IA para Engenharia Ambiental

Python e Ecossistema Científico

Cesar de Oliveira F. Silva

Setembro / 2026

Roteiro

Programação para ML: visão geral

- Aprendizado de Máquina é como um **laboratório virtual**: fazemos experiências com dados.
- Python é o **idioma** que usamos para conversar com esse laboratório.
- Objetivos da aula:
  - Entender o básico de Python como linguagem.
  - Ver estruturas fundamentais: variáveis, decisões, repetições, funções.
  - Conhecer bibliotecas centrais: **NumPy, Pandas, Matplotlib, Scikit-Learn**.

Por que Python?

- Sintaxe simples e próxima de pseudocódigo.
- Grande ecossistema científico:
  - **NumPy**: vetores e matrizes.
  - **Pandas**: tabelas de dados.
  - **Matplotlib**: gráficos.
  - **Scikit-Learn**: algoritmos de ML.
- Comunidade enorme e muitos exemplos prontos.
- Metáfora: Python é um **canivete suíço** para ciência de dados.

Ambientes de trabalho em Python

- **Interpretador** (terminal): bom para testes rápidos.
- **Notebooks Jupyter**: misturam texto, fórmulas, código e gráficos.
  - Ótimos para relatórios e experimentos em ML.
- **IDE** (VSCode, PyCharm, Spyder): edição de projetos maiores.
- Ideia: pense no notebook como um **caderno de laboratório digital**.

Variáveis: caixinhas com rótulo

Variável = **caixa na memória** com um nome (rótulo) e um valor dentro.
Exemplo:

```
idade = 21          # inteiro (int)
temp = 28.5         # ponto flutuante (float)
nome = "Ana"        # texto (str)
chovendo = True     # booleano (bool)

```

- Python descobre o tipo automaticamente (*tipagem dinâmica*).
- Metáfora: mudar o valor da variável é como trocar o conteúdo da caixa mantendo o mesmo rótulo.

Conversões e operações básicas

- Operadores: $+$, $-$, $\times$, $\div$, potência (`**`), módulo (`%`).
- Conversões: `int()`, `float()`, `str()`.
- Atenção: `"10" + "5"` concatena texto, não soma.

Decisões: `if`, `elif`, `else`

- Estruturas de decisão são como um **cruzamento** na estrada: o código escolhe qual caminho seguir.

```
temp = 30
if temp > 30:
    print("Muito quente")
elif temp >= 20:
    print("Clima agradável")
else:
    print("Frio")

```

- Condições lógicas: >, <, >=, <=, ==, !=.

Repetições: fazendo o computador repetir por você

Laços são como dizer: **“faça isso para cada elemento da lista”** ou **“repita enquanto esta condição for verdadeira”**.

`for` sobre uma coleção

```
nomes = ["Ana", "Bruno", "Carla"]

for n in nomes:
    print("Olá,", n)

```

`while` com condição

```
contador = 0

while contador < 3:
    print(contador)
    contador = contador + 1

```

Funções: receitas reutilizáveis

- Função = **receita de bolo**: recebe ingredientes (parâmetros), segue passos e devolve um resultado.

```
def soma(a, b):
    """Devolve a soma de a e b."""
    resultado = a + b
    return resultado

x = soma(3, 4)    # x = 7

```

- Vantagens: organização, reutilização, evita copiar e colar código.

Coleções em Python

- **Lista** (`list`): sequência mutável, ordenada.
- **Tupla** (`tuple`): sequência imutável.
- **Dicionário** (`dict`): mapeamento `chave` $\rightarrow$ `valor`.
- **Conjunto** (`set`): coleção sem ordem, sem repetição.

```
lista = [10, 20, 30]
tupla = (1.0, 2.0)
d = {"rio": "Tietê", "estado": "SP"}
conj = {1, 2, 2, 3}   # {1, 2, 3}

```

- Metáfora: listas e tuplas são **filas de caixas**; dicionários são **gavetas com etiquetas**.

Leitura e escrita de arquivos

- Precisamos ler dados de arquivos (ex.: CSVs de qualidade da água).
- Estrutura básica:

```
# Leitura linha a linha
with open("dados.txt", "r", encoding="utf-8") as f:
    for linha in f:
        print(linha.strip())

# Escrita
with open("saida.txt", "w", encoding="utf-8") as f:
    f.write("Resultado do modelo: 0.87\n")

```

- `with` garante que o arquivo será fechado corretamente.

Bibliotecas: caixas de ferramentas prontas

- Biblioteca = conjunto de funções e classes já implementadas.
- Importando bibliotecas:

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

```

- Metáfora: em vez de construir o martelo, você **importa** o conjunto de ferramentas.

NumPy: vetores e matrizes eficientes

- NumPy = motor numérico do Python científico.
- **Array NumPy** é como uma planilha só de números, muito rápida.

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([10, 20, 30])

soma = a + b          # [11 22 33]
media = a.mean()      # 2.0
mat = np.array([[1, 2],
                [3, 4]])
det = np.linalg.det(mat)

```

- Operações são **vetorizadas**: aplicadas a todos os elementos de uma vez.

Pandas: tabelas de dados (DataFrames)

- Pandas oferece o **DataFrame**: tabela com linhas e colunas nomeadas.
- Ideal para dados ambientais (estações, datas, variáveis).

```
import pandas as pd

df = pd.read_csv("qualidade_agua.csv")
print(df.head())            # primeiras linhas
print(df.columns)           # nomes das colunas

# Filtrar e calcular
df_sp = df[df["estado"] == "SP"]
media_ph = df_sp["pH"].mean()

```

Matplotlib: visualização de dados

- Matplotlib = **prancheta de desenho** para gráficos.

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(df["data"], df["OD"], label="Oxigênio dissolvido")
plt.xlabel("Data")
plt.ylabel("OD (mg/L)")
plt.title("Série temporal de OD")
plt.legend()
plt.show()

```

- Gráficos ajudam a enxergar padrões, outliers e tendências.

Scikit-Learn: algoritmos de ML em poucas linhas

- Biblioteca padrão de ML em Python.
- Metáfora: cada modelo é um **aparelho** que você pluga nos seus dados.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = df[["chuva", "temperatura"]]   # atributos
y = df["vazao"]                    # alvo

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2,
                     random_state=42)

modelo = LinearRegression()
modelo.fit(X_train, y_train)       # treinar
pred = modelo.predict(X_test)      # prever

```

Padrão de uso no Scikit-Learn

1. Escolher um modelo (`LinearRegression`, `RandomForest`, `SVC`, ...).
2. Criar o objeto do modelo.
3. Chamar `fit(X_train, y_train)` para treinar.
4. Chamar `predict(X_test)` para obter previsões.
5. Avaliar o desempenho com métricas (RMSE, acurácia, etc.).

Ideia central

Programar para ML é construir esse fluxo:
dados $\rightarrow$ **pré-processamento** $\rightarrow$ **modelo** $\rightarrow$ **avaliação**.

Mini-exemplo de pipeline em ML

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# 1. Ler dados
df = pd.read_csv("qualidade_agua.csv")

# 2. Escolher atributos e alvo
X = df[["chuva", "temperatura", "OD"]]
y = df["vazao"]

# 3. Separar treino e teste
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2, random_state=0)

# 4. Criar e treinar modelo
rf = RandomForestRegressor(n_estimators=100, random_state=0)
rf.fit(X_train, y_train)

# 5. Avaliar
y_pred = rf.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("RMSE =", rmse)

```

Resumo

- Vimos Python como **linguagem base** para ML.
- Exploramos:
  - Variáveis, decisões, laços e funções.
  - Leitura de arquivos e coleções.
  - Bibliotecas centrais: NumPy, Pandas, Matplotlib, Scikit-Learn.
- Próximos passos:
  - Praticar com pequenos scripts.
  - Reproduzir o mini-pipeline com seus próprios dados ambientais.
  - Preparar o terreno para **Redes Neurais Artificiais**.