

Introdução à linguagem de programação Python: conceitos básicos e requisitos para aplicação de algoritmos de IA

EDITAL FAPERGS 06/2024 - PROGRAMA DE PESQUISA E DESENVOLVIMENTO VOLTADO A DESASTRES CLIMÁTICOS

Leandro Rodrigues Oviedo
Engenheiro Químico
Doutor em Nanociências – UFN



GRUPO DE PESQUISA
EM NANOMATERIAIS APLICADOS

Tópicos abordados

Conceitos gerais – O que é Python?

Funcionalidades do Python

Ambientes: execução local e execução em nuvem

Principais bibliotecas (pesquisa científica – análise exploratória e algoritmos de IA (construção de *dataframes*, regressão não linear, teste de normalidade, correlação...))

Estudos de casos

O que é Python?



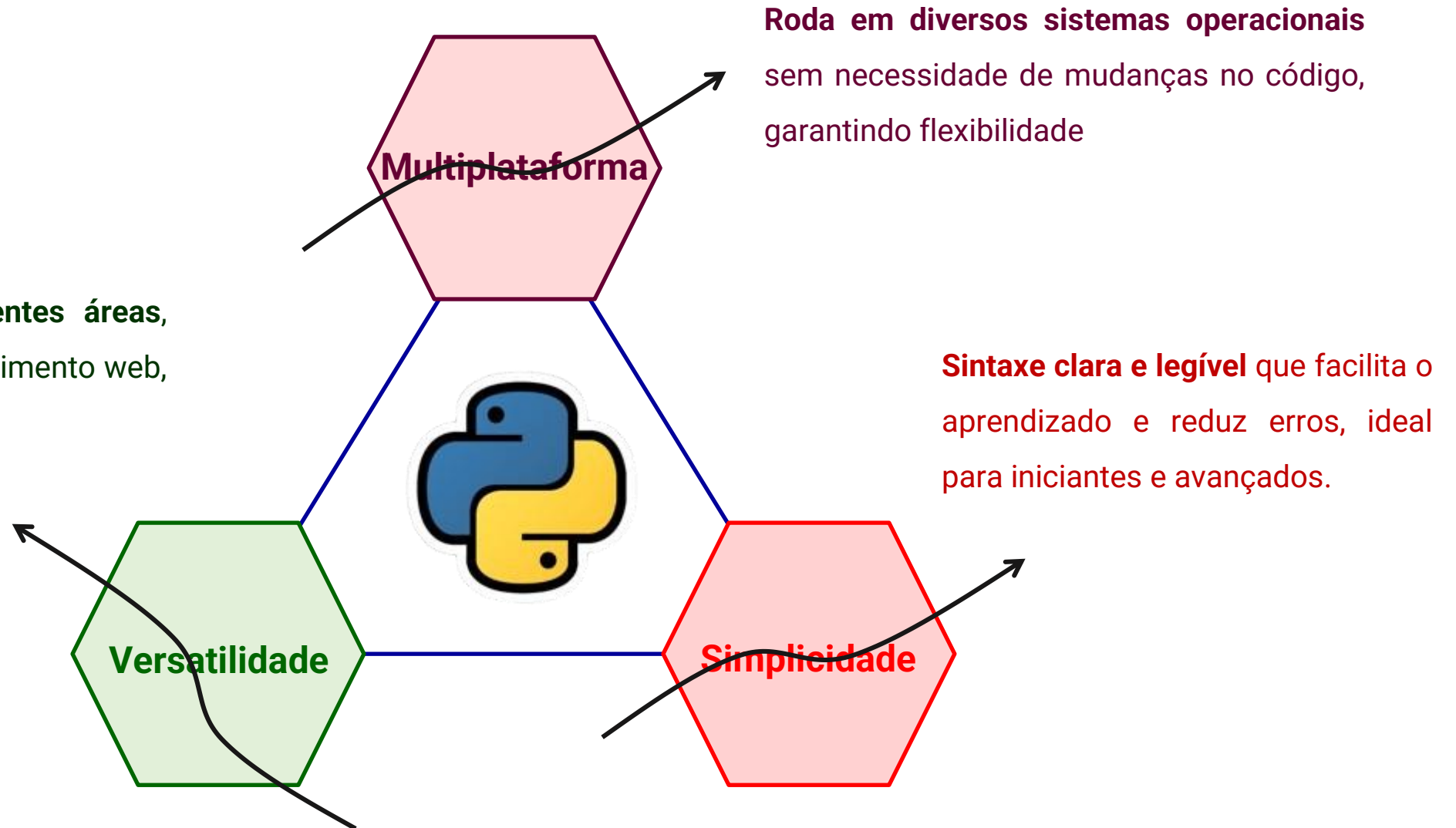
Linguagem de programação de uso geral, que permite criar uma grande variedade de aplicações diferentes. Destaca-se pela sua versatilidade e facilidade de uso, tornando-se uma das linguagens mais comuns atualmente.

Linguagem amplamente utilizada em desenvolvimento de software, análise de dados, inteligência artificial, automação de tarefas e criação de aplicativos web, contando com vasta biblioteca padrão e uma comunidade ativa que facilita encontrar soluções para diversos projetos.

Python é uma linguagem interpretada, ou seja, não necessita de compilação prévia, o que acelera o desenvolvimento.

Python e suas vantagens

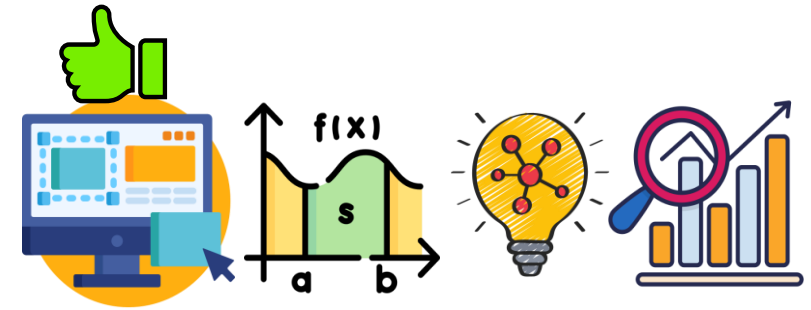
Pode ser **usado em diferentes áreas**, como automação, desenvolvimento web, ciência de dados, IA, e IoT



Funcionalidades

Operações matemáticas e análise estatística

- Regressão linear e não linear;
- Análise exploratória dos dados (EDA)
- Teste de normalidade, análise de correlação e teste de hipóteses.



Tratamento e visualização de dados

- Plotagem de gráficos interativos e não interativos;
- Construção de *dataframes*, normalização de dados e visualizações (*scatterplot*, *violinplot*, histogramas, gráficos 3D...);

Construção e compilação de algoritmos avançados

- Algoritmos de inteligência artificial (IA);
- Análise de extração de *features* (*feature extraction/selection*);
- Planejamento de experimentos (*design of experiments* – DOE);
- Elaboração de WebApp, páginas html e interfaces gráficas, *dashboards* e outros.

Ambientes de execução

Execução local / Computador pessoal/IDE/editor



VSCode



PyCharm



Jupyter



Spyder

Execução em nuvem / Browser, servidores remotos



Google Colab



JupyterHub



Kaggle Notebooks

Principais bibliotecas

Versão atual do *software*: Python 3.12.11

Bibliotecas utilizadas na pesquisa científica-----

numpy # computação numérica e manipulação eficiente de *arrays*

pandas # manipulação e análise de dados tabulares

matplotlib / **seaborn** / **plotly** #visualização de dados, estática e interativa

scipy # matemática aplicada, otimização e ferramentas científicas

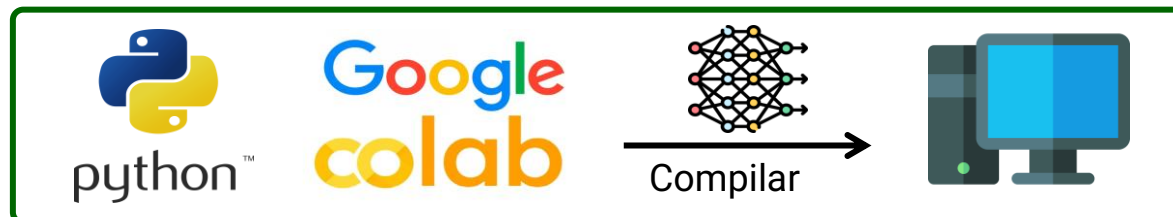
sympy # álgebra computacional e cálculo simbólico

statsmodels # análise estatística e modelagem econométrica

scikit-learn # machine learning e pré-processamento de dados

Levenberg-Marquardt # algoritmo para otimização e ajuste de modelos não lineares

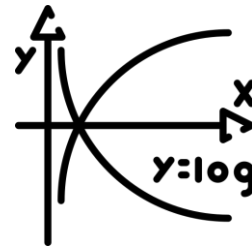
tensorflow / **keras** # desenvolvimento e treinamento de redes neurais (multicamadas e profundas)



Estudos de casos

Estudo de Caso 1

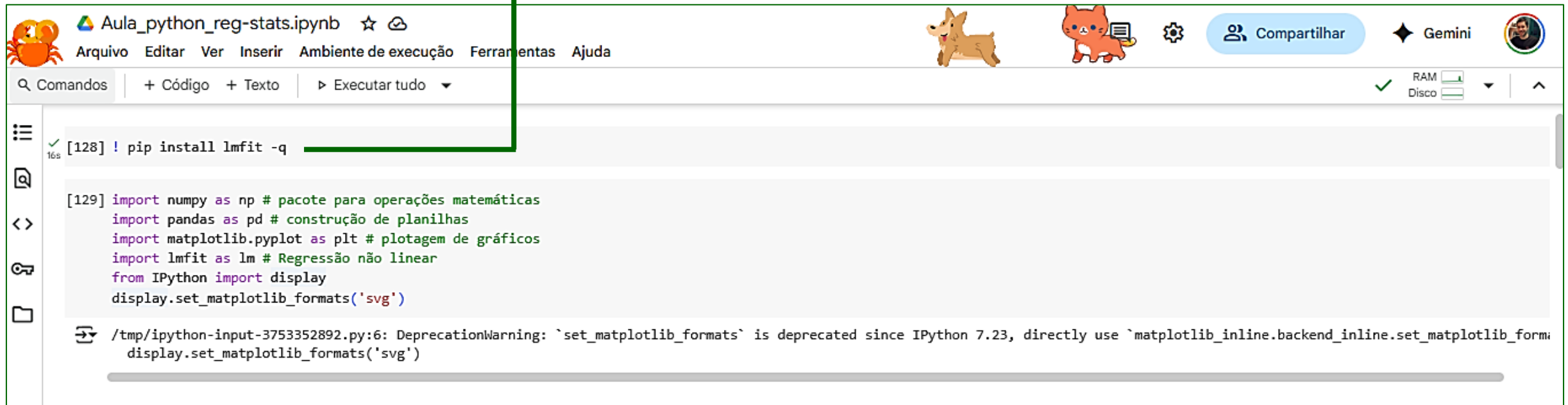
Temática: Deseja-se realizar uma regressão não linear dos dados experimentais de adsorção, utilizando um modelo cinético de 3 parâmetros. Para isso, defina qual sua equação, os parâmetros envolvidos e utilize as funções da biblioteca Levenberg-Marquardt. Qual a **confiabilidade** no modelo de regressão utilizado? Obtenha um relatório completo sobre **métricas** do modelo de regressão e compare com as métricas do modelo de 2 parâmetros. **Qual o tempo de processamento do algoritmo registrado em sua máquina?**



O método de **Levenberg-Marquardt** é um algoritmo usado para ajustar parâmetros de modelos não lineares por meio de otimização iterativa, combinando os métodos de Gauss-Newton e gradiente descendente para melhorar a convergência. Uma das bibliotecas que permitem sua implementação é Imfit (`> ! pip install Imfit > import Imfit as lm`)

Estudo de Caso 1

```
# Instala a biblioteca de regressão não linear (método Levenberg-Marquardt)  
! Pip install lmfit
```



The screenshot shows a Jupyter Notebook interface with the title "Aula_python_reg-stats.ipynb". The top bar includes a menu with "Arquivo", "Editar", "Ver", "Inserir", "Ambiente de execução", "Ferramentas", and "Ajuda". Below the menu are tabs for "Comandos", "+ Código", "+ Texto", and "Executar tudo". On the right side of the top bar, there are icons for a dog, a cat, a gear, a "Compartilhar" button, and a "Gemini" logo. The notebook content area shows two code cells. The first cell, labeled "[128]", contains the command `! pip install lmfit -q` and has a green checkmark and "16s" next to it. A green arrow points from this cell to the text above. The second cell, labeled "[129]", contains the following code:

```
import numpy as np # pacote para operações matemáticas  
import pandas as pd # construção de planilhas  
import matplotlib.pyplot as plt # plotagem de gráficos  
import lmfit as lm # Regressão não linear  
from IPython import display  
display.set_matplotlib_formats('svg')
```

Below the code, a deprecation warning is displayed: `/tmp/ipython-input-3753352892.py:6: DeprecationWarning: `set_matplotlib_formats` is deprecated since IPython 7.23, directly use `matplotlib_inline.backend_inline.set_matplotlib_formats`
display.set_matplotlib_formats('svg')`

Estudo de Caso 1



The screenshot shows a Jupyter Notebook titled "Aula_python_reg-stats.ipynb". The notebook contains two code cells. The first cell defines a list of time values, and the second cell defines a list of corresponding experimental data values. A green arrow points from the second list to the text "Tabulação/importação dos dados experimentais".

```
time= [0,
        3,
        5,
        10,
        15,
        20,
        30,
        45,
        60,
        75,
        90,
        120,
        150,
        180,
        210,
        240]

CCo_exp = [0.025,
            0.041,
            0.041,
            0.050,
            0.074,
            0.099,
            0.132,
            0.347,
            0.579,
            0.694,
            0.760,
            0.950,
            0.967,
            0.983,
            0.992,
            1.000]
```

Tabulação/importação dos dados experimentais

Estudo de Caso 1

Aula_python_reg-stats.ipynb ☆ ☁

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda

Comandos + Código + Texto ▶ Executar tudo

RAM Disco

Compartilhar Gemini

Yoon-Nelson Model

$$\frac{C_t}{C_0} = \frac{1}{1 + e^{K_{YN}(\tau - t)}}$$

where,

K_{YN} is the rate constant (min^{-1})

τ is time required for 50 % adsorbate breakthrough (min)

Ambiente Markdown

(permite registrar anotações em texto e escrever equações matemáticas/químicas)

Eixo $x \sim$ tempo (t) / Eixo $y \sim \frac{C_t}{C_0}$

```
[71] def Yoon_Nelson(t,kYN,tau):
      # C/Co = 1/(1+exp*(kYN(tau-t)))
      return 1 / (1 + np.exp(kYN*(tau-t)))
```

→ Digita equação matemática do modelo de regressão

```
[72] CCo_pred = np.empty(len(CCo_exp))

      modelo_YN = lm.Model(Yoon_Nelson)
      modelo_YN_parametros = modelo_YN.make_params(kYN=0.2,tau=0.5)
      reg_modelo_YN = modelo_YN.fit(CCo_exp,modelo_YN_parametros,t=time)

      reg_modelo_YN
```

→ Regressão não linear (em 3 linhas de código)

Variáveis Terminal

01:35 Python 3

Estudo de Caso 1



Fit Result

Model: Model(Yoon_Nelson)

Fit Statistics

fitting method leastsq
function evals 35
data points 16
variables 2

chi-square 0.01536572
reduced chi-square 0.00109755
Akaike info crit. -107.171281
Bayesian info crit. -105.626104
R-squared 0.99409827

Métricas do modelo de
regressão (ex.: $R^2 = 0,9941$)

Parameters

name	value	standard error	relative error	initial value	min	max	vary
kYN	0.05277767	0.00308866	(5.85%)	0.2	-inf	inf	True
tau	59.2701458	1.36101555	(2.30%)	0.5	-inf	inf	True

Correlations (unreported values are < 0.100)

Parameter1	Parameter 2	Correlation
kYN	tau	-0.2128

A função **Imfit** nos fornece um relatório completo de regressão

Estimativa dos parâmetros da equação
(neste caso, temos **2 parâmetros**)

Estudo de Caso 1

Cálculo do coeficiente de determinação ajustado (R^2_{adj})

```
[73] K = len(reg_modelo_YN.params)
      N = len(time)

      print(f'Tenho {K} parâmetros')
      print(f'Tenho {N} dados',N)

      print('Tenho {0:d} parâmetros e {1:d} dados'.format(K,N))
```

```
→ Tenho 2 parâmetros
    Tenho 16 dados 16
    Tenho 2 parâmetros e 16 dados
```

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where

R^2 = sample R-square
 p = Number of predictors
 N = Total sample size.

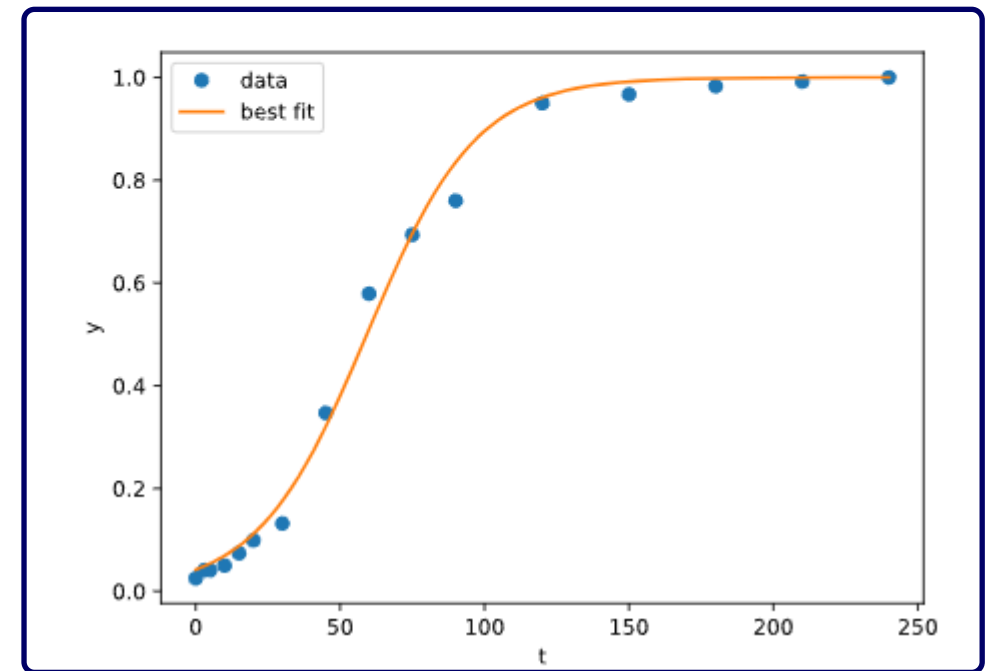
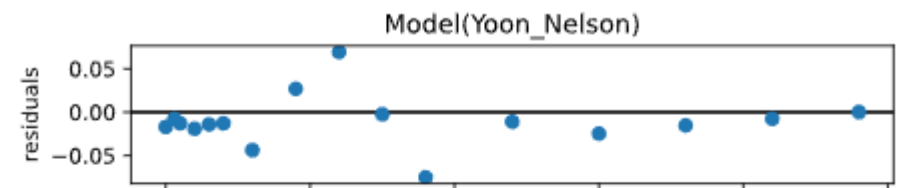
```
[74] R2 = reg_modelo_YN.rsquared
      R2_adj = 1 - ((1-R2)*(N-1)/(N-K-1))

      print(f'R²: {R2}\nR² ajustado: {R2_adj}')
```

```
→ R²: 0.9940982658793455
    R² ajustado: 0.9931903067838601
```

Plotagem do gráfico de regressão não linear

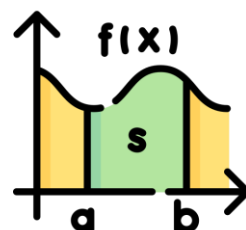
```
✓ [107] reg_modelo_YN.plot(numpoints=1000)
```



Estudos de casos

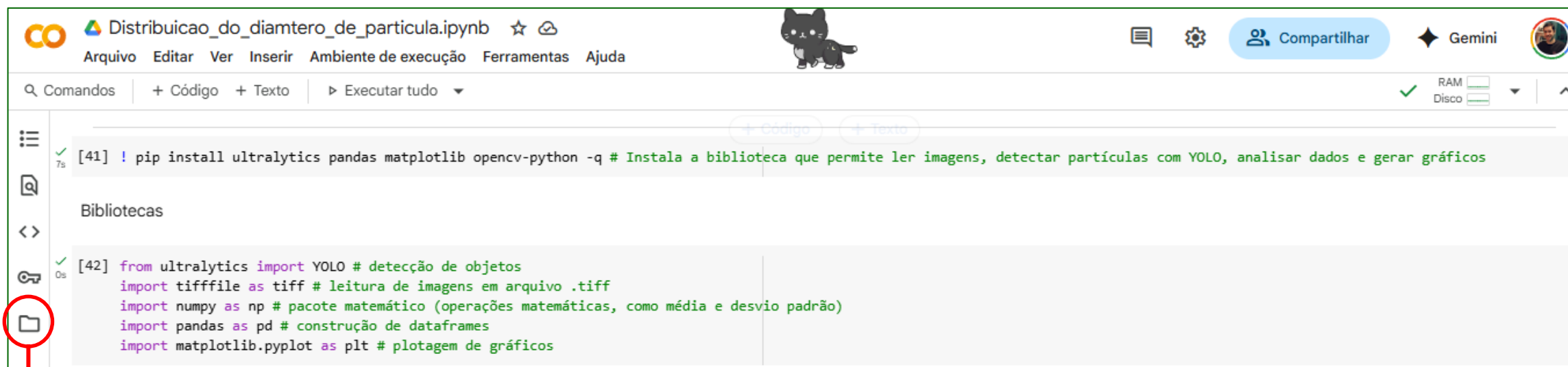
Estudo de Caso 2

Temática: Deseja-se determinar a distribuição do tamanho de partícula de uma nanopartícula de óxido de cobre (CuO-NPs) a partir de imagens de microscopia eletrônica de varredura de alta resolução (MEV-FEG). Para isso, um código em Python pode ser construído para ler a imagem de MEV-FEF (formato **.tif**), detectar o **número de partículas** na micrografia, fornecer uma planilha com os valores da **distribuição**, plotar um **histograma** e *printar os valores de média e desvio padrão para o tamanho/diâmetro de partícula*. Para isso, será utilizado o **YOLO** (You Only Look Once), um algoritmo de detecção de objetos preciso e rápido, ideal para identificar e contar partículas em imagens de MEV/TEM/MEV-FEG.



*o algoritmo **YOLO** funciona processando toda a imagem de uma vez e **localizando múltiplos objetos simultaneamente**, retornando as coordenadas e dimensões de cada partícula detectada, o que facilita a extração de tamanhos e análise estatística da distribuição dos diâmetros.*

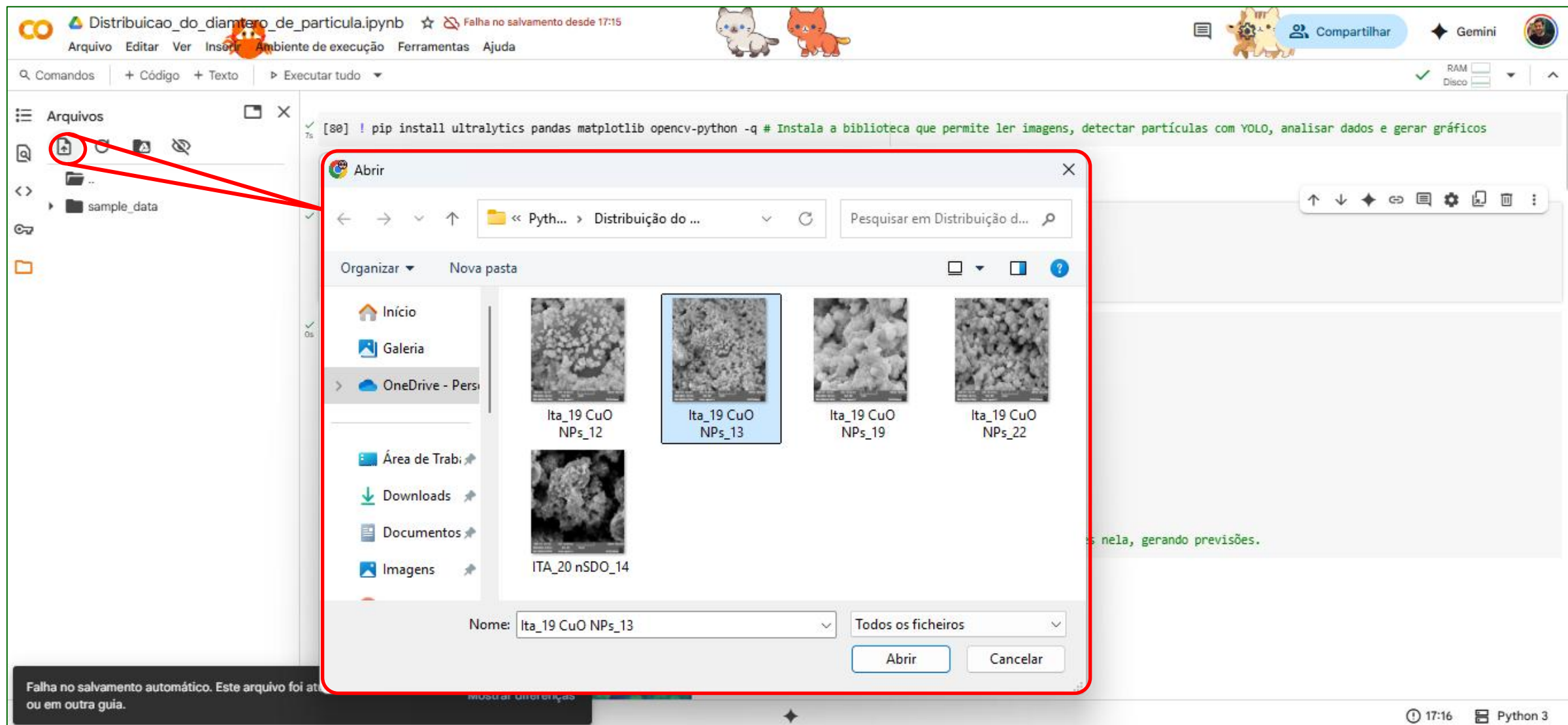
Estudo de Caso 2



Ferramenta que permite o carregando de arquivos no Python

(Opcionalmente, pode-se carregar, ler e processar dados diretamente contidos no Google Drive)

Estudo de Caso 2



The screenshot displays a Jupyter Notebook environment. The top bar shows the notebook title "Distribuicao_do_diametro_de_particula.ipynb" and a warning "Falha no salvamento desde 17:15". The left sidebar contains a file explorer with a "sample_data" folder. A red circle highlights the "Abrir" button in the bottom right corner of the file selection dialog box. The dialog box, titled "Abrir", shows a grid of image files: "Ita_19 CuO NPs_12", "Ita_19 CuO NPs_13", "Ita_19 CuO NPs_19", "Ita_19 CuO NPs_22", and "ITA_20 nSDO_14". The "Nome:" field at the bottom is set to "Ita_19 CuO NPs_13". The background code cell contains the command: `[80] ! pip install ultralytics pandas matplotlib opencv-python -q # Instala a biblioteca que permite ler imagens, detectar partículas com YOLO, analisar dados e gerar gráficos`. A status bar at the bottom indicates "Falha no salvamento automático. Este arquivo foi atualizado em outra guia." and "Python 3".

Estudo de Caso 2

Distribuição do diâmetro de partícula.ipynb ☆ ☁

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda

Comandos + Código + Texto ▶ Executar tudo ▼

Arquivos

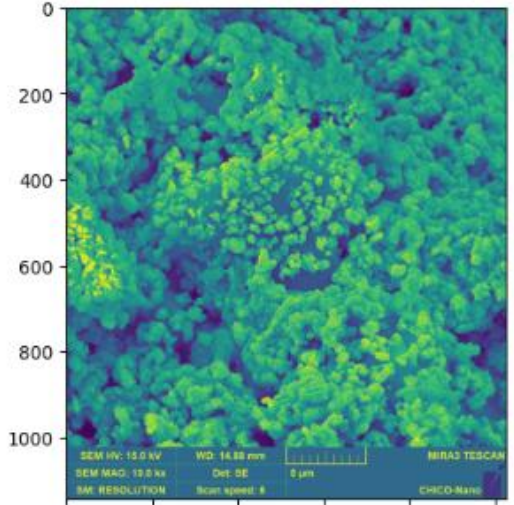
- sample_data
- Ita_19 CuO NPs_13.tif

```
img = tiff.imread('Ita_19 CuO NPs_13.tif') # lê a imagem de MEV de título 'nome_da_imagem.tif'

# mostrar imagem
plt.imshow(img)
plt.show()

# Se a imagem for monocanal, converte para RGB (YOLO espera 3 canais)
if img.ndim == 2:
    img = np.stack([img]*3, axis=-1)
# Carregue o modelo YOLOv8 treinado para partículas
model = YOLO('yolov8n.pt')

# Realize a inferência
results = model(img) # aplica o modelo treinado em uma nova imagem para detectar e reconhecer objetos presentes nela, gerando previsões.
```



Disco Disponível: 68.68 GB

Variáveis Terminal

17:16 Python 3

Estudo de Caso 2

Interface de um Jupyter Notebook no Google Colab, mostrando o código Python para análise de partículas.

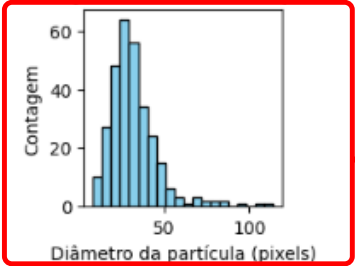
```
boxes = results[0].boxes.xyxy.cpu().numpy() # formato: [x1, y1, x2, y2]
# Conte o número de partículas detectadas
num_particles = len(boxes) # cada box é uma partícula
print(f'Número de partículas detectadas: {num_particles}')

# Calcule o diâmetro (média entre largura e altura da caixa)
sizes = []
for box in boxes:
    x1, y1, x2, y2 = box
    width = x2 - x1
    height = y2 - y1
    diameter = (width + height) / 2 # ou use sqrt(width*height) para área equivalente
    sizes.append(diameter)

# Crie o DataFrame
df = pd.DataFrame({'particle_diameter_px': sizes})
plt.figure(figsize=(2, 2))
plt.hist(df['particle_diameter_px'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Diâmetro da partícula (pixels)')
plt.ylabel('Contagem')
plt.title('Distribuição do tamanho de partículas')
plt.show()
```

Número de partículas detectadas: 300

Distribuição do tamanho de partículas

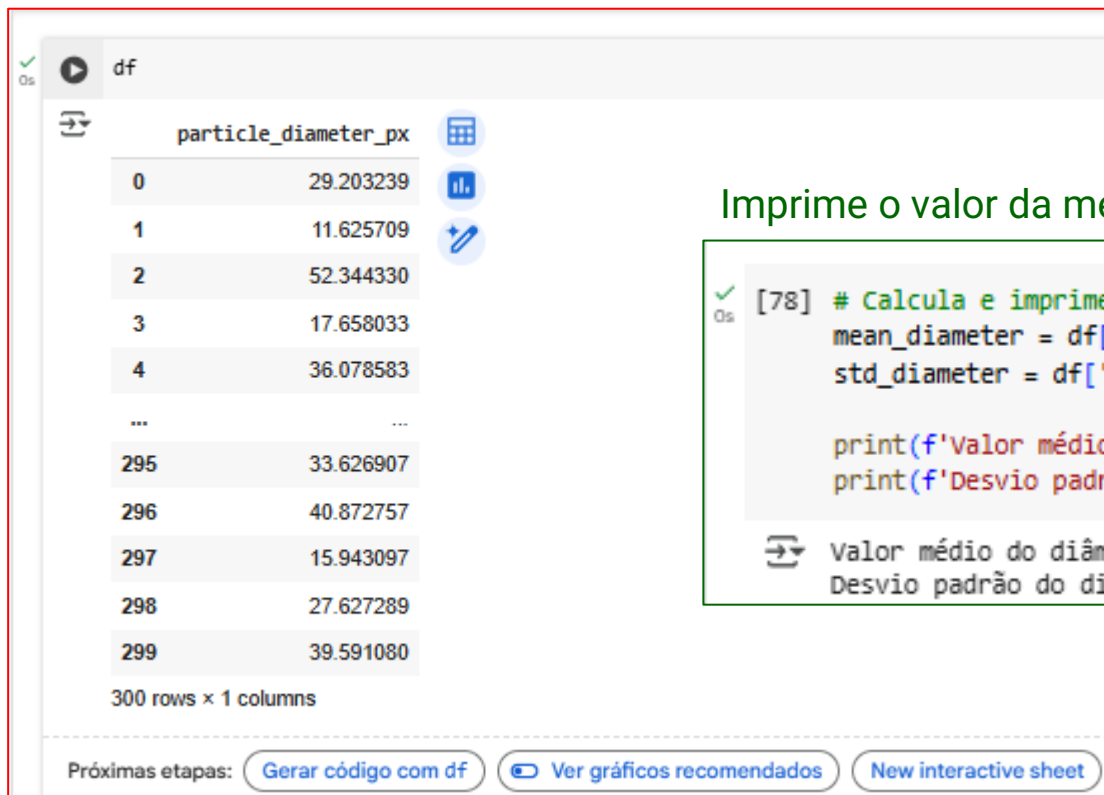


Imprime o resulta da contagem de partículas

Fornece o histograma da distribuição do diâmetro de partículas previsto pelo YOLO a partir da micrografia

Estudo de Caso 2

Fornece *dataframe* com a distribuição do diâmetro de partícula



	particle_diameter_px
0	29.203239
1	11.625709
2	52.344330
3	17.658033
4	36.078583
...	...
295	33.626907
296	40.872757
297	15.943097
298	27.627289
299	39.591080

300 rows × 1 columns

Próximas etapas: [Gerar código com df](#) [Ver gráficos recomendados](#) [New interactive sheet](#)

Imprime o valor da média e desvio padrão referente à diâmetro de partícula

```
[78] # Calcula e imprime o valor médio e o desvio padrão do diâmetro das partículas
      mean_diameter = df['particle_diameter_px'].mean()
      std_diameter = df['particle_diameter_px'].std() # desvio padrão amostral (ddof=1 por padrão)

      print(f'Valor médio do diâmetro das partículas (pixels): {mean_diameter:.2f}')
      print(f'Desvio padrão do diâmetro das partículas (pixels): {std_diameter:.2f}')
```

Valor médio do diâmetro das partículas (pixels): 29.04
Desvio padrão do diâmetro das partículas (pixels): 15.49

Imprime o resulta da contagem de partículas

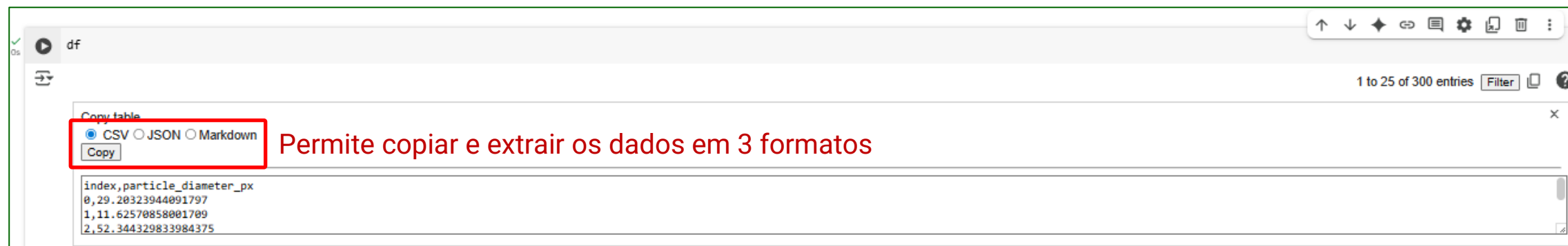
Estudo de Caso 2



1 to 25 of 300 entries **Filter** ?

index	particle_diameter_px
0	29.20323944091797
1	11.62570858001709
2	52.344329833984375
3	17.65803337097188
4	36.078582763671875
5	10.154022216796875
6	48.2249755859375
7	28.769073486328125
8	23.41033935546875
9	16.40843310546875
10	25.469482421875
11	26.68292236328125
12	38.7557373046875
13	97.4754638671875
14	26.579688045043945
15	26.1376953125
16	28.900726318359375
17	33.332122802734375
18	9.52966900634766
19	32.146240234375

Variáveis Terminal 17:33 Python 3



Copy table

☒ CSV ☐ JSON ☐ Markdown

Copy

index,particle_diameter_px
0,29.20323944091797
1,11.62570858001709
2,52.344329833984375

Permite copiar e extrair os dados em 3 formatos

Estudo de Caso 2

Sem Título * - Kate

Ficheiro Editar Selection Ver Ir Projectos Cliente LSP Sessões

Novo Abrir Gravar Salvar como Desfazer

Sem Título

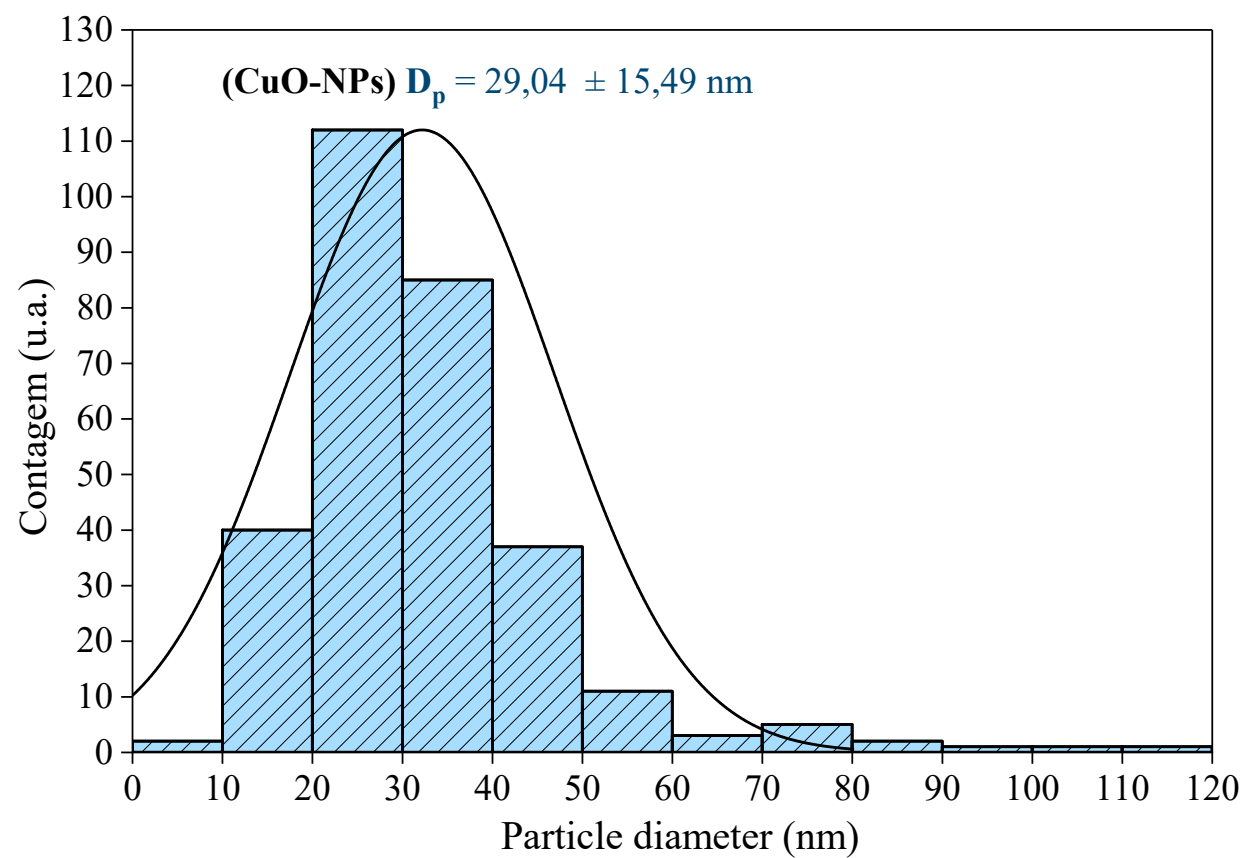
```

1 index,particle_diameter_px
2 0,29.20323944091797
3 1,11.62570858001709
4 2,52.344329833984375
5 3,17.65803337097168
6 4,36.078582763671875
7 5,10.154022216796875
8 6,48.2249755859375
9 7,28.769073486328125
10 8,23.41033935546875
11 9,16.40643310546875
12 10,25.469482421875
13 11,26.68292236328125
14 12,38.7557373046875
15 13,97.4754638671875
16 14,26.579668045043945
17 15,26.1376953125
18 16,28.900726318359375
19 17,33.332122802734375
20 18,9.529666900634766
21 19,32.146240234375
22 20,15.59051513671875
23 21,23.34979248046875
24 22,31.66429901123047
25 23,31.2362060546875
26 24,39.29547119140625
27 25,35.70189666748047
28 26,37.286956787109375
29 27,28.864349365234375
30 28,75.65414428710938
31 29,25.474411010742188
32 30,11.914674758911133
33 31,27.816055297851562
34 32,26.948394775390625
35 33,14.37877082824707
36 34,14.345916748046875
37 35,29.482131958007812
38 36,12.462894439697266
39 37,16.21123504638672

```

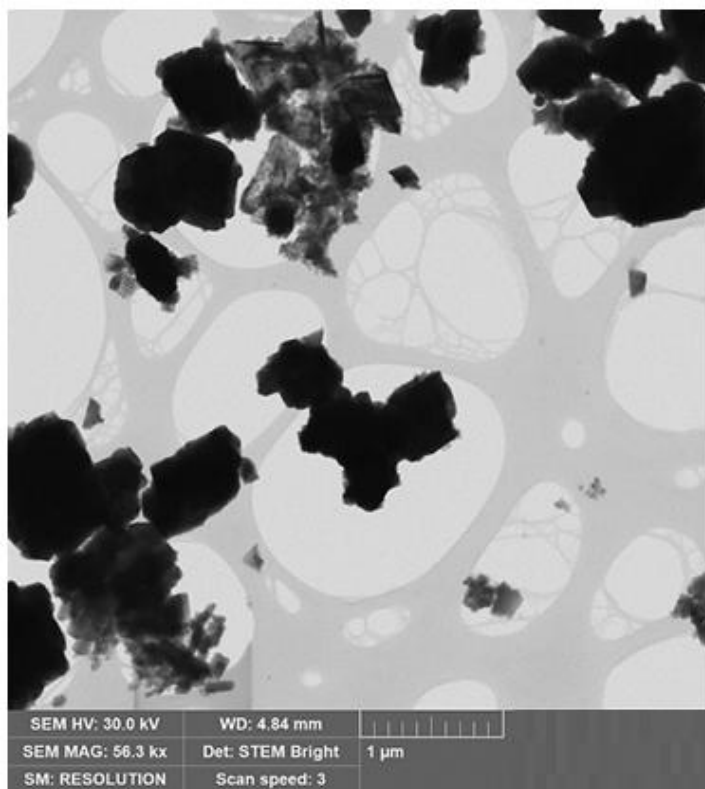
Resultado Diagnósticos Procurar Projecto Terminal

Figura 2. Distribuição do diâmetro médio de partícula das nSOD@CuO-NPs gerado a partir dos do Python e reproduzido em software OriginLab 2025b.

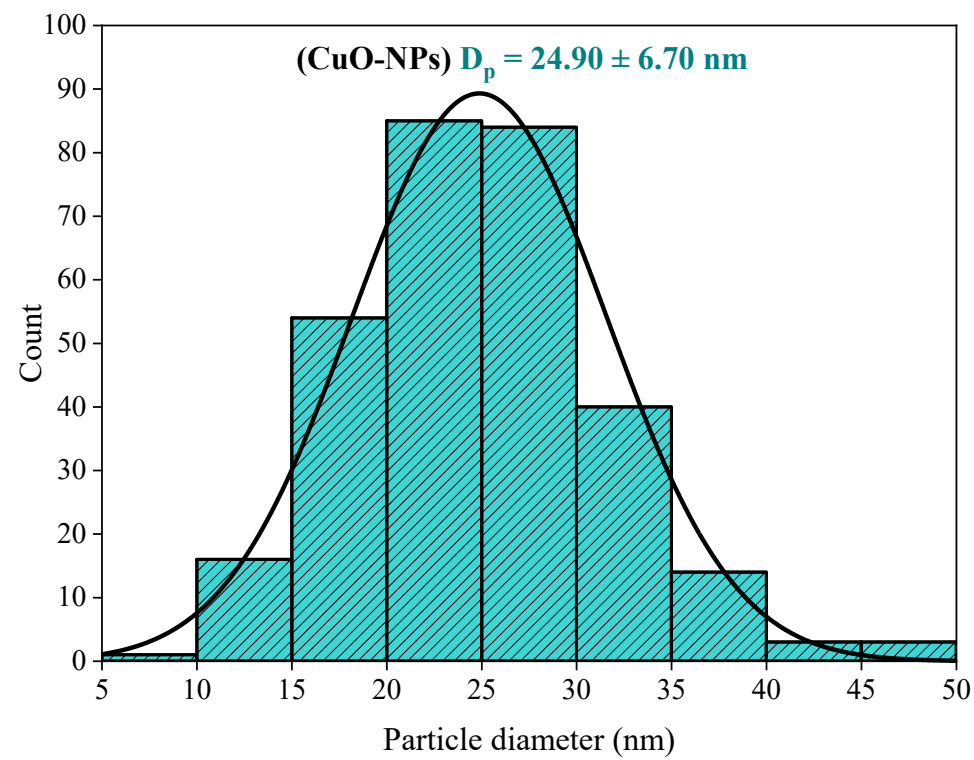


Estudo de Caso 2

Figura 3. (a) Micrografias de microscopia eletrônica de transmissão (TEM) de CuO-NPs com ampliação de 56,3 kx e (b) distribuição do tamanho médio de partícula das CuO-NPs



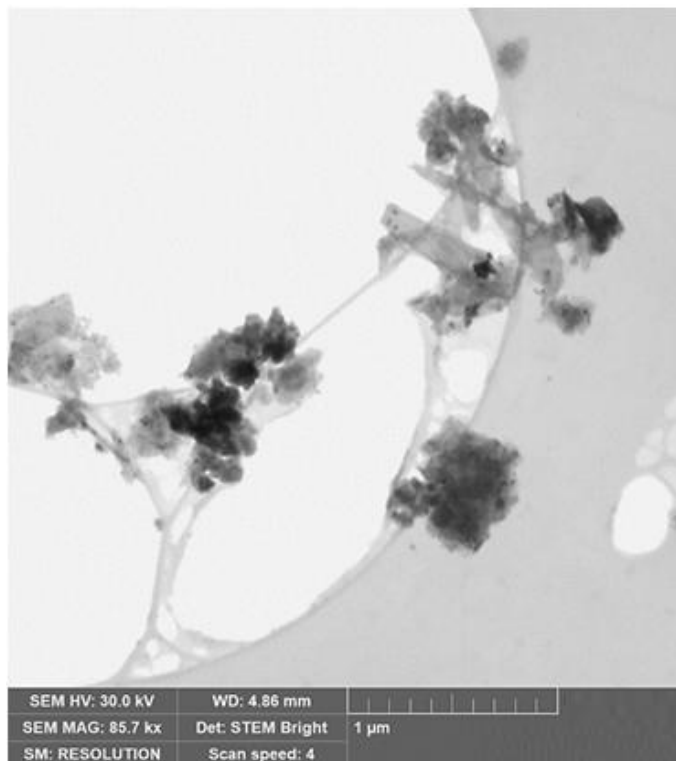
(a)



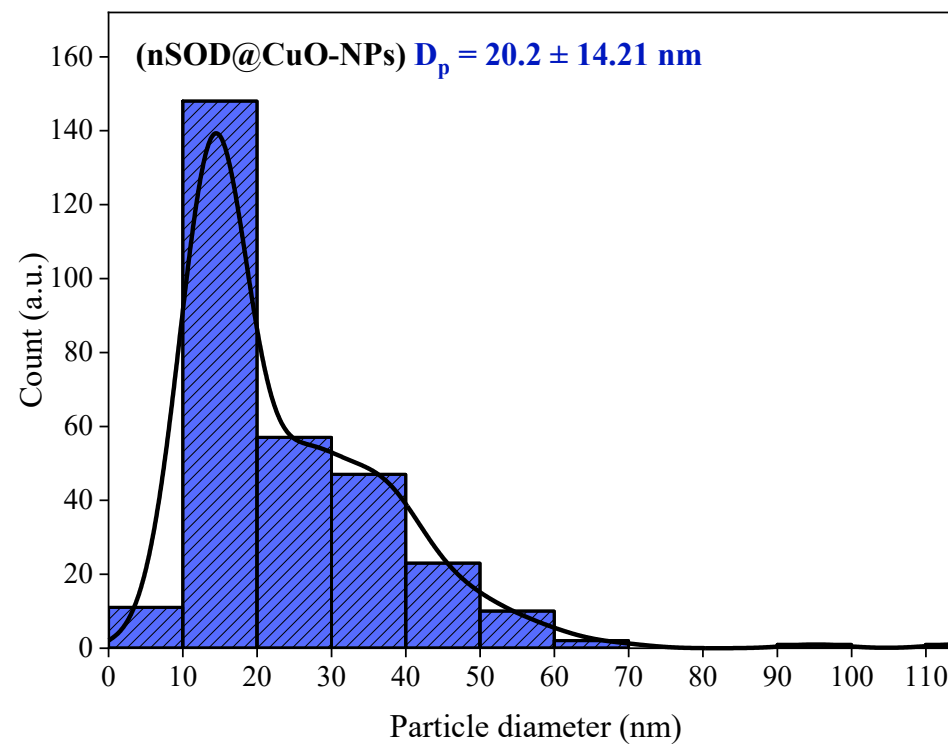
(b)

Estudo de Caso 2

Figura 4. (a) Micrografias de microscopia eletrônica de transmissão (TEM) de nSOD@CuO-NPs com ampliação de 85,7 kx e (b) distribuição do tamanho médio de partícula das nSOD@CuO-NPs



(a)



(b)

Estudos de casos

Estudo de Caso 3

Temática: Deseja-se realizar uma Análise Exploratória dos Dados (EDA – Exploratory Data Analysis) para verificar qual a melhor condição de um tratamento avançado de água (fotocatálise heterogênea para a degradação de uma mistura binária de corantes sintéticos). Para isso, sugere-se realizar uma análise de correlação e visualização de dados para identificar a condição ótima do processo oxidativo.



***EDA** é o processo inicial de examinar e visualizar dados para entender suas características principais, detectar **padrões**, **outliers** e **relações** entre variáveis. Sugere-se realizar a **análise de correlação** antes de aplicar **machine learning supervisionado**, pois isso permite identificar dependências e redundâncias entre variáveis, auxiliando na seleção das features mais relevantes e evitando modelos superajustados (**overfitting**) ou enviesados.*

Estudo de Caso 3

File Edit Selection View Go Run ... Search

CCRD 2to2 Teteh.ipynb × EDA_Statistics_Paper4_devolutiva.ipynb

C: > Users > Leandro Oviedo > Documents > Python > EDA_Statistics_Paper4_devolutiva.ipynb > import pandas as pd

memory usage: 27.6+ KB

Generate + Code + Markdown | Run All Restart Clear All Outputs Go To | Jupyter Variables Outline ... Python 3.13.5

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import shapiro

# 1. Teste de Normalidade para todas as variáveis numéricas
print("Teste de normalidade (Shapiro-Wilk) nas colunas numéricas:")
for col in data.select_dtypes(include=['float64', 'int64']).columns:
    stat, p_value = shapiro(data[col].dropna())
    status = "Normal" if p_value > 0.05 else "Não Normal"
    print(f"{col}: p-value = {p_value:.10f} -> {status}")

# 2. Correlação de Spearman (apenas colunas numéricas)
spearman_corr = data.select_dtypes(include=['float64', 'int64']).corr(method='spearman')

print("\nMatriz de correlação de Spearman:")
print(spearman_corr)
```

Bibliotecas utilizadas para construção de planilhas, plotagem de gráficos e estatística (scipy.stats ou statsmodels)

Código para teste de normalidade

Código para análise de correlação

Estudo de Caso 3

```
print("\nMatriz de correlação de Spearman:")
print(spearman_corr)

# 3. Heatmap da correlação com paleta Greens e barra de escala (colorbar)
plt.figure(figsize=(7, 7))
sns.heatmap(
    spearman_corr,
    annot=True,
    cmap='Greens',
    cbar=True,
    vmin=-1, vmax=1,
    square=True,
    linewidths=0.5,
    linecolor='gray'
)
plt.title('Matriz de Correlação de Spearman (Paleta Greens)')
plt.show()
```

Código para plotar a matriz de correlação
(biblioteca seaborn para gráficos)

Pode salvar em formatos png, svg, jpeg, PDF e tif.

Python



Para salvar a Figura, pode-se utilizar a **função** `plt.savefig('nome_da_figura.svg', dpi =600)`

`plt.tight_layout()` para ajustar automaticamente os espaços entre os elementos do gráfico para que não fiquem sobrepostos ou cortados,

Quanto maior o dpi, maior a qualidade de imagem (Python trabalha com **geração de Imagem** na faixa de **72 a 1400 dpi**)

Estudo de Caso 3

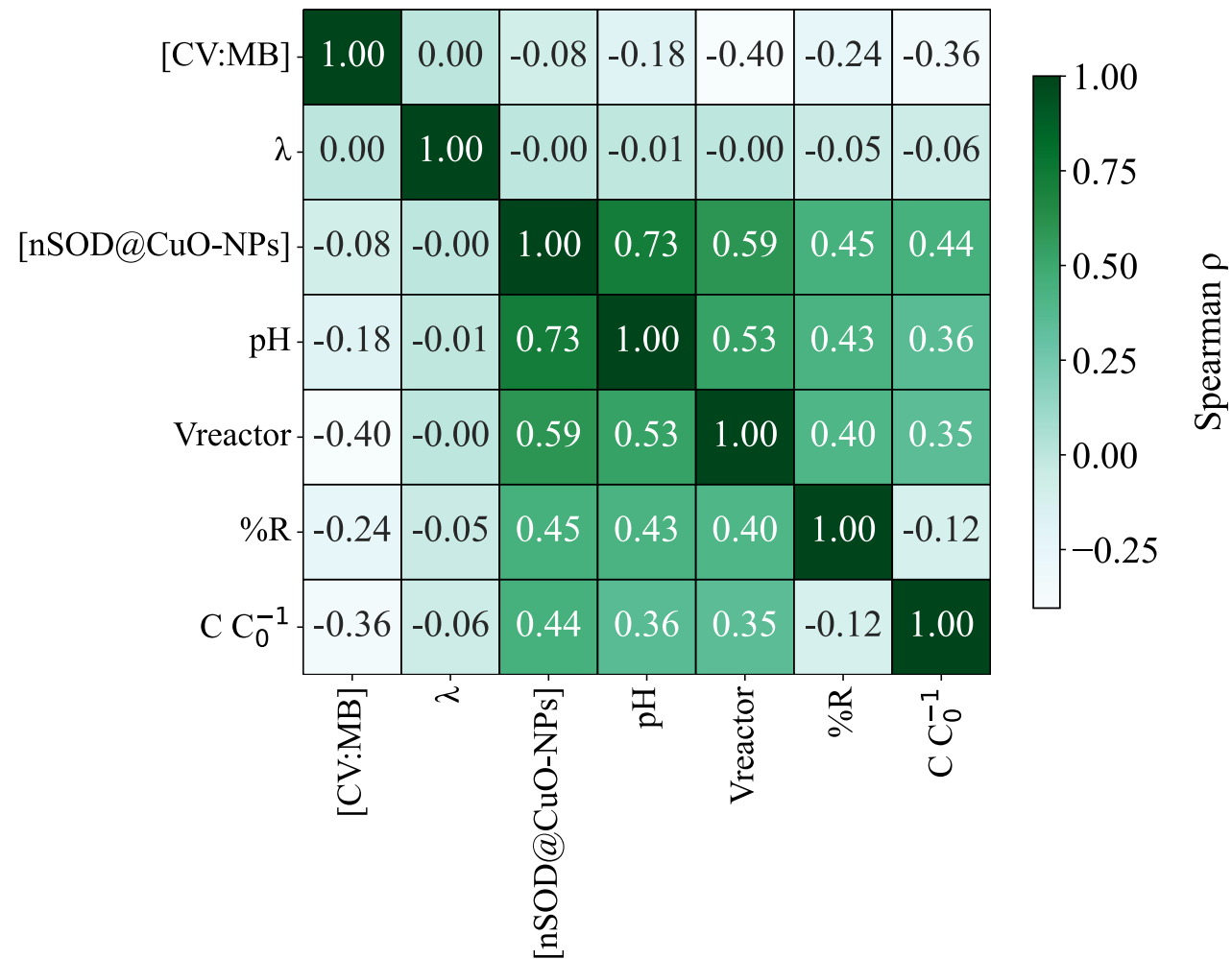


Figure 6. Spearman correlation for experimental data / **Fonte:** <https://doi.org/10.1016/j.jwpe.2025.108508>

Estudo de Caso 3

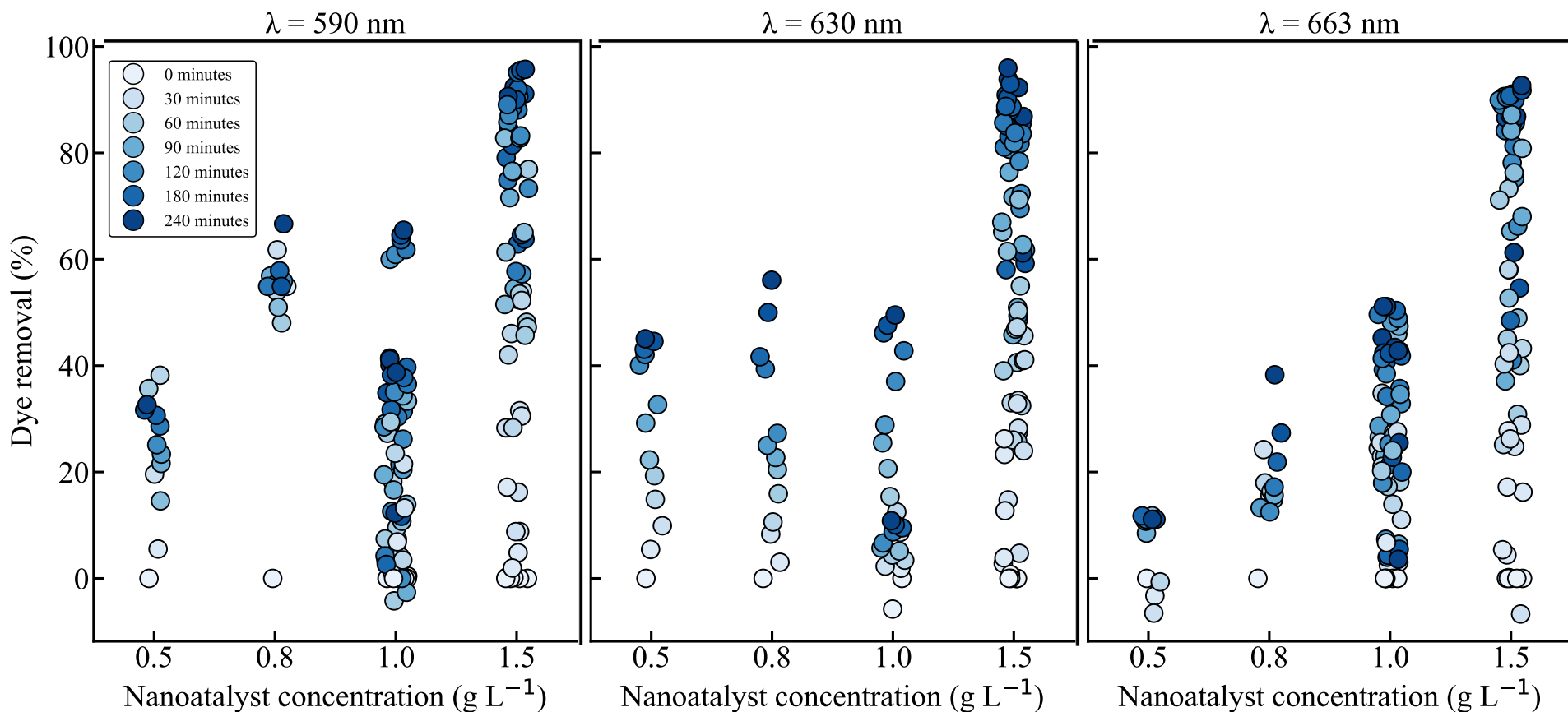


Figure 6. Effect of the nanocatalyst concentration and reaction time on dye removal (Sample size (N) for nanocatalyst concentration: 0.5 g L^{-1} (N = 51), 0.8 g L^{-1} (N = 51), 1.0 g L^{-1} (N = 122), 1.5 g L^{-1} (N = 176) / $V_{\text{reactor}} = 100 \text{ mL}$ | $T = 25 \pm 2^\circ\text{C}$ | $y_{\text{CV}} = 0.44$ and $y_{\text{MB}} = 0.56$ | under visible radiation with 600 W m^{-2}).

Estudo de Caso 3

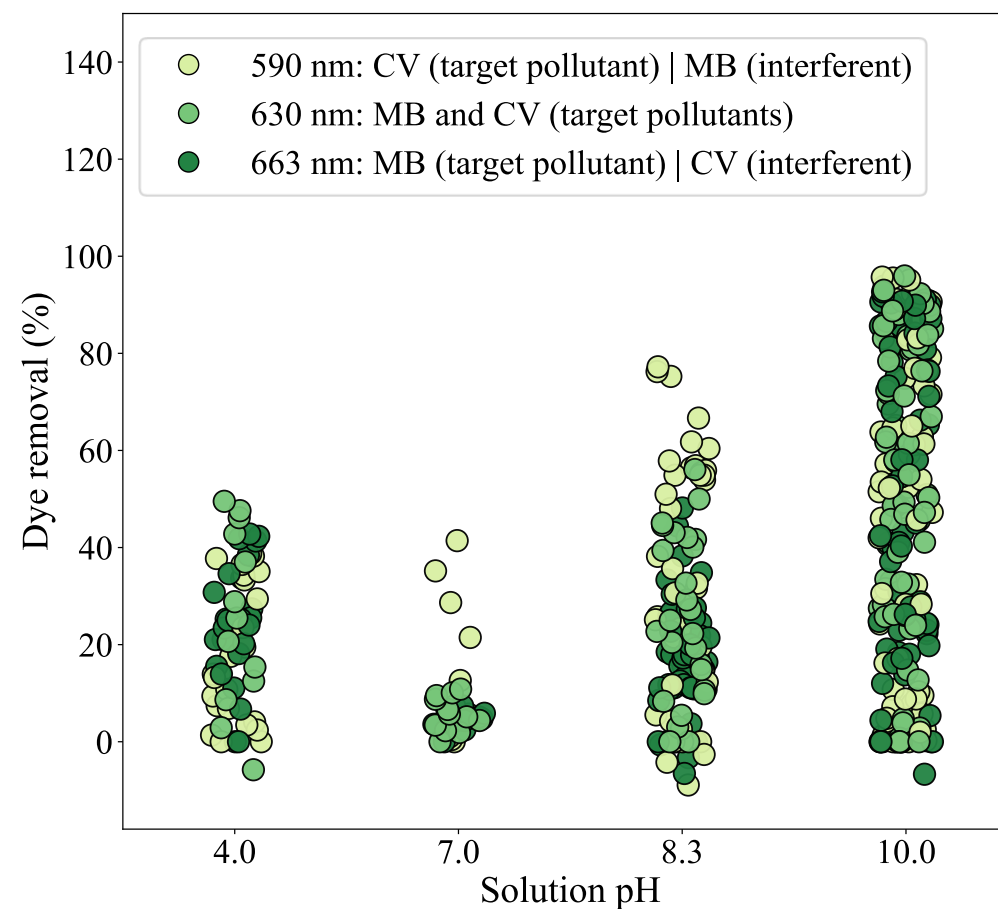
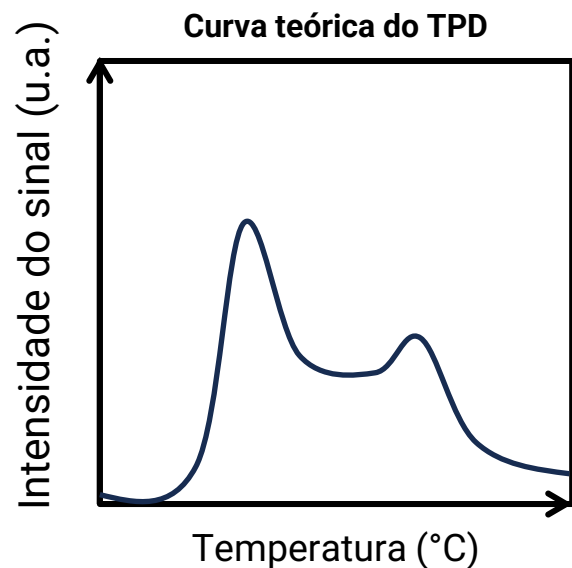


Figure 6. Effect of the pH and wavelength on dye removal. (Sample size (N): pH 4 (N = 35), pH 7 (N = 53), pH 8.3 (N = 136), pH 10 (N = 176)

$V_{\text{reactor}} = 100 \text{ mL}$ | $T = 25 \pm 2^\circ\text{C}$ | $y_{\text{CV}} = 0.44$ and $y_{\text{MB}} = 0.56$ | under visible radiation with 600 W m^{-2}).

Estudos de casos



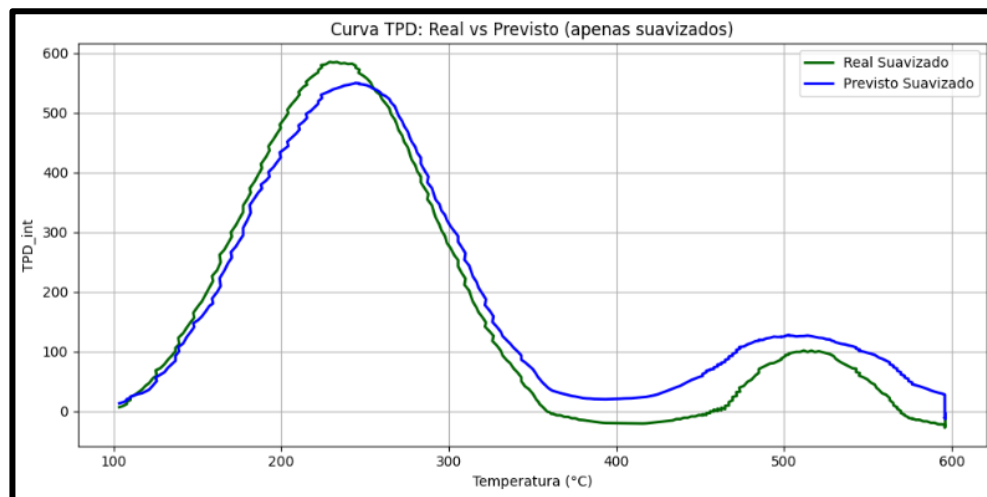
Estudo de Caso 4

Temática: Há disponível um difratômetro de raios X (DRX), um espectrofotômetro no infravermelho por Transformada Fourier (FTIR) e um analisador de área (porosimetria de N₂). Quero complementar os resultados dessas técnicas de caracterização com a técnica de Temperatura de Dessorção Programada (TPD) para determinar a acidez de um nanoadsorvente. No entanto, não há recursos financeiros nem disponibilidade do equipamento físico de TPD. Propõe-se a utilização de IA, com as técnicas acima e dados de TPD (retirados da literatura) para prever a curva TPD de um nanoadsorvente de interesse, cujos resultados de DRX, FTIR e Porosimetria de N₂ estão disponíveis.

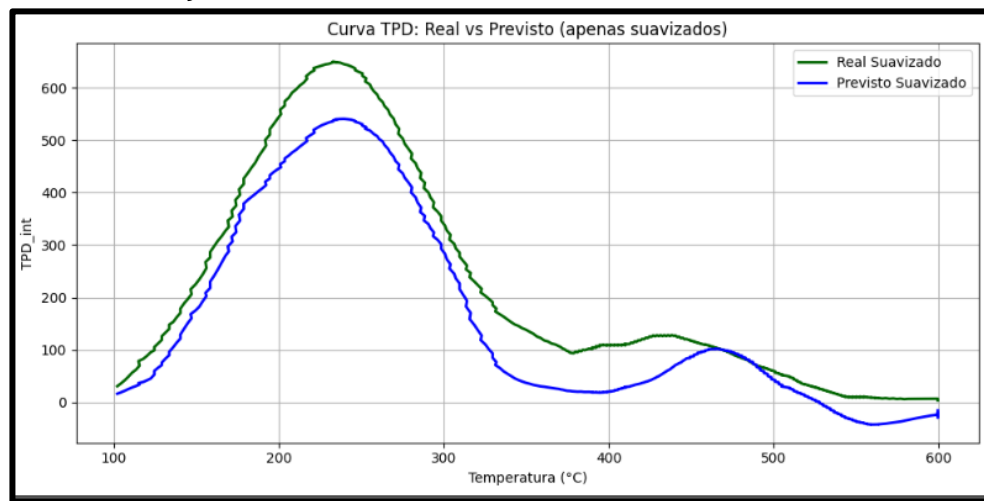
É possível **prever a curva TPD** para esse nanoadsorvente, tendo o resultado de caracterização das demais técnicas?

Estudo de Caso 4

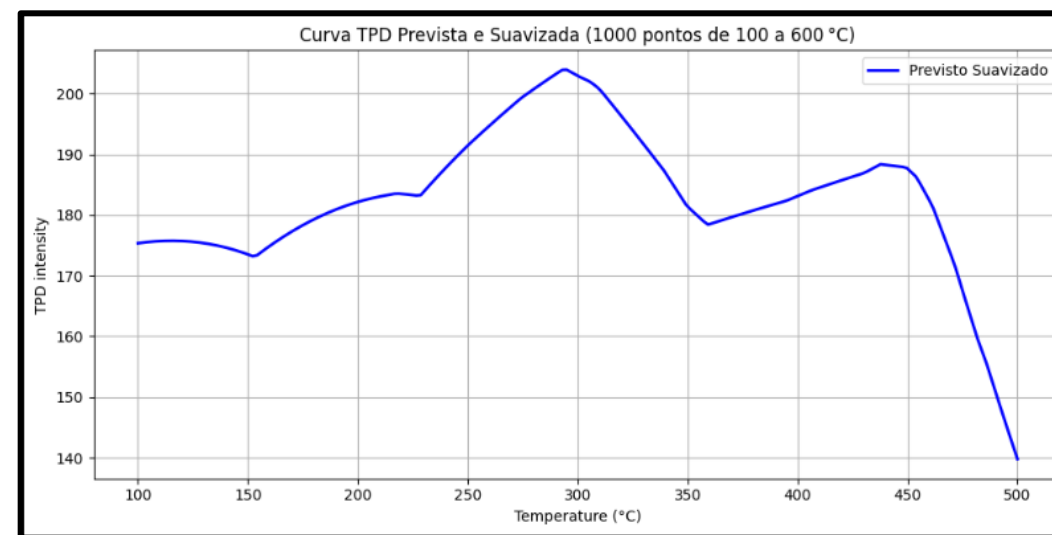
Predição 1. Nanozeólita FAU contendo 1% de Ni



Predição 2. Nanozeólita FAU contendo 10% de Ni



Predição 3. Nanozeólita FAU contendo 10% de Nb (amostra de interesse)



Estudos de casos

1

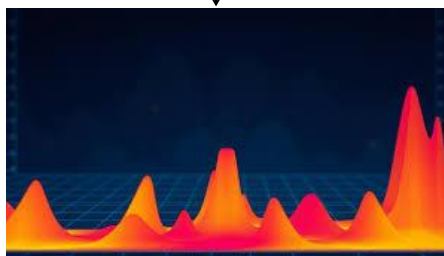


MEV-EDS

2



Adsorção de piridina em batelada

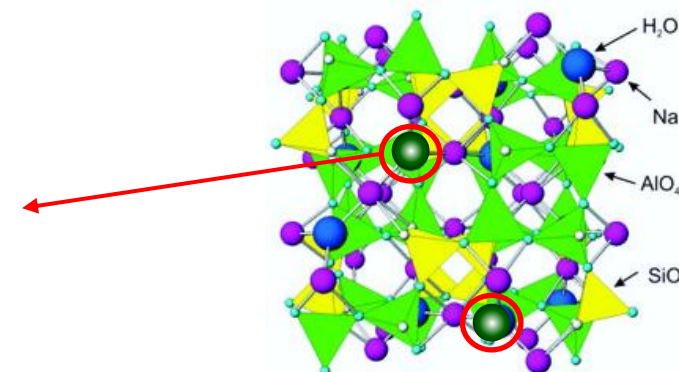


Estudo de Caso 5

Temática: Deseja-se prever a distribuição da acidez de nanomateriais (nanozeólitas modificadas e não modificadas com Nb) a partir de dados de adsorção em batelada (piridina), imagens de microscopia eletrônica de varredura e composição elementar (espectroscopia de energia dispersiva).

Substituição isomórfica

Átomos de Al e Si são trocados por átomos de Nb na estrutura zeolítica

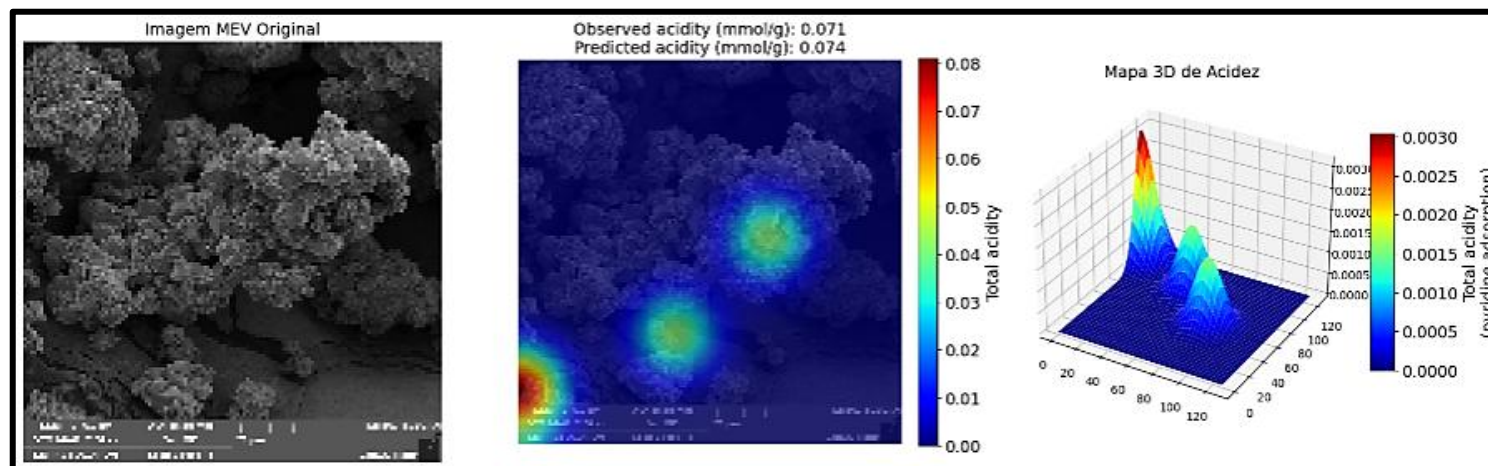


Nanozeólita

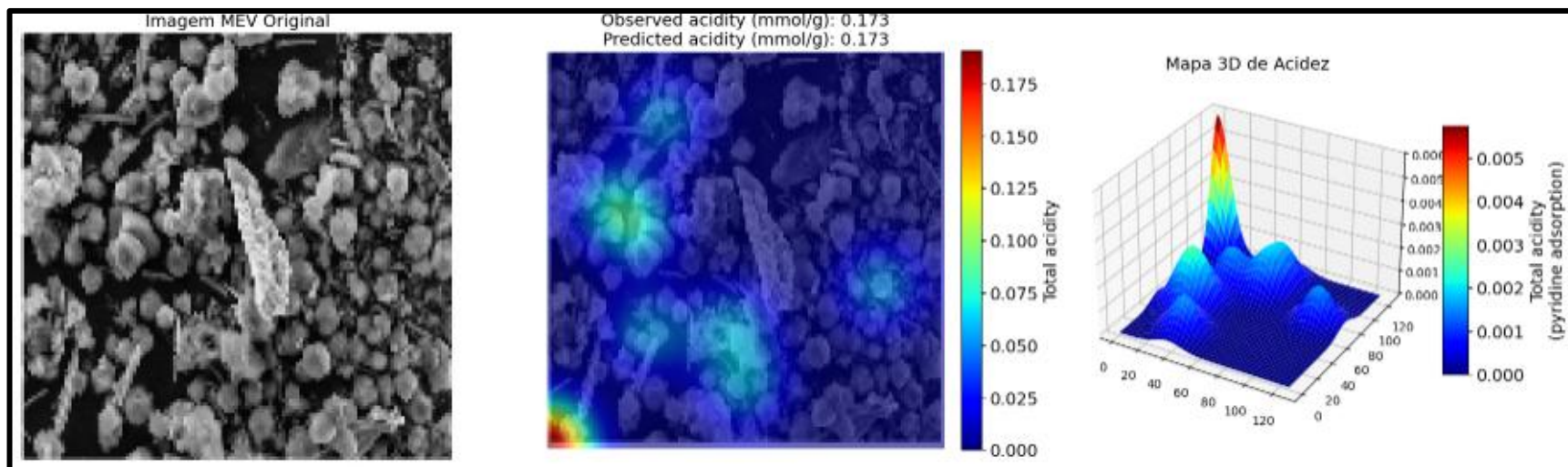
É possível **gerar um mapa de calor** para esse nanoadsorvente, mostrando a distribuição de uma alguma propriedade (acidez total) sobre sua estrutura

Estudo de Caso 5

Predição 1. Nanozeólita FAU contendo 10% Nb a partir de MEV-FEG.



Predição 2. Nanozeólita FAU contendo 10% Nb a partir de MEV.



Particularidades

A linguagem de programação Python é...



- **Case-sensitive**, isto é, uma variável k difere de uma variável K);
- **Zero-indexed**, isto é, inicia a contagem de elementos em 0 (0, 1, 2, 3...).

Informações adicionais



- Pode ser processada com CPU local ou GPU (processamento da Google)
- Alguns códigos podem ser muito pesados e demorados, necessitando máquinas mais potentes ou softwares mais sofisticados para uma aplicação;
- Execução em nuvem é recomendada para projetos pequenos;
- Busque registrar suas linhas de códigos para manter/garantir originalidade.

Links úteis

GitHub



Modelos de caixa branca e cinza

<https://github.com/LeandroOviedo>

Boas práticas de programação



YouTube

<https://www.youtube.com/watch?v=MwoA-pzzAQw>



<https://www.perplexity.ai/>



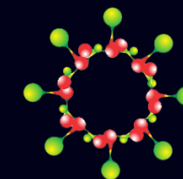
VAMOS PRATICAR???



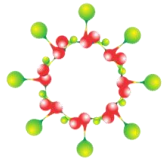
GRATO PELA ATENÇÃO!



Contato | leandro.oviedo@ufn.edu.br



Programa de Pós-Graduação em
Nanociências



Programa de Pós-Graduação em
Nanociências



UFPEL



Introdução à linguagem de programação Python: conceitos básicos e requisitos para aplicação de algoritmos de IA

EDITAL FAPERGS 06/2024 - PROGRAMA DE PESQUISA E DESENVOLVIMENTO VOLTADO A DESASTRES CLIMÁTICOS

Leandro Rodrigues Oviedo
Engenheiro Químico
Doutor em Nanociências – UFN



GRUPO DE PESQUISA
EM NANOMATERIAIS APLICADOS