>dashes</code>                         | sequence of floats (on/off ink in points) or (None, None)   |
| <code>drawstyle</code> or <code>ds</code>   | {'default', 'steps', 'steps-mid', 'steps-pre',  |

# Análise de dados com Python

25

## Matplotlib – Adicionando outros elementos

```
plt.plot(x,y)
plt.xlabel('Valores de X', fontsize=20)
plt.ylabel('Valores de Y', fontsize=15)
plt.title('Nome do Gráfico', c='purple')
plt.xticks(fontsize=15, c='red')
plt.yticks(fontsize=15, c='green')
```



» Podemos ainda adicionar labels aos eixos, nome ao gráfico, modificar o tamanho das fontes, dentre outras possibilidades.

» Tome cuidado para que seu gráfico não tenha muitos elementos e acabe tirando a atenção do que realmente importa: os dados!

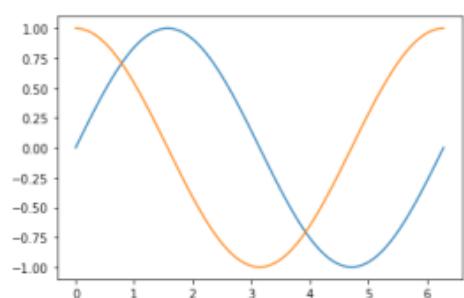
## Múltiplas curvas no mecanismo gráfico

26

```
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
z = np.cos(x)
```

```
plt.plot(x,y)
plt.plot(x,z)
```

[<matplotlib.lines.Line2D at 0x190bec8cb38>]

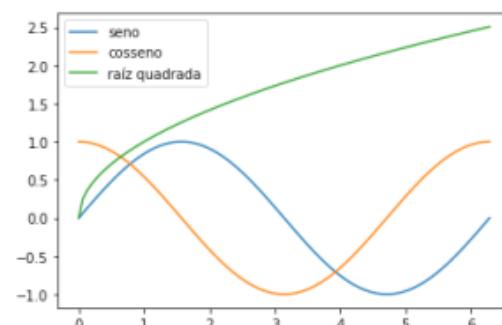


Dois gráficos

```
plt.plot(x,y,label='seno')
plt.plot(x,z,label='cosseno')
plt.plot(x,np.sqrt(x),label='raiz quadrada')
plt.legend()
```

<matplotlib.legend.Legend at 0x190beaa99b0>

Três gráficos e alguns elementos adicionais



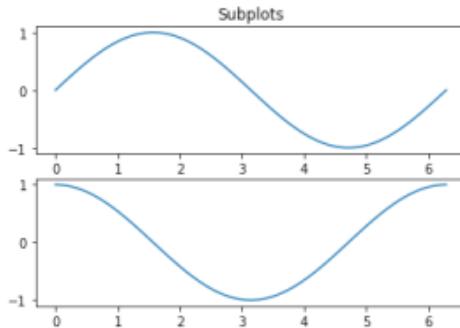
# Análise de dados com Python

27

## Matplotlib – Múltiplos subgráficos

```
plt.subplot(2, 1, 1)
plt.plot(x, y)
plt.title('Subplots')
plt.subplot(2, 1, 2)
plt.plot(x, z)
```

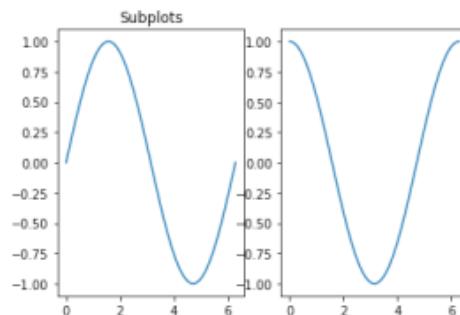
[<matplotlib.lines.Line2D at 0x190c000ce8b>]



2 gráficos na vertical (linha)

```
plt.subplot(1, 2, 1)
plt.plot(x, y)
plt.title('Subplots')
plt.subplot(1, 2, 2)
plt.plot(x, z)
```

[<matplotlib.lines.Line2D at 0x190c01452e8>]



2 gráficos na horizontal (coluna)

## Matplotlib – outros tipos de gráficos

28

- » Existem diversos tipos de gráficos que podem ser feitos com a biblioteca Matplotlib.
- » Alguns serão explorados a seguir (tanto em Matplotlib quanto em Seaborn), mas dentre as possibilidades, podemos citar:
  - » Gráficos lineares
  - » Gráficos de dispersão
  - » Gráficos de barras
  - » Histogramas
  - » Boxplots
  - » Imagens
  - » Tridimensionais
  - » Gráficos de fluxos de campo
  - » Gráficos polares



# Análise de dados com Python

## Exercícios - Matplotlib

29

1. Crie dois arrays x e y com as listas [-3,-2.5,-2,-1,0,1,2,2.5,3] e [-27,-15.62,-8,-1,0,1,8,15.62,27].
2. Use a função `plot()` e faça um gráfico de x e y.
3. Use a função `xlabel()` para adicionar o nome “Valor de X” ao eixo x. Deixe com fonte de tamanho 15.
4. Use a função `ylabel()` para adicionar o nome “Valor de Y” ao eixo y. Deixe com fonte de tamanho 15.
5. Use a função `legend()` para adicionar o nome “Função Cúbica” ao gráfico. Deixe com fonte de tamanho 20.
6. Rode novamente seu gráfico, mas mude agora a linha para a cor magenta.

## Seaborn – Gráficos mais elaborados

30

- » Seaborn é uma biblioteca baseada em Matplotlib que traz uma interface mais atraente para os gráficos.
- » Traz facilidades para modificarmos os parâmetros `default` do Matplotlib
- » Permite usar dataframes
- » Importando a biblioteca:

```
import seaborn as sns
```

# Análise de dados com Python

31

## Seaborn – Datasets embutidos

» O Seaborn possui diversos dados já embutidos que podem ser utilizados para estudar a própria biblioteca. Para sabermos quais os datasets disponíveis, podemos utilizar a função `get_dataset_names()`.

```
sns.get_dataset_names()
```

```
['anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'iris',
 'mpg',
 'planets',
```

32

## Seaborn – Datasets embutidos

» Para acessarmos o dataset escolhido, usamos a função `load_dataset()`.

Criando um dataframe com o dataset `iris`

```
iris = sns.load_dataset('iris')
```

```
iris.head(3)
```

|   | sepal_length | sepal_width | petal_length | petal_width | species |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         | setosa  |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         | setosa  |

Criando um dataframe com o dataset `planets`

```
planetas = sns.load_dataset('planets')
```

```
planetas.head(3)
```

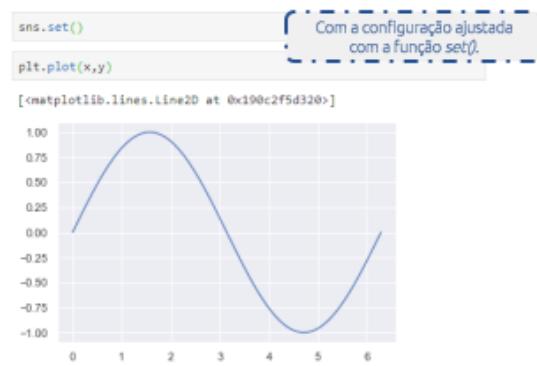
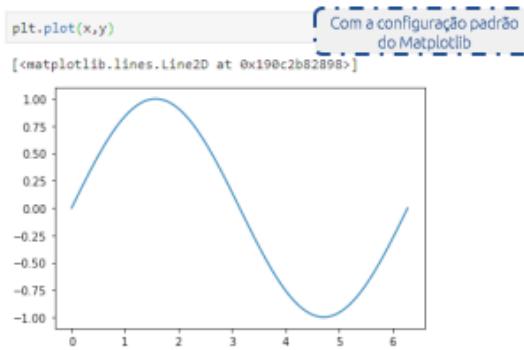
|   | method          | number | orbital_period | mass | distance | year |
|---|-----------------|--------|----------------|------|----------|------|
| 0 | Radial Velocity | 1      | 269.300        | 7.10 | 77.40    | 2006 |
| 1 | Radial Velocity | 1      | 874.774        | 2.21 | 56.95    | 2008 |
| 2 | Radial Velocity | 1      | 763.000        | 2.60 | 19.84    | 2011 |

# Análise de dados com Python

33

## Seaborn – Ajustando os gráficos

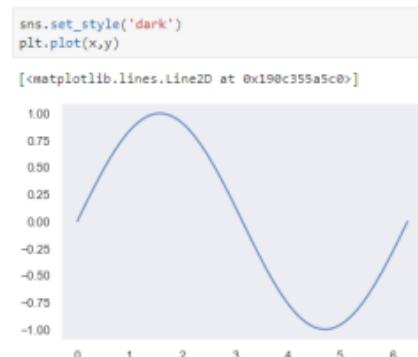
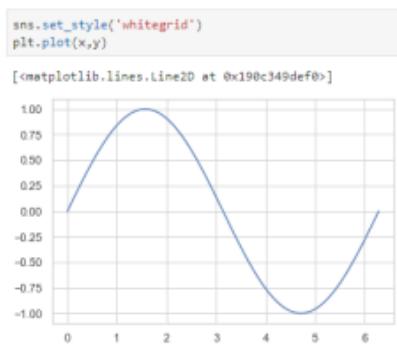
» O Seaborn possui a função `set()` que permite já ajustar estilos específicos para os gráficos, sem a necessidade de ajustar manualmente todos os argumentos.



## Seaborn – Ajustando os gráficos

34

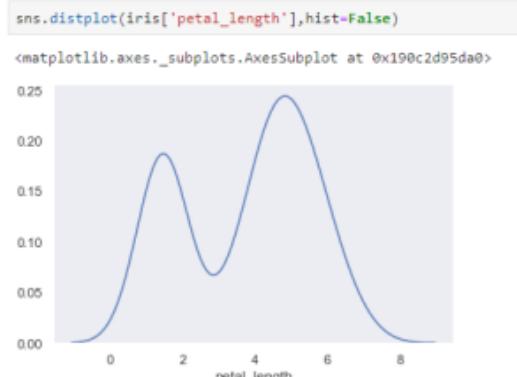
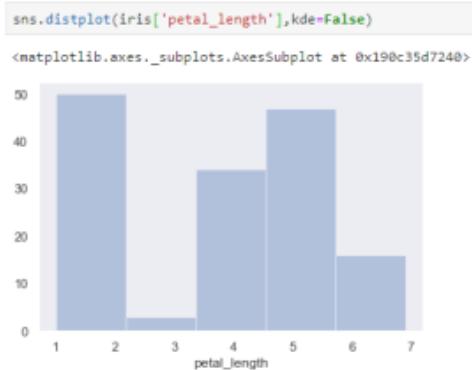
» O estilo default do Seaborn é o *darkgrid*, mas há ainda os estilos *whitegrid*, *dark*, *white*, e *ticks* que podem ser escolhidos com a função `set_style()`.



## Seaborn – Gráficos estatísticos

- » Outra vantagem do Seaborn é a possibilidade de criar gráficos estatísticos como, por exemplo, gráficos que estimem a função de densidade de probabilidade.
- » Entre as possibilidades de gráficos estatísticos, podemos criar gráficos estatísticos com as funções `distplot()` (univariada), `jointplot()` (bivariada).
- » Variando os parâmetros dentro dessas funções, podemos gerar visualização de histogramas, gráficos de hexágonos, dispersão e funções de distribuição.

## Seaborn – distplot (univariada)



## Seaborn – pairplot (bivariada)

- » A função pairplot() é outra função muito útil quando usamos os gráficos como recurso exploratório.
- » A função pairplot() exibe a relação 2 a 2 entre as variáveis, de modo que facilite a busca por correlações entre as variáveis de um conjunto de dados.
- » Exibe dados numéricos, embora dados categóricos possam ser utilizados para discriminar os dados.

## Seaborn – pairplot (bivariada)



# Análise de dados com Python

## Seaborn – Dados categóricos

39

- » Os gráficos vistos até o momento são apenas com dados numérico
- » E os dados categóricos? Como visualizá-los?
- » O Seaborn possui diversos gráficos que ajudam na exploração de dados categóricos:
  - » Stripplot
  - » Swarmplot
  - » Boxplot
  - » Violinplot
  - » Barplot
  - » Pointplot
  - » Etc.

## Seaborn – Dados categóricos

40

Os gráficos *stripplot()* e *swarmplot()* são gráficos em que analisamos a dispersão de uma variável numérica e outra categórica.



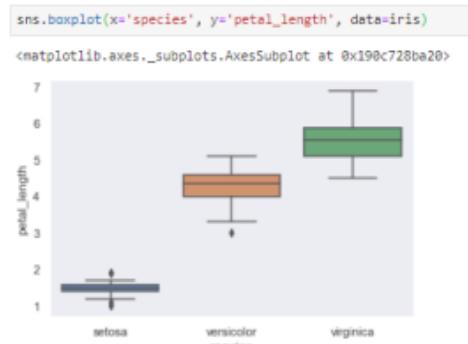
Qual a principal  
diferença entre os  
dois gráficos?

# Análise de dados com Python

## Seaborn – Dados categóricos

41

Os gráficos `barplot()` e `boxplot()` são gráficos em que analisamos a relação entre dados contínuos e categóricos.



## Exercícios - Seaborn

42

1. Importe a biblioteca Seaborn
2. Use a função `load_dataset()` e coloque o dataset `exercises` num dataframe chamado `exercicios`.
3. Use a função `set_style()` e mude para 'dark'.
4. Faça o histograma da variável `pulse` usando a função `distplot()` e o argumento `kde=False`.
5. Agora faça a distribuição de densidade da mesma variável, mas utilizando o argumento `hist=False`.
6. Use a função `swarmplot()` e analise a relação entre as variáveis `kind` (x) e `pulse` (y).

## Interpretação de gráficos

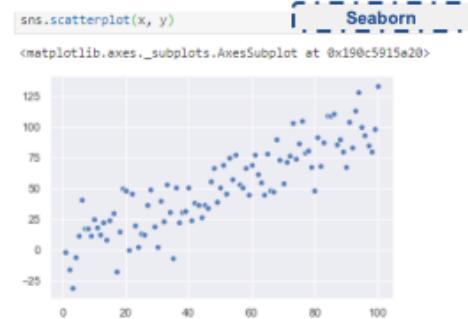
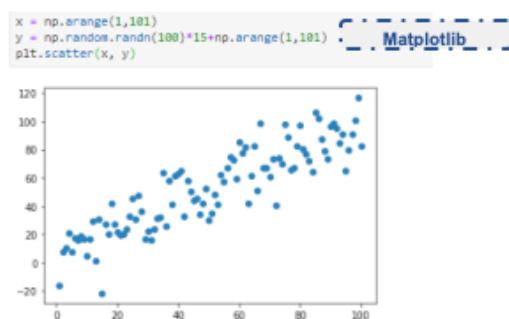
43

- » Mais do que apenas saber fazer gráficos, precisamos saber analisá-los para extrairmos informações.
- » Um gráfico sem uma interpretação é somente uma figura.
- » Iremos pontuar alguns elementos importantes a analisar quando utilizamos alguns tipos de dados.

## Gráficos de dispersão

44

- » O gráfico de dispersão (*scatterplot*) é muito útil quando precisamos verificar ou demonstrar a relação entre duas grandezas numéricas.
- » Cuidado com a sobreposição quando há muitos dados.

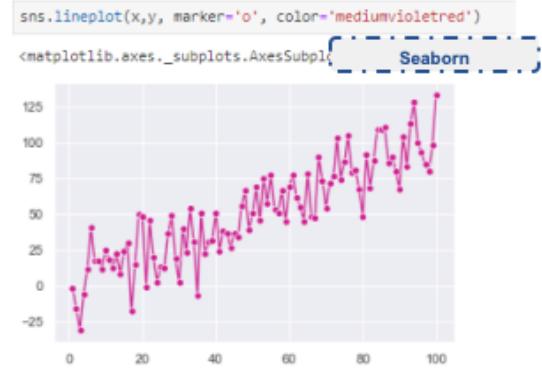
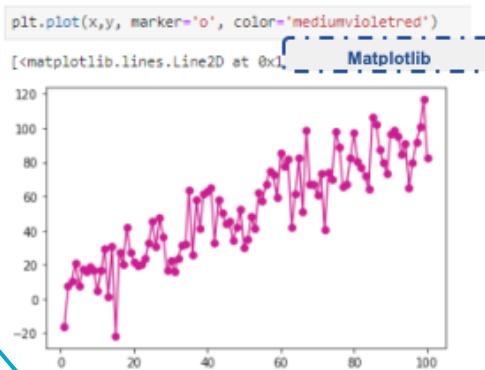


# Análise de dados com Python

## Gráficos de dispersão com linhas

45

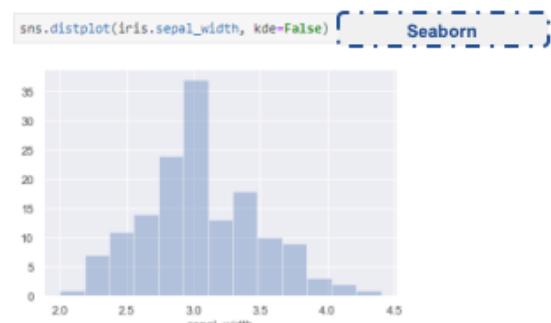
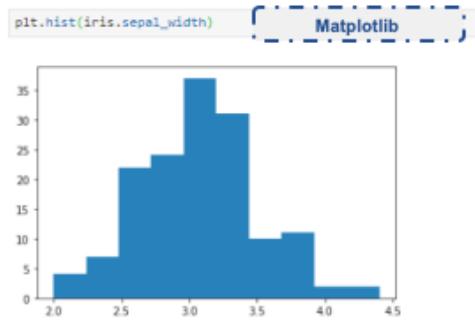
- » Similar ao gráfico de dispersão, mas com linhas que conectam os dados.
- » Muito útil para séries temporais ou dados em que a ordem das relações importam.



## Histogramas

46

- » Os histogramas exibem as frequências de uma ou mais variáveis, mostrando como os dados estão distribuídos.
- » É preciso atenção ao tamanho dos *bins* do histograma, pois pode induzir a conclusões diferentes.

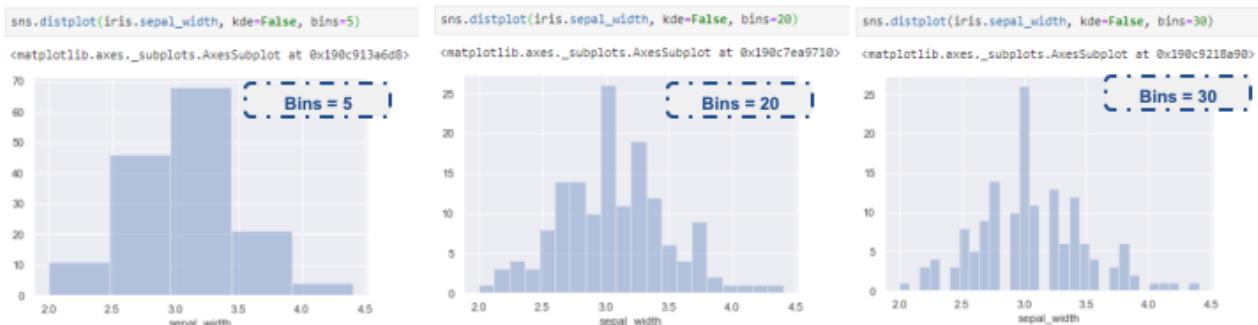


# Análise de dados com Python

47

## Histogramas

» É preciso atenção ao tamanho dos *bins* do histograma, pois pode induzir a conclusões diferentes.

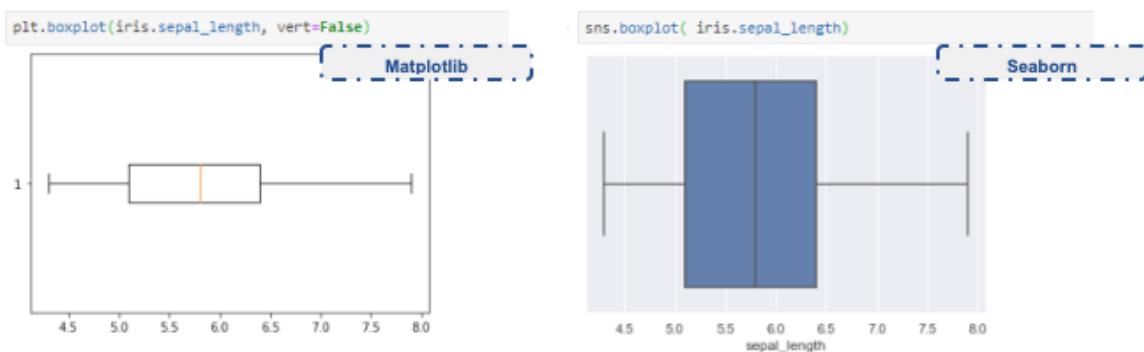


Mesmos dados, tamanhos diferentes de *bins*.

48

## Boxplots

» Um gráfico de boxplot permite visualizar a variação em uma ou mais variáveis através de seus quartis.

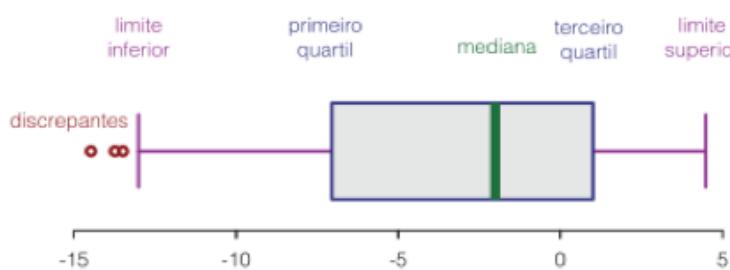


# Análise de dados com Python

49

## Boxplots

- » A linha central mostra a mediana dos valores e os limites da caixa exibem o primeiro e terceiro quartis.
- » As linhas externas exibem os limites inferior e superior
- » Se houver dados discrepantes (*outliers*), serão exibidos após as linhas externas.

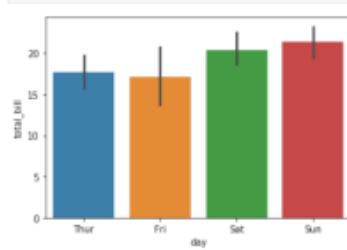


50

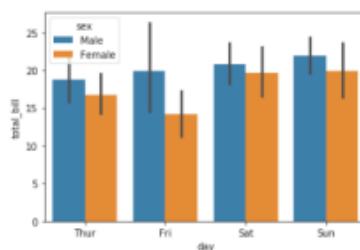
## Barplots

- » Gráficos de barras mostram a relação entre variável numérica e categórica.
- » Exibe o valor médio por categoria, mas pode calcular a mediana ou outro parâmetro estatístico.
- » O barplot do Seaborn exibe ainda um intervalo de confiança.

```
tips = sns.load_dataset("tips")
sns.barplot(x="day", y="total_bill", data=tips)
```



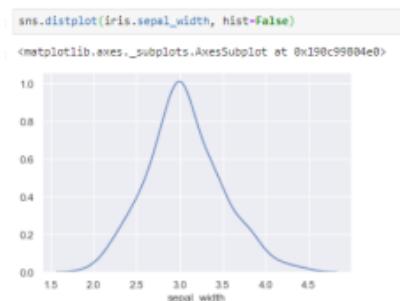
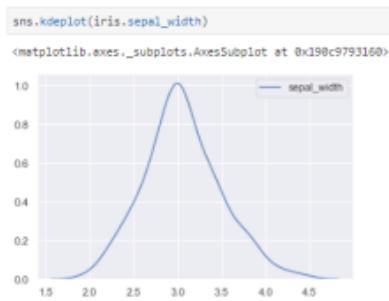
```
sns.barplot(x="day", y="total_bill", hue="sex", data=tips)
```



## Distribuição de densidade

51

- » O gráficos de distribuição de densidade, assim como o histograma, exibe a distribuição de uma ou mais variáveis.
- » Apenas para variáveis numéricas.



## Exercícios: gráficos

52

1. Utilizando os datasets do Seaborn, crie o dataframe *voo* com o datasets *flights*.
2. Faça um gráfico de barras com as variáveis *year* e *passengers*.
3. Que informação podemos extrair desse gráfico?
4. Use a função *boxplot()* e verifique a variável *passengers*.
5. O que podemos depreender desse gráfico?
6. Qual a diferença entre os resultados obtidos com o gráfico de barras e o boxplot?
7. Faça um gráfico de barras com as variáveis *month* e *passengers*.
8. Qual conclusão podemos extrair desse gráfico?

# Scikit Learning

Parceria:



# Análise de dados com Python

## Sumário

01

- » O que é Machine Learning
- » Modelos Supervisionados e Não Supervisionados
- » Biblioteca Scikit-Learning
- » Treino, teste e validação
- » Principais modelos
- » Regressão linear
- » Modelo de classificação
- » Modelo de clusterização

## Machine Learning – O que é?

02

- » Machine Learning (ML), ou Aprendizado de Máquina, é um método computacional que utiliza algoritmos e métodos estatísticos para analisar conjuntos de dados, descobrir padrões e realizar definições ou previsões.
- » ML é uma sub-área da Inteligência Artificial.

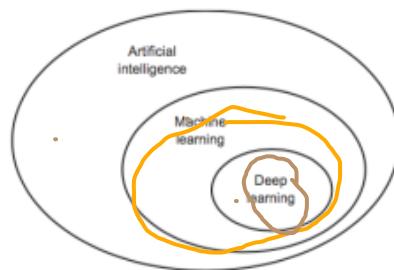


Figura obtida de François Chollet, Deep Learning with Python.

# Análise de dados com Python

## Machine Learning – O que é?

03

- » Muitos avanços estão sendo realizado na Ciência e Tecnologia devido ao desenvolvimento de novos métodos de análise utilizando aprendizado de máquina.
- » ML tem sido usado em diversas áreas, tanto em pesquisas acadêmicas (Medicina, Biologia, Astronomia, Ciência Sociais...) quanto no mundo corporativo (em prevenção à fraudes em bancos, estratificação de população em sistemas de saúde, sistemas de recomendação em e-commerce...)

## Machine Learning – Onde está sendo usada

04



# Análise de dados com Python

## Machine Learning – Onde esta sendo usada

05

### Google vai usar inteligência artificial para detectar vírus em e-mails

Sistema tem capacidade para verificar 300 bilhões de anexos por semana

### Pesquisadores holandeses usam inteligência artificial para prever o comportamento de asteroides

### Inteligência artificial é utilizada na operação de rebocadores no Porto

Saude  
Nova tecnologia

### Inteligência artificial descobre novo tipo de antibiótico

A halicina se mostrou capaz de combater bactérias consideradas super resistentes pela OMS.

Machine learning picks out hidden vibrations from earthquake data

Technique may help scientists more accurately map vast underground geologic structures.

### Inteligência artificial vai identificar futuros criminosos no Reino Unido

Plataforma desenvolvida pela polícia de West Midlands custou cerca de 65 milhões de reais e pode trazer segurança aos cidadãos

### Machine learning predicts behavior of biological circuits

Neural networks cut modeling times of complex biological circuits to enable new insights into their inner workings

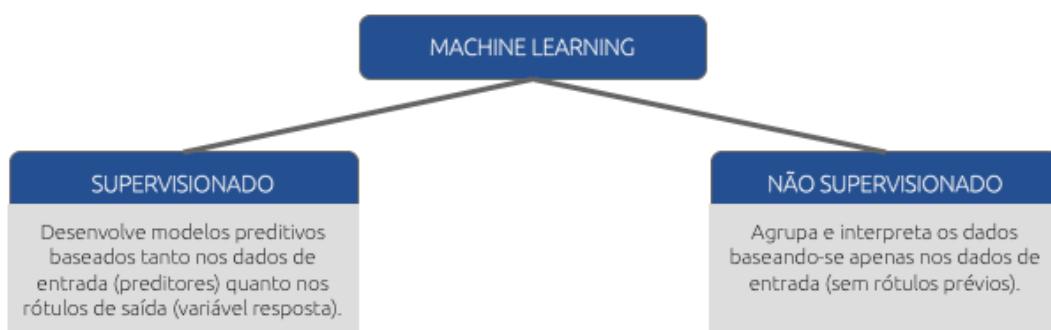
### Startup reduz desperdício em bares e restaurantes com ajuda de inteligência artificial

A empresa entrega para o cliente o levantamento e indicações do que pode estar gerando as perdas. Faz também um relatório para gerenciar os processos na prática.

## Machine Learning – Onde esta sendo usada

06

» Em ML temos dois tipos de modelagens: modelos supervisionados e modelos não supervisionados.

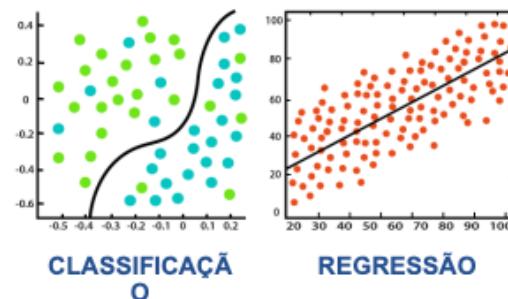


# Análise de dados com Python

## Modelos Supervisionados

07

- » No aprendizado supervisionado a classe (variável) que queremos predizer já foi rotulada nos dados que serão utilizados para treinar o modelo.
- » O aprendizado supervisionado pode ser utilizado para:
  - » Predição de uma categoria (classificação)
  - » Predição de um número (regressão)



## Modelos Supervisionados

08

Principais algoritmos de classificação/regressão:

- » Naive Bayes
- » Árvore de decisão
- » Random Forest
- » K-Nearest Neighbours (KNN)
- » Support Vector Machine (SVM)

# Análise de dados com Python

## Modelos Não Supervisionados

09

» No aprendizado não supervisionado não temos previamente os rótulos nos dados, de modo que o algoritmo irá procurar por padrões no conjunto de dados e agrupá-los segundo esses padrões.

» O aprendizado não supervisionado pode ser utilizado para:

- » Criar grupos similares (clusterização)
- » Diminuir número de dimensões em um conjunto de dados



## Modelos Não Supervisionados

10

Principais algoritmos Não Supervisionados:

- » K-means (agrupamento)
- » DBSCAN (agrupamento)
- » Isolation Forest (detecção de anomalia)
- » Auto Encoder (redes neurais)

# Análise de dados com Python

## Machine Learning – criando modelos

11

- » Pense: para quem e porquê você está criando o modelo?
- » Escolha a pergunta que estamos tentando responder.
- » Selecione o conjunto de dados para responder esta pergunta.
- » Identifique como medir o resultado.

» Para todas as anteriores é preciso de conversa e entendimento.

## Entendimento do negócio: 4P's

12

- » Para analisar dados e modelá-los é importante que estejamos sempre alinhados com as necessidades do negócio e/ou problema a ser resolvido.
- » Para isso, podemos utilizar a estratégia dos 4P's:

- » Problema - Qual a dor de negócio que seu estudo vai endereçar?
- » Potencial - Qual o potencial ganho que se obterá com o projeto?
- » Produto - Qual será o produto que você entregará? Qual modelo?
- » Proposta - Como seu modelo irá resolver a dor inicial?

# Análise de dados com Python

## Machine Learning – boas práticas

13

- » Simplesmente comece!
- » Defina o problema de negócio explicitamente
- » Estabeleça as métricas de sucesso (acurácia, precisão, recall...)
- » Colete os dados
- » Conheça os seus dados, explore e tire insights
- » Crie novas variáveis (features)
- » Escolha o algoritmo mais adequado para os seus dados e para seu projeto
- » Considere os problemas possíveis e teste antes do lançamento
- » Implante e automatize

## Exercícios – entendimento do negócio

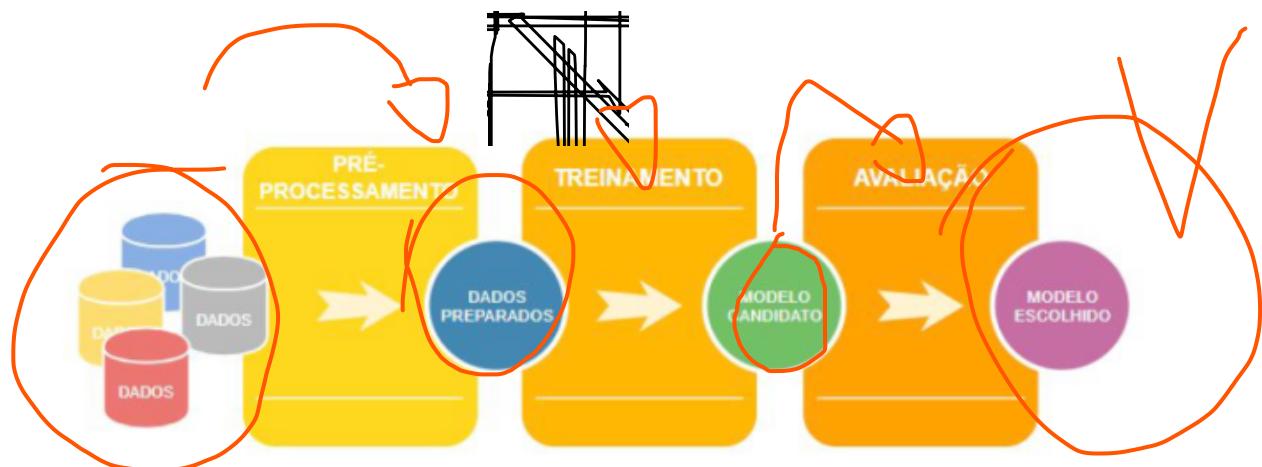
14

1. Pense em um problema a ser resolvido e defina-o de maneira clara e objetiva.
2. Qual a pergunta a ser respondida?
3. Qual sua hipótese?
4. Quais os dados necessários?
5. Como medir seus resultados? Como saber se meu modelo é bom?
6. Quem usaria os resultados obtidos?

# Análise de dados com Python

## Machine Learning – ciclo de processos

15



## Machine Learning – ciclo de processos

16

- » Pré-Processamento dos dados: verificação de dados faltantes, imputação de valores, interpolação, criação de novas variáveis, normalização.
- » Treinamento: separação de amostra de teste e treino, aplicação de modelos.
- » Avaliação: o modelo teve boa performance? Ele respondeu ou resolveu meu problema?

# Análise de dados com Python

## Amostras de Treino e Teste

17

» Quando trabalhamos com modelos supervisionados precisamos sempre separar nossas amostras de Treino e Teste, para evitar *overfitting*.

» Amostra de Treino: amostra utilizada para ensinar o modelo como Fazer a classificação/regressão.

» Amostra de Teste: amostra na qual aplicamos o modelo já treinado, conferindo a qualidade do modelo obtido.

## Exercícios – Ciclo de Processos

18

1. Pesquise o significado de overfitting e reescreva utilizando suas palavras.
2. Pesquise quais as melhores práticas para separação de amostra de treino e teste.
3. É razoável dividir os dados em amostra de treino e teste com os mesmos tamanhos? Por quê?

# Análise de dados com Python

## Scikit-Learn – biblioteca de Machine Learning

19

- » A biblioteca Scikit-Learn (sklearn) é uma das principais bibliotecas de machine learning em Python.
- » Com ela podemos desenvolver projetos com modelagem estatística e modelos de classificação, regressão, agrupamentos e redução de dimensionalidade.
- » Iremos abordar brevemente dois modelos: regressão linear simples e modelo de classificação.



## Sklearn – Datasets

20

- » Similar ao Seaborn, o Scikit-Learn também possui datasets embutidos que podem ser úteis para estudar as funcionalidades da biblioteca.

|   |   |
|---|---|
| <code>load_boston([return_X_y])</code>          | Load and return the boston house-prices dataset (regression).         |
| <code>load_iris([return_X_y])</code>            | Load and return the iris dataset (classification).                    |
| <code>load_diabetes([return_X_y])</code>        | Load and return the diabetes dataset (regression).                    |
| <code>load_digits([n_class, return_X_y])</code> | Load and return the digits dataset (classification).                  |
| <code>load_linnerud([return_X_y])</code>        | Load and return the linnerud dataset (multivariate regression).       |
| <code>load_wine([return_X_y])</code>            | Load and return the wine dataset (classification).                    |
| <code>load_breast_cancer([return_X_y])</code>   | Load and return the breast cancer wisconsin dataset (classification). |

```
from sklearn.datasets import load_wine  
wine = load_wine(return_X_y=True)
```

```
from sklearn.datasets import load_boston  
boston = load_boston(return_X_y=True)
```

- » Os dados retornados serão do tipo array (Numpy) e terão um metadados se não utilizarmos o argumento `return_X_y=True`.

# Análise de dados com Python

## Sklearn – Treino e Teste

21

- » O Sklearn possui uma função própria para fazer a separação entre amostra de treino e teste: a função `train_test_split()`.

```
X, y = boston  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)  
  
len(X)  
506  
  
len(X_train)  
379  
  
len(X_test)  
127
```

Separando o dataset em variáveis preditoras (X) e variável resposta (y)

Confirmando o tamanho do conjunto de dados inteiro e das amostra de treino e teste.

## Sklearn – Treino e Teste

22

- » A função `train_test_split()` separa, por default, 25% dos dados para teste.
- » Para modificar a proporção, usamos o argumento `test_size`.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,  
                                                 random_state=0)  
  
print('Treino: %d' % len(X_train))  
print('Teste: %d' % len(X_test))  
  
Treino: 354  
Teste: 152  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,  
                                                 random_state=0)  
  
print('Treino: %d' % len(X_train))  
print('Teste: %d' % len(X_test))  
  
Treino: 303  
Teste: 203
```

# Análise de dados com Python

## Exercícios– Treino e Teste

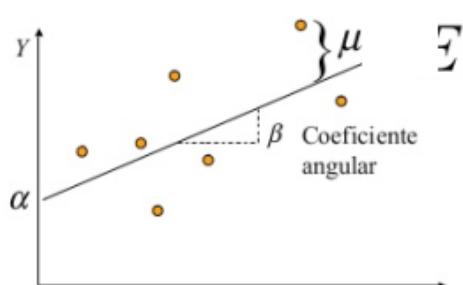
23

1. Utilizando os datasets embutidos no Scikit-Learn crie o conjunto de dados diabetes, utilizando o argumento `return_X_y=True`.
2. Utilizando o dataset diabetes, crie as variáveis preditoras (X) e a variável resposta(y).
3. Procure a documentação da função `train_test_split` e verifique qual o significado do argumento `random_state` e explique qual sua importância.
4. Faça uma separação de amostra de treino e teste onde a amostra de treino contenha 65% dos dados e com `random_state=42`.

## Regressão Linear

24

» A regressão linear é um modelo matemático em que tentamos prever um valor y (variável resposta) baseado em diversas outras variáveis X (variáveis preditoras).



$$y = \alpha + \beta X + \mu$$

y: variável de interesse  
X: variáveis preditoras  
 $\alpha$ : interceptação do eixo y  
 $\beta$ : inclinação da reta ajustada  
 $\mu$ : erro da previsão

# Análise de dados com Python

25

## Regressão Linear

» O dataset Boston possui 13 variáveis preditoras e uma variável resposta (preço das casas em Boston).

1. CRIM - per capita crime rate by town
2. ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS - proportion of non-retail business acres per town.
4. CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. NOX - nitric oxides concentration (parts per 10 million)
6. RM - average number of rooms per dwelling
7. AGE - proportion of owner-occupied units built prior to 1940
8. DIS - weighted distances to five Boston employment centres
9. RAD - index of accessibility to radial highways
10. TAX - full-value property-tax rate per \$10,000
11. PTRATIO - pupil-teacher ratio by town
12. B -  $1000(Bk - 0.63)^2$  where Bk is the proportion of blacks by town
13. LSTAT - % lower status of the population
14. MEDV - Median value of owner-occupied homes in \$1000's

» Iremos treinar um modelo de regressão que encontre o valor de MEDV baseado nas 13 variáveis preditoras.

26

## Regressão Linear

» Na biblioteca Scikit-Learn há diversos modelos para regressão linear: LinearRegression, Ridge, RidgeCV, SGDRegressor e outros.

» Para o nosso estudo iremos utilizar o modelo LinearRegression.

```
from sklearn.linear_model import LinearRegression
```

» Iniciando o modelo de regressão:

```
model = LinearRegression()
```

# Análise de dados com Python

## Regressão Linear

27

- » Para treinar o modelo, aplicamos a função fit() aos dados X e y.

```
model.fit(X_train,y_train)  
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

- » Para predizer os valores de y utilizando já o modelo treinado, aplicamos o modelo aos dados de teste.

```
predito = model.predict(X_test)  
  
predito  
  
array([24.58155243, 24.51629253, 29.71379915, 12.51132696, 21.34965428,  
       19.05443022, 20.94614567, 20.95753329, 19.54644456, 20.53025981,  
       6.96153725, 17.1707288 , 16.85608802, 5.74921859, 40.74378524,  
       32.62964196, 22.88997064, 37.11387241, 30.94054261, 23.12796161,
```

## Regressão Linear

28

- » Agora que já treinamos e aplicamos o modelo aos dados de teste, podemos verificar qual os coeficientes da reta ajustada.
- » `coef_` é o valor de b que multiplica cada uma das variáveis preditoras.
- » `intercept_` é o valor onde a reta intercepta o eixo y.

```
model.coef_  
  
array([-1.03747356e-01,  5.58589924e-02,  5.88240770e-02,  2.50523544e+00,  
       -1.90284888e+01,  3.25353601e+00, -3.22150522e-03, -1.57603462e+00,  
       2.58716068e-01, -1.14681299e-02, -1.10777478e+00,  5.50051783e-03,  
       -5.59569992e-01])  
  
model.intercept_  
  
45.481419593250976
```

# Análise de dados com Python

29

## Regressão Linear

- » Para sabermos quão boa foi nossa estimativa, podemos utilizar a função `r2_score()`.
- » Quanto mais próximo de 1 o r2\_score for, mais precisa está sendo nossa predição.

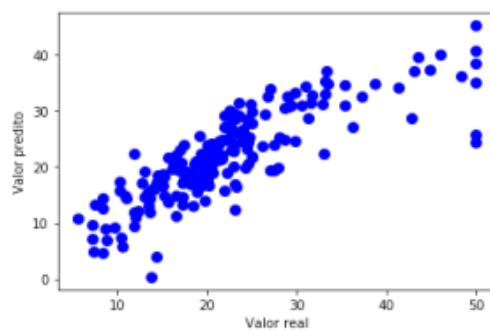
```
from sklearn.metrics import r2_score  
  
r2_score(y_test, predito)  
0.6882607142538016
```

30

## Regressão Linear

- » Podemos ainda visualizar a relação entre os valores reais e os valores preditos:

```
plt.scatter(y_test, predito, color='blue', linewidth=3)  
plt.xlabel('Valor real')  
plt.ylabel('Valor predito')
```



# Análise de dados com Python

## Exercícios – Regressão Linear

31

1. Utilize o dataset diabetes criado anteriormente. Pesquise sobre o que se trata.
2. Separe-o em X,y.
3. Faça a separação em amostra de treino e teste, deixando 30% para teste. Use random\_state=42.
4. Importe o LinearRegression, crie o modelo model.
5. Treino o modelo aplicando a função fit() aos dados de treino.
6. Faça a predição, aplicando a função predict aos dados X\_train, chamando o resultado de predito.
7. Meça o valor de r2\_score. O quanto bom está seu modelo?
8. Faça um gráfico de espalhamento com os valores de y\_train e predito

## Classificação

32

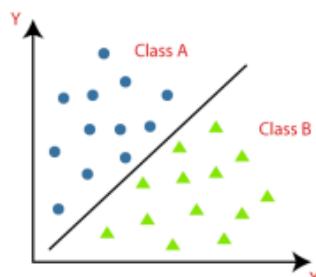
» Algoritmos de classificação são muito úteis em diversas tarefas:

- » Separar estrelas e Galáxias
- » Aprovação de crédito
- » Classificar diagnósticos
- » Separar tipos de flores

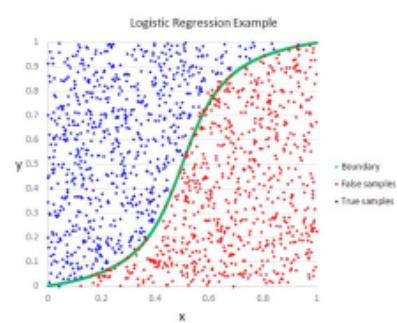
» Existem diversos tipos de algoritmos para classificação:

- » Regressão logística
- » KNN
- » Naive Bayes
- » Árvores de decisão
- » Redes neurais

» Iremos utilizar/trabalhar o modelo de Regressão Logística.



## Regressão Logística



» É um modelo de classificação muito comum.

» Não é um algoritmo de regressão.

» É usado para estimar valores discretos (em geral, valores binários) com base em determinado conjunto de variáveis independentes.

» A Regressão Logística prevê a probabilidade de ocorrência de um evento, ajustando os dados a uma [Função logit](#).

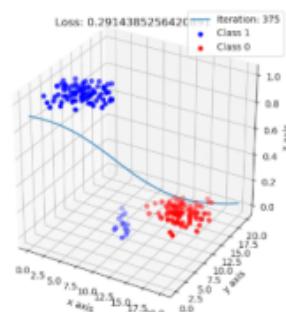
$$f(x) = \log\left(\frac{x}{1-x}\right)$$

## Regressão Logística

» O modelo de regressão logística irá sempre retornar uma probabilidade de um certo dado pertencer (ou não) à uma dada classe.

» Dentre os casos de uso, podemos citar:

- » Detecção de spams
- » Predição de Diabetes
- » Cálculo de churn
- » Se um cliente irá clicar ou não em uma propaganda
- » Estudos de mercado



## Regressão Logística

» Existem três tipos de Regressão Logística:

- » Regressão logística binária: usada em classificações binárias (ex.: spam e não-spam, sim e não).
- » Regressão logística multinomial: três ou mais categorias em que não há ordem nas categorias (ex.: classificação de vinhos).
- » Regressão logística ordinária: três ou mais categorias onde a ordem da classificação importa (ex.: as notas de um restaurante.)

## Regressão Logística

» Na biblioteca do Scikit-Learn temos o modelo LogisticRegression para a modelagem de regressão logística.

```
from sklearn.linear_model import LogisticRegression
```

» Nesse exemplo usaremos o dataset clássico Iris.

```
from sklearn.datasets import load_iris
iris = load_iris(return_X_y=True)
```

» Iremos classificar as espécies das flores em 3 categorias: Iris setosa, Iris virginica e Iris versicolor.

# Análise de dados com Python

## Regressão Logística

37

- » Separando as variáveis preditoras (X) da variável resposta (y), que é a classificação que queremos obter.

```
X,y = iris
```

- » Separamos nossas amostras de treino e teste

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,  
random_state=0)
```

- » Note que deixamos 30% para a amostra de teste.

## Regressão Logística

38

- » Criando o modelo de classificação

```
clf = LogisticRegression()
```

- » Treinando o modelo de classificação na amostra de treino

```
clf.fit(X_train,y_train)  
  
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
intercept_scaling=1, l1_ratio=None, max_iter=100,  
multi_class='warn', n_jobs=None, penalty='l2',  
random_state=None, solver='warn', tol=0.0001, verbose=0,  
warm_start=False)
```

# Análise de dados com Python

39

## Regressão Logística

» Aplicando agora o modelo treinado nos dados de teste

```
predito = clf.predict(X_test)

predito
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 2, 1, 0, 0, 2, 2,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 2, 0, 2, 2, 2, 0, 2, 1, 1, 2, 0, 2, 0,
       0])
```

» Para verificar as probabilidades calculadas para cada item do dataset, usamos a função predict\_proba().

```
clf.predict_proba(X_test)
Probabilidade de cada classe,
para cada item do dataset.

array([[1.01514149e-03, 1.68979201e-01, 8.30005658e-01],
       [2.77325788e-02, 8.19480925e-01, 1.52786497e-01],
       [9.35029110e-01, 6.49599289e-02, 1.09613940e-05],
       [2.42826242e-04, 3.78006486e-01, 6.21750688e-01],
```

40

## Regressão Logística

» Para avaliar o resultado, podemos usar a função classification\_report e obter rapidamente as principais métricas de avaliação.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, predito))

      precision    recall  f1-score   support

          0       1.00     1.00      1.00      16
          1       1.00     0.72      0.84      18
          2       0.69     1.00      0.81      11

  accuracy                           0.89      45
  macro avg       0.90     0.91      0.88      45
weighted avg       0.92     0.89      0.89      45
```

## Exercícios – Regressão Logística

1. Utilize os datasets embutidos no Scikit-Learn crie o conjunto de dados cancer, utilizando o argumento `return_X_y=True`. Pesquise sobre o dataset.
2. Separe-o em X,y.
3. Faça a separação em amostra de treino e teste, deixando 30% para teste. Use `random_state=42`.
4. Importe o `LogisticRegression`, crie o modelo de classificação `clf`.
5. Treino o modelo aplicando a função `fit()` aos dados de treino.
6. Faça a predição, aplicando a função `predict` aos dados `X_train`, , chamando o resultado de `predito`.
7. Utilize a função de métrica `classification_report`. O quanto bom está seu modelo?

**Mãos à obra!**

# Análise de dados com Python

---

## Introdução ao Python - Variáveis

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Atribua os seguintes valores à variáveis, um a um:
  - a. 347
  - b. 2.71
  - c. “347”
  - d.  $2+3j$
3. Quais os tipos das variáveis acima?
4. Faça uma atribuição múltipla das variáveis do exercício 2.
5. Declare a seguinte variável e verifique o que acontece:
  - a. teste = true
6. Transforme as variáveis do exercício 2 conforme segue:
  - a. para float
  - b. para inteiro
  - c. para float
  - d. para string
7. Crie uma variável complexa com os valores de 2.a e 2.b.

## Introdução ao Python - Operadores

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Defina as seguintes variáveis  $x = 3$ ,  $y = 4.0$ ,  $z = 12$  e  $t = 'banana'$
3. Calcule:
  - a.  $x + y$
  - b.  $z - y$
  - c.  $y ** x$
  - d.  $x * z$
  - e.  $z / x$
  - f.  $y / x$
  - g.  $y \% x$
4. Teste as seguintes relações
  - a.  $x > y$
  - b.  $x * y == z$
  - c.  $t == y$
  - d.  $x > y$  and  $x < z$

## Introdução ao Python - Operadores com strings

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie uma variável com seu nome completo.
3. Escreva a variável em lowercase.
4. Escreva em uppercase.
5. Verifique se o nome começa com a letra P.
6. Verifique se o nome termina com a letra J.
7. Fatie a string para apenas o 1o nome.
8. Fatie a string de 2 em 2.
9. Considerando os espaços, qual o tamanho do seu nome?

# Análise de dados com Python

---

## Introdução ao Python - Listas

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie uma lista com nomes de 4 times de futebol.
3. Acesse o time que está na 3a posição.
4. Crie uma nova lista com duas listas de 3 times de futebol, cada uma de uma divisão diferente.
5. Crie uma lista com 3 diferentes moedas. Acrescente mais 2 outras moedas à essa mesma lista.
6. Crie uma string com a lista do exercício anterior.
7. Agora utilize a string do exercício 6 para recriar uma lista.

## Introdução ao Python - Dicionários

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie um dicionário chamado cardapio em que as chaves são os dias da semana e os respectivos valores sejam os pratos do dia.
3. Crie um dicionário chamado hemograma que contenha as seguintes chave:valor:
  - *hemacias*: 4.71
  - *hemoglobina*: 14.1
  - *hematocrito*: 41.2
  - *linfocitos*: 38
  - *monocitos*: 7
  - *resultado*: saudável
4. Corrija o dicionário acima com monocitos:12.

# Análise de dados com Python

---

## Introdução ao Python - Leitura e Escrita

1. Abra um bloco de notas e crie um código chamado imc.py que:
  - a. Pergunte o nome da pessoa e atribua na variável nome.
  - b. Pergunte a altura (em metros) e atribua na variável alt. Não esqueça de que a variável deve ser do tipo float.
  - c. Pergunte o peso (em quilos) e atribua na variável kg. Não esqueça de que a variável deve ser do tipo float.
  - d. Calcule o IMC através da fórmula  $\text{IMC} = \text{peso}/(\text{alt}^*\text{alt})$
  - e. Escreva o resultado do cálculo do IMC como “Olá <Fulano>, seu IMC é <xx>”, em que <Fulano> seja o nome da pessoa e <xx> seja o valor do IMC.

# Análise de dados com Python

## Introdução ao Python - Condicionais

1. Atualize o programa `imc.py` feito anteriormente para que o resultado exibido seja

“Olá <Fulano>, seu IMC é <xx>, logo você está <situação>.” em que a <situação> segue as condições abaixo:

| Resultado           | Situação                |
|---------------------|-------------------------|
| Abaixo de 17        | Muito abaixo do peso    |
| Entre 17 e 18,49    | Abaixo do peso          |
| Entre 18,50 e 24,99 | Peso normal             |
| Entre 25 e 29,99    | Acima do peso           |
| Entre 30 e 34,99    | Obesidade I             |
| Entre 35 e 39,99    | Obesidade II (severa)   |
| Acima de 40         | Obesidade III (mórbida) |

# Análise de dados com Python

---

## Introdução ao Python - Loops: for

1. CAbra um bloco de notas e crie um programa chamado tabuada.py que faça:
  - a. Declare uma lista multiplos = [1,2,3,4,5,6,7,8,9,10]
  - b. Peça ao usuário um número inteiro de 1 a 10 e atribua na variável number.
  - c. Faça um laço for que imprima os valores da tabuada de number.

## Introdução ao Python - Loops: while

1. Abra um bloco de notas e crie um programa chamado factorial.py que:
  - a. Peça um número inteiro entre 2 e 15 ao usuário e atribua na variável valor.
  - b. Crie uma variável fat e atribua um valor inicial igual a zero.
  - c. Crie um contador cont e atribua um valor inicial igual a zero.
  - d. Usando um loop while, enquanto cont for menor que valor, atualize fat como fat = fat x valor
  - e. Quando o loop terminar, imprima “O factorial de <valor> é <fat>”, em que <valor> é o número dado pelo usuário e <fat> seja o resultado do loop.

# Análise de dados com Python

---

## Numpy - Fatiamento de arrays

1. Abra o Jupyter lab
2. Crie um array data com a seguinte lista: [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
3. Selecione apenas os dados diferentes de 4,6,8,10. Aloque em um array chamado data\_2.
4. Crie um array dim2 que tenha 2 dimensões e a lista [1,2,3,4,5] na primeira linha e a lista [6,7,8,9,10] na 2a linha.
5. Crie um array chamado dim2\_2 que seja uma fatia de dim2 e que contenha apenas os valores 3,4,8,9.

## Numpy - Funções úteis

1. Abra o Jupyter lab
2. Crie um array data com a seguinte lista: [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
3. Verifique a dimensão, a quantidade de elementos e a forma desse array.
4. Crie um novo array chamado data2, com 4x4 elementos, aplicando a função reshape no array data.
5. Verifique a dimensão, a quantidade de elementos e a forma desse array.
6. Crie um novo array chamado data\_t, que seja a transposição do array data2.
7. Use a função np.flatten() e crie o array data\_flat.
8. Compare o array data com o array data\_flat.

# Análise de dados com Python

---

## Numpy - Operações com arrays

1. Abra o Jupyter lab
2. Crie 5 arrays a, b, c, d e e com as seguintes listas:
  - a. [7]
  - b. [[4,6,8],[3,5,7]]
  - c. [3,3,3]
  - d. [[3,2,1],[1,2,3]]
  - e. [5,10]
3. Faça as seguintes operações:
  - a. a \* b
  - b. d - b
  - c. c + b
  - d. c + e
4. O que aconteceu no item d) do último exercício? Explique.

# Análise de dados com Python

---

## Numpy - Funções e métodos matemáticos

1. Abra o Jupyter lab
2. Crie 2 arrays a e b com as seguintes listas:
  - a. [[4,6,8],[3,5,7]]
  - b. [[11,13,17],[23,29,31]]
3. Use a função `cbrt()` e calcule a raiz cúbica dos arrays a e b.
4. Calcule a soma cumulativa de a.
5. Calcule a média de a e b.
6. Aplique a função `np.negative()` no array b e some com b. O que aconteceu?

## Numpy - Concatenação

**6.** Abra o Jupyter lab

**7.** Crie 2 arrays arrays com as seguintes listas:

**a.** [[1,2,3],[4,5,6]]

**b.** [[2,4,6,8],[10,12,14,16]]

**c.** [[11,13,17],[23,29,31]]

**d.** [[13,14,5],[19,21,23]]

**8.** Concatene a e b. O que aconteceu?

**9.** Concatene b e c, tanto usando o eixo 0 (linha) quanto o eixo 1 (coluna).

**10.** Concatene a e d, tanto usando o eixo 0 (linha) quanto o eixo 1 (coluna).

# Análise de dados com Python

## Pandas - Manipulando Dataframes

1. Abra o arquivo world\_happiness\_report\_2015.csv e o aloque em um dataframe.
2. Verifique o cabeçalho e o final do dataframe.
3. Quais as colunas desse dataframe?
4. Quais os tipos de dados temos no dataframe?
5. Há valores faltantes ou nulos? Em quais colunas?
6. Renomeie as variáveis como segue:

|                                 |    |                         |
|---------------------------------|----|-------------------------|
| <b>happiness rank</b>           | => | <b>rank_felicidade</b>  |
| <b>happiness score</b>          | => | <b>score_felicidade</b> |
| <b>standard error</b>           | => | <b>stand_error</b>      |
| <b>economy (GDP per Capita)</b> | => | <b>PIB</b>              |

|                                      |    |                    |
|--------------------------------------|----|--------------------|
| <b>health (Life Expectancy)</b>      | => | <b>expect_vida</b> |
| <b>trust (Government Corruption)</b> | => | <b>corrupcao</b>   |

7. Quais os valores médios de expect\_vida? E o valor mediano? E o máximo da variável PIB?
8. Crie uma series que contenha a altura de 5 colegas e deixe seus nomes como índice.

# Análise de dados com Python

---

## Pandas - loc e iloc

1. Selecione apenas os dados de country, region, family e freedom (usando loc)
2. Selecione apenas os dados de country, region, family e freedom (usando iloc)
3. Selecione apenas as primeiras 15 linhas de country e PIB (usando loc)
4. Selecione apenas as primeiras 15 linhas de country e PIB (usando iloc)
5. Selecione apenas os dados cujo score\_felicidade seja maior que 5.
6. Selecione apenas os dados que sejam da Southern Asia.

## Pandas - Operações com Dataframes

1. Qual a média do score\_felicidade?
2. Qual a soma do PIB?
3. Qual a soma do freedom e corrupcao?
4. Há dados duplicados? Quantos? Verifique quais são eles.
5. Verifique a quantidade de dados faltantes.
6. Crie um novo dataframe onde os valores faltantes de score\_felicidade sejam substituídos por -9999.
7. Quantas e quais são regions existentes nos dados?
8. Verifique a frequência dos dados segundo suas regiões. Qual a região com maior quantidade de dados? E a região com a menor quantidade?

## Pandas - merge, concat

1. Crie um dataframe para cada um dos arquivos nba\_2015\_a.csv, nba\_2015\_b.csv, nba\_2015\_c.csv, e bust\_nba\_2015.csv. Chame esses dataframes de df\_a, df\_b, df\_c, bust, respectivamente.
2. Visualize o head() de cada um dos dataframes.
3. Concatene os arquivos df\_a, df\_b e df\_c usando a função concat() usando os índices. Salve um dataframe chamado df\_total.
4. Faça a concatenação do dataframe df\_total com o dataframe bust utilizando a função merge() e a variável ID.
5. Busque a documentação das funções concat() e merge() e veja que outros parâmetros podem ser utilizados.

# Análise de dados com Python

---

## Pandas - groupby

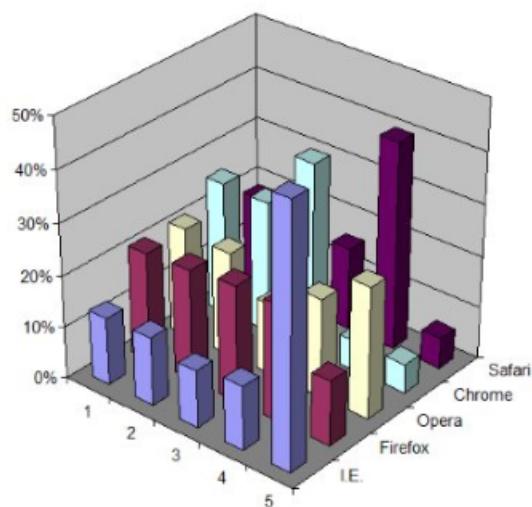
1. Abra o arquivo preferencias.csv como um dataframe chamado pref.
2. Visualize os 5 primeiras linhas do arquivo.
3. Agrupe os dados pela variável Gender (gênero).
4. Verifique a contagem de itens por cada gênero.
5. Agrupe os dados pelas variáveis Gender e Favorite Color (cor favorita).
6. Quantos itens de gênero F também possuem cor favorita Cool?
7. Agrupe os dados pelas variáveis Gender e Favorite Color e Favorite Beverage (bebida favorita).
8. Verifique a quantidade de itens de gênero M que têm cor preferida Warm e que preferem Beer.

# Análise de dados com Python

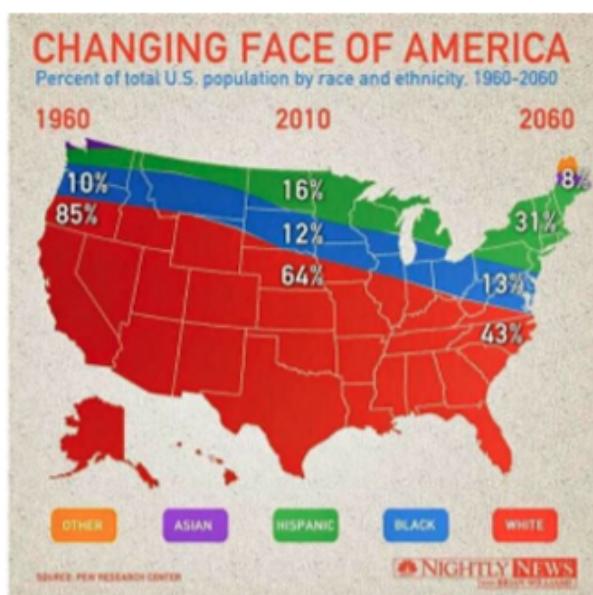
## Visualização de Dados - Information Design

1. Como exibir as informações dos gráficos abaixo de maneira mais eficiente? Faça, usando apenas papel e caneta, um esboço de como os dados poderiam ser melhor apresentados.

a)



b)



## Visualização de Dados - Matplotlib

1. Crie dois arrays x e y com as listas [-3,-2.5,-2,-1,0,1,2,2.5,3] e [-27,-15.62,-8,-1,0,1,8,15.62,27].
2. Use a função plot() e faça um gráfico de x e y.
3. Use a função xlabel() para adicionar o nome “Valor de X” ao eixo x. Deixe com fonte de tamanho 15.
4. Use a função ylabel() para adicionar o nome “Valor de Y” ao eixo y. Deixe com fonte de tamanho 15.
5. Use a função legend() para adicionar o nome “Função Cúbica” ao gráfico. Deixe com fonte de tamanho 20.
6. Rode novamente seu gráfico, mas mude agora a linha para a cor magenta.

# Análise de dados com Python

---

## Visualização de Dados - Seaborn

1. Importe a biblioteca Seaborn
2. Use a função `load_dataset()` e coloque o dataset `exercises` num dataframe chamado `exercicios`.
3. Use a função `set_style()` e mude para 'dark'.
4. Faça o histograma da variável `pulse` usando a função `distplot()` e o argumento `kde=False`.
5. Agora faça a distribuição de densidade da mesma variável, mas utilizando o argumento `hist=False`.
6. Use a função `swarmplot()` e analise a relação entre as variáveis `kind` (`x`) e `pulse` (`y`).

# Análise de dados com Python

---

## Visualização de Dados - Interpretando gráficos

1. Utilizando os datasets do Seaborn, crie o dataframe voo com o datasets flights.
2. Faça um gráfico de barras com as variáveis year e passengers.
3. Que informação podemos extrair desse gráfico?
4. Use a função boxplot() e verifique a variável passengers.
5. O que podemos depreender desse gráfico?
6. Qual a diferença entre os resultados obtidos com o gráfico de barras e o boxplot?
7. Faça um gráfico de barras com as variáveis month e passengers.
8. Qual conclusão podemos extrair desse gráfico?

## Data Mining - Limpeza dos Dados

1. Crie um dataframe com o arquivo carros.csv
2. Faça uma primeira visualização dos dados.
3. Faça o describe das variáveis numéricas.
4. Faça o describe das variáveis categóricas.
5. Verifique se há dados nulos ou faltantes. Caso haja, substitua pela média das variáveis.
6. Verifique se há dados duplicados. Caso haja, elimine esses dados duplicados do dataframe.

# Análise de dados com Python

---

## Data Mining - Entendendo as variáveis

1. Use o dataframe anterior.
2. Faça histogramas das variáveis numéricas.
3. Escreva pequenas conclusões sobre os histogramas.
4. Faça countplots das variáveis categóricas.
5. Escreva pequenas conclusões sobre os gráficos.

## Data Mining - Alguma estatísticas

1. Qual a média da variável losses? Qual a mediana da variável highway mpg?
2. Qual a média da variável price? Qual a mediana da variável price? Discuta os valores encontrados.
3. Qual a moda da variável fuel type? Qual a moda da variável make?
4. Calcule a moda das variáveis horsepower e price.
5. Escolha 3 variáveis numéricas e faça seus boxplots. Que informações é possível obter?

# Análise de dados com Python

---

## Data Mining - Correlações

1. Use a função corr() e faça a correlação entre as variáveis do dataframe utilizado anteriormente.
2. Quais as variáveis com maior correlação?
3. Quais as variáveis com maior anticorrelação?
4. Escolha 2 correlações e 2 anticorrelações e tentem argumentar o porquê dessa relação.

## Data Mining - Estratificação

1. Vamos analisar os dados sob a perspectiva da quantidade de portas.
2. Faça gráficos de countplot simples para as variáveis riskiness e body.
3. Agora refaça os gráficos anteriores, mas estratificando pela variável doors. (Dica: coloque a variável estratificadora no parâmetro hue.)
4. O que podemos concluir a partir desses gráficos?
5. Faça o countplot da variável aspiration estratificada pela variável doors.
6. O que podemos concluir?
7. Comparando os resultados, qual deles foi mais fácil de obter?

# Análise de dados com Python

---

## Data Mining - Hipóteses

1. Crie mais 2 outras hipóteses que poderiam ser feitas com base nos dados sobre carros.
2. Explique em que situações essas hipóteses poderiam ser utilizadas.
3. Cite 2 hipóteses que não poderiam ser testadas com os dados sobre carros.
4. Que outros dados seriam necessários para testá-las?

## Machine Learning - Entendimento de Negócio

1. Pense em um problema a ser resolvido e defina-o de maneira clara e objetiva.
2. Qual a pergunta a ser respondida?
3. Qual sua hipótese?
4. Quais os dados necessários?
5. Como medir seus resultados? Como saber se meu modelo é bom?
6. Quem usaria os resultados obtidos?

# Análise de dados com Python

---

## Machine Learning - Ciclo de Processos

1. Pesquise o significado de overfitting e reescreva utilizando suas palavras.
2. Pesquise quais as melhores práticas para separação de amostra de treino e teste.
3. É razoável dividir os dados em amostra de treino e teste com os mesmos tamanhos? Por quê?

## Machine Learning - Treino e Teste

1. Utilizando os datasets embutidos no Scikit-Learn crie o conjunto de dados diabetes, utilizando o argumento `return_X_y=True`.
2. Utilizando o dataset diabetes, crie as variáveis preditoras (X) e a variável resposta(y).
3. Procure a documentação da função `train_test_split` e verifique qual o significado do argumento `random_state` e explique qual sua importância.
4. Faça uma separação de amostra de treino e teste onde a amostra de treino contenha 65% dos dados e com `random_state=42`.

## Machine Learning - Regressão

1. Utilize o dataset diabetes criado anteriormente. Pesquise sobre o que se trata.
2. Separe-o em X,y.
3. Faça a separação em amostra de treino e teste, deixando 30% para teste. Use random\_state=42.
4. Importe o LinearRegression, crie o modelo model.
5. Treino o modelo aplicando a função fit() aos dados de treino.
6. Faça a predição, aplicando a função predict aos dados X\_train, chamando o resultado de preditos.
7. Meça o valor de r2\_score. O quanto bom está seu modelo?
8. Faça um gráfico de espalhamento com os valores de y\_train e predito

## Machine Learning - Classificação

1. Utilize os datasets embutidos no Scikit-Learn crie o conjunto de dados cancer, utilizando o argumento `return_X_y=True`. Pesquise sobre o dataset.
2. Separe-o em `X,y`.
3. Faça a separação em amostra de treino e teste, deixando 30% para teste. Use `random_state=42`.
4. Importe o `LogisticRegression`, crie o modelo de classificação `clf`.
5. Treino o modelo aplicando a função `fit()` aos dados de treino.
6. Faça a predição, aplicando a função `predict` aos dados `X_train`, , chamando o resultado de `preditos`.
7. Utilize a função de métrica `classification_report`. O quanto bom está seu modelo?