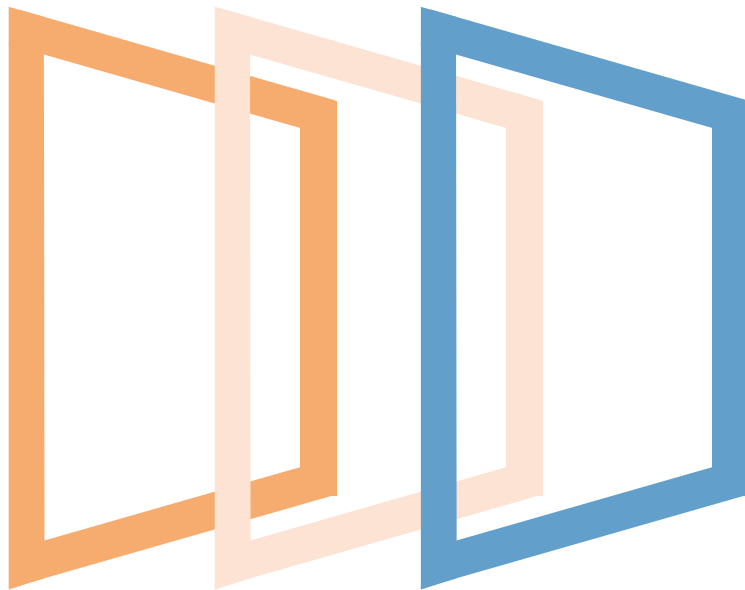


Trilhando Caminhos em Ciência de Dados

Thaís Ratis

Práticas Tecnológicas, 12.08.2023

minsoit



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

Aula 1: Revisão manipulação de dados

Aula 2: Visualização de dados

Aula 3: Pré-processamento e estatística

Aula 4: Visão Geral de ML

Aula 5: Prova prática

Conteúdo programático

Revisão:

1. Numpy
2. Pandas

**Manipulação
de dados**

Conteúdo programático

- Revisão:
1. Numpy
 2. Pandas

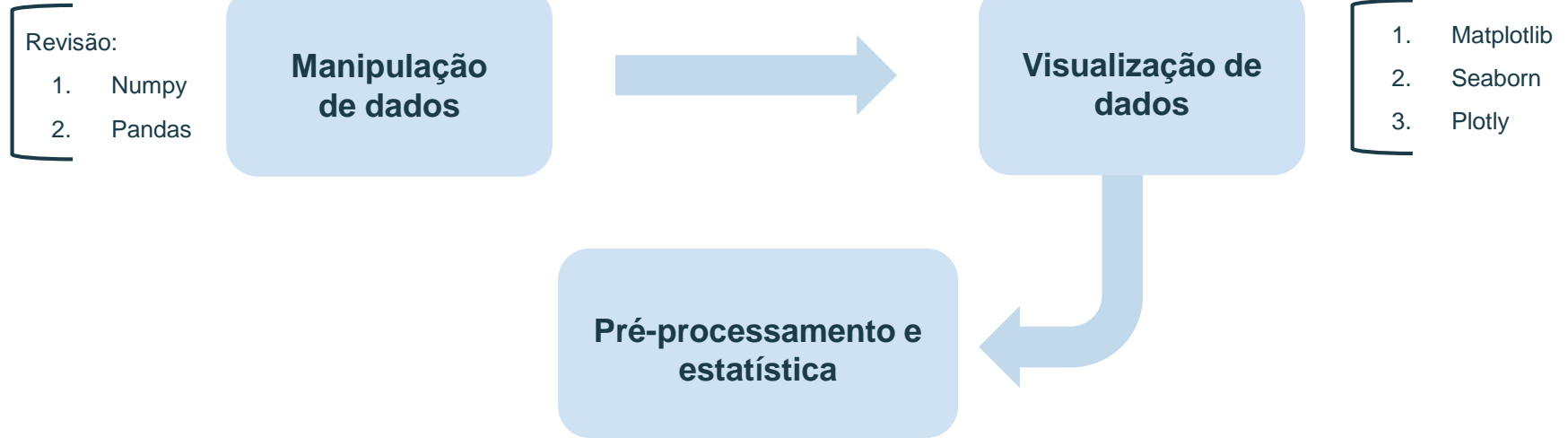
**Manipulação
de dados**



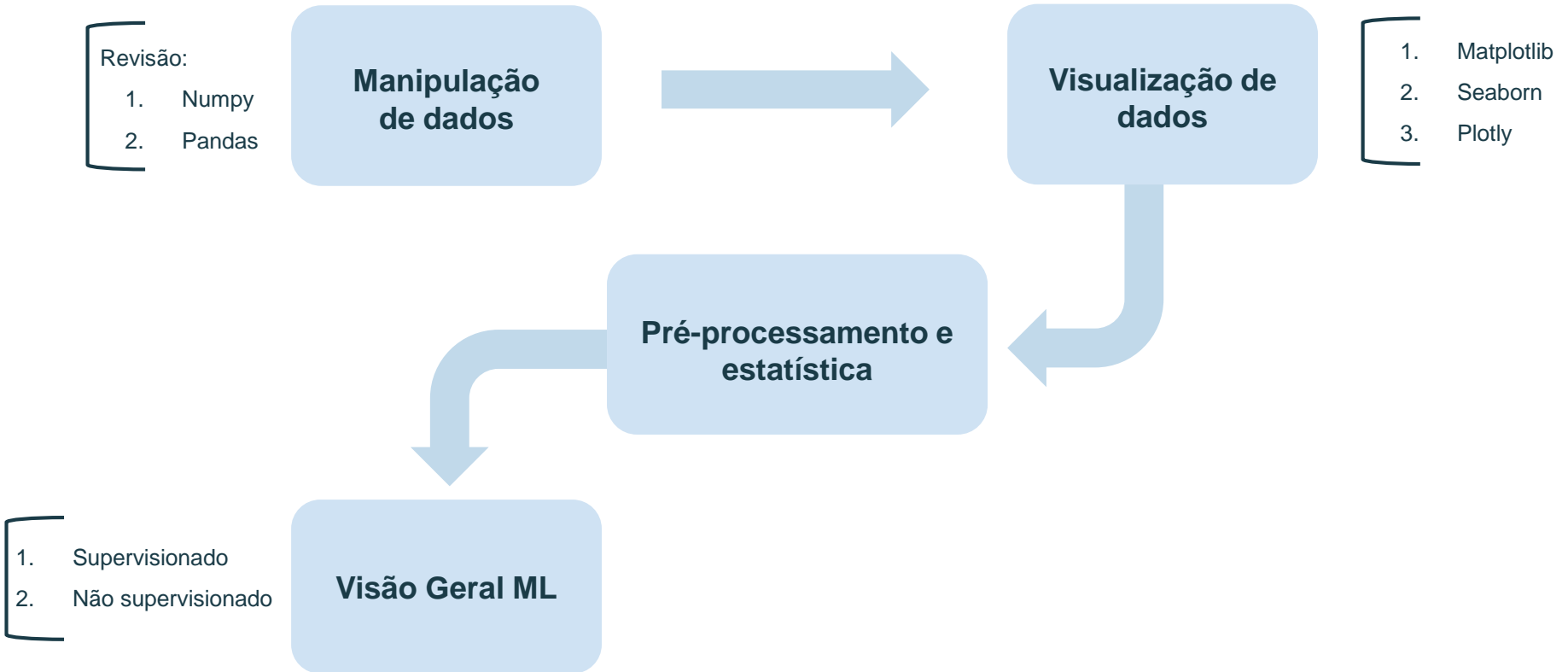
**Visualização de
dados**

1. Matplotlib
2. Seaborn
3. Plotly

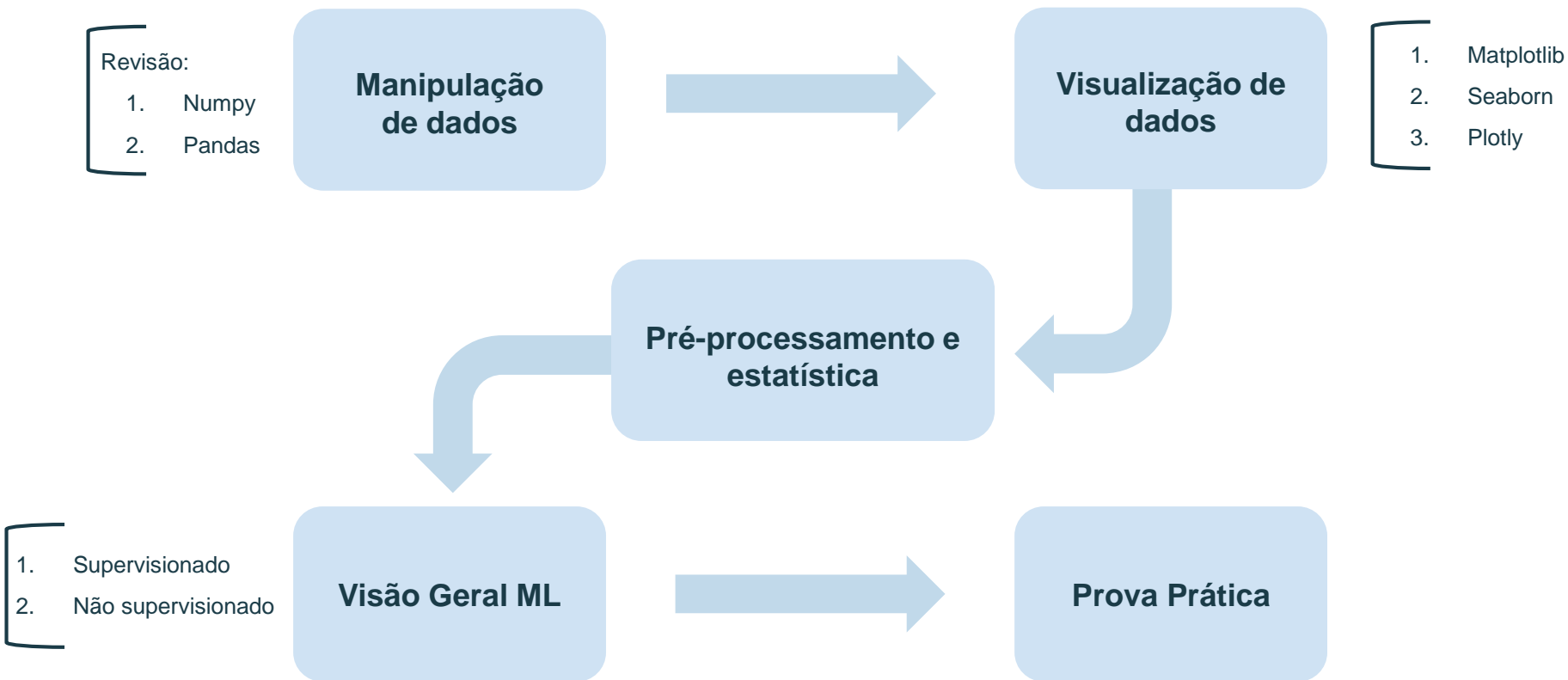
Conteúdo programático



Conteúdo programático



Conteúdo programático



Revisão manipulação de dados

Aula 01

Objetivo

Reforçar e aprofundar o conhecimento de NumPy e Pandas para ciência de dados, garantindo aos alunos um domínio sólido dessas bibliotecas, para manipulação e análise eficiente de dados, visando igualar o conhecimento da turma em ferramentas de ciência de dados.

Visualização de dados

Aula 02

Objetivo

Capacitar os estudantes em técnicas e ferramentas de visualização de informações complexas, desenvolvendo habilidades para criar gráficos e diagramas que identifiquem padrões e tendências nos dados. Promove a representação visual compreensível dos dados, facilitando a tomada de decisões informadas e compartilhamento de descobertas impactantes.

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.

Visão Geral de ML

Aula 04

Objetivo

Oferecer uma base sólida em conceitos e técnicas fundamentais de Machine Learning (ML), abordando diferentes tipos de algoritmos e princípios de aprendizado supervisionado e não supervisionado, como regressão, classificação e agrupamento. Os alunos serão capacitados a identificar problemas adequados para ML, selecionar e avaliar algoritmos, interpretar resultados e aplicar técnicas em análise de dados, preparação e avaliação de modelos.

Prova prática

Aula 05

Objetivo

Avaliar a capacidade dos alunos de aplicar conhecimentos em problemas práticos, incluindo análise e interpretação de dados, pré-processamento, seleção de técnicas de Machine Learning e interpretação de resultados. Além disso, proporciona uma oportunidade para demonstrar habilidades colaborativas e eficazes em cenários do mundo real.

Formato avaliação

Avaliação teórica: formulário;

Avaliação prática: será realizada em trio. Cada trio escolherá uma base no site, realizará a EDA e a predição com ML. 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 e os resultados da predição.

Revisão manipulação de dados

Aula 01

Conteúdo programático

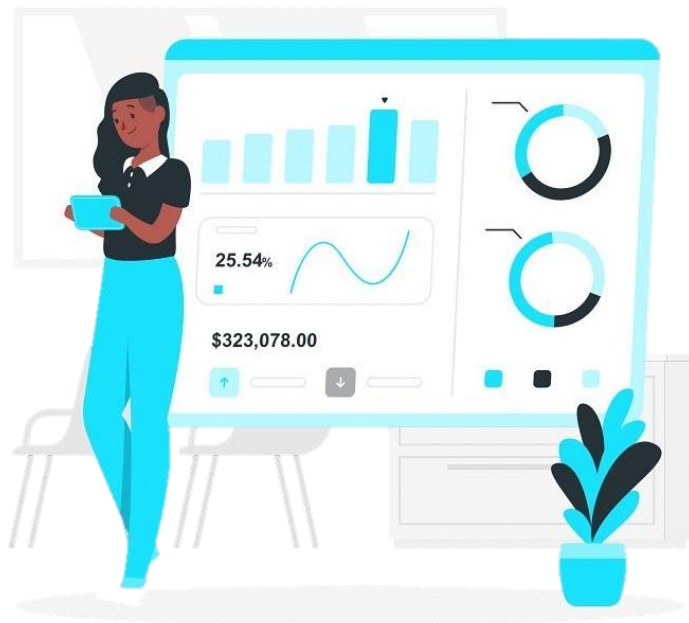
1. Carreira de Dados
2. Estatísticas Publicadas – State of data 2022
3. Por onde começar?
4. Principais bibliotecas
5. Numpy
6. Pandas

Carreiras de Dados

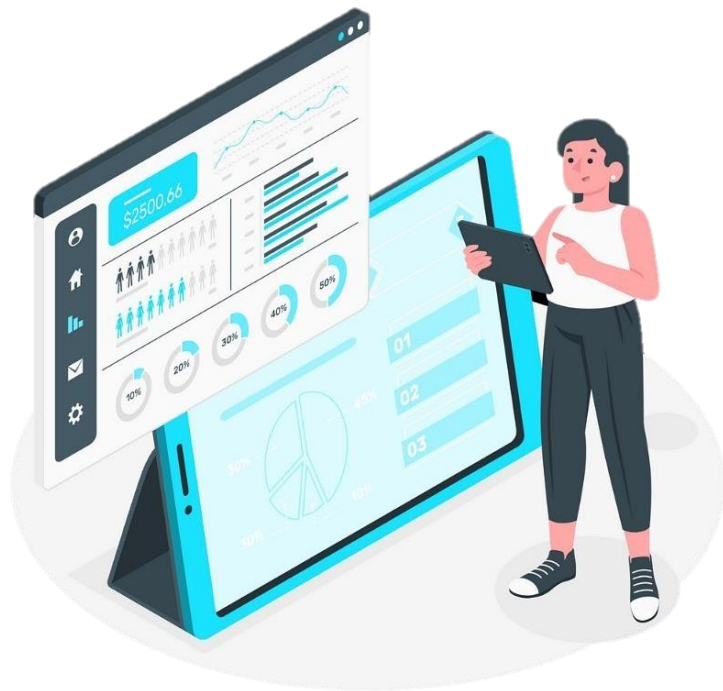
01

Analista de Dados

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



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.



Principais Linguagens e Ferramentas



 SQL

 Hadoop

 Python

 Spark

 Kafka

 Hive

Engenheiro de *Machine Learning*

Conhece os vários modelos de aprendizado de máquina existentes atualmente, e sabe qual deles se aplica à análise de dados que está sendo realizada naquele momento.



Principais Linguagens e Ferramentas



 SQL

 Mlflow

 Python

 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 e organizá-lo.



Principais Linguagens e Ferramentas



 SQL

 Pandas

 Python

 Matplotlib

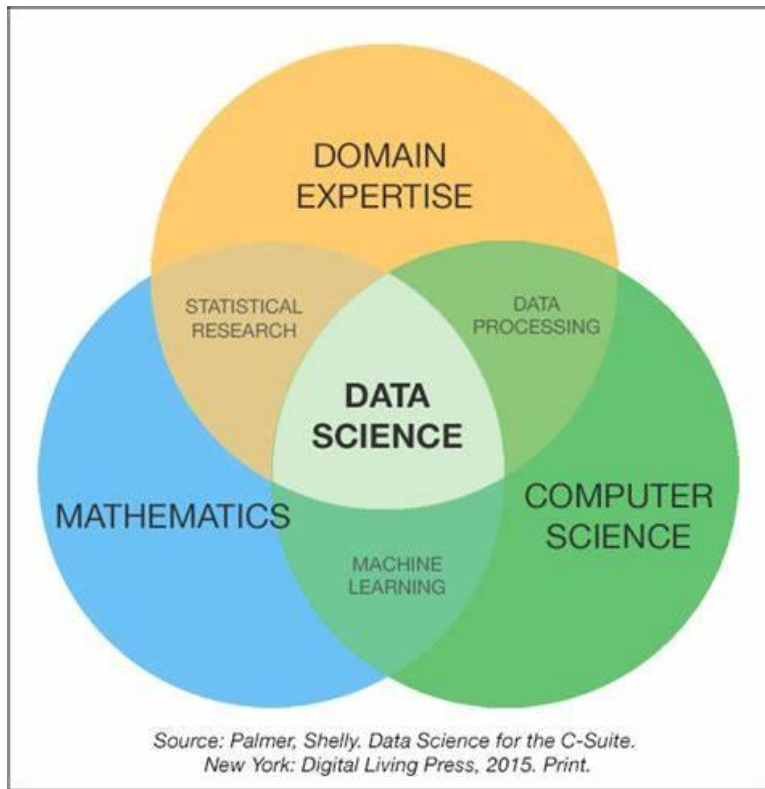
 Seaborn

 R

 Scikit-learn

 Tensorflow

Ciência de dados

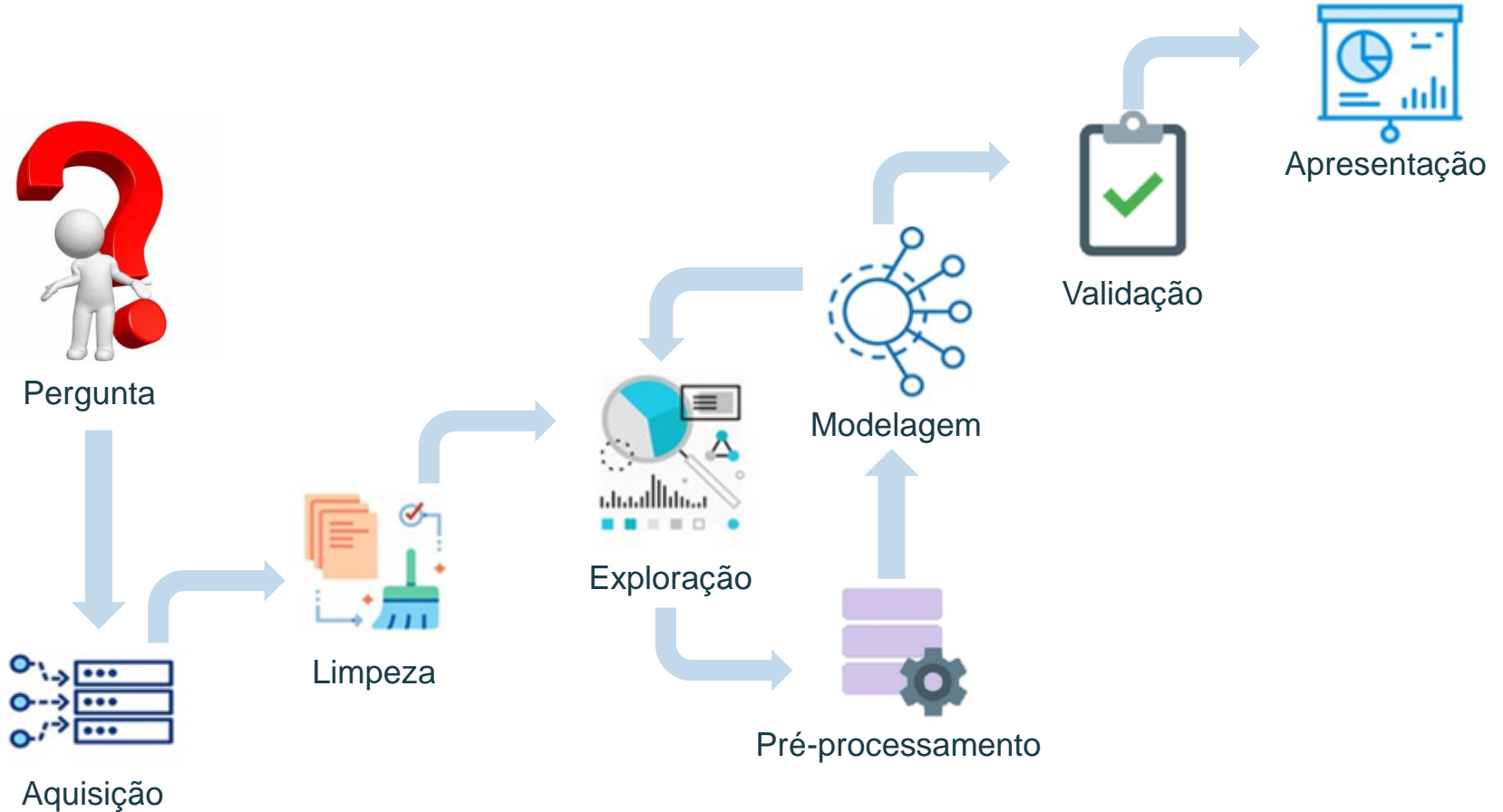


A ciência de dados é uma área multidisciplinar que combina conhecimentos de estatística, programação e expertise em um domínio específico para extrair *insights* valiosos e informações úteis a partir de grandes conjuntos de dados.

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



Estatísticas Publicadas

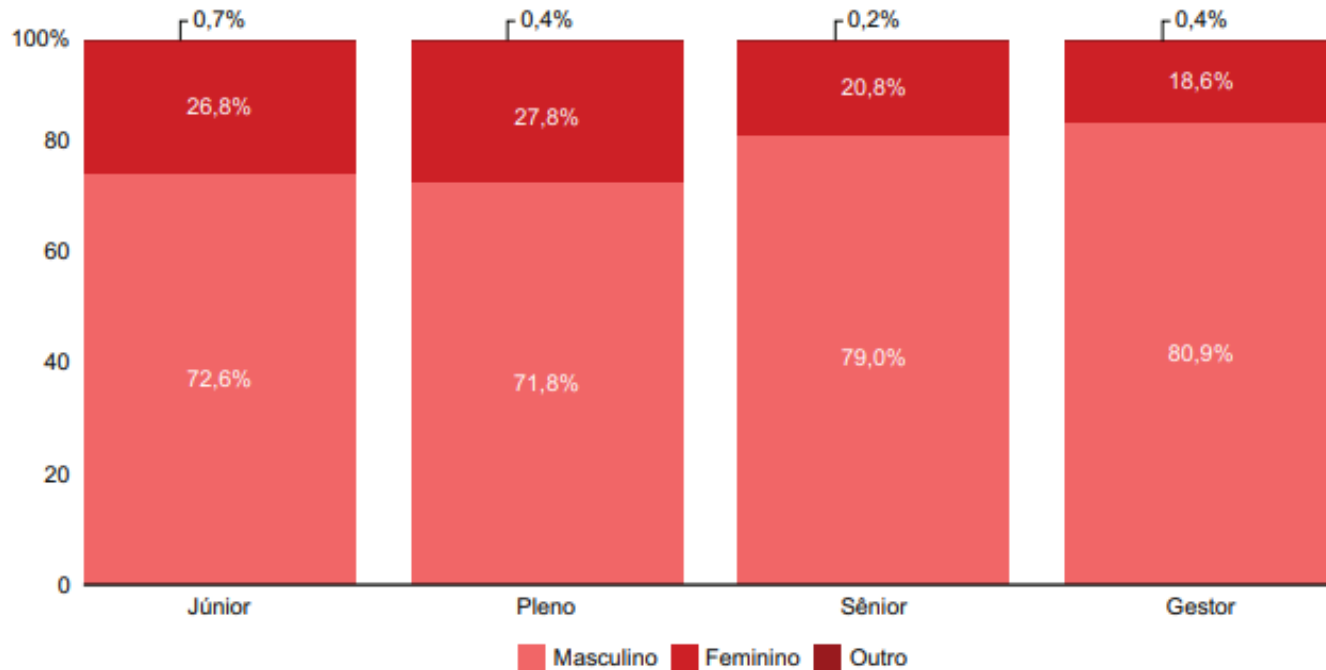
State of Data 2022

Um raio-x dos profissionais de dados do Brasil

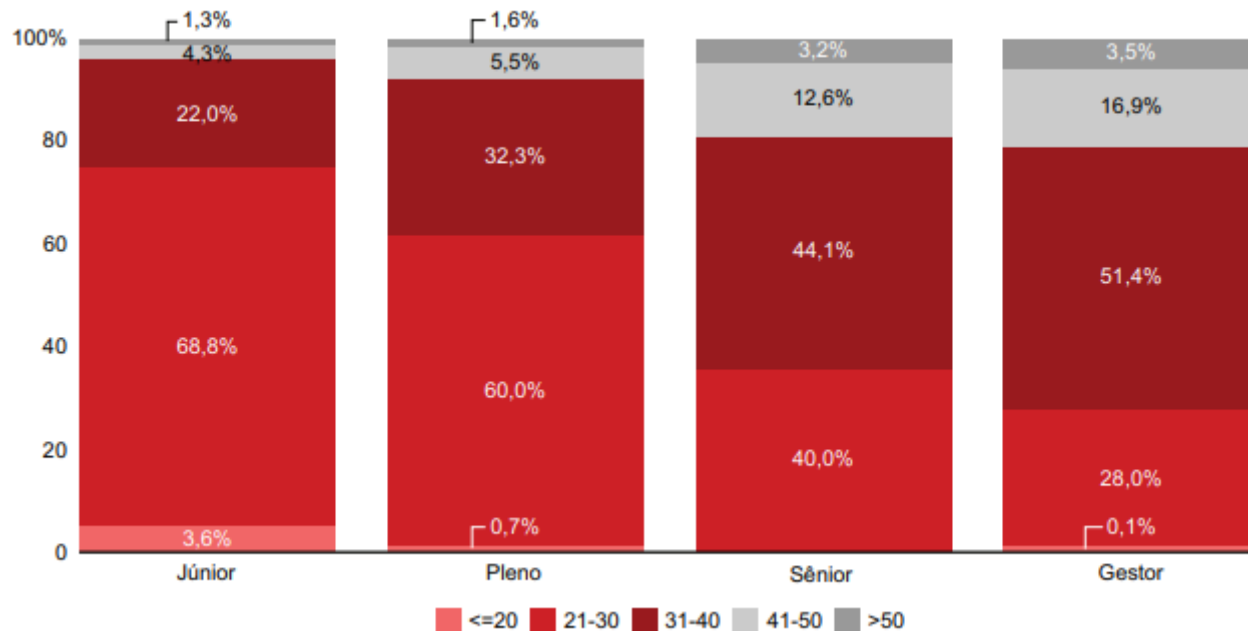


02

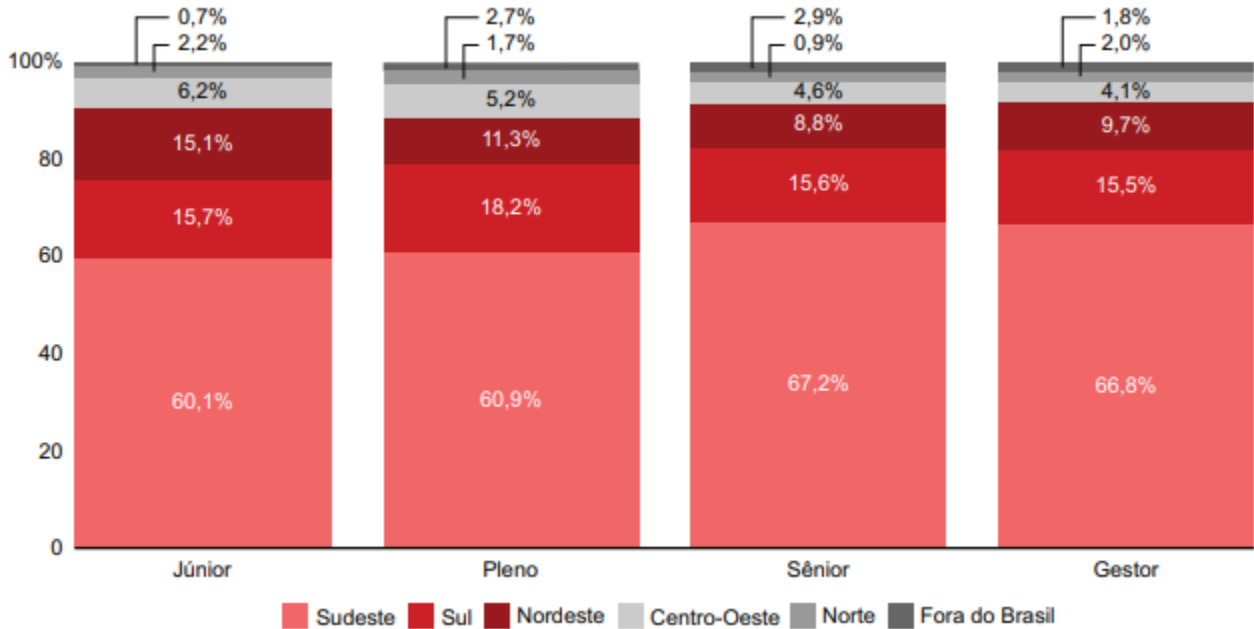
Distribuição de gênero por níveis de cargo



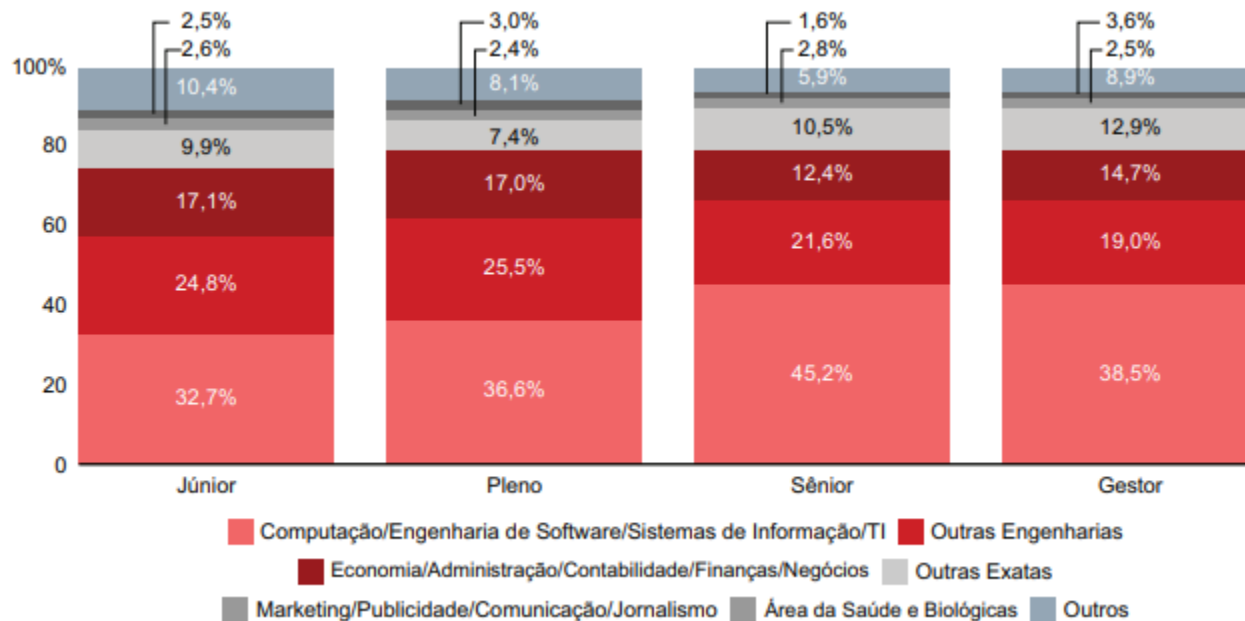
Distribuição de idade por nível de cargo



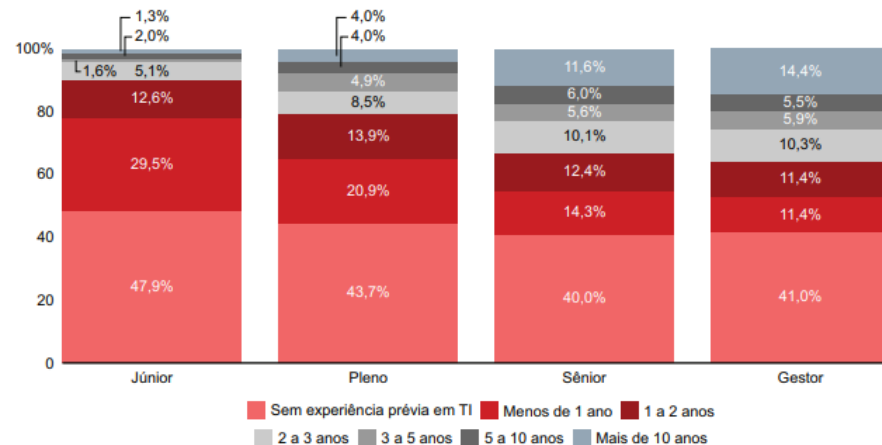
Distribuição de local de residência por níveis de cargo



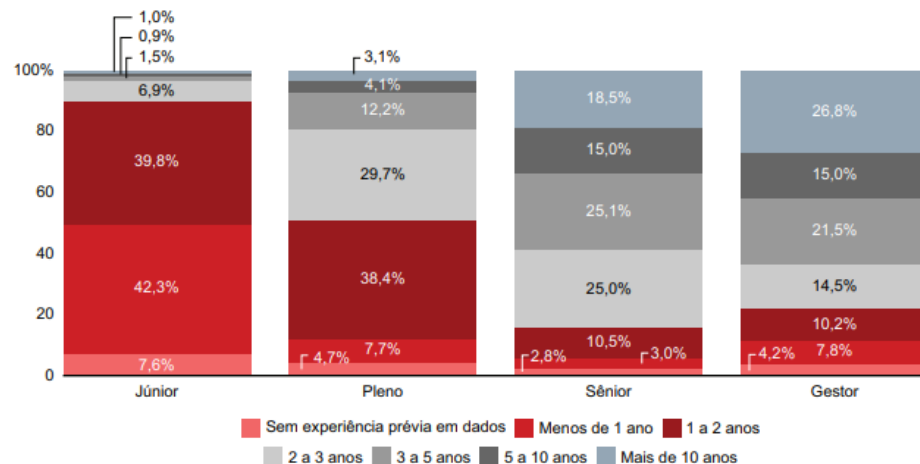
Distribuição de área de formação por nível de cargo



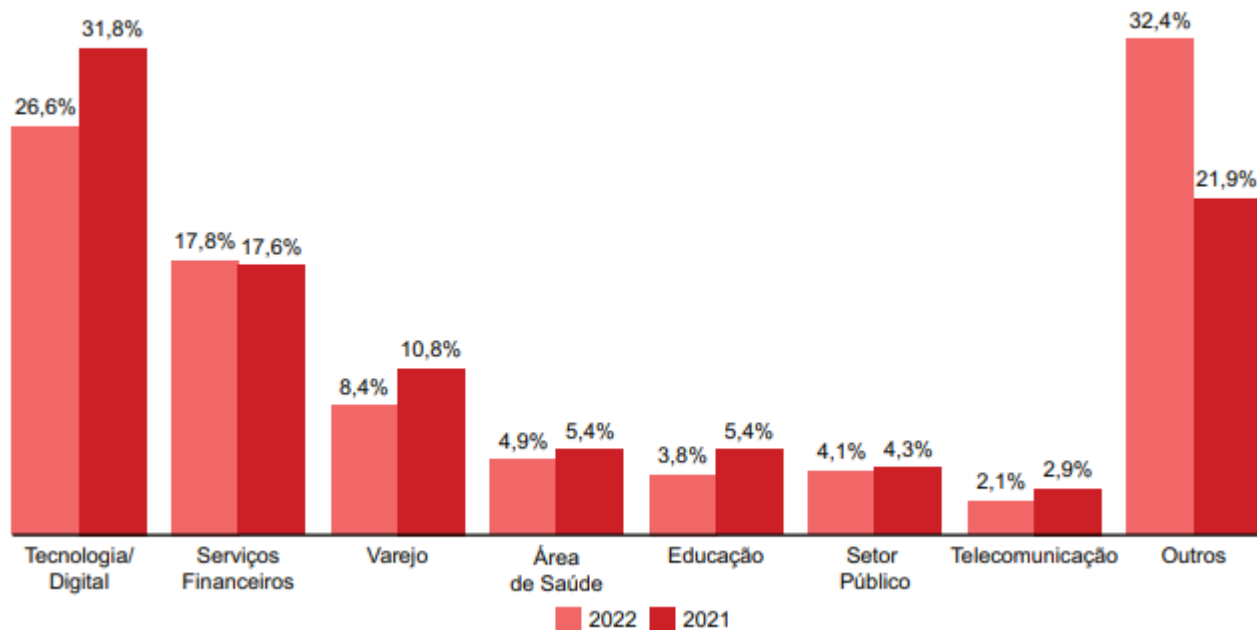
Anos de experiência prévia em TI por nível de cargo



Anos de experiência em dados por nível de cargo



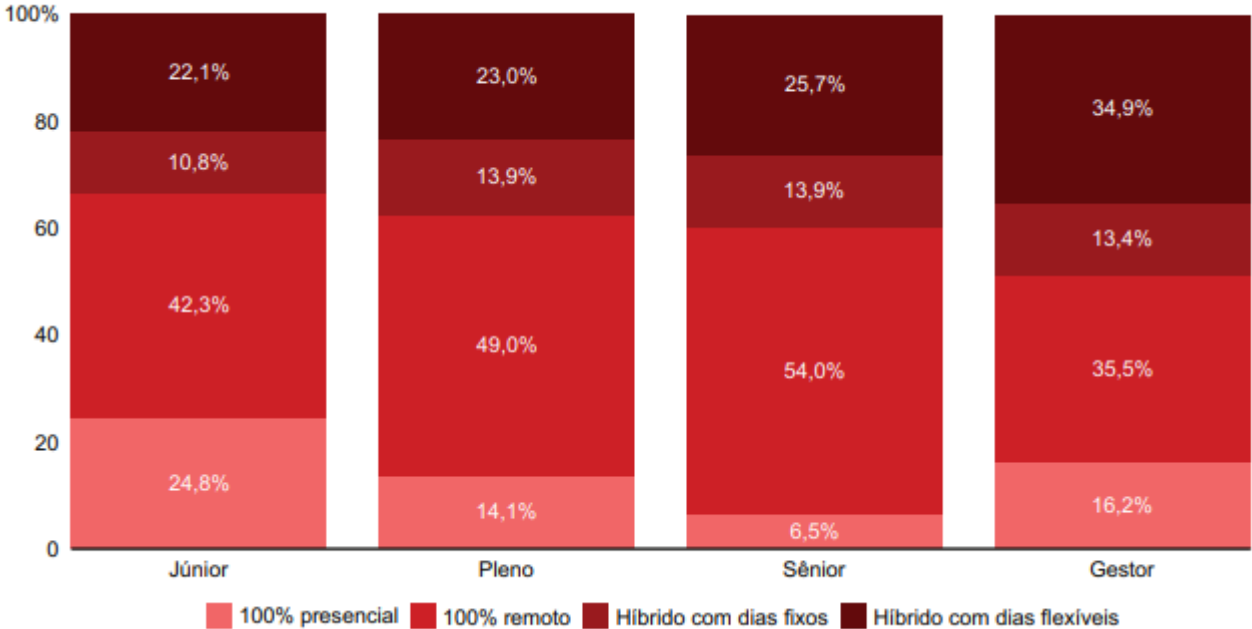
Segmento de atuação dos atuais empregadores por ano



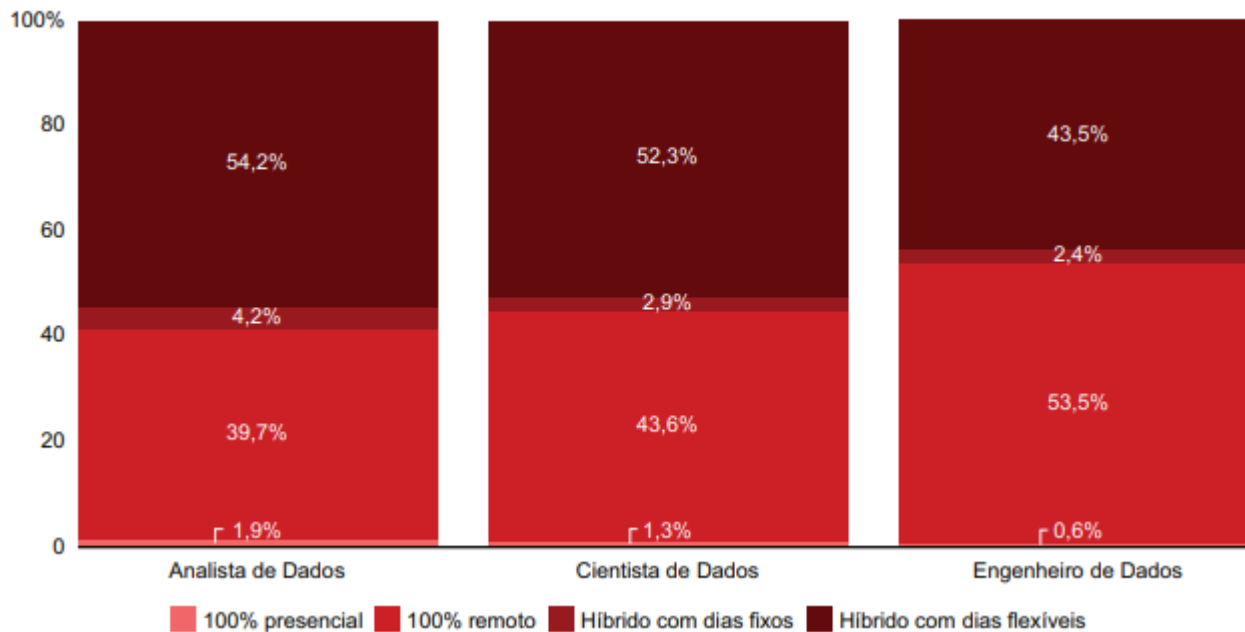
Critérios levados em consideração para escolha de empresa



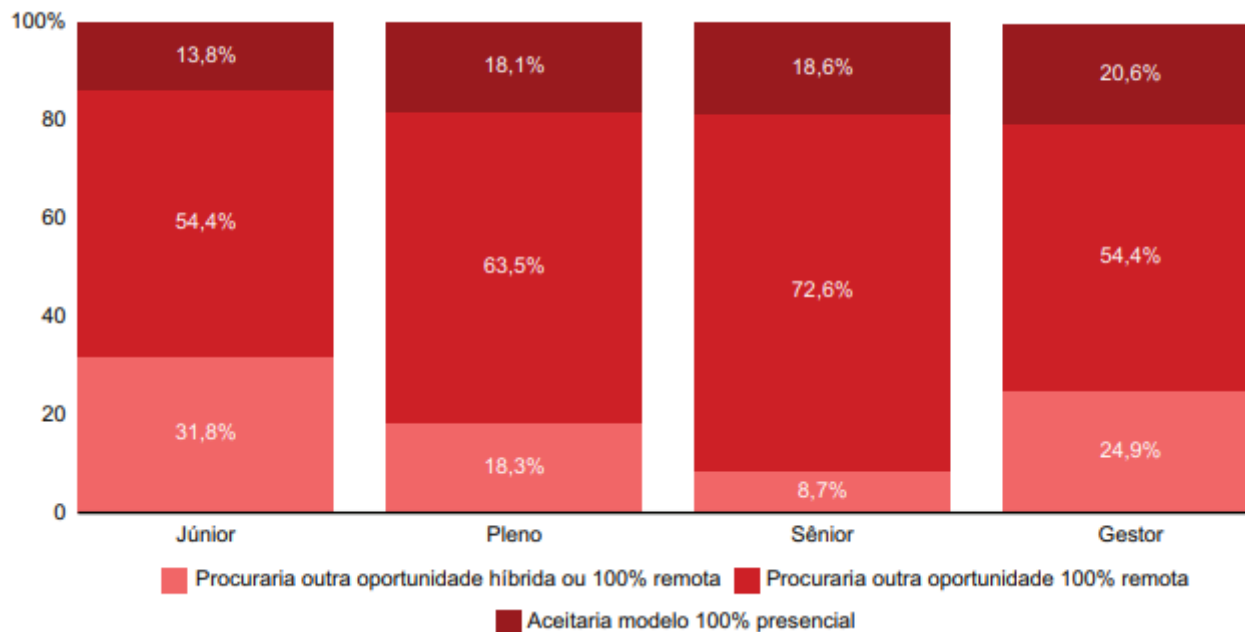
Atual forma de trabalho por nível de cargo



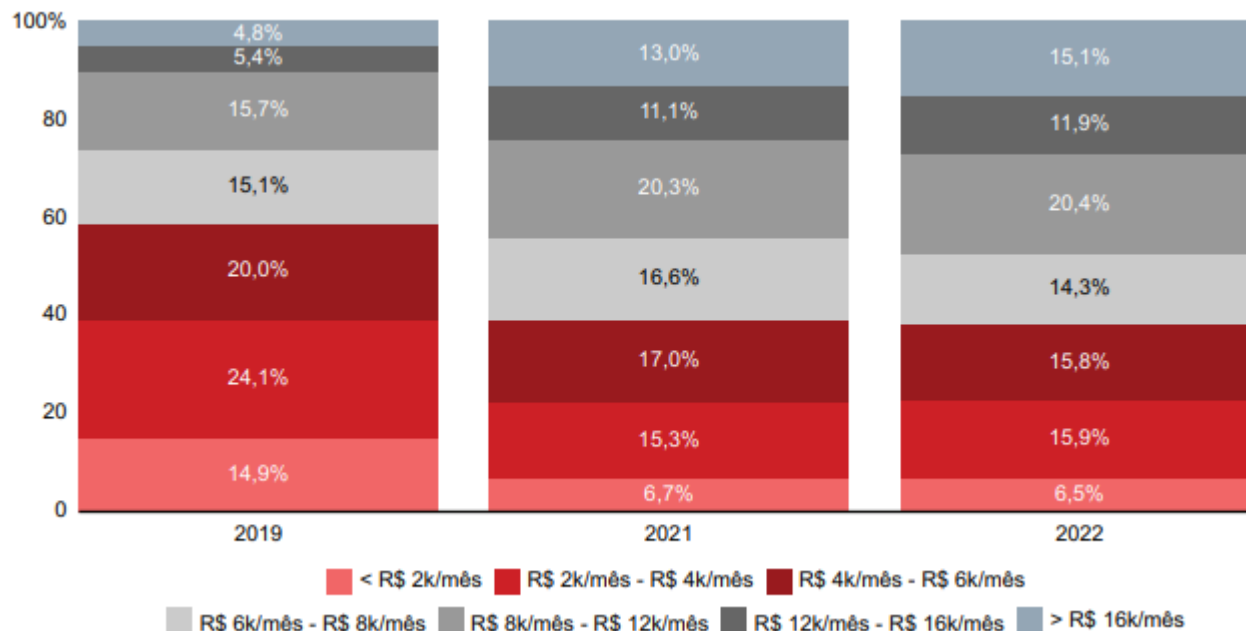
Modalidade de trabalho preferencial por função



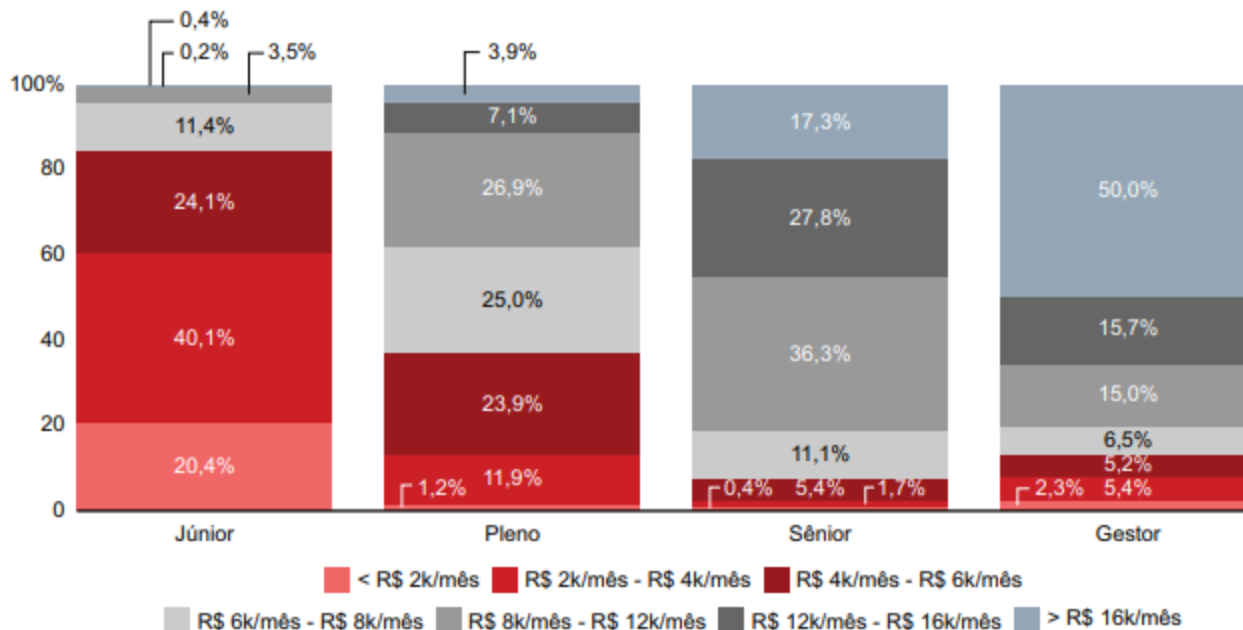
Decisão caso a empresa volte ao modelo 100% presencial por nível de cargo



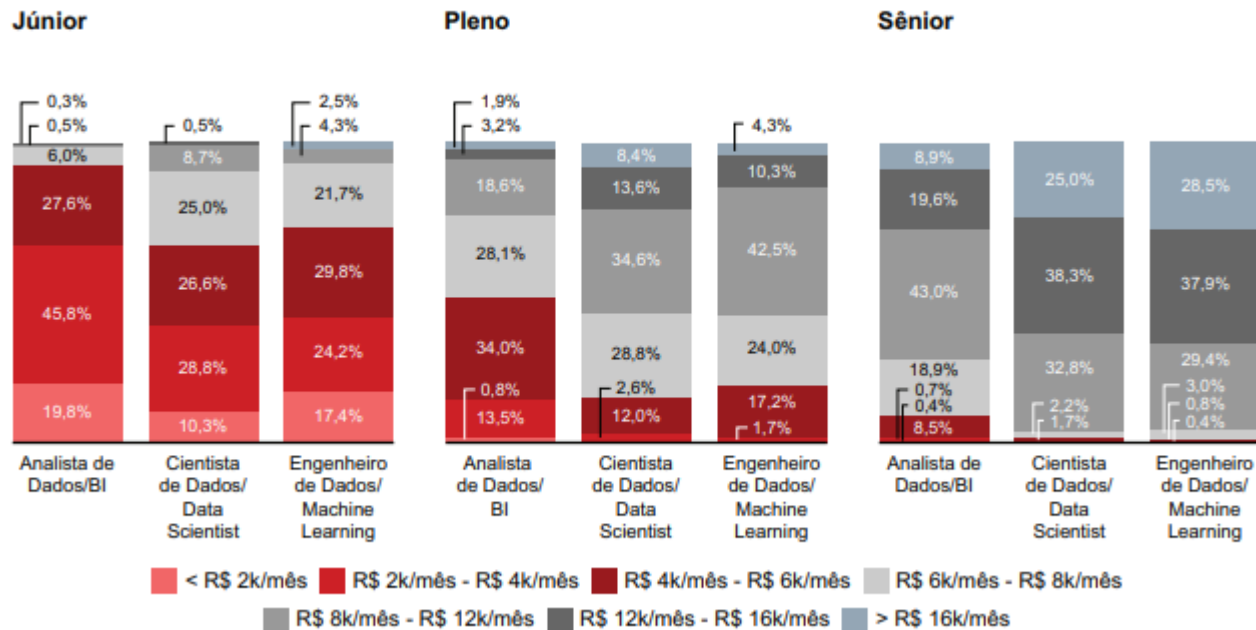
Comparação da remuneração entre 2019 e 2022



Remuneração em 2022 por nível de cargo



Comparação da remuneração por função e nível de cargo



Por onde começar?

03

[Explorar](#)
[Trilhas de aprendizado](#)

[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](#)

[Produtividade no escritório](#)

[Análises](#)

[Big Data](#)

[Visualização de dados](#)

[Machine Learning](#)

[Análise estatística](#)

públicas da Indra aqui.

Mais novo

Criar trilha


es


o Java

e mais 2 editores

Codecademy


codecademy [Catalog](#) [Resources ▼](#) [Community ▼](#) [Pricing ▼](#) [Business Solutions](#)

 [Log In](#)



Join the millions learning to code with Codecademy for free





Email

Password 

[Sign up](#)

By signing up for Codecademy, you agree to Codecademy's [Terms of Service](#) & [Privacy Policy](#).

Or sign up using:

The screenshot shows the Kaggle homepage. At the top is a navigation bar with links for Competitions, Datasets, Code, Discussions, and Courses. A search bar is on the right. The main content area features a large heading 'Start with more than a blinking cursor' and a subtext: 'Kaggle offers a no-setup, customizable, Jupyter Notebooks environment. Access GPUs at no cost to you and a huge repository of community published data & code.' Below this are two buttons: 'REGISTER WITH GOOGLE' and 'Register with Email'. To the right, a preview of a Jupyter Notebook is shown. The notebook's title is 'XGBoost model that predicts whether a website is malicious or not.' The code in the notebook includes imports for numpy, pandas, xgboost, and matplotlib, followed by data loading, cleaning, splitting into training and testing sets, and model specification.

Start with more than a blinking cursor

Kaggle offers a no-setup, customizable, Jupyter Notebooks environment. Access GPUs at no cost to you and a huge repository of community published data & code.

REGISTER WITH GOOGLE

Register with Email

This kernel has an XGBoost model that predicts whether a website is malicious or not.

```

import numpy as np
import pandas as pd
import xgboost as xgb

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels

data = pd.read_csv('../input/dataset.csv')

# clean up column names
data.columns = data.columns.\
    str.strip().\
    str.lower()

# remove non-numeric columns
data = data.select_dtypes(['number'])

# split data into training & testing
train, test = train_test_split(data, shuffle=True)

# peek @ dataframe
train.head()

# split training data into inputs & outputs
x = train.drop(['type'], axis=1)
y = train['type']

# specify model (xgboost defaults are generally fine)
model = xgb.XGBRegressor()
    
```

Console: Draft Session (4m03s) | CPU 45% | GPU Off | RAM 4.5/8GB | Disk 32MB/128GB

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.

Principais bibliotecas

04

Principais bibliotecas



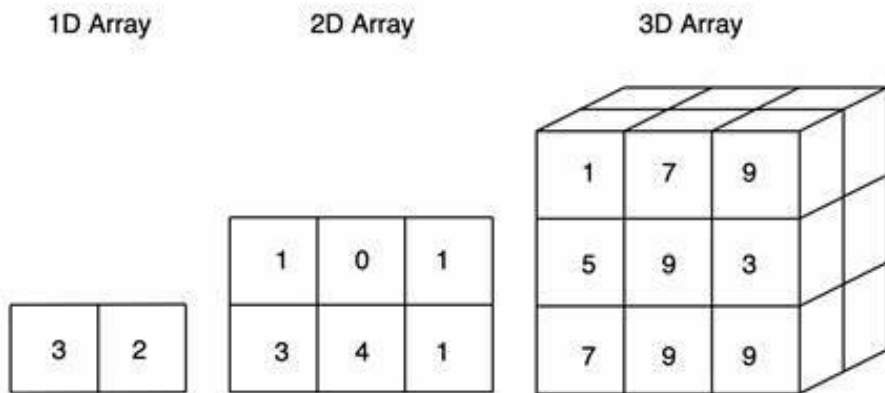
Numpy

05

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

```
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 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]
```

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 [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'
```

```
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 – 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 + ATIVIDADE PRÁTICA

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.

Pandas

06

Pandas

O Pandas não substitui o Numpy, mas o estende. Pandas é uma biblioteca poderosa para análise de dados. Ela oferece estruturas de dados flexíveis, como DataFrames, que permitem a manipulação, limpeza, transformação e análise de dados de forma eficiente. O Pandas também possui recursos para lidar com dados ausentes e integrar-se com outras bibliotecas de análise e visualização.

Column Label/ Header		0	1	2	3	4
Index Label		Name	Age	Marks	Grade	Hobby
0	S1	Joe	20	85.10	A	Swimming
1	S2	Nat	21	77.80	B	Reading
2	S3	Harry	19	91.54	A	Music
3	S4	Sam	20	88.78	A	Painting
4	S5	Monica	22	60.55	B	Dancing

Column Index

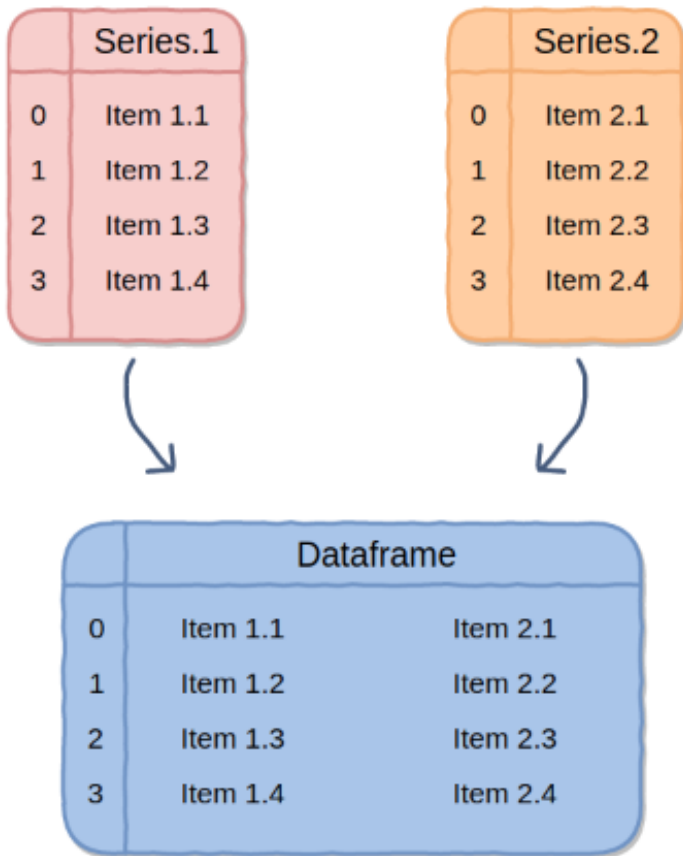
Row Index

Row

Column

Element/ Value/ Entry

Pandas – Tipos de dados



- Numéricos;
- Datas;
- Object, que o pandas usa para colunas que possuem dados, que não se encaixam em nenhum outro tipo.
 - Também normalmente é usado em dados do tipo *string*.

Pandas vs Numpy

```
In [1]: import numpy as np
import pandas as pd

a = np.array([1,2,3,4])
series = pd.Series([1,2,3,4])

print (series.describe(),'\n') # Função não disponível no numpy
```

```
count    4.000000
mean     2.500000
std      1.290994
min      1.000000
25%      1.750000
50%      2.500000
75%      3.250000
max      4.000000
dtype: float64
```

```
In [4]: print (a[0],'\n')
print (a[:2],'\n')
```

1

[1 2]

```
In [5]: print (series[0],'\n')
print (series[:2],'\n')
```

1

0 1

1 2

dtype: int64

Pandas vs Numpy

```
In [13]: print ('mean', a.mean())  
         print ('std', a.std())  
         print ('max', a.max())
```

```
mean 2.5  
std 1.118033988749895  
max 4
```

```
In [14]: print ('mean', series.mean())  
         print ('std', series.std())  
         print ('max', series.max())
```

```
mean 2.5  
std 1.2909944487358056  
max 4
```

```
In [17]: a = pd.Series([1, 2, 3, 4])  
         b = pd.Series([1, 2, 1, 2])
```

```
print (a + b)  
print (a * 2)  
print (a >= 3)  
print (a[a >= 3])
```

```
0    2  
1    4  
2    4  
3    6  
dtype: int64  
0    2  
1    4  
2    6  
3    8  
dtype: int64  
0    False  
1    False  
2     True  
3     True  
dtype: bool  
2    3  
3    4  
dtype: int64
```

```
In [16]: a = np.array([1, 2, 3, 4])  
         b = np.array([1, 2, 1, 2])
```

```
print (a + b)  
print (a * 2)  
print (a >= 3)  
print (a[a >= 3])
```

```
[2 4 4 6]  
[2 4 6 8]  
[False False  True  True]  
[3 4]
```

Pandas vs Numpy

```
In [22]: population = pd.Series([1415045928,1354051854,326766748], index = ["China", "India", "US"])
```

```
print (population,'\n')
print ('Population of {} is {}'.format(population.index[1], population['India']))
```

```
China    1415045928
India    1354051854
US        326766748
dtype: int64
```

Population of India is 1354051854

```
In [23]: import numpy as np
```

```
population = np.array([1415045928,1354051854,326766748])
index = np.array(["China", "India", "US"])

print (population,'\n')
print ('Population of {} is {}'.format(index[1], population[1]))
```

```
[1415045928 1354051854 326766748]
```

Population of India is 1354051854

Pandas vs Lista

Pandas

```
In [28]: # Busca por um valor específico  
print(population==1415045928, '\n')
```

```
China      True  
India      False  
US          False  
dtype: bool
```

```
In [30]: value = population[population==1415045928]  
value
```

```
Out[30]: China      1415045928  
dtype: int64
```

Lista

```
In [31]: population = [1415045928, 1354051854, 326766748]  
  
for i in population:  
    if i == 1415045928:  
        value = population[population==1415045928]  
  
print (value)
```

```
1415045928
```

Map, apply, applymap

O método `map()` só funciona em séries pandas onde diferentes tipos de operação podem ser aplicados aos itens da série.

Quando você aplica o método `map(função)` em uma série, a função `map()` pega cada elemento da série e aplica a função a ele e retorna a série transformada.

```
: from sklearn.datasets import load_iris
data = load_iris()
features = pd.DataFrame(data = data['data'], columns= data['feature_names'])
features.head()
```

```
:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1                3.5                1.4                0.2
1                4.9                3.0                1.4                0.2
2                4.7                3.2                1.3                0.2
3                4.6                3.1                1.5                0.2
4                5.0                3.6                1.4                0.2
```



```
In [59]: def cm_to_mm(cm):
          mm = cm * 10
          return mm

features['conversion_SL'] = features['sepal length (cm)'].map(cm_to_mm)
features.head()
```

```
Out[59]:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  conversion_SL
0                5.1                3.5                1.4                0.2            51.0
1                4.9                3.0                1.4                0.2            49.0
2                4.7                3.2                1.3                0.2            47.0
3                4.6                3.1                1.5                0.2            46.0
4                5.0                3.6                1.4                0.2            50.0
```

Map, apply, applymap

O método `apply()` funciona em séries pandas e dataframes com uma variedade de funções facilmente aplicadas dependendo do tipo de dados.

Semelhante a `map()`, quando você usa o método `apply()` em uma série ou dataframe, a função pega cada elemento da série e aplica a função no elemento, então retorna a série ou dataframe transformados.

```
In [61]: features[['sepal length (cm)', 'sepal width (cm)']].map(cm_to_mm)
```

```
-----
AttributeError                                Traceback (most recent call last)
C:\Users\TALMEI~1\AppData\Local\Temp\ipykernel_17636\3128970834.py in <module>
----> 1 features[['sepal length (cm)', 'sepal width (cm)']].map(cm_to_mm)

~\Anaconda3\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)
    5485         ):
    5486             return self[name]
-> 5487         return object.__getattr__(self, name)
    5488
    5489     def __setattr__(self, name: str, value) -> None:
AttributeError: 'DataFrame' object has no attribute 'map'
```

```
In [62]: features[['sepal length (cm)', 'sepal width (cm)']].apply(cm_to_mm).head()
```

Out[62]:

	sepal length (cm)	sepal width (cm)
0	51.0	35.0
1	49.0	30.0
2	47.0	32.0
3	46.0	31.0
4	50.0	36.0

Map, apply, applymap

O método `applymap()` funciona em todo o dataframe do pandas, onde a função de entrada é aplicada a cada elemento individualmente.

features

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2



In [64]: `features.applymap(cm_to_mm).head()`

Out[64]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	51.0	35.0	14.0	2.0
1	49.0	30.0	14.0	2.0
2	47.0	32.0	13.0	2.0
3	46.0	31.0	15.0	2.0
4	50.0	36.0	14.0	2.0

JUPYTER NOTEBOOK + ATIVIDADE PRÁTICA

Comparações

Característica	Listas	Numpy	Pandas
Estrutura de Dados	Lista ordenada de elementos	Arrays N-dimensionais homogêneos	Estruturas de dados tabulares
Tipo dos elementos	Pode conter tipos diversos	Homogêneo (mesmo tipo de dado)	Pode conter tipos diversos
Performance	Menos eficiente para cálculos e operações com grandes conjuntos de dados	Altamente otimizado para cálculos e operações vetorizadas	Eficiente para manipulação de dados grandes
Funcionalidades	Possui funcionalidades básicas para manipulação de elementos	Oferece funções matemáticas e operações vetorizadas	Oferece ferramentas de análise de dados sofisticadas
Indexação	Acesso aos elementos é realizado por índices	Acesso aos elementos é realizado por índices	Oferece indexação personalizada (por rótulos) além de indexação por índices
Operações matemáticas	Suporta algumas operações básicas	Oferece uma ampla gama de funções matemáticas e operações vetorizadas	Oferece operações matemáticas e estatísticas
Funções de Agregação	Limitadas	Oferece funções de agregação poderosas (por exemplo, soma, média, mínimo, máximo)	Oferece funções de agregação avançadas
Manipulação de dados	Limitada	Possui métodos para filtragem, seleção, transformação e agrupamento de dados	Oferece recursos completos para manipulação e limpeza de dados
Compatibilidade	Integra-se bem com Python e outras bibliotecas	Compatível com outras bibliotecas científicas e matemáticas	Compatível com outras bibliotecas de análise de dados e visualização

Referências

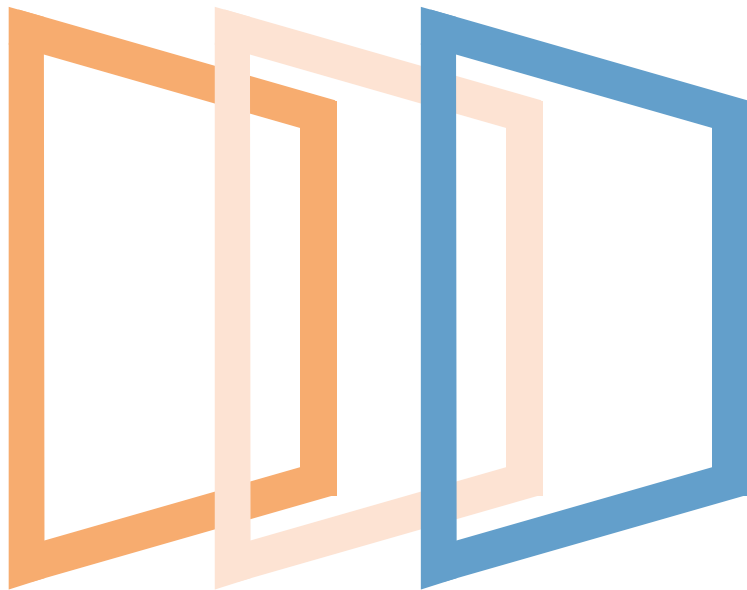
- [state-of-data2022_um-raio-x-dos-profissionais-de-dados-do-brasil.pdf \(bain.com\)](#)
- [NumPy](#)
- [Entendendo a biblioteca NumPy. O que é o NumPy? | by Luiz Santiago Jr. | Ensina.AI | Medium](#)
- [Introdução ao Numerical Python \(Numpy\) | OPL \(ufc.br\)](#)
- [pandas - Python Data Analysis Library \(pydata.org\)](#)
- [Pandas Tutorial \(w3schools.com\)](#)
- [Tutorial Pandas slides \(jmsevillam.github.io\)](#)
- [https://stack-academy.memberkit.com.br/32408-data-science-do-zero](#)

Trilhando Caminhos em Ciência de Dados

Thaís Ratis

Práticas Tecnológicas, 12.08.2023

minsoit



An Indra company