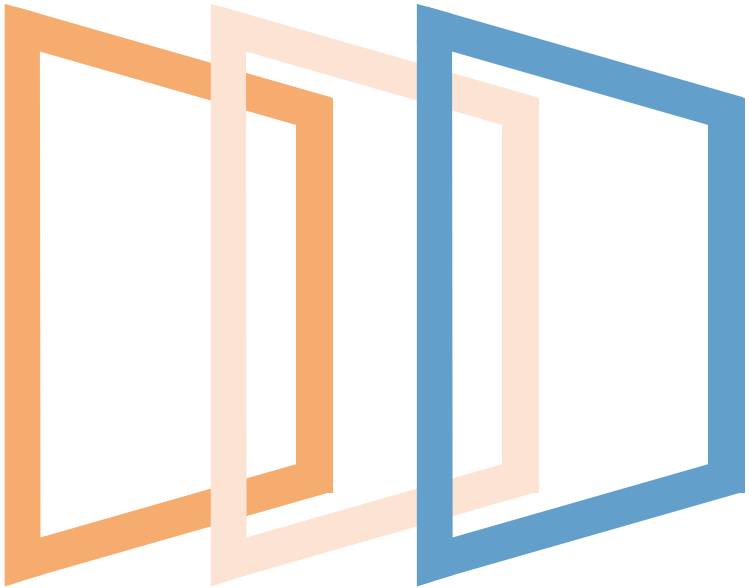
Python Power: Exploração, Manipulação e Análise de Dados com Numpy e Pandas

Thaís Ratis

Inteligência Artificial Brasil, 24.02.2024



minsait

An Indra company

Thaís Ratis

Instrutora

Formação Acadêmica

B.Sc. Ciência da Computação

Esp. Bioinformática

M.Sc. Bioinformática

Ph.D. Bioinformática

Área Profissional

Cientista de Dados

Contato no Teams

talmeidar@minsait.com

Conteúdo programático

Semana 1: Numpy

Semana 2: Pandas

Semana 3: Pré-processamento e estatística

Semana 4: Prova prática

Numpy

Aula 01

Objetivo

Reforçar e aprofundar o conhecimento de NumPy, garantindo aos alunos um domínio sólido da biblioteca, para manipulação e análise eficiente de dados, visando igualar o conhecimento da turma na ferramenta.

Pandas

Aula 02

Objetivo

Capacitar os alunos a manipular e analisar dados de forma eficiente, utilizando as estruturas de dados flexíveis oferecidas pelo Pandas, como DataFrames e Series, para realizar tarefas como limpeza, transformação e análise exploratória.

Pré-processamento e estatística

Aula 03

Objetivo

Capacitar os alunos em pré-processamento e técnicas estatísticas relevantes para análise de dados, incluindo limpeza, imputação de valores ausentes e inferências, garantindo resultados precisos e confiáveis. Objetiva fornecer as ferramentas necessárias para tomar decisões embasadas em dados com qualidade e precisão.

Prova prática

Aula 04

Objetivo

Avaliar a capacidade dos participantes em aplicar os conceitos aprendidos para resolver problemas reais de análise de dados. Os alunos serão desafiados a realizar tarefas como manipulação avançada de dados, aplicação de técnicas estatísticas, e interpretação de resultados utilizando as bibliotecas NumPy e Pandas.

Formato avaliação

Avaliação teórica: formulário;

Avaliação prática: será realizada em trio. Cada trio escolherá uma base no site, realizará a EDA. Por fim, deverão apresentar um *pitch* com um pptx em torno de 5 minutos com os principais pontos analisados, quais os *insights* e conclusões chegaram.

Numpy

Aula 01

Conteúdo programático

1. Carreira de Dados
2. Por onde começar?
3. Revisão Python
4. Numpy

Carreiras de Dados

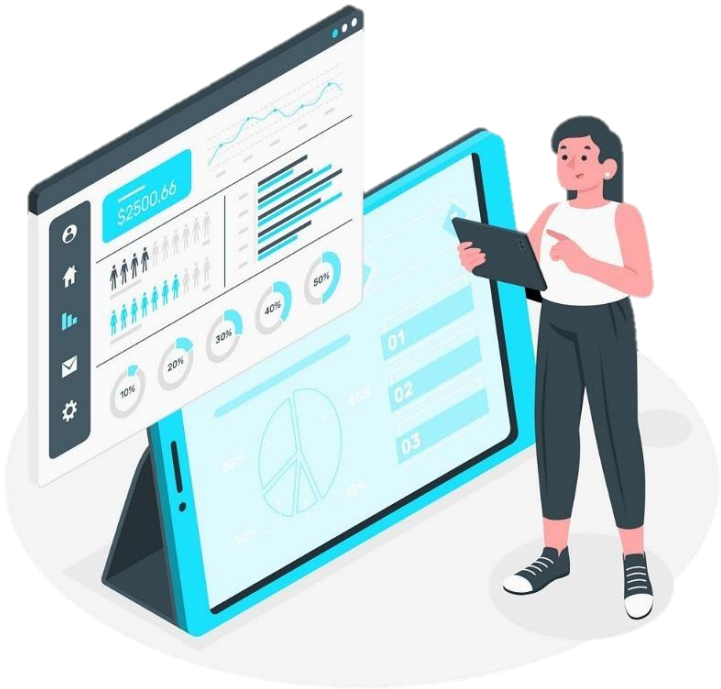
01

Analista de Dados

Utiliza técnicas de análise, visualização e modelagem de dados para conseguir identificar tendências que possam ajudar na tomada de decisão.

1.1

Principais Linguagens e Ferramentas



- SQL
- Power BI
- Tableau
- Google Data Studio
- Qlik Sense

Engenheiro de Dados

Profissional dedicado ao desenvolvimento, construção, teste e manutenção de arquitetura, como um sistema de processamento em grande escala.

1.2

Principais Linguagens e Ferramentas

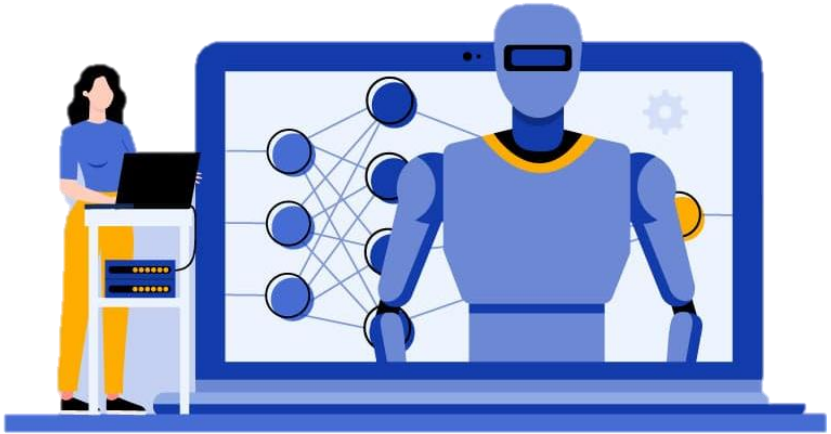
- SQL
- Python
- Hadoop
- Spark
- Kafka
- Hive

Engenheiro de *Machine Learning*

Responsáveis por desenvolver modelos e criar soluções de aprendizado de máquina, com maior foco em garantir que modelos funcionem de forma otimizada e possam ser escalados para dar conta de um grande volume de dados.

1.3

Principais Linguagens e Ferramentas



- SQL
- Python
- Mlflow
- Spark
- Tensorflow

Ciência de Dados

Recebe os dados (estruturados ou não estruturados) e usa suas habilidades em Matemática, Estatística e Ciência da Computação para limpar, tratar, organizá-lo e desenvolver modelos preditivos.

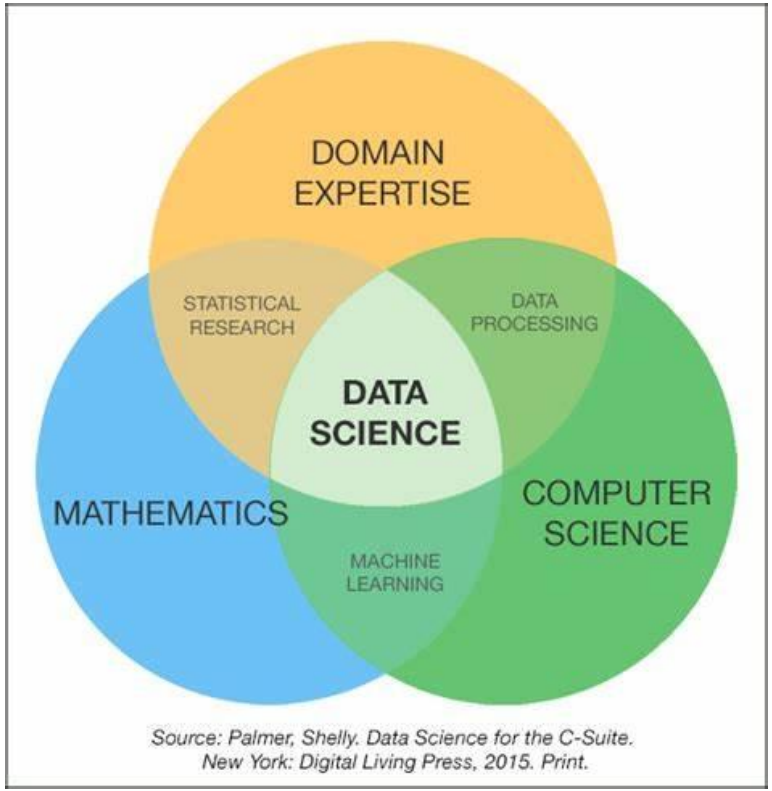
1.4

Principais Linguagens e Ferramentas

- SQL
- Python
- R

- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- Tensorflow

Ciência de dados



Source: Palmer, Shelly. Data Science for the C-Suite. New York: Digital Living Press, 2015. Print.

A ciência de dados é uma área multidisciplinar que combina conhecimentos de estatística, programação e expertise em um domínio específico para extrair *insights* valiosos e informações úteis a partir de grandes conjuntos de dados.

Um cientista de dados é responsável por identificar as perguntas que precisam ser respondidas, coletar os dados relevantes, prepará-los para análise, aplicar modelos estatísticos ou algoritmos de aprendizado de máquina, e interpretar os resultados para oferecer insights acionáveis.

Benefícios: Ajudar as organizações a tomar decisões mais informadas, identificar padrões ocultos, otimizar processos, prever tendências, personalizar experiências do usuário e muito mais.

Pipeline cientista de dados

Pergunta

Aquisição

Limpeza

Exploração

Modelagem

Validação

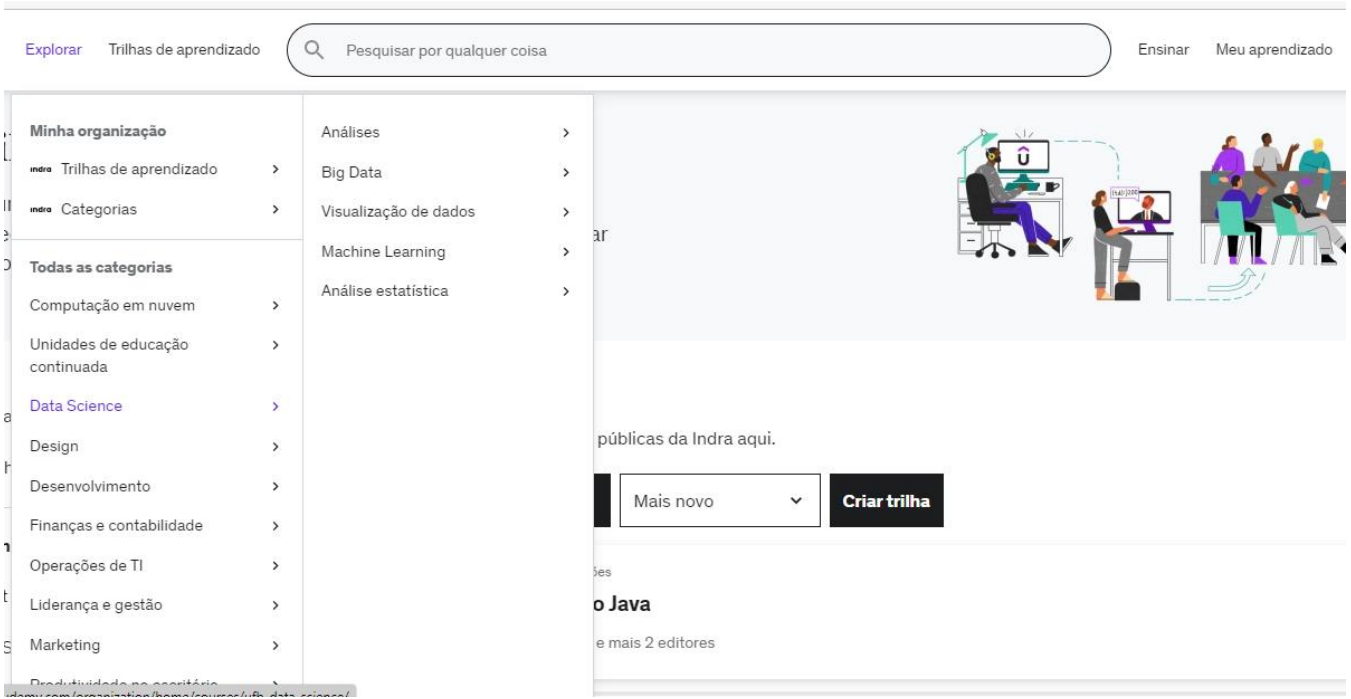
Apresentação

Pré-processamento

Por onde começar?

02

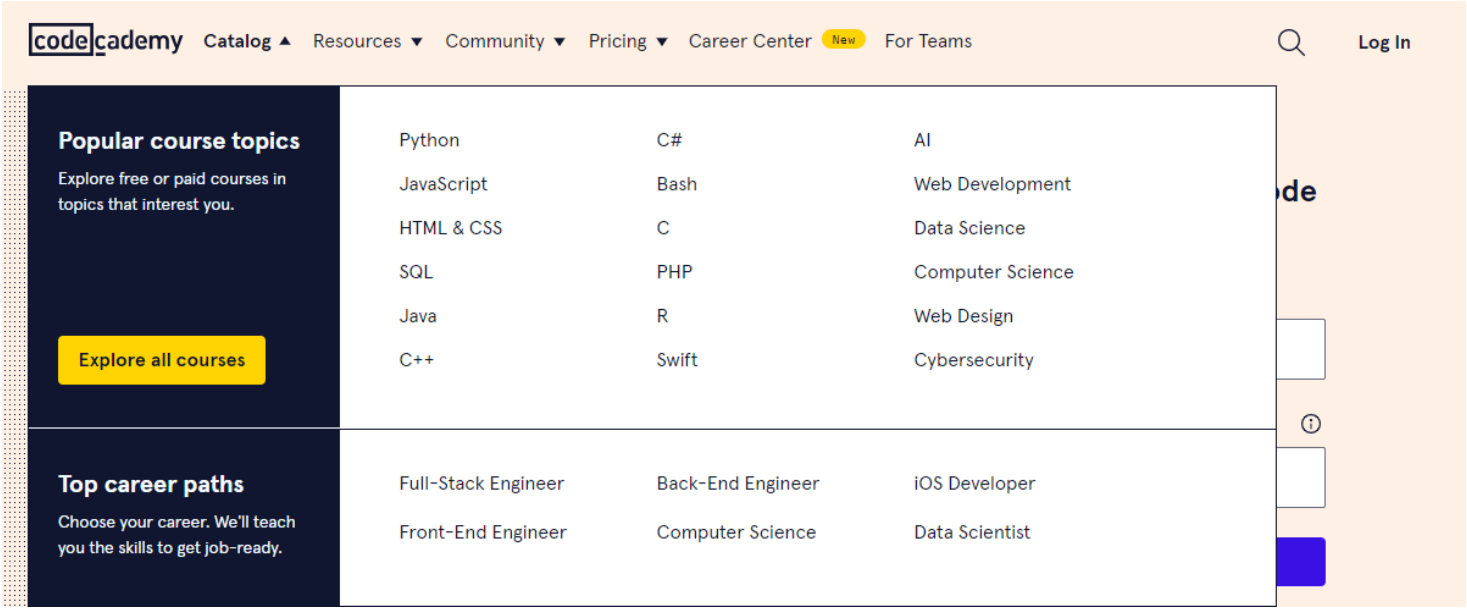
Udemy


Explorar
Trilhas de aprendizado
Pesquisar por qualquer coisa
Ensinar
Meu aprendizado
Minha organização
Trilhas de aprendizado
Categorias
Todas as categorias
Computação em nuvem
Unidades de educação continuada
Data Science
Design
Desenvolvimento
Finanças e contabilidade
Operações de TI
Liderança e gestão
Marketing
Análises
Big Data
Visualização de dados
Machine Learning
Análise estatística
públicas da Indra aqui.
Mais novo
Criar trilha
o Java
e mais 2 editores

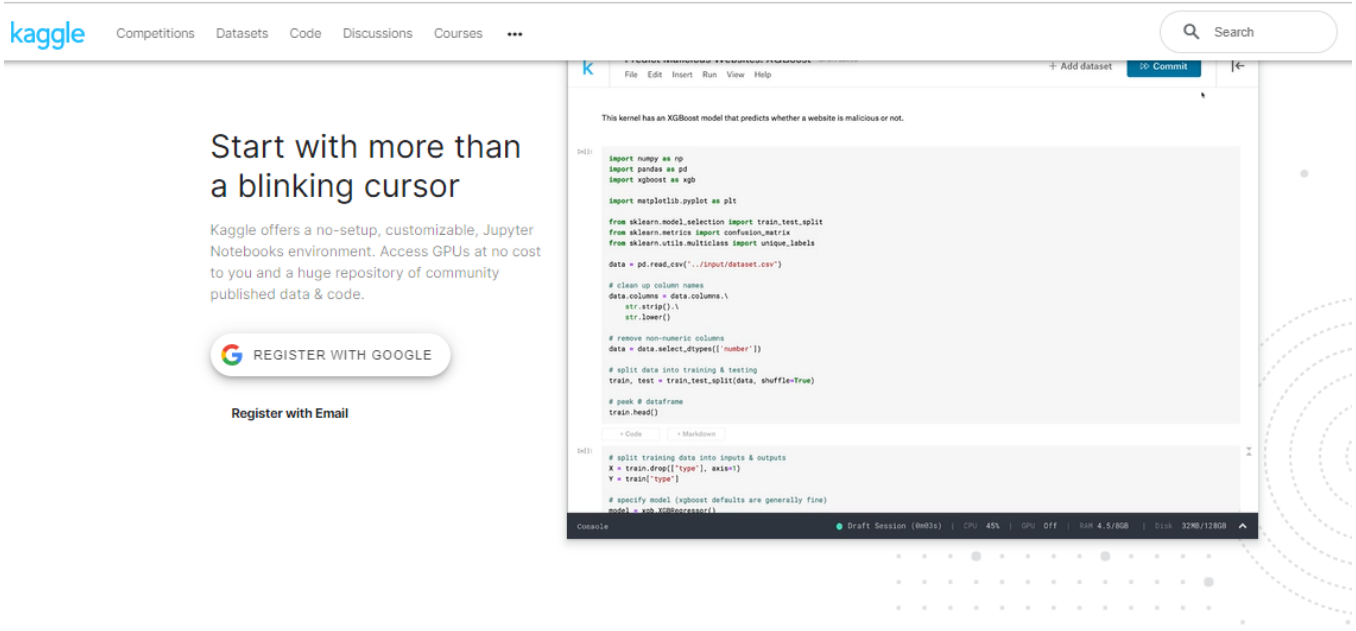
Udemy

https://indra.udemy.com/learning-paths/6258416/

Codecademy


codecademy
Catalog
Resources
Community
Pricing
Career Center
New
For Teams
Log In
Popular course topics
Explore free or paid courses in topics that interest you.
Explore all courses
Python
C#
AI
JavaScript
Bash
Web Development
HTML & CSS
C
Data Science
SQL
PHP
Computer Science
Java
R
Web Design
C++
Swift
Cybersecurity
Top career paths
Choose your career. We'll teach you the skills to get job-ready.
Full-Stack Engineer
Back-End Engineer
iOS Developer
Front-End Engineer
Computer Science
Data Scientist

Kaggle



Inside Kaggle you'll find all the code & data you need to do your data science work. Use over 50,000 public datasets and 400,000 public notebooks to conquer any analysis in no time.

Revisão Python

03

Tipos de Dados Básicos

- Inteiros
- Flutuantes
- Strings
- Booleanos
- Listas
- Tuplas
- Dicionários
- Sets

```python
# Exemplo de tipos de dados básicos
inteiro = 10
flutuante = 3.14
string = "Olá, mundo!"
booleano = True
lista = [1, 2, 3, 4, 5]
tupla = (1, 2, 3)
dicionario = {'nome': 'João', 'idade': 25}
```

Listas

- São mutáveis, o que significa que você pode adicionar, remover ou modificar elementos após a criação da lista;
- Elas são definidas usando colchetes [ ];
- Os elementos de uma lista podem ser acessados por meio de índices;
- As listas são adequadas quando você precisa de uma coleção de itens que possa ser modificada e que possua elementos duplicados.

```
minha_lista = [1, 2, 3]
minha_lista.append(4)
minha_lista[0] = 0
print(minha_lista)  # Saída: [0, 2, 3, 4]
```

Tuplas

- São imutáveis, o que significa que, uma vez criadas, não é possível adicionar, remover ou modificar elementos;
- Elas são definidas usando parênteses ( );
- Os elementos de uma tupla também podem ser acessados por meio de índices;
- As tuplas são ideais quando você tem uma coleção de itens que não deve ser alterada.

```
minha_tupla = (1, 2, 3)
# minha_tupla.append(4)  # Isso resultaria em um erro
# minha_tupla[0] = 0      # Isso resultaria em um erro
print(minha_tupla)  # Saída: (1, 2, 3)

print(minha_tupla[0])  # Saída: 1
```

Dicionários

- Os dicionários são coleções não ordenadas de pares chave-valor;
- Eles são definidos usando chaves { };
- Os elementos de um dicionário são acessados por suas chaves, não por índices;
- Os dicionários são adequados quando você precisa associar um valor a uma chave para fins de busca eficiente.

```
meu_dicionario = {'nome': 'João', 'idade': 25, 'cidade': 'São Paulo'}
print(meu_dicionario['idade'])  # Saída: 25
meu_dicionario['profissão'] = 'Engenheiro'
print(meu_dicionario)  # Saída: {'nome': 'João', 'idade': 25, 'cidade': 'São Paulo', 'profissão': 'Engenheiro'}
```

Dicionários - Acessos

Acessando valores pelas chaves:

```python
meu_dicionario = {'nome': 'João', 'idade': 25, 'cidade': 'São Paulo'}

# Acessando valores pelas chaves
print(meu_dicionario['nome'])    # Saída: João
print(meu_dicionario['idade'])   # Saída: 25
print(meu_dicionario['cidade'])  # Saída: São Paulo
```

Acessando chaves e valores usando métodos:

```python
# Acessando chaves
chaves = meu_dicionario.keys()
print(chaves)  # Saída: dict_keys(['nome', 'idade', 'cidade'])

# Acessando valores
valores = meu_dicionario.values()
print(valores)  # Saída: dict_values(['João', 25, 'São Paulo'])
```

Iterando sobre um dicionário:

```python
# Iterando sobre chaves e valores simultaneamente
for chave, valor in meu_dicionario.items():
    print(f'Chave: {chave}, Valor: {valor}')
# Saída:
# Chave: nome, Valor: João
# Chave: idade, Valor: 25
# Chave: cidade, Valor: São Paulo
```

Sets

- São mutáveis, o que significa que você pode adicionar, remover ou modificar elementos após a criação;
- São definidos por chaves { }, mas apenas com elementos, não com pares chave-valor;
- Os elementos podem ser acessados por usando operações como in ou not in, ou então iterar sobre os elementos do conjunto;
- Útil para realizar operações de conjunto como união, interseção e diferença;
- Não permite elementos duplicados.

```python
# Criando um conjunto
meu_conjunto = {1, 2, 3, 4, 5}

# Verificando se um elemento está presente no conjunto
print(1 in meu_conjunto)   # Saída: True
print(6 in meu_conjunto)   # Saída: False

# Iterando sobre os elementos do conjunto
for elemento in meu_conjunto:
    print(elemento)
# Saída:
# 1
# 2
# 3
# 4
# 5
```

Sets

```python
# União de conjuntos
conjunto1 = {1, 2, 3}
conjunto2 = {3, 4, 5}
uniao = conjunto1.union(conjunto2)
print(uniao)  # Saída: {1, 2, 3, 4, 5}

# Interseção de conjuntos
intersecao = conjunto1.intersection(conjunto2)
print(intersecao)  # Saída: {3}

# Diferença de conjuntos
diferenca = conjunto1.difference(conjunto2)
print(diferenca)  # Saída: {1, 2}

# Teste de subconjunto
subconjunto = {1, 2}
print(subconjunto.issubset(conjunto1))  # Saída: True
```

Operadores Básicos

- Aritméticos
- Comparação
- Lógicos
- Atribuição

```python
# Exemplo de operadores básicos
a = 10
b = 5

# Aritméticos
soma = a + b
subtracao = a - b
multiplicacao = a * b
divisao = a / b

# Comparação
igual = a == b
maior_que = a > b

# Lógicos
e_logico = (a > 0) and (b > 0)
ou_logico = (a > 0) or (b > 0)

# Atribuição
a += 1  # Equivalente a: a = a + 1
```

Estruturas de Controle

- Condicionais (if, elif, else)
- Loops (for, while)

```python
# Exemplo de estruturas de controle
idade = 18

# Condicionais
if idade >= 18:
    print("Você é maior de idade")
elif idade < 0:
    print("Idade inválida")
else:
    print("Você é menor de idade")

# Loops
for i in range(5):
    print(i)

contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Funções

- Definindo funções
- Parâmetros e argumentos
- Retorno de valores

```python
# Exemplo de funções
def soma(a, b):
    return a + b

resultado = soma(5, 3)
print(resultado)  # Saída: 8
```

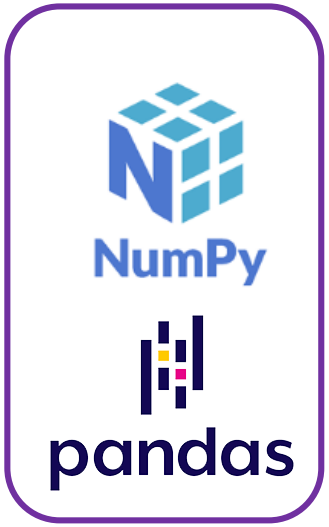
Funções

- Importando módulos
- Criando e usando pacotes

```python
# Exemplo de módulos e pacotes
import math

raiz_quadrada = math.sqrt(16)
print(raiz_quadrada)  # Saída: 4.0

# Criando e usando pacotes
# Exemplo de estrutura de pacote: meu_pacote/minha_funcao.py
# Dentro de minha_funcao.py
def minha_funcao():
    print("Esta é a minha função")

# No arquivo principal
from meu_pacote.minha_funcao import minha_funcao
minha_funcao()  # Saída: Esta é a minha função
```

Principais bibliotecas para dados e ML

Principais bibliotecas para dados e ML



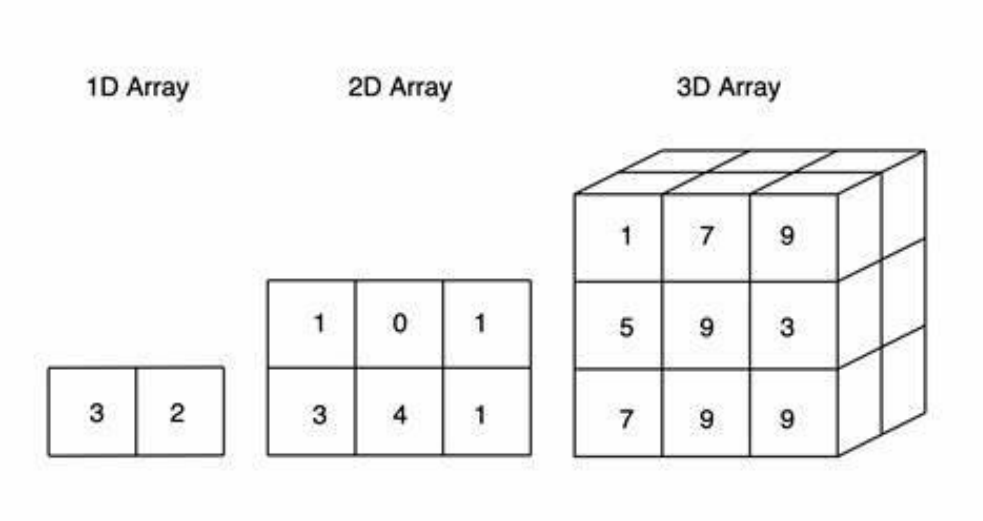
Numpy

04

Numpy

NumPy é uma biblioteca fundamental para computação científica em Python. Ela fornece suporte para matrizes multidimensionais e funções matemáticas de alto desempenho, tornando-a essencial para operações numéricas e manipulação de dados.

A estrutura de dados principal do Numpy é o Array.



Numpy

Tipos de cálculos numéricos realizados por numpy são amplamente utilizados em tarefas como:

- **Tarefas matemáticas:** NumPy é bastante útil para executar várias tarefas matemáticas como integração numérica, diferenciação, interpolação, extrapolação e muitas outras. O NumPy possui também funções incorporadas para álgebra linear e geração de números aleatórios. É uma biblioteca que pode ser usada em conjunto do SciPy e Matplotlib.

- **Processamento de Imagem e Computação Gráfica:** Imagens no computador são representadas como Arrays Multidimensionais de números. NumPy torna-se a escolha mais natural para o mesmo. O NumPy, na verdade, fornece algumas excelentes funções de biblioteca para rápida manipulação de imagens. Alguns exemplos são o espelhamento de uma imagem, a rotação de uma imagem por um determinado ângulo etc.

- **Modelos de Machine Learning:** Ao escrever algoritmos de Machine Learning, supõe-se que se realize vários cálculos numéricos em Array. Por exemplo, multiplicação de Arrays, transposição, adição, etc. O NumPy fornece uma excelente biblioteca para cálculos fáceis (em termos de escrita de código) e rápidos (em termos de velocidade). Os Arrays NumPy são usados para armazenar os dados de treinamento, bem como os parâmetros dos modelos de Machine Learning.

Numpy

```
In [1]: # Importando a biblioteca numpy
        import numpy as np

In [2]: # Criando um array de 1 dimensão
        one_dim = np.array([1,2,3,4])

In [3]: type(one_dim)

Out[3]: numpy.ndarray

In [4]: # Imprimindo um array.
        one_dim.ndim

Out[4]: 1

In [5]: # Criando um array de 2 dimensões.
        two_dim = np.array([(1,2,3), (4,5,6)])

In [6]: # Imprimindo o array
        two_dim.ndim

Out[6]: 2
```

```
In [7]: # Cria um array de números aleatórios.
        # Um array de 5 linhas e duas dimensões.
        np.random.random((5,2))

Out[7]: array([[0.15602492, 0.00436481],
               [0.0056985 , 0.64874345],
               [0.33440417, 0.62976834],
               [0.78467376, 0.92567222],
               [0.59192061, 0.90153326]])

In [8]: # Cria um array com valores esparsos iniciando com o valor 10, menor que 50 e incrementando de 5 em 5.
        np.arange(10,50,5)

Out[8]: array([10, 15, 20, 25, 30, 35, 40, 45])

In [9]: # cria um array linear de 0 a 2 de no máximo 9 elementos.
        np.linspace(0,2,9)

Out[9]: array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])

In [10]: # Cria um array de valores zero.
         # Cria um array com 3 linhas e 4 dimensões.
         np.zeros((3,4))

Out[10]: array([[0., 0., 0., 0.],
                [0., 0., 0., 0.],
                [0., 0., 0., 0.]])
```

Numpy - Observações

- Não é possível ter mais de um tipo de dado em um único array do Numpy.
- É possível criar arrays a partir de dados obtidos de arquivos (por exemplo, extensões txt e csv).
- Podemos transformar uma lista em um array.
- Podemos transformar arrays em listas.
- Podemos ver quantas linhas e colunas tem num array.
- Entre outras operações.

Numpy vs Listas

- Arrays Numpy permitem fazermos operações em **arrays inteiros** de forma rápida.
- Listas não permitem operações em todos os elementos da lista.
- Para operações em todos os elementos é preciso iterar sobre toda a lista.
- Listas em Python armazenam diferentes tipos de objetos.
- Arrays Numpy considera todos os elementos de tipos distintos como strings.

Listas vs Arrays

Listas

```
In [11]: # Criando uma lista em Python.
         lista = [1,2,3]

In [12]: lista
Out[12]: [1, 2, 3]

In [13]: # Multiplicar valores da lista por 2.
         lista * 2
Out[13]: [1, 2, 3, 1, 2, 3]
```

```
In [17]: # Calcular IMC de pessoas.
         pesos = [67,81,120,90]
         altura = [1.68,1.70,1.75,1.85]

In [18]: # Faz o calculo usando as listas
         pesos / altura ** 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\TALMEI~1\AppData\Local\Temp/ipykernel_12632/2643097884.py in <module>
      1 # Faz o calculo usando as listas
----> 2 pesos / altura ** 2

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Arrays

```
In [14]: # Transforme a variável lista em um array Numpy
         lista = np.array(lista)

In [15]: # Imprimindo o tipo do objeto.
         type(lista)
Out[15]: numpy.ndarray

In [16]: # Multiplicando cada elemento por 2.
         lista * 2
Out[16]: array([2, 4, 6])
```

```
In [19]: # Transforme isso em arrays numpy
         pesos = np.array(pesos)
         altura = np.array(altura)

In [20]: # Imprime o calculo de cada valor
         pesos / altura **2
Out[20]: array([23.73866213, 28.02768166, 39.18367347, 26.29656684])
```

Listas vs Arrays

Listas

```
In [52]: lista = [1,3,'Casa',True]
         lista

Out[52]: [1, 3, 'Casa', True]

In [53]: lista * 2

Out[53]: [1, 3, 'Casa', True, 1, 3, 'Casa', True]
```

Arrays

```
In [6]: # Arrays Numpy armazena elementos como strings quando estes não são inteiros ou float
        a = np.array([1,3,'Casa',True])

In [7]: a

Out[7]: array(['1', '3', 'Casa', 'True'], dtype='<U11')

In [23]: a * 2

---------------------------------------------------------------------------
UFuncTypeError                            Traceback (most recent call last)
C:\Users\TALMEI~1\AppData\Local\Temp/ipykernel_12632/2172713831.py in <module>
----> 1 a * 2

UFuncTypeError: ufunc 'multiply' did not contain a loop with signature matching types (dtype('<U11'), dtype('<U11')) -> dtype
('<U11')

In [8]: # repete o valor do array 2 vezes.
        np.tile(a, 2)

Out[8]: array(['1', '3', 'Casa', 'True', '1', '3', 'Casa', 'True'], dtype='<U11')
```

Listas vs Arrays – Operações matemáticas

Listas

```
In [55]: lista = [i for i in range(10,50,5)]
         lista

Out[55]: [10, 15, 20, 25, 30, 35, 40, 45]
```

```
In [57]: soma = 0
         for i in lista:
             soma +=i

         print('Média:' + str(soma/len(lista)))

         Média:27.5
```

Arrays

```
In [56]: n_array = np.arange(10,50,5)
         n_array

Out[56]: array([10, 15, 20, 25, 30, 35, 40, 45])
```

```
In [58]: n_array.mean()

Out[58]: 27.5
```

Listas vs Arrays – Acessando elementos

Listas

```
In [65]: lista
Out[65]: [10, 15, 20, 25, 30, 35, 40, 45]

In [68]: lista[0]
Out[68]: 10

In [72]: lista[lista>20]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\TALMEI~1\AppData\Local\Temp/ipykernel_12632/2359260420.py in <module>
----> 1 lista[lista>20]

TypeError: '>' not supported between instances of 'list' and 'int'
```

Arrays

```
In [67]: n_array
Out[67]: array([10, 15, 20, 25, 30, 35, 40, 45])

In [69]: n_array[0]
Out[69]: 10

In [73]: n_array[n_array>20]
Out[73]: array([25, 30, 35, 40, 45])
```

Listas vs Arrays – Acessando elementos

Listas

```
1 ano = [[1990, 1995, 1997, 1999], [2000, 2005, 2010, 2015]]
2 print(ano)
3 print(type(ano))
```

```
[[1990, 1995, 1997, 1999], [2000, 2005, 2010, 2015]]
<class 'list'>
```

```
1 #retornar os anos maiores que 1999
2 #usando listas:
3 print(ano > 1999)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-96c707c125bd> in <cell line: 3>()
      1 #retornar os anos maiores que 1999
      2 #usando listas:
----> 3 print(ano > 1999)

TypeError: '>' not supported between instances of 'list' and 'int'

PESQUISAR NO STACK OVERFLOW
```

Arrays

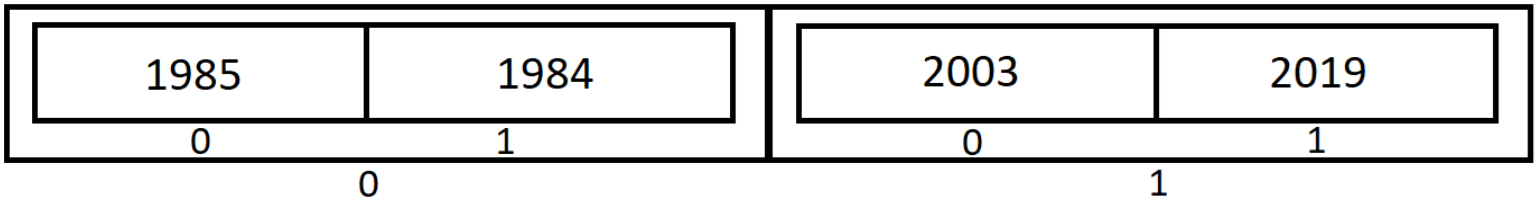
```
1 matriz_anos = np.array(ano)
2 print(matriz_anos)
3 print(type(matriz_anos))
```

```
[[1990 1995 1997 1999]
 [2000 2005 2010 2015]]
<class 'numpy.ndarray'>
```

```
1 #retornar os anos maiores que 1999
2 #usando matriz do numpy:
3 print(matriz_anos > 1999)
4 print(matriz_anos[matriz_anos > 1999])
```

```
[[False False False False]
 [ True  True  True  True]]
[2000 2005 2010 2015]
```

Arrays – Acessando elementos



anos[0][0] = 1985

anos[0][1] = 1984

anos[1][0] = 2003

anos[1][1] = 2019

anos[0,0] = 1985

anos[0,1] = 1984

anos[1,0] = 2003

anos[1,1] = 2019

Listas vs Arrays – Tempo de execução

```
import time
l1 = list(range(100000))
l2 = np.arange(100000)

start_time = time.time()
%time for i in range(len(l1)): l1[i] = l1[i]*2
end_time = time.time()
cpu_time = end_time - start_time

print("Tempo de CPU:", cpu_time, "segundos")

start_time = time.time()
%time l = [i*2 for i in l1]
end_time = time.time()
cpu_time = end_time - start_time

print("Tempo de CPU:", cpu_time, "segundos")

start_time = time.time()
%time l2 = l2 * 2
end_time = time.time()
cpu_time = end_time - start_time

print("Tempo de CPU:", cpu_time, "segundos")
#%time for i in range(100)
```

```
Wall time: 26.2 ms
Tempo de CPU: 0.028210878372192383 segundos
Wall time: 11 ms
Tempo de CPU: 0.01307225227355957 segundos
Wall time: 0 ns
Tempo de CPU: 0.0010726451873779297 segundos
```

JUPYTER NOTEBOOK

Numpy

Embora NumPy forneça estruturas e ferramentas fundamentais que facilitam o trabalho com dados, existem algumas limitações para sua utilidade:

- A falta de suporte para nomes de colunas nos força a enquadrar as questões como arrays;
- O suporte a apenas um tipo de dado, torna difícil trabalhar com dados que contenham números e strings;
- Não existem muitos padrões para análises.

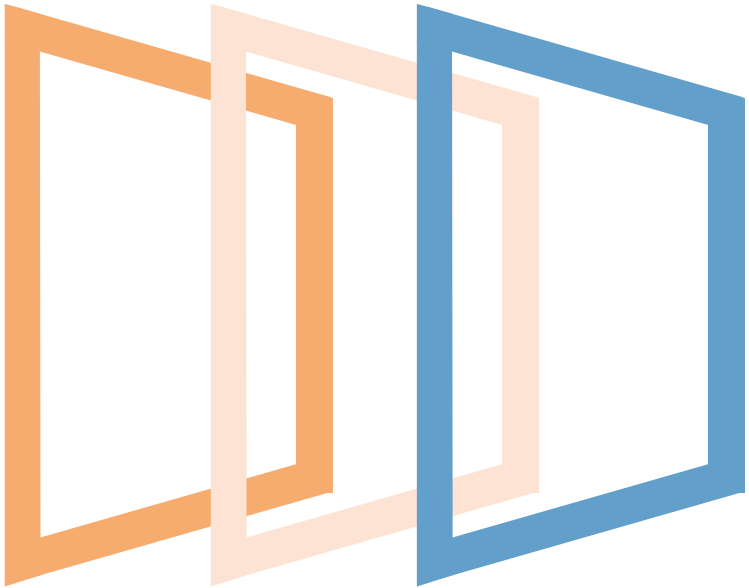
Referências

- NumPy
- Entendendo a biblioteca NumPy. O que é o NumPy? | by Luiz Santiago Jr. | Ensina.AI | Medium
- Introdução ao Numerical Python (Numpy) | OPL (ufc.br)
- https://stack-academy.memberkit.com.br/32408-data-science-do-zero

Python Power: Exploração, Manipulação e Análise de Dados com Numpy e Pandas

Thaís Ratis

Inteligência Artificial Brasil, 24.02.2024



minsait

An Indra company