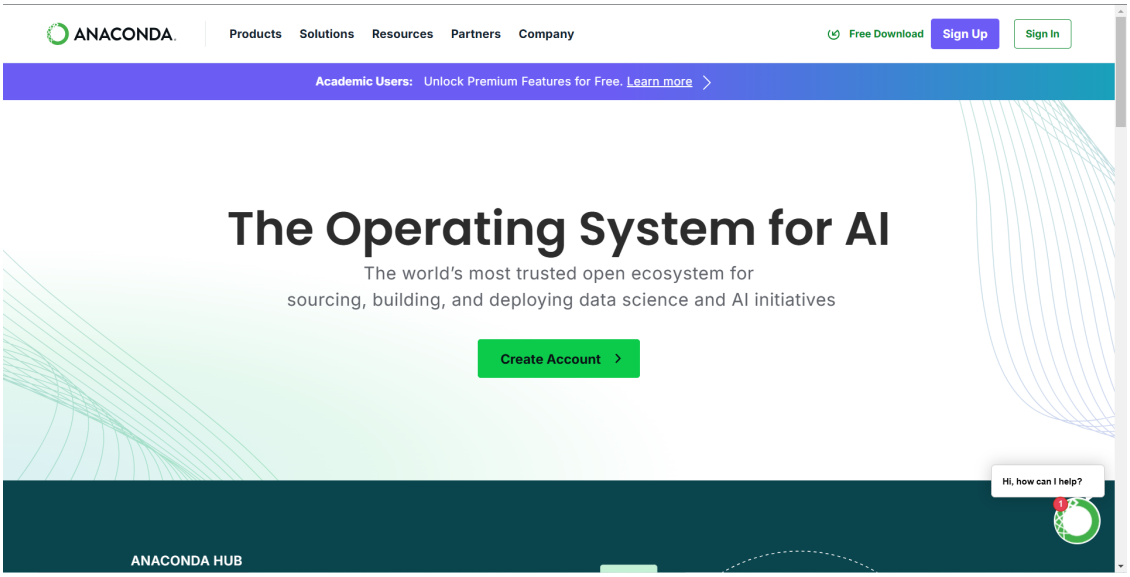
Day 1 - Cài đặt môi trường cho lập trình Python

Đăng-Nhã Nguyễn, Đoàn Ngọc Cường



1 Anaconda là gì và tại sao cần Anaconda?

1.1 Anaconda là gì?

Anaconda là một bộ công cụ giúp chúng ta dễ dàng cài đặt và quản lý Python cùng với các thư viện quan trọng để làm việc với dữ liệu, trí tuệ nhân tạo (AI), và khoa học máy tính. Nó bao gồm trình quản lý gói conda, Python, và nhiều công cụ hỗ trợ như Jupyter Notebook, Spyder.

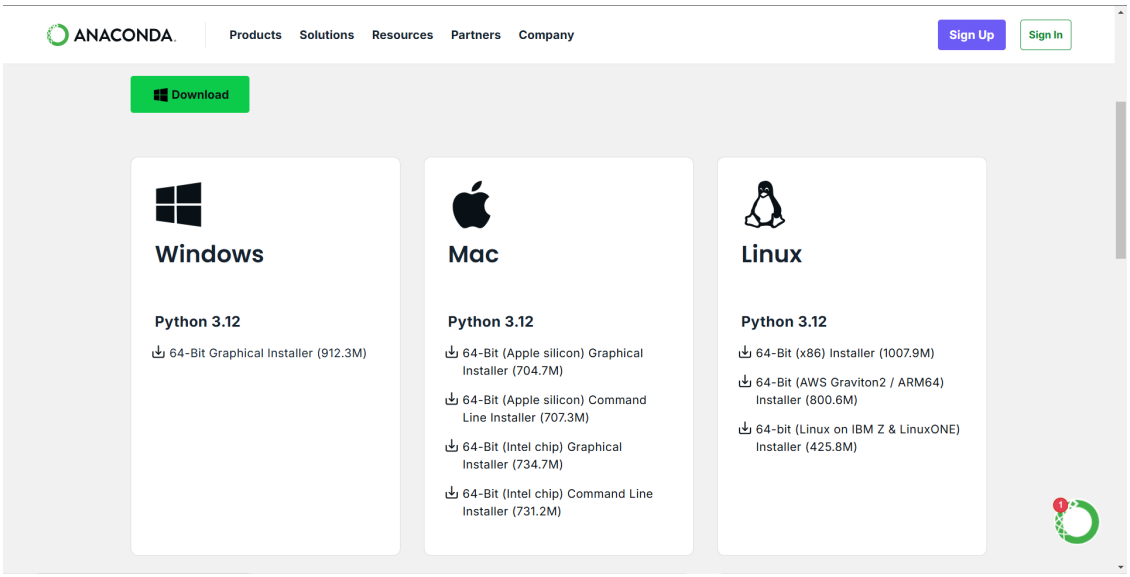
1.2 Lợi ích khi sử dụng Anaconda:

- **Quản lý môi trường dễ dàng:** Bạn có thể tạo, kích hoạt hoặc xóa các môi trường riêng biệt mà không ảnh hưởng đến môi trường chính của hệ thống.
- **Cài đặt thư viện nhanh chóng**: Anaconda đi kèm với nhiều thư viện phổ biến như `numpy`, `pandas`, `matplotlib`, v.v., giúp tiết kiệm thời gian cài đặt.
- **Hỗ trợ đa nền tảng:** Windows, macOS và Linux.

2 Cài đặt Anaconda

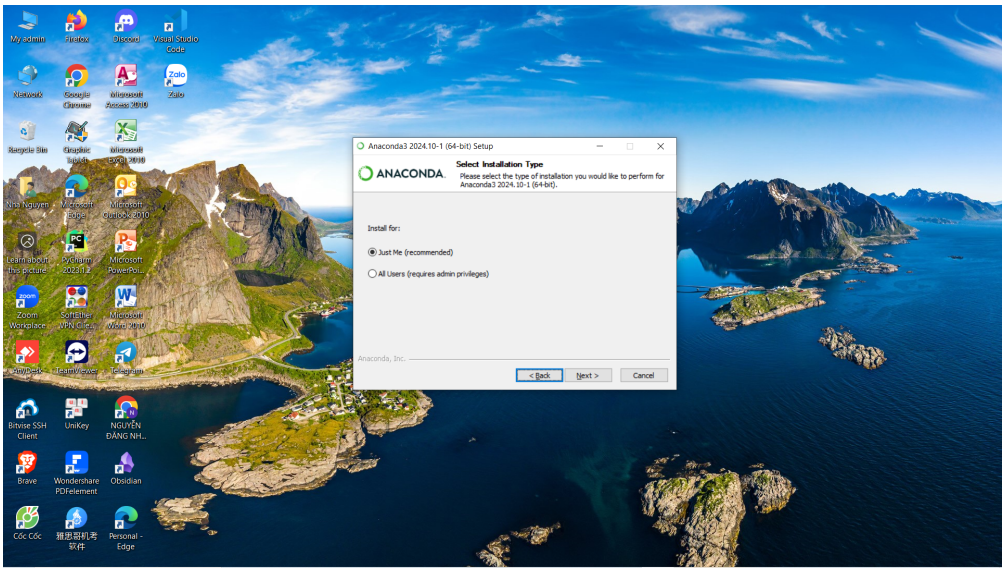
Bước 1: Tải và cài đặt Anaconda

- Truy cập trang web chính thức của Anaconda tại: https://www.anaconda.com/download.
- Tải xuống phiên bản phù hợp với hệ điều hành của bạn (window, macOS hoặc Linux).

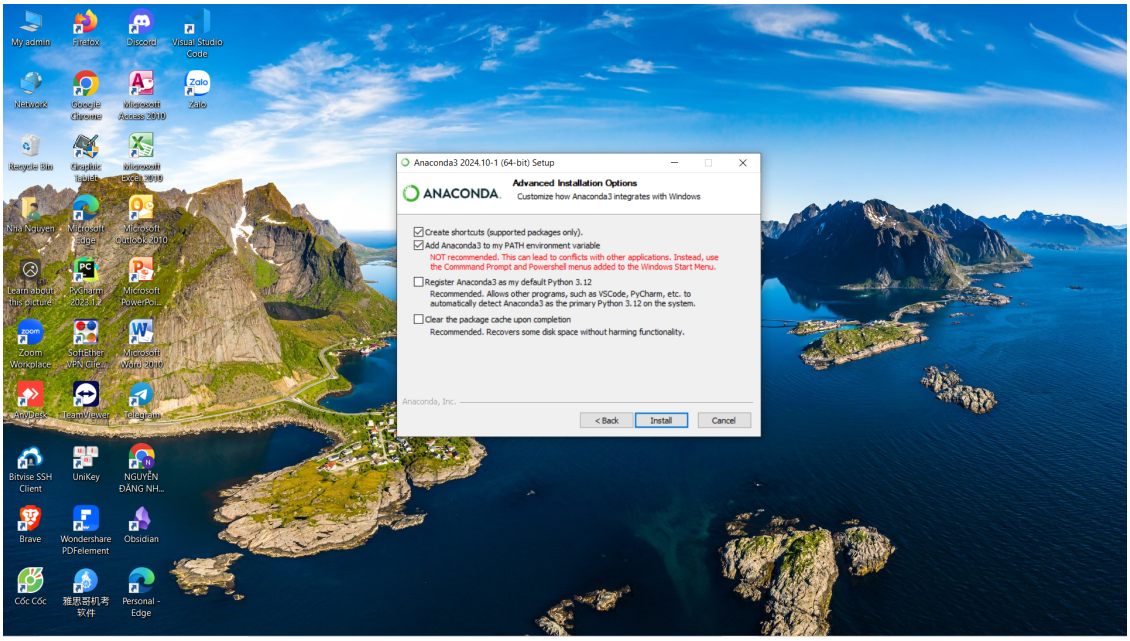


- Chạy file `.exe` vừa tải về để bắt đầu quá trình cài đặt.

Bước 2: Lưu ý trong quá trình cài đặt



- Chọn loại cài đặt, gồm có 2 loại: `Just me` và `All Users`.
  - **Just me:** Cài đặt Anaconda chỉ dành cho tài khoản người dùng Microsoft hiện tại. Tức là khi login vào tài khoản khác sẽ không có phần mềm này.
  - **All Users:** Cài đặt Anaconda cho tất cả người dùng trên máy tính. Nghĩa là cho dù dùng bất kỳ account nào đăng nhập cũng có thể sửa dụng anaconda.
  - **Khuyến nghị:** Trên máy tính cá nhân, mọi người có thể chọn bất kỳ tùy chọn nào trong hai tùy chọn trên. Ở đây mình sẽ chọn "Just Me" để tiến hành cài đặt mà không gặp trở ngại (ví dụ cần quyền admin).
- Chọn nơi lưu trữ: Đảm bảo mọi người nhớ đường dẫn cài đặt để dễ dàng quản lý sau này khi gặp lỗi.
- Tick vào ô "Add Anaconda3 to my PATH environment variable": Điều này giúp chúng ta có thể chạy Anaconda trực tiếp từ terminal bằng lệnh ‘conda‘.



- Tiếp tục các bước còn lại cho tới khi cài đặt thành công.

Bước 3: Kiểm tra cài đặt

Mở terminal hoặc command prompt, gõ lệnh:

```
conda --version
```

Nếu hiển thị phiên bản của Anaconda, bạn đã thêm PATH thành công.

Lưu ý: Nếu vẫn gặp lỗi, bạn có thể cần khởi động lại máy để cập nhật các thay đổi.

2.1 Không thể dùng lệnh ‘conda‘ trong terminal (Phần này dành cho những người gặp lỗi)

Nếu bạn đã cài đặt anaconda nhưng quên tick vào ô "Add Anaconda3 to my PATH environment variable" trong quá trình cài đặt hoặc vì lí do nào đó dẫn tới việc không thể dùng lệnh ‘conda –version‘ thì có thể thêm thủ công bằng các bước sau

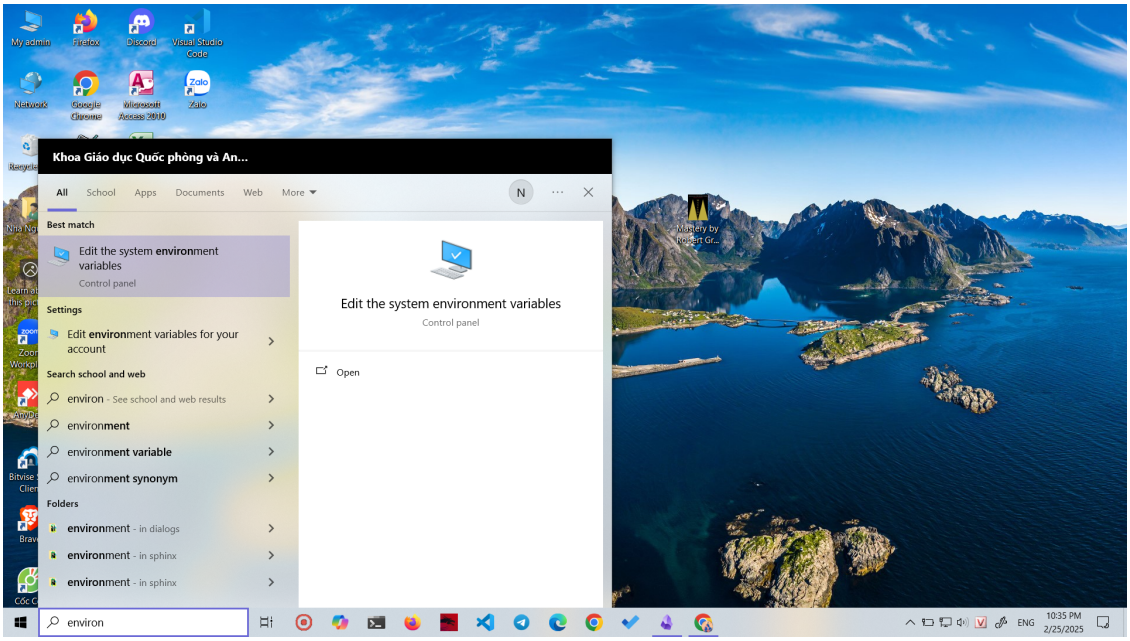
Bước 1: Xác định đường dẫn thư mục Anaconda3

- **Windows:** Mặc định, Anaconda thường được cài vào:
  - `C:\Users\<Tên người dùng>\Anaconda3`
  - hoặc `C:\Anaconda3`
- **Mac/Linux:** Đường dẫn mặc định thường là:
  - `/Users/<Tên người dùng>/anaconda3`
  - hoặc `/home/<Tên người dùng>/anaconda3`

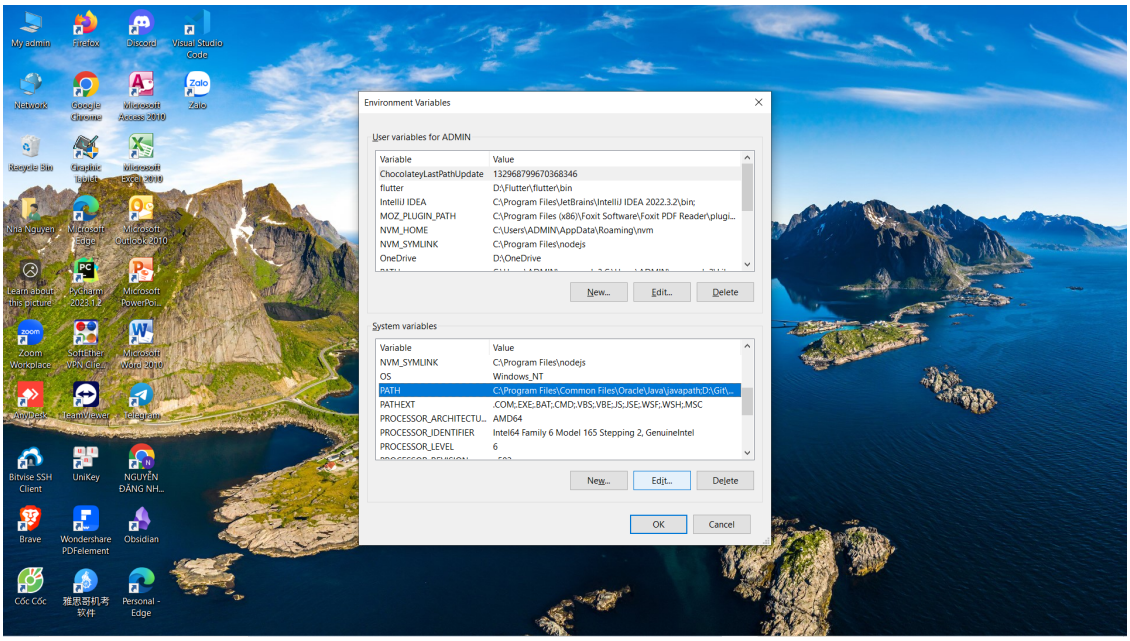
Bước 2: Thêm đường dẫn vào PATH

Cách thêm PATH trên Windows:

1. Mở **Start Menu** và tìm **Environment Variables** hoặc vào **Control Panel** → **System** → **Advanced System Settings** → **Environment Variables**.



2. Trong cửa sổ **System Properties**, chọn **Environment Variables**.
3. Ở phần **System variables**, tìm biến `Path` và nhấn **Edit**.



4. Nhấn **New** và thêm các đường dẫn sau:
   - `C:\Users\<Tên người dùng>\Anaconda3`
   - `C:\Users\<Tên người dùng>\Anaconda3\Scripts`
   - `C:\Users\<Tên người dùng>\Anaconda3\Library\bin`
5. Nhấn **OK** để lưu thay đổi.

Cách thêm PATH trên Mac/Linux:

1. Mở terminal.
2. Gõ lệnh:

```
nano ~/.bashrc  # Hoặc ~/.zshrc nếu dùng zsh

```

3. Thêm dòng sau vào cuối file:

```
export PATH="/Users/<Tên người dùng>/anaconda3/bin:$PATH"

```

4. Lưu file và tải lại cấu hình:

```
source ~/.bashrc  # Hoặc ~/.zshrc nếu dùng zsh

```

3. Khởi tạo môi trường với `conda` (sử dụng CMD)

Môi trường (environment) trong Conda giống như một hộp riêng biệt chứa các thư viện và phiên bản Python cụ thể để chạy code mà không ảnh hưởng đến hệ thống chung. Dưới đây là hướng dẫn cơ bản sử dụng `conda` trong CMD.

Bước 1: Kiểm tra phiên bản conda

Mở CMD và nhập lệnh kiểm tra:

```
conda --version
```

Nếu lệnh trả về phiên bản, bạn đã cài đặt thành công.

Bước 2: Liệt kê môi trường hiện có

Xem các môi trường ảo bằng lệnh:

```
conda env list
```

Bước 3: Tạo môi trường mới

Tạo môi trường với tên và phiên bản Python cụ thể:

```
conda create --name <ten_moi_truong> python=<phien_ban_python>
```

Ví dụ: `conda create -name myenv python=3.12`
Nhấn `y` để xác nhận khi được yêu cầu.

Bước 4: Kích hoạt và hủy kích hoạt môi trường

Kích hoạt môi trường

Dùng lệnh sau để kích hoạt môi trường:

```
conda activate <ten_moi_truong>
```

Hủy kích hoạt môi trường

Thoát môi trường khi hoàn tất:

```
conda deactivate
```

Bước 5: Cài đặt gói cần thiết

Khi môi trường được kích hoạt, bạn có thể cài đặt các gói:

```
conda install <ten_goi>
pip install <ten_goi>
```

Ưu tiên `conda install` để đảm bảo tương thích.

Bước 6: Quản lý môi trường

Liệt kê môi trường

Xem lại các môi trường:

```
conda env list
```

Xóa môi trường

Xóa môi trường không còn sử dụng:

```
conda remove --name <ten_moi_truong> --all
```

Việc sử dụng môi trường ảo giúp cô lập dự án, tránh xung đột và đảm bảo tính ổn định cho hệ thống.

4. Cài đặt Jupyter Notebook

Bước 1: Truy cập Anaconda Navigator

Mở **Anaconda Navigator**, tìm **Jupyter Notebook**, nhấn **Install** nếu chưa cài hoặc **Launch** để mở.

Bật nhanh Jupyter Notebook bằng CMD

Bước 1: Mở CMD tại thư mục dự án

Chuột phải vào thư mục, chọn **Open Terminal** hoặc **Open Command Prompt**.

Bước 2: Gõ lệnh mở Jupyter Notebook

```
jupyter notebook
```

Nếu gặp lỗi không nhận diện lệnh, thêm đường dẫn sau vào `PATH`:

- `C:\Users\Username\anaconda3\Scripts`

Bước 3: Mở tệp và bắt đầu làm việc

Trình duyệt sẽ mở giao diện Jupyter, chọn tệp `.ipynb` hoặc tạo tệp mới để lập trình.

- Chúc bạn thành công! -

BÀI TẬP

Yêu cầu cài đặt Anaconda (2 bài test)

1. **Test 1**: Mở Jupyter Notebook trên máy local, tạo một tệp notebook mới và chạy đoạn code:

```
print("Hello World")
```

2. **Test 2**:

   - Tạo môi trường `conda` mới (ví dụ: `conda create -name myenv python=3.8`).
   - Kích hoạt môi trường (`conda activate myenv`).
   - Cài `matplotlib`:

```
pip install matplotlib
```

   hoặc

```
conda install matplotlib
```

   - Chạy thử đoạn code sau (có thể do ChatGPT gợi ý) để kiểm tra việc vẽ đồ thị:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [i**2 for i in x]
plt.plot(x, y, marker='o')
plt.title("Matplotlib Test")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

   - Đảm bảo code chạy thành công và hiển thị được đồ thị.

Ngoài Anaconda, còn công cụ nào khác?

Ngoài Anaconda, bạn có thể sử dụng các công cụ khác để tạo môi trường ảo:

- `python -m venv`: Một công cụ tích hợp sẵn trong Python để tạo môi trường ảo.
- `virtualenv`: Công cụ mạnh mẽ hơn `venv` cho phép tùy chỉnh và quản lý môi trường ảo một cách linh hoạt.

Ví dụ: Tạo môi trường ảo bằng `venv`

```
# Tạo môi trường ảo
python -m venv myenv

# Kích hoạt môi trường ảo
# Windows:
myenv\Scripts\activate.bat
```

```

# macOS/Linux:
source myenv/bin/activate
```

Lựa chọn nào phù hợp hơn?

- Nếu bạn cần quản lý nhiều dự án phức tạp hoặc dùng thư viện lớn như TensorFlow, PyTorch, Anaconda là sự lựa chọn tối ưu. Vì Anaconda tích hợp sẵn nhiều thư viện khoa học dữ liệu, hỗ trợ quản lý phụ thuộc tốt và cài đặt dễ dàng các gói lớn, đặc biệt với GPU.
- Nếu chỉ cần môi trường nhẹ cho một vài dự án nhỏ, `venv` hoặc `virtualenv` có thể đáp ứng tốt. Vì chúng tạo môi trường nhanh, gọn và chỉ cài đặt đúng các gói cần thiết mà không gây nặng máy.

- Hết -

Day 2 - Basic Python

Đăng-Nhã Nguyễn

1 Lý thuyết:

Python là một ngôn ngữ lập trình dễ học và mạnh mẽ. Dưới đây là một số khái niệm cơ bản:

- **Biến (Variable):** Dùng để lưu trữ giá trị, có thể thay đổi trong quá trình chạy chương trình. Ví dụ: x = 5 (giá trị 5 được gán cho x), name = "python" (chuỗi "python" được gán cho biến name).
- **Kiểu dữ liệu (Data Types):** Python hỗ trợ nhiều kiểu dữ liệu như:
  - int (số nguyên):5, 10, 11, 13
  - float (số thực): 3.14, 3.2, 0.55, 1.10
  - str (chuỗi): "Hello", "aivietnam", "opeanai"
  - bool (giá trị logic): True, False (chú ý in hoa chữ cái đầu)
- **Toán tử (Operators):** Python hỗ trợ các toán tử như:
  - Toán tử số học: +, -, *, /, //,
  - Toán tử so sánh: ==, !=, >, <, >=, <=
  - Toán tử logic: and, or, not
- Cấu trúc điều kiện (if-else): Dùng để kiểm tra điều kiện và thực hiện lệnh tương ứng.

```python
if điều_kiện:
    # Thực hiện nếu điều kiện đúng
else:
    # Thực hiện nếu điều kiện sai
```

Ví dụ:

```python
age = 18
if age >= 18:
    print("Bạn đã đủ tuổi trưởng thành.")
else:
    print("Bạn chưa đủ tuổi trưởng thành.")
```

2 Bài tập

Bạn là một giáo viên muốn quản lý điểm số của học sinh. Hãy viết một chương trình Python thực hiện các bước sau:

- **Khai báo biến:**

  - Tạo biến *student_name* (kiểu chuỗi) để lưu tên học sinh.
  - Tạo hai biến *math_score* và *science_score* (kiểu số thực) để lưu điểm Toán và Khoa học của học sinh.

- **In thông tin ban đầu:**
  - In ra thông tin học sinh theo hai cách khác nhau bằng hàm `print`:
    * Cách 1: Dùng dấu + để nối chuỗi.
    * Cách 2: Dùng f-string (Ví dụ: `f"Tên: student_name"`).
  - Ví dụ output sẽ là:

    Tên: An, Điểm Toán: 8.5, Điểm Khoa học: 7.0
    Tên: An - Toán: 8.5 - Khoa học: 7.0

- **Tính toán với toán tử:**
  - Tính và in ra:
    * Tổng điểm: `math_score + science_score`.
    * Hiệu điểm: `math_score - science_score`.
    * Tích điểm: `math_score * science_score`.
    * Thương điểm: `math_score / science_score`.
  - Ví dụ output:

    Tổng điểm: 15.5
    Hiệu điểm: 1.5
    Tích điểm: 59.5
    Thương điểm: 1.2142857142857142

- **Thay đổi giá trị:** Giả sử học sinh được cộng thêm 1 điểm môn Toán vì làm bài tập tốt. Cập nhật giá trị math_score và in lại thông tin học sinh (dùng một trong hai cách ở bước 2).

- **Kiểm tra bằng `if-else`:**
  - Tính điểm trung bình mới sau khi cập nhật.
  - Dùng `if-elif-else` để đánh giá.
    * Nếu điểm trung bình >= 8.0: In "Xuất sắc".
    * Nếu điểm trung bình >= 6.5: In "Khá".
    * Nếu điểm trung bình >= 5.0: In "Trung bình".
    * Còn lại: In "Cần cố gắng".
  - Ví dụ đầu ra (giả sử điểm trung bình là 8.25):

    "Xuất sắc".

Day 3 - Implement Activation Function

Đăng-Nhã Nguyễn

1 Lý thuyết: Khai báo và gọi hàm

Hàm (function) trong Python là một khối mã được đặt tên, có thể tái sử dụng, thực hiện một nhiệm vụ cụ thể. Việc sử dụng hàm giúp mã nguồn trở nên dễ đọc, dễ bảo trì và tránh lặp lại code.

1.1 Khai báo hàm:

Cú pháp cơ bản:

```
def ten_ham(tham_so):
    # Khối mã thực hiện công việc
    return ket_qua  # (Tùy chọn)
```

- `def`: Từ khóa để định nghĩa hàm.
- `ten_ham`: Tên hàm do người dùng đặt (nên mô tả chức năng của hàm).
- `tham_so`: Các tham số (parameter) hay gọi là đối số (argument) là giá trị đầu vào mà hàm nhận được (có thể có hoặc không).
- `return`: Trả về kết quả (nếu cần), nếu không có `return`, hàm mặc định trả về `None`.

Ví dụ:

```
def cong_hai_so(a, b):
    return a + b
```

1.2 Gọi hàm

Bước khai báo hàm chỉ là bước khai báo logic của một khối code, muốn sử dụng nó ta phải thực sự gọi tới hàm đó, truyền vào tham số cần thiết thì mới có thể lấy được output mong muốn.

- Để sử dụng hàm, ta gọi tên hàm kèm theo các đối số (argument) tương ứng:

```
ket_qua = cong_hai_so(3, 5)  # ket_qua = 8
```

- Đối số phải khớp với số lượng và kiểu dữ liệu của tham số mà hàm yêu cầu.

1.3 Cách truyền đối số (argument):

- **Tham số vị trí (positional arguments):** Truyền theo thứ tự tham số trong định nghĩa hàm.

```
print(cong_hai_so(3, 5))  # 8
```

- **Tham số từ khóa (keyword arguments):** Chỉ định rõ tên tham số khi gọi hàm, không cần theo thứ tự.

```
print(cong_hai_so(b=5, a=3))  # 8
```

- Tham số mặc định (default arguments): Gán giá trị mặc định cho tham số, nếu không truyền thì dùng giá trị đó.

```
def nhan_so(x, y=2):
    return x * y
print(nhan_so(3))  # 6 (y mặc định là 2)
print(nhan_so(3, 4))  # 12 (y được truyền là 4)
```

1.4 Lưu ý

- Tên hàm và tham số nên đặt ngắn gọn, rõ nghĩa.
- Hàm có thể không có tham số hoặc nhiều tham số tùy mục đích sử dụng.
- là tùy chọn, nhưng nếu muốn hàm trả kết quả để dùng sau, cần có return.

2 Bài tập

Yêu cầu: Viết chương trình Python bao gồm:

- Khai báo các hàm activation function phổ biến: sigmoid, tanh, Softplus, ReLU, LeakyReLU, PReLU, Swish. (Công thức chi tiết xem lại ở phần bài tập)
- Thiết kế một hàm tổng quát `activate` nhận đầu vào là tên activation function (dạng chuỗi) và giá trị x (cùng tham số bổ sung nếu cần), sau đó sử dụng `if-elif` để gọi hàm tương ứng.

Gợi ý:

```
import math

# Sigmoid
def sigmoid(x):
    #Your code here

# Tanh
def tanh(x):
    #Your code here

```

```python
# Softplus
def softplus(x):
    #Your code here

# ReLU
def relu(x):
    #Your code here

# LeakyReLU
def leaky_relu(x, alpha=0.01):
    #Your code here

# PReLU
def prelu(x, alpha):
    #Your code here

# Swish
def swish(x):
    #Your code here
```

```python
def activate(function_name, x, alpha=None):
    if function_name == "":

    #Your code here

    else:
        raise ValueError("Unknown activation function")
```

Ví dụ:

activate("sigmoid", math.e)
» 0.9380968325850065
activate("softplus", 5)
» 5.006715348489118
activate("prelu", 2, 0.1)
» 2
activate("prelu", 2, 0.1)
» ValueError: PReLU requires an alpha parameter

Day 4 - Taylor Series for (e) and Exponential Functions

Đăng-Nhã Nguyễn

1 Lý thuyết

1.1 Vòng lặp for:

Cú pháp cơ bản: Vòng lặp `for` trong Python là một cấu trúc lặp cho phép duyệt qua các phần tử của một dãy (sequence) hoặc thực hiện lặp với số lần xác định.

- **For x in string:** Lặp qua từng ký tự trong chuỗi

```python
text = "Python"
for char in text:
    print(char)  # In lần lượt: P, y, t, h, o, n
```

- **For với range:** Sử dụng hàm `range()` để tạo dãy số và lặp qua chúng.

```python
for i in range(3):  # Lặp từ 0 đến 2
    print(i)  # Output: 0, 1, 2

# range(start, stop, step): bắt đầu, kết thúc (không bao gồm), bước nhảy
for i in range(1, 6, 2):  # Lặp: 1, 3, 5
    print(i)  # Outputn: 1, 3, 5
```

1.2 Số e (hằng số Euler) và công thức Taylor

- **Số** e: Là hằng số toán học xấp xỉ 2.71828, được định nghĩa qua chuỗi Taylor:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots$$

  Trong đó, $n!$ là giai thừa của n (ví dụ: $4! = 4 \times 3 \times 2 \times 1 = 24$).

- **Công thức Taylor cho** e^x: Số e^x (hàm mũ cơ số e) có thể được tính gần đúng bằng chuỗi Taylor:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

  - x: Giá trị thực bất kỳ (ví dụ: $x = 2$ thì tính e^2).
  - Chuỗi này hội tụ đến giá trị thực của e^x khi số hạng tăng lên.

- **Liên hệ với Python**: Dùng vòng lặp `for` để tính tổng chuỗi Taylor, kết hợp với giai thừa và lũy thừa (x^n).

1.3 Giai thừa trong Python

- Giai thừa $n!$ là tích của tất cả các số nguyên dương từ 1 đến n.
- Có thể tính bằng vòng lặp `for` hoặc dùng `math.factorial()` từ thư viện `math`.

2 Bài tập

Bài 1: Tính giai thừa bằng vòng lặp `for`

Yêu cầu: Viết hàm `factorial(n)` sử dụng vòng lặp `for` để tính giai thừa của một số nguyên dương n. In kết quả cho $n = 6$.

```python
def factorial(n):
    # Your code here
    return result

# Test case
n = 6
print(f"Giai thừa của {n} là: {factorial(n)}")  # Kết quả: 720
```

Bài 2: Tính số e bằng chuỗi Taylor

Yêu cầu: Viết hàm `calculate_e(n)` sử dụng vòng lặp `for` để tính gần đúng số e với n số hạng đầu tiên của chuỗi Taylor.

- Tính e với $n = 8$.
- So sánh với `math.e`.

```python
import math

def calculate_e(n):
    # Your code here
    return e_approx

# Test case
n = 8
e_approx = calculate_e(n)
print(f"Số e gần đúng với {n} số hạng: {e_approx}")
print(f"Số e thực tế: {math.e}")
print(f"Độ lệch: {abs(math.e - e_approx)}")
```

Bài 3: Tính e^x bằng chuỗi Taylor

Yêu cầu: Viết hàm `calculate_exp(x, n)` sử dụng vòng lặp `for` để tính gần đúng e^x với n số hạng đầu tiên của chuỗi Taylor.

- Tính e^2 với $n = 10$.

- So sánh với giá trị thực từ `math.exp(2)`.

```python
import math

def calculate_exp(x, n):
    # Your code here
    return exp_approx

# Test case
x = 2
n = 10
exp_approx = calculate_exp(x, n)
print(f"e^{x} gần đúng với {n} số hạng: {exp_approx}")
print(f"e^{x} thực tế: {math.exp(x)}")
print(f"Độ lệch: {abs(math.exp(x) - exp_approx)}")
```

Gợi ý mở rộng

- Thử thay đổi giá trị n trong bài 2 và bài 3 để quan sát độ chính xác thay đổi.
- Thêm kiểm tra đầu vào (ví dụ: $n > 0$) để tránh lỗi.

Day 5 - Practice Programming

Đăng-Nhã Nguyễn

1 Bài tập

Bài tập 1: Tìm tất cả các số nguyên tố trong khoảng từ 1 đến N

Mô tả bài toán

Viết một chương trình yêu cầu người dùng nhập một số nguyên dương N, sau đó in ra tất cả các số nguyên tố từ 1 đến N. Một số nguyên tố là số chỉ chia hết cho 1 và chính nó (ví dụ: 2, 3, 5, 7,...).

Mô tả Input, Output

- **Input**: Một số nguyên dương N (do người dùng nhập).
- **Output**: Các số nguyên tố từ 1 đến N, cách nhau bởi dấu phẩy và khoảng trắng (", ").
- **Ví dụ**:
  - Input: 10
  - Output: $2, 3, 5, 7$

Gợi ý làm bài

Một số n là số nguyên tố nếu $n > 1$ và không tồn tại số nguyên d từ 2 đến $\sqrt{n}$ sao cho n chia hết cho d.

- Với mỗi số n từ 2 đến N, thử chia n cho các số từ 2 đến $\sqrt{n}$.
- Nếu không có số nào trong khoảng này mà n chia hết (dư bằng 0), thì n là số nguyên tố.
- Nếu tồn tại ít nhất một số mà n chia hết, thì n không phải số nguyên tố.

```python
def find_primes(N):
    # Your code here
    return _

# Nhập N và gọi hàm
N = int(input("Nhập N: "))
result = find_primes(N)
print(result)
```

```
Nhập N: 15
Output: 2, 3, 5, 7, 11, 13
```

Bài tập 2: Tính tổng các số hoàn hảo nhỏ hơn N

Mô tả bài toán

Viết một chương trình yêu cầu người dùng nhập một số nguyên dương N, sau đó tính tổng tất cả các số hoàn hảo nhỏ hơn N. Một số hoàn hảo là số mà tổng các ước số thực sự của nó (không tính chính nó) bằng chính số đó (ví dụ: $6 = 1 + 2 + 3$).

Mô tả Input, Output

- **Input**: Một số nguyên dương N (do người dùng nhập).
- **Output**: Tổng của các số hoàn hảo nhỏ hơn N.
- **Ví dụ**:
  - Input: 10
  - Output: 6 (vì 6 là số hoàn hảo duy nhất nhỏ hơn 10)

Gợi ý làm bài

Một số n là số hoàn hảo nếu tổng các ước số thực sự của n (từ 1 đến $n-1$) bằng n.

- Với mỗi số n, tìm tất cả các ước số thực sự bằng cách thử chia n cho các số từ 1 đến $n-1$.
- Nếu n chia hết cho một số d (dư bằng 0), thì d là ước số thực sự.
- Tính tổng các ước số này và so sánh với n.

```python
def sum_perfect_numbers(N):
    # Your code here
    return _

# Nhập N và gọi hàm
N = int(input("Nhập N: "))
result = sum_perfect_numbers(N)
print(result)
```

```
Nhập N: 30
Output: 34 (6, 28)
```

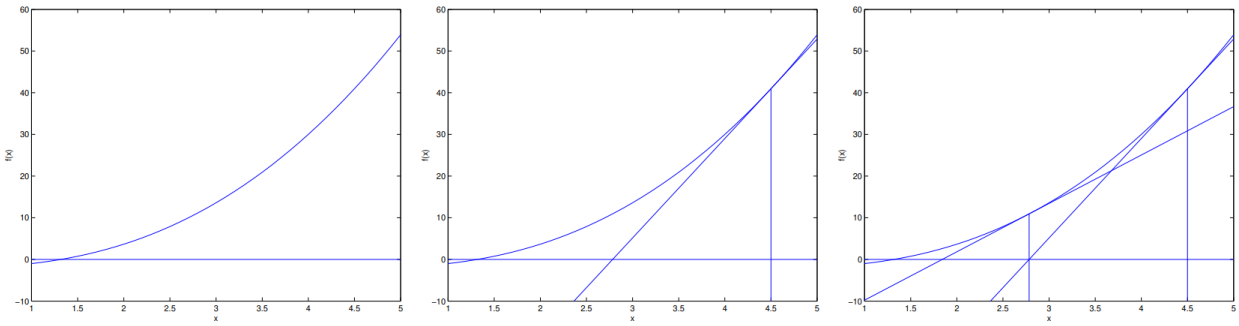
Day 6 - Derivative and Its Application: Newton Method

Đăng-Nhã Nguyễn

1 Lý thuyết

1.1 Phương pháp Newton

Phương pháp Newton (Newton-Raphson) là một kỹ thuật lặp để tìm nghiệm gần đúng của phương trình phi tuyến $f(x) = 0$. Phương pháp này dựa trên việc sử dụng đạo hàm và tiếp tuyến của hàm tại một điểm để dự đoán nghiệm tiếp theo.



Hình 1: Mô phòng từng bước tính toán của phương pháp Newton

- **Ý tưởng cơ bản**: Từ một giá trị ban đầu x_0 (ước lượng gần đúng của nghiệm), ta xây dựng một chuỗi các giá trị $x_1, x_2, x_3, \ldots$ sao cho mỗi bước tiến gần hơn đến nghiệm thực sự. Công thức lặp của phương pháp Newton là:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

  Trong đó:

  - $f(x_n)$: Giá trị của hàm tại x_n.
  - $f'(x_n)$: Đạo hàm bậc nhất của $f(x)$ tại x_n.

- **Các bước thực hiện**:

  1. Chọn một giá trị khởi đầu x_0 (dựa trên ước lượng ban đầu hoặc đồ thị của hàm).
  2. Tính $f(x_0)$ và $f'(x_0)$.
  3. Áp dụng công thức Newton để tính x_1.
  4. Lặp lại quá trình cho đến khi $|x_{n+1} - x_n|$ nhỏ hơn một sai số cho phép (thường gọi là ϵ), hoặc $f(x_n)$ đủ gần 0.

- **Ưu điểm**:

  - Hội tụ rất nhanh (hội tụ bậc hai) nếu giá trị ban đầu gần nghiệm.

- **Nhược điểm**:
  - Cần tính đạo hàm, điều này có thể phức tạp với một số hàm.
  - Nếu $f'(x_n) = 0$ hoặc giá trị ban đầu x_0 quá xa nghiệm, phương pháp có thể không hội tụ.

- **Ví dụ**: Tìm nghiệm của $f(x) = x^2 - 2 = 0$ (nghiệm là $\sqrt{2}$).
  - Đạo hàm: $f'(x) = 2x$.
  - Công thức lặp: $x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n}$.
  - Với $x_0 = 1$, ta tính:
    * $x_1 = 1 - \frac{1-2}{2 \cdot 1} = 1 + 0.5 = 1.5$.
    * $x_2 = 1.5 - \frac{1.5^2 - 2}{2 \cdot 1.5} = 1.5 - \frac{0.25}{3} \approx 1.4167$.
    * Tiếp tục lặp, giá trị tiến gần $\sqrt{2} \approx 1.4142$.

1.2 Lý thuyết về vòng lặp while, continue, break

Vòng lặp `while` trong lập trình được sử dụng để lặp lại một khối lệnh miễn là điều kiện của nó vẫn đúng. Cùng với đó, các từ khóa `continue` và `break` giúp kiểm soát luồng thực thi.

- **Vòng lặp while**:
  - Cú pháp (trong Python):

```python
while dieu_kien:
    # Khoi lenh
```

  - Ý nghĩa: Thực hiện khối lệnh bên trong miễn là `dieu_kien` trả về `True`. Nếu `dieu_kien` trở thành `False`, vòng lặp kết thúc.
  - Ví dụ:

```python
i = 0
while i < 5:
    print(i)
    i += 1
# In ra: 0, 1, 2, 3, 4
```

- **Từ khóa** `continue`:
  - Khi gặp `continue`, chương trình sẽ bỏ qua phần còn lại của khối lệnh trong lần lặp hiện tại và quay lại kiểm tra điều kiện để bắt đầu lần lặp tiếp theo.
  - Ví dụ:

```python
i = 0
while i < 5:
    i += 1
    if i == 3:
        continue
    print(i)
# In ra: 1, 2, 4, 5 (bo qua 3)
```

- **Từ khóa** `break`:
  - Khi gặp `break`, chương trình sẽ thoát hoàn toàn khỏi vòng lặp, bất kể điều kiện của `while` có còn đúng hay không.
  - Ví dụ:

```python
i = 0
while i < 5:
    if i == 3:
        break
    print(i)
    i += 1
# In ra: 0, 1, 2 (dung tai 3)
```

2 Bài Tập

Bài tập 1: Ứng dụng vòng lặp While

Đề bài: Viết chương trình sử dụng vòng lặp `while` yêu cầu người dùng nhập vào một số nguyên dương n, sau đó tính tổng các chữ số của n.

Gợi ý:

- Nhập số nguyên dương n.
- Khởi tạo tổng ban đầu $S = 0$.
- Sử dụng vòng lặp `while` để lấy từng chữ số cuối cùng bằng phép chia dư.
- Cộng chữ số vào S, sau đó loại bỏ chữ số cuối bằng phép chia nguyên.
- Lặp lại cho đến khi $n = 0$, sau đó in kết quả.

Bài tập 2: Dùng vòng lặp While để giải phương trình tìm nghiệm bằng phương pháp Newton

Đề bài: Viết chương trình sử dụng vòng lặp `while` để tìm nghiệm của phương trình $f(x) = x^3 - 3x + 1 = 0$ bằng phương pháp Newton. Yêu cầu:

- Chọn giá trị ban đầu $x_0 = 5$.
- Dừng khi sai số giữa hai lần lặp liên tiếp nhỏ hơn $\epsilon = 0.0001$.
- In ra giá trị nghiệm gần đúng và số lần lặp.

Gợi ý:

- Hàm $f(x) = x^3 - 3x + 1$ và đạo hàm $f'(x) = 3x^2 - 3$ được định nghĩa.
- Vòng lặp `while True` chạy vô hạn, nhưng sử dụng `break` để thoát khi:
  - Sai số $|x_{n+1} - x_n| < \epsilon$.
  - Hoặc vượt quá số lần lặp tối đa (tránh vòng lặp vô hạn).
- Mỗi lần lặp, công thức Newton được áp dụng: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
- Kết quả là nghiệm gần đúng của phương trình.

Day 7 - Basic List and It's Build-in Function

Đăng-Nhã Nguyễn

Phần 1: Lý thuyết - Các khái niệm cơ bản về List

1. Khai báo List và Truy xuất phần tử

- **Khai báo List**:

```
my_list = [1, 2, 3, 4]       % Khai báo trực tiếp
empty_list = []              % List rỗng
mixed_list = [1, "hello", 3.14]  % Chứa nhiều kiểu dữ liệu
```

- **Truy xuất phần tử**: Dùng chỉ số (index), bắt đầu từ 0. Chỉ số âm truy xuất từ cuối về đầu (-1 là phần tử cuối).

```
my_list = [10, 20, 30, 40]
print(my_list[0])   % Output: 10
print(my_list[-1])  % Output: 40
```

2. Thêm, sửa, xóa phần tử

- **Thêm phần tử**:
  - `.append(x)`: Thêm phần tử `x` vào cuối List.

```
my_list = [1, 2, 3]
my_list.append(4)  % my_list = [1, 2, 3, 4]
```

  - `.insert(i, x)`: Chèn phần tử `x` vào vị trí `i`.

```
    my_list.insert(1, 10)  % my_list = [1, 10, 2, 3, 4]
```

- **Sửa phần tử**: Gán giá trị mới cho chỉ số cụ thể.

```
my_list[0] = 100  % my_list = [100, 10, 2, 3, 4]
```

- **Xóa phần tử**:
  - `.remove(x)`: Xóa phần tử `x` đầu tiên tìm thấy.

```
my_list.remove(10)  % my_list = [100, 2, 3, 4]
```

  - `.pop(i)`: Xóa và trả về phần tử tại chỉ số `i` (mặc định là cuối).

```
value = my_list.pop()  % value = 4, my_list = [100, 2, 3]
```

3. Duyệt List bằng vòng lặp

- **Dùng `for`:**
  - Duyệt trực tiếp qua phần tử:

```
my_list = [1, 2, 3]
for item in my_list:
    print(item)  % Output: 1, 2, 3
```

  - Duyệt bằng chỉ số:

```
for i in range(len(my_list)):
    print(my_list[i])  % Output: 1, 2, 3
```

- **Dùng `while`:**

```
i = 0
while i < len(my_list):
    print(my_list[i])
    i += 1  % Output: 1, 2, 3
```

4. Một số thao tác cơ bản

- `len()`: Trả về độ dài của List.

```
my_list = [1, 2, 3]
print(len(my_list))  % Output: 3
```

- `min()`, `max()`: Tìm giá trị nhỏ nhất/lớn nhất.

```
print(min(my_list))  % Output: 1
print(max(my_list))  % Output: 3
```

- `sum()`: Tính tổng các phần tử.

```
print(sum(my_list))  % Output: 6
```

- `sorted()`: Trả về List mới đã sắp xếp.

```
my_list = [3, 1, 2]
print(sorted(my_list))  % Output: [1, 2, 3]
```

- `reverse()`: Đảo ngược thứ tự List.

```
my_list.reverse()  % my_list = [2, 1, 3]
```

- `index(x)`: Trả về chỉ số đầu tiên của `x`.

```
print(my_list.index(1))  % Output: 1
```

- `count(x)`: Đếm số lần xuất hiện của `x`.

```
my_list = [1, 2, 1, 3]
print(my_list.count(1))  % Output: 2
```

Phần 2: Bài tập

Bài 1: Tạo danh sách số nguyên từ bàn phím

Yêu cầu:

1. Nhập số lượng `n` từ bàn phím (số phần tử của List).
2. Nhập `n` số nguyên từ bàn phím và lưu vào List.
3. In ra List vừa tạo.

Ví dụ:

```
Nhập số lượng phần tử: 4
Nhập số thứ 1: 5
Nhập số thứ 2: 2
Nhập số thứ 3: 8
Nhập số thứ 4: 1
List: [5, 2, 8, 1]
```

Bài 2: Tính toán trên List

Yêu cầu:

1. Dùng List từ Bài 1 (hoặc tạo List cố định: `[5, 2, 8, 1]`).
2. Tính tổng các phần tử bằng `sum()`.
3. Tìm giá trị lớn nhất và nhỏ nhất bằng `max()` và `min()`.
4. In kết quả.

Ví dụ::

```
List: [5, 2, 8, 1]
Tổng: 16
Giá trị lớn nhất: 8
Giá trị nhỏ nhất: 1
```

Bài 3: Thêm và xóa phần tử

Yêu cầu:

1. Tạo List ban đầu: `numbers = [10, 20, 30, 40]`.
2. Thêm một phần tử (nhập từ bàn phím) vào cuối List bằng `append()`.
3. Chèn một phần tử (nhập từ bàn phím) vào vị trí bất kỳ (nhập chỉ số từ bàn phím) bằng `insert()`.

4. Xóa một phần tử theo giá trị (nhập từ bàn phím) bằng `remove()`.
5. In List sau mỗi thao tác.

Ví dụ::

```
List ban đầu: [10, 20, 30, 40]
Nhập phần tử để thêm vào cuối: 50
List sau khi thêm: [10, 20, 30, 40, 50]
Nhập vị trí để chèn: 2
Nhập phần tử để chèn: 25
List sau khi chèn: [10, 20, 25, 30, 40, 50]
Nhập giá trị để xóa: 20
List sau khi xóa: [10, 25, 30, 40, 50]
```

Bài 4: Duyệt List bằng vòng lặp

Yêu cầu:

1. Dùng List từ Bài 3 (hoặc tạo List cố định: `[10, 25, 30, 40, 50]`).
2. Dùng vòng lặp `for` để in từng phần tử kèm chỉ số index của nó.
3. Dùng vòng lặp `while` để in lại tương tự.

Ví dụ::

```
Duyệt bằng for:
Phần tử tại chỉ số 0 là 10
Phần tử tại chỉ số 1 là 25
Phần tử tại chỉ số 2 là 30
Phần tử tại chỉ số 3 là 40
Phần tử tại chỉ số 4 là 50
Duyệt bằng while:
Phần tử tại chỉ số 0 là 10
Phần tử tại chỉ số 1 là 25
Phần tử tại chỉ số 2 là 30
Phần tử tại chỉ số 3 là 40
Phần tử tại chỉ số 4 là 50
```

Bài 5: Tìm kiếm phần tử trong List

Yêu cầu:

1. Dùng List từ Bài 4 (hoặc tạo List cố định: `[10, 25, 30, 40, 50]`).
2. Nhập một số từ bàn phím.
3. Kiểm tra số đó có trong List không (dùng `in`).

4. Nếu có, in ra vị trí đầu tiên của nó bằng `index()`. Nếu không, thông báo "Không tìm thấy".

Ví dụ::

```
List: [10, 25, 30, 40, 50]
Nhập số cần tìm: 30
30 có trong List, vị trí: 2
Nhập số cần tìm: 15
Không tìm thấy 15 trong List
```

Day 8 - List Comprehension & Dictionary

Đăng-Nhã Nguyễn

Phần 1: Lý thuyết

1. List Comprehension

- List comprehension là cách tạo List ngắn gọn bằng một dòng code, thay thế cho vòng lặp `for` truyền thống.
- **Cú pháp cơ bản**: `[biểu_thức for biến in iterable]`.

```
new_list = [expression for item in iterable if condition]
```

  Trong đó:

  - **expression:** Biểu thức áp dụng lên từng phần tử.
  - **item:** Phần tử lấy từ iterable (list, tuple, range, v.v.).
  - **condition:** Chỉ thêm phần tử vào list nếu điều kiện đúng.

- **Ví dụ**:

```
even_numbers = [x for x in range(10) if x % 2 == 0]
# Output: even_numbers = [0, 2, 4, 6, 8]
```

2. Copy List & Aliasing

- **Aliasing**: Gán một List cho biến khác chỉ tạo tham chiếu, không tạo bản sao.

```
list1 = [1, 2, 3]
list2 = list1  % list2 tham chiếu đến list1
list2[0] = 10  % list1 cũng thay đổi: [10, 2, 3]
```

- `copy()`: Tạo một bản sao nông (shallow copy) của danh sách gốc, trong đó danh sách cha được sao chép, nhưng các phần tử bên trong vẫn tham chiếu đến cùng một đối tượng

```
original_list = [[1, 2, 3], [4, 5, 6]]
shallow_copy = original_list.copy()

# Thay đổi một phần tử trong danh sách con
shallow_copy[0][0] = 100

print("Original List:", original_list)
print("Shallow Copy:", shallow_copy)

# Output
# Original List: [[100, 2, 3], [4, 5, 6]]
# Shallow Copy: [[100, 2, 3], [4, 5, 6]]
```

- `deepcopy()` (từ module `copy`): Tạo một bản sao sâu (deep copy) của danh sách gốc, trong đó toàn bộ danh sách và các phần tử bên trong đều được sao chép thành các đối tượng độc lập.

```
import copy
original_list = [[1, 2, 3], [4, 5, 6]]
deep_copy = copy.deepcopy(original_list)

deep_copy[0][0] = 100

print("Original List:", original_list)
print("Deep Copy:", deep_copy)

# Output
Original List: [[1, 2, 3], [4, 5, 6]]  # Không bị thay đổi
Deep Copy: [[100, 2, 3], [4, 5, 6]]    # Chỉ bản sao bị thay đổi
```

3. Dictionary

- **Khái niệm**: Dictionary là container mutable, lưu trữ dữ liệu dạng cặp `key-value`.
- **Khai báo**:

```
my_dict = {"name": "An", "age": 20}  % key: "name", value: "An"
empty_dict = {}  % Dictionary rỗng
```

- **Truy xuất và thêm/sửa**:

```
print(my_dict["name"])  % Output: An
my_dict["age"] = 21  % Sửa giá trị
my_dict["grade"] = "A"  % Thêm cặp key-value mới
```

- **Phương thức cơ bản**:
  - `.keys()`: Trả về danh sách các key.
  - `.values()`: Trả về danh sách các value.
  - `.items()`: Trả về danh sách các cặp (key, value).
  - `.get(key, default)`: Trả về giá trị của key, nếu không có thì trả về default.

```
print(my_dict.keys()) # Output: dict_keys(["name", "age", "grade"])
print(my_dict.get("name"))  # Output: An
```

Phần 2: Bài tập

Bài 1: List Comprehension

Yêu cầu:

1. Nhập một số `n` từ bàn phím.
2. Dùng List comprehension để tạo danh sách các số lẻ từ 1 đến `n`.
3. In danh sách kết quả.

Ví dụ:

```
Nhập n: 10
Danh sách số lẻ: [1, 3, 5, 7, 9]
```

Bài 2: Copy List & Aliasing

Yêu cầu:

1. Tạo List ban đầu: `original = [1, 2, [3, 4]]`.
2. Tạo 2 bản sao: một bằng `copy()` và một bằng `deepcopy()`.
3. Thay đổi phần tử lồng nhau trong cả hai bản sao (VD: đổi `[3, 4]` thành `[10, 4]`) và in cả 3 List để so sánh.

Ví dụ:

```
Original ban đầu: [1, 2, [3, 4]]
Shallow copy sau khi sửa: [1, 2, [10, 4]]
Deep copy sau khi sửa: [1, 2, [10, 4]]
Original sau khi sửa: [1, 2, [10, 4]]
```

Bài 4: Dictionary

Yêu cầu:

1. Tạo Dictionary: `students = "An": 8.5, "Bình": 7.0`.
2. Nhập tên và điểm của một học sinh mới từ bàn phím, thêm vào Dictionary.
3. In tất cả tên học sinh (dùng `.keys()`) và điểm trung bình (dùng `.values()`).

Ví dụ:

```
Dictionary ban đầu: 'An': 8.5, 'Bình': 7.0
Nhập tên học sinh: Cường
Nhập điểm: 9.0
Dictionary sau khi thêm: 'An': 8.5, 'Bình': 7.0, 'Cường': 9.0
Danh sách học sinh: dict_keys(['An', 'Bình', 'Cường'])
Điểm trung bình: 8.17
```

Day 9 - List 2D & Set, Tuple

Đăng-Nhã Nguyễn

Phần 1: Lý thuyết

1. List và Ma trận 2D

- **Khái niệm**: Ma trận 2D là List lồng nhau (nested list), biểu diễn dữ liệu dạng bảng (hàng và cột).
- **Khai báo**:

```
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]  # Ma trận 3x3
```

- **Truy xuất phần tử**: Dùng chỉ số kép `[hàng][cột]`.

```
print(matrix[0][1])
# Output: 2  (hàng 0, cột 1)
```

- **Duyệt ma trận**: Dùng vòng lặp lồng nhau.

```
for row in matrix:
    for element in row:
        print(element, end=" ")
# Output: 1 2 3 4 5 6 7 8 9
```

2. List Slicing

- **Khái niệm**: Slicing là cách trích xuất một phần của List bằng cú pháp `[start:stop:step]`.
- **Cú pháp**:
  - `start`: Chỉ số bắt đầu (bao gồm).
  - `stop`: Chỉ số kết thúc (không bao gồm).
  - `step`: Bước nhảy (mặc định là 1).

```
my_list = [0, 1, 2, 3, 4, 5]
print(my_list[1:4])     # Output: [1, 2, 3]
print(my_list[::2])     # Output: [0, 2, 4] (lấy cách 2 phần tử)
print(my_list[::-1])    # Output: [5, 4, 3, 2, 1, 0] (đảo ngược)
```

3. Set và Tuple

- **Set**:
  - Là tập hợp chỉ chứa các phần tử duy nhất, không thể thực hiện truy xuất các phần tử thông qua index.
  - Khai báo:

```
my_set = {1, 2, 3}  # hoặc my_set = set([1, 2, 3])
```

  - Thao tác: `.add()`, `.remove()`, `.union()`, `.intersection()`.

```
my_set.add(4)  # my_set = {1, 2, 3, 4}
```

- **Tuple**:
  - Là tập hợp chưa các phần tử không thể bị thay đổi.
  - Khai báo:

```
my_tuple = (1, 2, 3)  # hoặc my_tuple = tuple([1, 2, 3])
```

  - Truy xuất: Dùng chỉ số như List, nhưng không thể sửa đổi.

```
print(my_tuple[0])  # Output: 1
```

Phần 2: Bài tập

Bài 1: Áp dụng List 2D

Yêu cầu:

1. Tạo ma trận 2D kích thước 3x3 chứa các số nguyên nhập từ bàn phím.
2. Tính tổng các phần tử trong ma trận.
3. In ma trận theo định dạng bảng (mỗi hàng trên một dòng).

Ví dụ:

```
Nhập phần tử [0][0]: 1
Nhập phần tử [0][1]: 2
Nhập phần tử [0][2]: 3
Nhập phần tử [1][0]: 4
Nhập phần tử [1][1]: 5
Nhập phần tử [1][2]: 6
Nhập phần tử [2][0]: 7
Nhập phần tử [2][1]: 8
Nhập phần tử [2][2]: 9
Ma trận:
1 2 3
4 5 6
7 8 9
Tổng các phần tử: 45
```

Bài 2: List Slicing

Yêu cầu:

1. Tạo List: `numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`.
2. Dùng slicing để:
   - Trích xuất các số từ chỉ số 2 đến 6.
   - Lấy các số chẵn (bắt đầu từ 0, bước nhảy 2).
   - Đảo ngược List.
3. In kết quả từng bước.

Ví dụ:

```
List ban đầu: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Từ chỉ số 2 đến 6: [2, 3, 4, 5, 6]
Các số chẵn: [0, 2, 4, 6, 8]
List đảo ngược: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Bài 3: Set và Tuple

Yêu cầu:

1. Tạo một List: `[1, 2, 2, 3, 4, 4, 5]`.
2. Chuyển List thành Set để loại bỏ trùng lặp, rồi thêm số 6 vào Set.
3. Chuyển Set thành Tuple và in ra.
4. Tìm giá trị lớn nhất trong Tuple bằng `max()`.

Ví dụ:

List ban đầu: [1, 2, 2, 3, 4, 4, 5]
Set sau khi loại trùng và thêm 6: 1, 2, 3, 4, 5, 6
Tuple: (1, 2, 3, 4, 5, 6)
Giá trị lớn nhất: 6

Day 10 - List, Dictionary, Set, & Tuple Exercises

Đăng-Nhã Nguyễn

Bài tập

Bài 1: Quản lý danh sách môn học

Mô tả đề: Cho một danh sách các tuple *subjects*, mỗi tuple chứa thông tin về một môn học gồm `(tên môn, số tín chỉ, điểm trung bình)`. Hãy tạo một từ điển (`dict`) để lưu số lượng môn học theo số tín chỉ, một tập hợp (`set`) chứa các điểm trung bình duy nhất, và trả về môn học có điểm trung bình cao nhất dưới dạng tuple.

Ví dụ Input/Output:

- **Ví dụ 1:**
  Input: $subjects = [("Ton", 3, 8.5), ("L", 2, 7.0), ("Ha", 3, 9.0)]$
  Output:
  Từ điển số tín chỉ: $\{2 : 1, 3 : 2\}$
  Tập hợp điểm TB: $\{7.0, 8.5, 9.0\}$
  Môn học cao nhất: $("Ha", 3, 9.0)$

- **Ví dụ 2:** Input: $subjects = [("Vn", 2, 6.5)]$
  Output:
  Từ điển số tín chỉ: $\{2 : 1\}$
  Tập hợp điểm TB: $\{6.5\}$
  Môn học cao nhất: $("Vn", 2, 6.5)$

- **Ví dụ 3:** Input: $subjects = [("Sinh", 3, 8.0), ("S", 3, 8.0), ("\text{Œ}a", 4, 7.5)]$
  Output:
  Từ điển số tín chỉ: $\{3 : 2, 4 : 1\}$
  Tập hợp điểm TB: $\{7.5, 8.0\}$
  Môn học cao nhất: $("Sinh", 3, 8.0)$

Gợi ý làm bài: Dùng vòng lặp để duyệt qua danh sách *subjects*. Với từ điển, dùng số tín chỉ làm khóa và đếm số môn học. Với tập hợp, thêm điểm trung bình vào `set`. Dùng hàm `max` với khóa so sánh là điểm trung bình để tìm tuple có điểm cao nhất.

Bài 2: Phân loại số theo tần suất

Mô tả đề: Cho một danh sách số nguyên *numbers*, hãy tạo một từ điển (`dict`) lưu tần suất xuất hiện của mỗi số, một tập hợp (`set`) chứa các số xuất hiện đúng một lần, và trả về danh sách các tuple `(số, tần suất)` được sắp xếp theo tần suất giảm dần.

Ví dụ Input/Output:

- **Ví dụ 1:**
  Input: $numbers = [1, 2, 2, 3, 1, 4]$
  Output:
  Từ điển tần suất: $\{1 : 2, 2 : 2, 3 : 1, 4 : 1\}$
  Tập hợp số xuất hiện 1 lần: $\{3, 4\}$
  Danh sách sắp xếp: $[(1, 2), (2, 2), (3, 1), (4, 1)]$

- **Ví dụ 2:**
  Input: $numbers = [5, 5, 5]$
  Output:
  Từ điển tần suất: $\{5 : 3\}$
  Tập hợp số xuất hiện 1 lần: $\emptyset$
  Danh sách sắp xếp: $[(5, 3)]$

- **Ví dụ 3:**
  Input: $numbers = [1, 2, 3]$
  Output:
  Từ điển tần suất: $\{1 : 1, 2 : 1, 3 : 1\}$
  Tập hợp số xuất hiện 1 lần: $\{1, 2, 3\}$
  Danh sách sắp xếp: $[(1, 1), (2, 1), (3, 1)]$

Gợi ý làm bài: Dùng `dict` để đếm tần suất bằng cách duyệt qua *numbers*. Tạo `set` bằng cách kiểm tra các khóa trong từ điển có giá trị là 1. Chuyển từ điển thành danh sách các tuple và sắp xếp bằng hàm `sorted` với khóa là tần suất.

Bài 3: Tìm cặp giá trị gần nhất

Mô tả đề: Cho một danh sách các tuple *pairs*, mỗi tuple chứa `(giá trị, chỉ số)`, hãy tìm cặp hai tuple có giá trị gần nhau nhất (chênh lệch nhỏ nhất). Trả về một tuple chứa chỉ số của hai phần tử này, một từ điển lưu chênh lệch giữa các cặp, và một tập hợp các giá trị duy nhất.

Ví dụ Input/Output:

- **Ví dụ 1:**
  Input: $pairs = [(10, 0), (15, 1), (13, 2), (12, 3)]$
  Output:
  Cặp chỉ số gần nhất: $(2, 3)$
  Từ điển chênh lệch: $\{(0, 1) : 5, (0, 2) : 3, (0, 3) : 2, (1, 2) : 2, (1, 3) : 3, (2, 3) : 1\}$
  Tập hợp giá trị: $\{10, 12, 13, 15\}$
  Giải thích: Chênh lệch nhỏ nhất là $|13 - 12| = 1$ giữa chỉ số 2 và 3.

- **Ví dụ 2:**
  Input: $pairs = [(5, 0), (7, 1)]$
  Output:
  Cặp chỉ số gần nhất: $(0, 1)$
  Từ điển chênh lệch: $\{(0, 1) : 2\}$
  Tập hợp giá trị: $\{5, 7\}$

- **Ví dụ 3:**
  Input: $pairs = [(1, 0), (4, 1), (2, 2)]$
  Output:
  Cặp chỉ số gần nhất: $(0, 2)$
  Từ điển chênh lệch: $\{(0, 1) : 3, (0, 2) : 1, (1, 2) : 2\}$
  Tập hợp giá trị: $\{1, 2, 4\}$

Gợi ý làm bài: Sắp xếp danh sách $pairs$ theo giá trị để dễ tìm chênh lệch nhỏ nhất giữa các phần tử kề nhau. Dùng `dict` để lưu chênh lệch giữa mọi cặp chỉ số. Tạo `set` từ các giá trị trong $pairs$. So sánh chênh lệch để tìm cặp gần nhất.

Day 11 - List, Dictionary & Set Advanced Exercises

Đăng-Nhã Nguyễn

Bài tập

Bài 1: Kiểm tra phần tử trùng lặp

Mô tả đề: Cho một mảng số nguyên *nums*, hãy xác định xem mảng có chứa bất kỳ phần tử nào xuất hiện nhiều hơn một lần hay không. Trả về `True` nếu tồn tại ít nhất một phần tử trùng lặp, và `False` nếu tất cả các phần tử đều khác nhau.

Ví dụ Input/Output:

- **Ví dụ 1:**
  Input: $nums = [1, 2, 3, 1]$
  Output: `True`
  Giải thích: Số 1 xuất hiện hai lần trong mảng.
- **Ví dụ 2:**
  Input: $nums = [1, 2, 3, 4]$
  Output: `False`
  Giải thích: Tất cả các số trong mảng đều khác nhau.
- **Ví dụ 3:**
  Input: $nums = [1, 1, 1, 3, 3, 4, 3, 2, 4, 2]$
  Output: `True`
  Giải thích: Các số $1, 3, 4, 2$ xuất hiện nhiều hơn một lần.

Gợi ý làm bài: Sử dụng tập hợp (`set`) để loại bỏ các phần tử trùng lặp và so sánh độ dài của tập hợp với mảng ban đầu. Nếu độ dài khác nhau, mảng có phần tử trùng lặp. Ngoài ra, có thể dùng từ điển (`dict`) để đếm số lần xuất hiện của mỗi phần tử.

```python
# Cách 1: List + dict
def contain_duplicate(nums):
    # your code here
    return _

# Cách 2: List + set
def contain_duplicate(nums):
    # Your code here
    return _
```

Bài 2: Nhóm các chuỗi đảo chữ

Mô tả đề: Cho một mảng các chuỗi *strs*, hãy gom các chuỗi là đảo chữ của nhau (các chuỗi có cùng tập hợp ký tự với số lần xuất hiện như nhau) thành từng nhóm. Trả về danh sách các nhóm, trong đó mỗi nhóm là một danh sách các chuỗi đảo chữ. Thứ tự của các nhóm và chuỗi trong nhóm không quan trọng.

Ví dụ Input/Output:

- **Ví dụ 1:**
  Input: $strs = ["eat", "tea", "tan", "ate", "nat", "bat"]$
  Output: $[["bat"], ["nat", "tan"], ["ate", "eat", "tea"]]$
  Giải thích: "bat" là một nhóm riêng; "nat" và "tan" có cùng ký tự (n, a, t); "ate", "eat", "tea" có cùng ký tự (e, a, t).

- **Ví dụ 2:**
  Input: $strs = [""]$
  Output: $[[""]]$
  Giải thích: Chuỗi rỗng tự tạo thành một nhóm.

- **Ví dụ 3:**
  Input: $strs = ["a"]$
  Output: $[["a"]]$
  Giải thích: Chuỗi "a" tự tạo thành một nhóm.

Gợi ý làm bài: Sắp xếp từng chuỗi theo thứ tự bảng chữ cái để tạo "dấu ấn" duy nhất cho các đảo chữ, sau đó dùng từ điển (`dict`) để nhóm các chuỗi có cùng dấu ấn. Mỗi khóa trong từ điển là một chuỗi đã sắp xếp, và giá trị là danh sách các chuỗi tương ứng.

```python
def group_anagrams(strs):
    # Your code here
    return _
```

Bài 3: Tìm cặp số có tổng bằng đích

Mô tả đề: Cho một mảng số nguyên $nums$ và một số nguyên $target$, hãy tìm và trả về chỉ số của hai phần tử trong mảng sao cho tổng của chúng bằng $target$. Giả định rằng mỗi trường hợp chỉ có duy nhất một cặp giải pháp, và không được sử dụng cùng một phần tử hai lần. Thứ tự các chỉ số trong kết quả không quan trọng.

Ví dụ Input/Output:

- **Ví dụ 1:**
  Input: $nums = [2, 7, 11, 15]$, $target = 9$
  Output: $[0, 1]$
  Giải thích: $nums[0] + nums[1] = 2 + 7 = 9$.

- **Ví dụ 2:**
  Input: $nums = [3, 2, 4]$, $target = 6$
  Output: $[1, 2]$
  Giải thích: $nums[1] + nums[2] = 2 + 4 = 6$.

- **Ví dụ 3:**
  Input: $nums = [3, 3]$, $target = 6$
  Output: $[0, 1]$
  Giải thích: $nums[0] + nums[1] = 3 + 3 = 6$.

Gợi ý làm bài: Sử dụng từ điển (`dict`) để lưu giá trị đã duyệt qua cùng chỉ số của chúng. Với mỗi phần tử $nums[i]$, kiểm tra xem $target - nums[i]$ có trong từ điển hay không. Nếu có, trả về chỉ số của nó cùng i. Nếu không, thêm $nums[i]$ và chỉ số i vào từ điển.

```python
def two_sum(nums, target):
    # Your code here
    return _
```

Day 12 - Step into Streamlit

Đăng-Nhã Nguyễn

1 Lý Thuyết

Streamlit là một framework mã nguồn mở cho phép tạo ứng dụng web từ mã Python một cách dễ dàng. Nó thường được sử dụng trong khoa học dữ liệu và machine learning để xây dựng giao diện trực quan cho các mô hình.

1.1 Cài đặt Streamlit

Để cài đặt Streamlit, sử dụng lệnh:

```
pip install streamlit
```

1.2 Chạy ứng dụng Streamlit

Một tệp Streamlit thường được chạy bằng lệnh:

```
streamlit run your_script.py
```

1.3 Các thành phần cơ bản trong Streamlit

- **st.title()** - Tiêu đề ứng dụng
- **st.header()** - Tiêu đề cấp nhỏ hơn
- **st.text()** - Hiển thị văn bản đơn giản

- **st.write()** - Hiển thị nhiều kiểu dữ liệu khác nhau
- **st.button()** - Tạo nút bấm tương tác
- **st.sidebar()** - Tạo sidebar cho ứng dụng

1.4 Chương trình mẫu: Website tính toán đơn giản

Dưới đây là một ví dụ về cách sử dụng Streamlit để tạo một website tính toán cơ bản:

```
import streamlit as st

# Thêm tiêu đề cho ứng dụng
st.title("Website Tính Toán Cơ Bản")

# Thêm mô tả
st.write("Đây là một ứng dụng đơn giản để tính tổng và tích của hai số.")

# Nhập dữ liệu từ người dùng
a = st.text_input("Nhập số thứ nhất:", "0")  # Ô nhập số a
b = st.text_input("Nhập số thứ hai:", "0")   # Ô nhập số b

# Chuyển đổi dữ liệu thành số nguyên
a = int(a) if a.isdigit() else 0  # Nếu không phải số, mặc định là 0
b = int(b) if b.isdigit() else 0  # Nếu không phải số, mặc định là 0

# Nút tính toán
if st.button("Tính toán"):
    sum_result = a + b  # Tính tổng
    product_result = a * b  # Tính tích
    st.write(f"Tổng của {a} và {b} là: {sum_result}")
    st.write(f"Tích của {a} và {b} là: {product_result}")
```

2 Bài tập

2.1 Bài 1: Chương trình Streamlit đơn giản

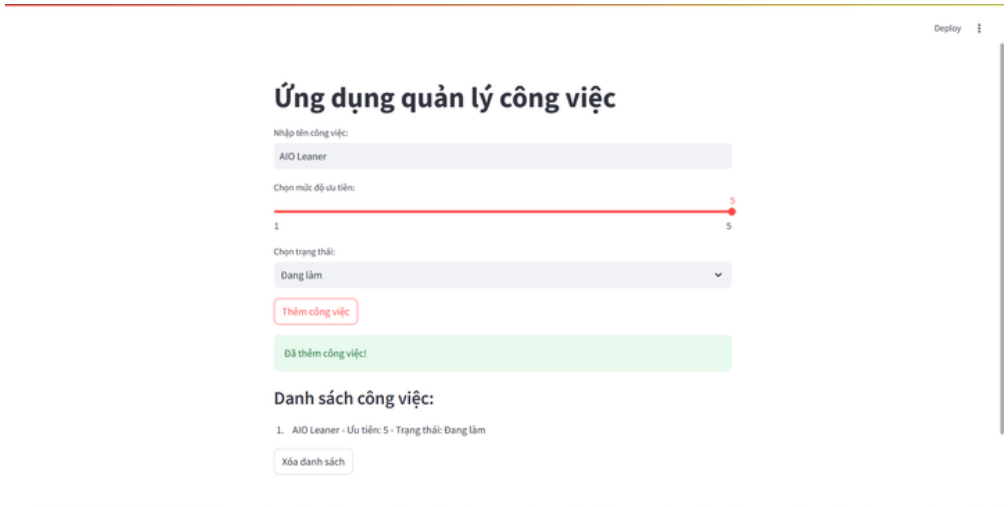
1. Cài đặt Streamlit trên máy tính.
2. Tạo file `app.py` với nội dung:
   - Hiển thị tiêu đề "Ứng dụng đầu tiên của tôi".
   - Hiển thị dòng chữ "Thành công chạy ứng dụng Streamlit đầu tiên" bên dưới.
3. Chạy chương trình và kiểm tra kết quả trên trình duyệt.

2.2 Bài 2: Taọ chương trình Streamlit phức tạp hơn

Tạo một ứng dụng Streamlit với các yêu cầu sau:

1. Thêm tiêu đề "Ứng dụng quản lý công việc".
2. Tạo ô nhập để người dùng nhập tên công việc.
3. Tạo thanh trượt để chọn mức độ ưu tiên (từ 1 đến 5).
4. Tạo hộp chọn để chọn trạng thái công việc (Chưa làm, Đang làm, Hoàn thành).
5. Tạo nút "Thêm công việc".
6. Khi nhấn nút, hiển thị thông tin công việc bao gồm: tên, mức ưu tiên, trạng thái.
7. Thêm nút "Xóa danh sách" để xóa toàn bộ công việc đã thêm (nếu có).

Kết quả:



Day 13 - Practice NumPy

Đăng-Nhã Nguyễn

Lý Thuyết

1. Giới Thiệu về NumPy

NumPy (Numerical Python) là thư viện mạnh mẽ trong Python hỗ trợ tính toán khoa học. NumPy cung cấp:

- Thư viện tính toán số học hiệu năng cao cho Python
- Cung cấp đối tượng mảng đa chiều (ndarray)
- Tối ưu cho tính toán vector và ma trận

2. Các khái niệm cơ bản của NumPy

```
import numpy as np
```

- **ndarray**: Mảng N chiều, kiểu dữ liệu đồng nhất

```
import numpy as np
arr = np.array([1, 2, 3])  # Tạo mảng 1 chiều
matrix = np.array([[1, 2], [3, 4]])  # Mảng 2 chiều
```

- **Shape**: Kích thước các chiều của mảng (ví dụ: (3,4))

```
print(arr.shape)      # (3,)
print(matrix.shape)   # (2, 2)
```

- **Dtype**: Kiểu dữ liệu của phần tử (float32, int64,...)

```
print(arr.dtype)      # int64 (mặc định)
float_arr = np.array([1.0, 2.0], dtype=np.float32)
print(float_arr.dtype) # float32
```

3. Các hàm quan trọng

- **Tạo mảng**: `np.array()`, `np.zeros()`, `np.ones()`, `np.arange()`, `np.linspace()`

```
zeros = np.zeros((2, 3))    # Mảng 2x3 toàn 0
ones = np.ones((3,))        # Mảng 3 phần tử toàn 1
range_arr = np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
lin_arr = np.linspace(0, 1, 5)   # [0., 0.25, 0.5, 0.75, 1.]
```

- **Truy cập và thay đổi phần tử**: Slicing, Indexing

```python
arr = np.array([1, 2, 3, 4, 5])
print(arr[1])         # 2 (truy cập phần tử)
print(arr[1:4])       # [2, 3, 4] (slicing)
arr[2] = 10           # Thay đổi phần tử

matrix = np.array([[1, 2], [3, 4]])
print(matrix[0, 1])  # 2 (truy cập phần tử 2D)
```

- **Các phép toán trên mảng**: Cộng, trừ, nhân, chia, ma trận

```python
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)    # [5, 7, 9] (cộng từng phần tử)
print(a * b)    # [4, 10, 18] (nhân từng phần tử)
print(a @ b)    # 32 (tích vô hướng)

# Phép nhân ma trận
mat1 = np.array([[1, 2], [3, 4]])
mat2 = np.array([[5, 6], [7, 8]])
print(np.dot(mat1, mat2))  # [[19 22], [43 50]]
```

- **Hàm thống kê**: `mean()`, `sum()`, `min()`, `max()`, `std()`

```python
data = np.array([1, 2, 3, 4, 5])
print(np.mean(data))    # 3.0 (trung bình)
print(np.sum(data))     # 15 (tổng)
print(np.max(data))     # 5 (giá trị lớn nhất)
print(np.std(data))     # 1.414... (độ lệch chuẩn)
```

- **Thao tác với hình dạng mảng**: `reshape()`, `flatten()`, `transpose()`

```python
arr = np.arange(6)      # [0, 1, 2, 3, 4, 5]
matrix = arr.reshape(2, 3)  # [[0, 1, 2], [3, 4, 5]]
flat = matrix.flatten() # [0, 1, 2, 3, 4, 5]
transposed = matrix.T   # [[0, 3], [1, 4], [2, 5]]
```

Bài Tập

Bài 1: Làm quen với cú pháp

Thực hiện gõ lại các ví dụ code ở phần lý thuyết, thực hiện các thử nghiệm, thay đổi input, argument của các hàm, in ra màn hình.

Bài 2: Sử dụng np.where()

Đề bài: Cho mảng:

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

Thực hiện:

1. Thay các số âm bằng 0
2. Thay số chẵn bằng "even", số lẻ bằng "odd"
3. Tìm vị trí các phần tử > 2

Ví dụ:

```
# Ví dụ thay số > 3 bằng 100
new_arr = np.where(arr > 3, 100, arr)
```

Bài 3: Kết hợp nhiều điều kiện

Đề bài: Cho mảng nhiệt độ:

```
temps = np.array([25, 28, 30, 22, 35, 18, 20])
```

Dùng `np.select()` phân loại:

- "Cold" nếu < 20
- "Mild" nếu 20-29
- "Hot" nếu $>= 30$

Gợi ý:
Hàm `np.select()` cho phép áp dụng nhiều điều kiện phức tạp trên mảng NumPy. Cú pháp:

```
np.select(condlist, choicelist, default=0)
```

Trong đó:

- `condlist`: Danh sách các điều kiện (dạng boolean array)
- `choicelist`: Giá trị tương ứng với từng điều kiện
- `default`: Giá trị mặc định khi không thỏa điều kiện nào

Ví dụ:

```python
# Phân loại điểm số học sinh
scores = np.array([55, 72, 88, 42, 90, 60])

conditions = [
    scores < 50,
    (scores >= 50) & (scores < 70),
    scores >= 70
]
grades = ["Trượt", "Đạt", "Xuất sắc"]
```

Bài 4: Áp dụng hàm theo trục

Đề bài: Cho ma trận:

```python
mat = np.random.randint(1,100, (5,5))
```

Tính:

1. Chuẩn hóa từng hàng (trừ đi min, chia cho max - min)
2. Áp dụng hàm softmax theo cột: $softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

Gợi ý:

- `np.apply_along_axis()` giúp áp dụng hàm theo từng trục (axis) của mảng
- Axis=0: Theo cột, Axis=1: Theo hàng
- Rất hữu ích cho các phép chuẩn hóa và biến đổi dữ liệu

Ví dụ:

```python
# Áp dụng cho ma trận điểm ảnh
pixels = np.random.randint(0, 256, (4, 4))

def adjust_contrast(channel):
    return np.clip(channel * 1.5, 0, 255)

enhanced = np.apply_along_axis(adjust_contrast, 1, pixels)
```

Bài 5: Xử lý dữ liệu phức tạp

Đề bài: Cho mảng 2D:

```python
data = np.array([
    [1, 2, np.nan],
    [4, np.nan, 6],
    [7, 8, 9]
])
```

Xử lý:

1. Thay các giá trị NaN bằng trung bình cột

2. Tính z-score cho mỗi cột
3. Chuẩn hóa min-max theo cột

Gợi ý:

- `np.nanmean`: Tính trung bình bỏ qua NaN
- `np.isnan`: Kiểm tra giá trị NaN
- Z-score: $(x - \mu)/\sigma$
- Min-max scaling: $(x - min)/(max - min)$

Ví dụ

```
# Xử lý dữ liệu thời tiết
weather_data = np.array([
    [25.1, np.nan, 1010],
    [np.nan, 80, 1008],
    [22.5, 75, np.nan]
])

# Điền giá trị thiếu
filled_temp = np.where(np.isnan(weather_data[:,0]),
                    np.nanmean(weather_data[:,0]),
                    weather_data[:,0])
```

Day 14 - Practice OpenCV

Đăng-Nhã Nguyễn

Lý Thuyết

Để làm việc với OpenCV trong Python, người học cần nắm các khái niệm cơ bản sau (Lưu ý: Có một số thao tác thư viện cv2 không hỗ trợ trên google colab):

- **Cài đặt OpenCV**: Sử dụng lệnh `pip install opencv-python` để cài đặt thư viện.
- **Đọc và hiển thị ảnh**: Sử dụng các hàm `cv2.imread()`, `cv2.imshow()`, `cv2.waitKey()`, và `cv2.destroyAllWindows()`.
- **Thao tác cơ bản với ảnh**: Chuyển đổi màu sắc (`cv2.cvtColor()`), thay đổi kích thước (`cv2.resize()`), cắt ảnh.
- **Vẽ hình học**: Các hàm như `cv2.line()`, `cv2.rectangle()`, `cv2.circle()`.
- **Xử lý ảnh cơ bản**: Làm mờ (`cv2.GaussianBlur()`), phát hiện cạnh (`cv2.Canny()`).

Bài Tập

Bài 1: Đọc và Hiển Thị Ảnh

Đề bài: Viết chương trình cho phép người dùng tải ảnh lên Colab và hiển thị ảnh đó.

Giải thích: Trong Colab, ảnh cần được tải lên từ máy tính bằng `files.upload()`. Hàm `cv2.imread()` đọc ảnh, và `cv2_imshow()` thay thế `cv2.imshow()` để hiển thị ảnh trong notebook.

Ví dụ: Tải file ảnh bất kỳ (aivn.png) và hiển thị.

Code tham khảo:

```
from google.colab import files
from google.colab.patches import cv2_imshow
import cv2

# Tai anh len tu may tinh
uploaded = files.upload()
img_name = list(uploaded.keys())[0]  # Lay ten file anh dau tien
img = cv2.imread(img_name)

# Hien thi anh
cv2_imshow(img)
```

Bài 2: Chuyển Đổi Màu Sắc

Đề bài: Viết chương trình tải ảnh lên, chuyển ảnh sang không gian màu xám và hiển thị cả ảnh gốc lẫn ảnh xám.

Giải thích: Sử dụng `cv2.cvtColor()` để chuyển từ BGR sang GRAY. Trong Colab, hiển thị cả hai ảnh bằng `cv2_imshow()`.

Code tham khảo:

```
# Tai anh len
uploaded = files.upload()
img_name = list(uploaded.keys())[0]
img = cv2.imread(img_name)

# Chuyen sang anh xam
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Hien thi ca hai anh
print("Anh Goc:")
cv2_imshow(img)
print("Anh Xam:")
cv2_imshow(gray_img)
```

Bài 3: Vẽ Hình Học Trên Ảnh

Đề bài: Viết chương trình tải ảnh lên, vẽ một hình chữ nhật đỏ và một đường thẳng xanh lên ảnh, sau đó hiển thị kết quả.

Giải thích: Sử dụng `cv2.rectangle()` và `cv2.line()` với tọa độ, màu sắc (BGR) và độ dày đường vẽ cụ thể.

Code tham khảo:

```
# Tai anh len
uploaded = files.upload()
img_name = list(uploaded.keys())[0]
img = cv2.imread(img_name)

# Ve hinh chu nhat do (diem tren trai, diem duoi phai, mau do, do day 2)
cv2.rectangle(img, (50, 50), (200, 200), (0, 0, 255), 2)

# Ve duong thang xanh (diem dau, diem cuoi, mau xanh la, do day 2)
cv2.line(img, (50, 50), (200, 200), (0, 255, 0), 2)

# Hien thi ket qua
cv2_imshow(img)
```

Bài 4: Làm Mờ và Phát Hiện Cạnh

Đề bài: Viết chương trình tải ảnh lên, làm mờ ảnh bằng Gaussian Blur, phát hiện cạnh bằng Canny, và hiển thị cả ba ảnh (gốc, mờ, cạnh).

Giải thích: `cv2.GaussianBlur()` làm mờ ảnh với kernel cụ thể, `cv2.Canny()` phát hiện cạnh dựa trên ngưỡng gradient. Hiển thị bằng `cv2_imshow()`.

Code tham khảo:

```
# Tai anh len
uploaded = files.upload()
img_name = list(uploaded.keys())[0]
img = cv2.imread(img_name)

# Lam mo anh voi Gaussian Blur
blurred = cv2.GaussianBlur(img, (5, 5), 0)

# Phat hien canh voi Canny
edges = cv2.Canny(blurred, 100, 200)

# Hien thi ket qua
print("Anh Goc:")
cv2_imshow(img)
print("Anh Mo:")
cv2_imshow(blurred)
print("Canh:")
cv2_imshow(edges)
```

Day 15 - OpenCV Advanced

Đăng-Nhã Nguyễn

Bài tập

Đối với bài tập ngày hôm nay, chúng ta sẽ thực hiện code theo hướng dẫn để luyện tập thêm về OpenCV

Bài 1: Xử lý Image Enhancement (Cải thiện chất lượng ảnh)

Yêu cầu: Áp dụng các kỹ thuật xử lý ảnh để cải thiện độ sáng, độ tương phản và làm nét ảnh.

- Đọc ảnh đầu vào (ví dụ: input.jpg - ảnh tối hoặc mờ).
- Áp dụng các phương pháp enhancement:
  - Điều chỉnh gamma correction để thay đổi độ sáng.
  - Áp dụng histogram equalization để cân bằng độ tương phản.
  - Làm nét ảnh bằng bộ lọc unsharp masking.
- Hiển thị ảnh gốc và ảnh đã xử lý.

Triển khai: Import các thư viện cần thiết

```
import cv2
import numpy as np
from google.colab import files
from google.colab.patches import cv2_imshow
```

Đọc ảnh từ link, path hoặc từ file upload

```
uploaded = files.upload()
img_name = list(uploaded.keys())[0]
img = cv2.imread(img_name, cv2.IMREAD_COLOR)
# img = cv2.imread('input.jpg', cv2.IMREAD_COLOR)
cv2_imshow(img)
```

Thay đổi độ sáng bằng Gamma Correction

```
# 2. Gamma Correction (Điều chỉnh độ sáng của ảnh)
# Gamma Correction giúp điều chỉnh độ sáng tổng thể của ảnh bằng cách áp d
    ụng phép biến đổi phi tuyến tính.
# Công thức: output = (input / 255)^gamma * 255
# Nếu gamma > 1: Ảnh tối hơn (giảm độ sáng)
# Nếu gamma < 1: Ảnh sáng hơn (tăng độ sáng)

gamma = 1.5  # Giá trị gamma được đặt là 1.5 (giảm độ sáng)

```

```
# Chuẩn hóa ảnh về khoảng [0,1] trước khi tính gamma correction
gamma_corrected = np.power(img / 255.0, gamma) * 255.0

# Chuyển đổi dữ liệu về kiểu uint8 để hiển thị bằng OpenCV
gamma_corrected = np.uint8(gamma_corrected)

# Hiển thị ảnh sau khi áp dụng Gamma Correction
cv2_imshow(gamma_corrected)
```

Xủ lý tương phản

```
# 3. Histogram Equalization (Cân bằng độ tương phản)
# Phương pháp này giúp cải thiện độ tương phản của ảnh bằng cách dàn trải histogram của kênh độ sáng (Y channel).
# Điều này giúp làm rõ các chi tiết trong vùng sáng và tối mà không làm thay đổi màu sắc tổng thể.

# Chuyển đổi ảnh từ không gian màu BGR sang YUV
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

# Cân bằng histogram chỉ trên kênh độ sáng (Y channel)
# Kênh Y chứa thông tin độ sáng, còn U và V chứa thông tin màu sắc
img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])

# Chuyển đổi ảnh từ không gian màu YUV trở lại BGR để hiển thị
equalized = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)

# Hiển thị ảnh sau khi áp dụng cân bằng histogram
cv2_imshow(equalized)
```

Làm nét ảnh bằng phương pháp Gaussian

```
# 4. Unsharp Masking (Làm nét ảnh)
# Kỹ thuật này giúp tăng độ sắc nét bằng cách khuếch đại các chi tiết cạnh trong ảnh.
# Phương pháp hoạt động bằng cách lấy ảnh gốc trừ đi ảnh bị làm mờ Gaussian, sau đó cộng lại vào ảnh gốc.

# Làm mờ ảnh bằng bộ lọc Gaussian
# Bộ lọc Gaussian giúp làm mờ các chi tiết nhỏ, giữ lại các vùng tổng thể của ảnh
gaussian = cv2.GaussianBlur(img, (0, 0), 2.0)

# Tạo ảnh làm nét bằng cách kết hợp ảnh gốc với ảnh bị làm mờ
# Công thức: unsharp = alpha * img + beta * blurred + gamma
# Ở đây:
# - alpha = 2.0 (tăng cường ảnh gốc)
# - beta = -1.0 (trừ đi ảnh bị làm mờ)
# - gamma = 0 (không thay đổi mức độ sáng tổng thể)
unsharp = cv2.addWeighted(img, 2.0, gaussian, -1.0, 0)

# Hiển thị ảnh sau khi làm nét
cv2_imshow(unsharp)
```

Bài 2: Truy cập Camera và Thao tác (Thực hiện ở máy local)

Yêu cầu: Đọc video từ webcam, áp dụng hiệu ứng và hiển thị kết quả.

- Mở camera mặc định (hoặc video file).
- Áp dụng hiệu ứng:
  - Chuyển ảnh sang grayscale.
  - Phát hiện cạnh bằng Canny edge detection.
  - Vẽ hình chữ nhật quanh mặt người dùng (nếu có) bằng Haar Cascade.
- Hiển thị video kết quả và thoát khi nhấn phím q

Triển khai:
Install thư viện `opencv-python` về máy.
Mở VScode hoặc python ide, tạo một file python có đuôi `.py`.
Thực hiện đọc hiểu và chạy thành công code sau đây:

```python
import cv2

# 1. Mở camera
cap = cv2.VideoCapture(0)

# Load Haar Cascade để phát hiện khuôn mặt
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # 2.1 Chuyển sang grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # 2.2 Phát hiện cạnh (Canny)
    edges = cv2.Canny(gray, 100, 200)

    # 2.3 Phát hiện khuôn mặt và vẽ hình chữ nhật
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

    # Hiển thị kết quả
    cv2.imshow('Original', frame)
    cv2.imshow('Edges', edges)

    # Thoát khi nhấn 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

```

```
cap.release()
cv2.destroyAllWindows()
```

Nếu đã chạy thành công, chụp hình thành quả và dán vào file colab.

Day 16 - Pandas: Basics & Data Manipulation

Đăng-Nhã Nguyễn

Lý thuyết

1. Data Frame và CSV File

- *Data Frame*: Trong Python, đặc biệt khi sử dụng thư viện `pandas`, một Data Frame là một cấu trúc dữ liệu dạng bảng hai chiều (hàng và cột), tương tự như bảng tính Excel hoặc bảng trong cơ sở dữ liệu. Nó được sử dụng để lưu trữ và thao tác dữ liệu có cấu trúc.
- *CSV File*: CSV (Comma-Separated Values) là một định dạng tệp văn bản đơn giản, trong đó các giá trị được phân tách bằng dấu phẩy (hoặc ký tự phân tách khác). Đây là cách phổ biến để lưu trữ dữ liệu dạng bảng và dễ dàng được đọc bởi `pandas`.

2. Các hàm và thuộc tính cơ bản trong pandas

- `pd.read_csv()`: Hàm này thuộc thư viện `pandas` (thường được nhập dưới dạng `import pandas as pd`), dùng để đọc tệp CSV và chuyển nó thành một Data Frame. Ví dụ:

```
df = pd.read_csv('ten_tep.csv')
```

- `df.head()`: Trả về 5 hàng đầu tiên của Data Frame (có thể thay đổi số lượng bằng cách truyền tham số, ví dụ: `df.head(10)`).
- `df.info()`: Cung cấp thông tin tổng quan về Data Frame, bao gồm số lượng hàng, cột, kiểu dữ liệu của từng cột và số giá trị không rỗng.
- `df.describe()`: Tạo thống kê mô tả (số lượng, trung bình, độ lệch chuẩn, giá trị min/max, phân vị) cho các cột số trong Data Frame.
- `df.shape`: Thuộc tính trả về một tuple biểu thị kích thước của Data Frame (số hàng, số cột). Ví dụ: `(100, 5)` nghĩa là 100 hàng và 5 cột.
- `df.columns`: Thuộc tính trả về danh sách tên các cột trong Data Frame.
- `df.dtypes`: Thuộc tính trả về kiểu dữ liệu của từng cột (ví dụ: `int64`, `float64`, `object`).

3. Truy cập dữ liệu trong Data Frame

- `.loc[]`: Truy cập dữ liệu bằng nhãn (label-based). Ví dụ:

```
df.loc[0, 'ten_cot'] #truy cập giá trị tại hàng 0, cột 'ten_cot'}
```

- `.iloc[]`: Truy cập dữ liệu bằng chỉ số index (integer-based). Ví dụ:

```
df.iloc[0, 1] truy cập giá trị tại hàng 0, cột 1}
```

- `.at[]`: Truy cập một giá trị đơn bằng nhãn, nhanh hơn `.loc[]` cho truy cập đơn lẻ. Ví dụ:

```
df.at[0, 'ten_cot']
```

- `.iat[]`: Truy cập một giá trị đơn bằng chỉ số index, nhanh hơn `.iloc[]`. Ví dụ:

```
df.iat[0, 1]
```

- `df[df['ten_cot'] > gia_tri]`: Lọc các hàng trong Data Frame thỏa mãn điều kiện, ví dụ:

```
df[df['tuoi'] > 25] lọc các hàng có giá trị cột 'tuoi' lớn hơn 25)}
```

4. Thêm, xóa, sửa cột trong Data Frame

- Thêm/sửa cột: Có thể tạo hoặc sửa cột mới bằng cách gán giá trị. Ví dụ:

```
# tạo cột mới bằng tổng của hai cột
df['cot_moi'] = df['cot1'] + df['cot2']
```

- Xóa cột: Sử dụng `df.drop()`. Ví dụ:

```
# xóa cột 'cot1'
df = df.drop(columns=['cot1'])
```

Lưu ý: Tham số `inplace=True` có thể được sử dụng để sửa trực tiếp Data Frame mà không cần gán lại.

Bài tập

Tải file dữ liệu student.csv, upload lên google colab, đọc file trên thành dataframe và thực hiện chạy các dòng lệnh ở phần 1, quan sát và hiểu các lệnh đó.

Day 17 - Pandas: Data Cleaning & Aggregation

Đăng-Nhã Nguyễn

Lý thuyết

1. Xử lý giá trị thiếu (Handling Missing Values)

- `df.dropna()`: Loại bỏ các hàng hoặc cột chứa giá trị thiếu (NaN). Mặc định, hàm này loại bỏ bất kỳ hàng nào có ít nhất một giá trị NaN. Có thể tùy chỉnh bằng tham số, ví dụ: `df.dropna(axis=1)` để loại bỏ cột hoặc `df.dropna(how='all')` để chỉ loại bỏ hàng toàn NaN.
Ví dụ:

```
# loại bỏ tất cả hàng chứa NaN
df = df.dropna()
```

- `df.fillna(value)`: Điền giá trị thay thế vào các ô chứa NaN. Tham số `value` có thể là một số, chuỗi, hoặc từ điển để điền khác nhau cho từng cột.
Ví dụ:

```
# điền 0 vào tất cả giá trị NaN
df = df.fillna(0)
```

- `df.isnull().sum()`: Kiểm tra và đếm số lượng giá trị NaN trong mỗi cột. `df.isnull()` trả về Data Frame với giá trị `True`/`False`, và `.sum()` tính tổng số `True` theo cột.
Ví dụ:

```
# trả về số lượng NaN trong từng cột
df.isnull().sum()
```

2. Tổng hợp và nhóm dữ liệu (Aggregation & Grouping)

- `df.groupby('column').agg({'col1': 'sum', 'col2': 'mean'})`: Nhóm dữ liệu theo một cột và áp dụng các hàm tổng hợp (aggregation) như tổng (`sum`), trung bình (`mean`), v.v. trên các cột được chỉ định.
Ví dụ:

```
# tổng 'diem' và trung bình 'tuoi' theo 'nhom'
df.groupby('nhom').agg({'diem': 'sum', 'tuoi': 'mean'})
```

Kết quả trả về một Data Frame với các giá trị đã được tổng hợp theo nhóm.

3. Hàm apply và lambda (Apply Functions & Lambda)

- `df['new_col'] = df['col'].apply(lambda x: x * 2 if x > 10 else x + 1)`: Áp dụng một hàm (có thể là hàm tự định nghĩa hoặc lambda) lên từng phần tử của cột. Hàm `apply()` cho phép xử lý linh hoạt dữ liệu.
Ví dụ:

```
df['new_col'] = df['col'].apply(lambda x: x * 2 if  x > 10 else x + 1)
```

Trong ví dụ này:

- Nếu giá trị trong cột `col` lớn hơn 10, nó sẽ được nhân đôi.
- Nếu không, giá trị sẽ được cộng thêm 1.

Kết quả được lưu vào cột mới `new_col`.

Bài tập

Bài 1: Xử lý giá trị thiếu

Sử dụng file `student.csv` đã có ở day 16 để thực hiện các yêu cầu:

- Kiểm tra số lượng giá trị thiếu trong mỗi cột.
- Điền giá trị 0 vào các ô còn lại.
- Xóa tất cả các hàng có ít nhất một giá trị thiếu (nếu có).

Bài 2: Tổng hợp và nhóm dữ liệu

Sử dụng Data Frame `df` từ tệp `student.csv`.

- Nhóm dữ liệu theo cột `gender` và tính trung bình của `math.grade` và tổng của `sciences.grade` cho mỗi nhóm.
- Hiển thị kết quả dưới dạng Data Frame.

Kết quả sẽ là một Data Frame với các cột `math.grade` (trung bình) và `sciences.grade` (tổng), được nhóm theo `gender`.

Bài 3: Áp dụng hàm và lambda

Sử dụng Data Frame `df` từ tệp `student.csv`.

- Tạo cột mới `adjusted_age` dựa trên cột `age`, trong đó: nếu `age` lớn hơn 18 thì tăng thêm 2, ngược lại giảm đi 1.
- Tạo cột mới `total_grade` bằng tổng của `english.grade`, `math.grade`, `sciences.grade`, và `language.grade`.
- Hiển thị 5 hàng đầu tiên của Data Frame sau khi thêm các cột mới.

Kết quả sẽ hiển thị Data Frame với các cột mới `adjusted_age` và `total_grade`.

Day 18 - Pandas: Advanced Operations & Visualization

Đăng-Nhã Nguyễn

Lý thuyết

1. Merging & Joining

- `pd.merge(df1, df2, on='key', how='inner')`: Hàm `pd.merge()` của thư viện `pandas` được sử dụng để hợp nhất hai Data Frame dựa trên một hoặc nhiều cột chung (khóa).
Các tham số chính:

- `df1, df2`: Hai Data Frame cần hợp nhất.
- `on='key'`: Cột khóa chung để hợp nhất (có thể là danh sách các cột).
- `how`: Kiểu hợp nhất, bao gồm:
  - `'inner'`: Chỉ giữ các hàng có khóa chung trong cả hai Data Frame.
  - `'outer'`: Giữ tất cả các hàng, điền NaN nếu không có giá trị tương ứng.
  - `'left'`: Giữ tất cả hàng từ `df1`, điền NaN nếu không có trong `df2`.
  - `'right'`: Giữ tất cả hàng từ `df2`, điền NaN nếu không có trong `df1`.

Ví dụ:

```
# hợp nhất df1 và df2 dựa trên cột 'id'
df_merged = pd.merge(df1, df2, on='id', how='inner')
```

2. Pivot Tables & Reshaping

- `df.pivot_table(values='col', index='group', columns='category', aggfunc='sum')`: Hàm `pivot_table()` tạopivot table từ Data Frame, cho phép tổng hợp dữ liệu theo nhóm và danh mục.
Các tham số chính:

- `values='col'`: Cột chứa giá trị cần tổng hợp.
- `index='group'`: Cột dùng làm chỉ số (hàng) của pivot table.
- `columns='category'`: Cột dùng làm cột của pivot table.
- `aggfunc='sum'`: Hàm tổng hợp (ví dụ: `'sum'`, `'mean'`, `'count'`).

Ví dụ:

```
df_pivot = df.pivot_table(values='diem', index='lop',
                          columns='mon', aggfunc='sum')
```

Kết quả là một pivot table với tổng `diem` theo `lop` (hàng) và `mon` (cột).

3. Trực quan hóa cơ bản (Basic Visualization)

- `df.plot(kind='bar')`: Vẽ biểu đồ cột từ Data Frame bằng thư viện `matplotlib` tích hợp trong `pandas`.
Ví dụ:

```
# vẽ biểu đồ cột cho tất cả cột số trong df
df.plot(kind='bar')
```

- `df['col'].hist()`: Vẽ biểu đồ histogram cho một cột cụ thể, hiển thị phân phối giá trị.
Ví dụ:

```
# vẽ histogram cho cột 'age'
df['age'].hist()
```

- `sns.heatmap(df.corr())`: Sử dụng thư viện `seaborn` (thường nhập dưới dạng `import seaborn as sns`) để vẽ bản đồ nhiệt (heatmap) hiển thị ma trận tương quan giữa các cột số trong Data Frame.
Ví dụ:

```
# vẽ bản đồ nhiệt của ma trận tương quan
sns.heatmap(df.corr())
```

Lưu ý: Cần import `matplotlib` và `seaborn` trước khi sử dụng.

Bài tập

Bài tập 1: Hợp nhất dữ liệu

Bạn hãy tải thêm một tệp csv khác (có thể tìm từ kaggle) tên là `extra.csv`.

- Tải cả hai tệp `student.csv` và `extra.csv` vào hai Data Frame `df1` và `df2`.
- Hợp nhất hai Data Frame dựa trên cột `id` với kiểu hợp nhất "inner".
- Hiển thị 5 hàng đầu tiên của Data Frame sau khi hợp nhất.

Kết quả sẽ là một Data Frame chứa tất cả cột từ `df1` và `df2`, chỉ giữ các hàng có `id` xuất hiện trong cả hai tệp.

Bài 2: Pivot table

Sử dụng Data Frame `df` từ tệp `student.csv`.

- Tạo pivot table tính tổng `math.grade` theo `nationality` (hàng) và `gender` (cột).
- Tạo pivot table thứ hai tính trung bình `english.grade` theo `ethnic.group` (hàng) và `gender` (cột).
- Hiển thị cả hai pivot table.

Kết quả sẽ là hai pivot table với các giá trị tương ứng được tổng hợp.

Bài tập 3: Trực quan hóa dữ liệu

Sử dụng Data Frame `df` từ tệp `student.csv`.

- Vẽ biểu đồ cột hiển thị trung bình `sciences.grade` theo `gender`.
- Vẽ histogram cho cột `age`.
- Vẽ bản đồ nhiệt (heatmap) của ma trận tương quan giữa các cột điểm (`english.grade, math.grade, sciences.grade, language.grade`).

Lưu ý: Sử dụng thư viện `matplotlib` và `seaborn`.
Kết quả
- Biểu đồ cột so sánh trung bình `sciences.grade` giữa các giới tính.
- Histogram hiển thị phân phối độ tuổi.
- Bản đồ nhiệt thể hiện mức độ tương quan giữa các cột điểm

Day 19 - Simple Gradient Descent for 1D Square Function

Đăng-Nhã Nguyễn

Lý thuyết

Mục tiêu của bài tập này là tìm giá trị tối ưu của hàm một biến (1 chiều) bằng phương pháp *Gradient Descent* (GD). Chúng ta sẽ làm việc với hàm mục tiêu là hàm bình phương đơn giản, nhằm tìm giá trị x sao cho hàm đạt giá trị nhỏ nhất.

Hàm mục tiêu: Xét hàm $f(x) = x^2$. Đây là một hàm lồi, có giá trị nhỏ nhất tại $x = 0$, với $f(0) = 0$.

Gradient Descent: Phương pháp GD là một thuật toán lặp để tìm cực trị địa phương (cục bộ, local extremum) của hàm số. Ý tưởng là di chuyển ngược hướng với đạo hàm (gradient) của hàm tại điểm hiện tại để tiến gần đến điểm tối ưu. Công thức cập nhật của GD cho hàm một biến là:

$$x_{t+1} = x_t - \eta \cdot f'(x_t),$$

trong đó:

- x_t: Giá trị của x tại bước thứ t.
- η: Hằng số học (learning rate), điều khiển kích thước bước di chuyển.
- $f'(x_t)$: Đạo hàm của hàm f tại x_t.

Đạo hàm của hàm mục tiêu: Với $f(x) = x^2$, đạo hàm là:

$$f'(x) = 2x.$$

Thực hiện GD: Từ một điểm khởi tạo x_0, ta lặp lại quá trình cập nhật:

$$x_{t+1} = x_t - \eta \cdot 2x_t,$$

cho đến khi x_t hội tụ (ví dụ, $|x_{t+1} - x_t| < \epsilon$ với ϵ nhỏ) hoặc sau một số lần lặp cố định.

Lưu ý về η:

- Nếu η quá lớn, thuật toán có thể không hội tụ, dao động quanh điểm tối ưu.
- Nếu η quá nhỏ, thuật toán hội tụ chậm.

Độ phủ: Trong bài tập này, "độ phủ" được hiểu là khả năng thuật toán tiến gần đến giá trị tối ưu $x = 0$ (sự chênh lệch giữa giá trị cũ và mới). Chúng ta sẽ đánh giá độ phủ bằng cách quan sát giá trị $|x_t|$ sau mỗi lần lặp.

Bài tập

1. **Tính toán thủ công:**
   - Chọn $x_0 = 10$ và $\eta = 0.1$. Tính toán bằng tay 3 bước lặp đầu tiên của GD cho $f(x) = x^2$. Ghi lại giá trị x_1, x_2, x_3.
   - Tính giá trị hàm mục tiêu $f(x_t)$ tại các điểm đó.
2. **Lập trình:**
   - Viết chương trình bằng Python thực hiện GD cho $f(x) = x^2$.
   - Khởi tạo $x_0 = 10$, $\eta = 0.1$, và lặp 100 lần hoặc cho đến khi $|x_{t+1} - x_t| < 10^{-6}$.
   - In ra giá trị x_t và $f(x_t)$ sau mỗi 10 lần lặp.
   - Vẽ đồ thị của $f(x) = x^2$ và đánh dấu các điểm x_t qua các lần lặp (phần vẽ đồ thị có thể tham khảo solution vì chưa học tới matplotlib).
3. **Thử nghiệm với η:**
   - Chạy lại chương trình với $\eta = 0.01$ và $\eta = 0.5$. So sánh tốc độ hội tụ (số lần lặp để $|x_t| < 0.01$) và rút ra kinh nghiệm về chọn giá trị này trong quá trình training.

* **Lưu ý:** Phần code solution sẽ là lời giải của phần lập trình, chúng ta có thể dựa vào code để tự kiểm tra lại kết quả tính toán thủ công, cũng như việc thay đổi η, rút ra kết luận, kinh nghiệm cá nhân có thể điền vào file colab làm bài tập, TA sẽ thực hiện comment trên kết luận rút ra đó.

Day 20 - Gradient Descent for 2D Convex Function

Đăng-Nhã Nguyễn

Lý thuyết

Bài tập này mở rộng sang tối ưu hóa hàm hai biến (2 chiều), phức tạp hơn hàm 1 chiều ở Day 19. Chúng ta sẽ áp dụng Gradient Descent để tìm điểm tối ưu của một hàm mục tiêu hai chiều.

Hàm mục tiêu: Xét hàm $f(x, y) = x^2 + 2y^2$. Đây là một hàm lồi, có giá trị nhỏ nhất tại $(x, y) = (0, 0)$, với $f(0, 0) = 0$.

Gradient Descent cho 2 chiều: Với hàm $f(x, y)$, gradient là một vector chứa các đạo hàm riêng:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}.$$

Công thức cập nhật của GD là:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} - \eta \cdot \nabla f(x_t, y_t).$$

Gradient của hàm mục tiêu: Với $f(x, y) = x^2 + 2y^2$, gradient là:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 4y \end{bmatrix}.$$

Thực hiện GD: Từ điểm khởi tạo (x_0, y_0), ta cập nhật:

$$x_{t+1} = x_t - \eta \cdot 2x_t, \quad y_{t+1} = y_t - \eta \cdot 4y_t,$$

cho đến khi gradient nhỏ (ví dụ, $||\nabla f(x_t, y_t)|| < \epsilon$) hoặc sau số lần lặp cố định.

Độ phủ: Độ phủ được đánh giá bằng khoảng cách từ (x_t, y_t) đến điểm tối ưu $(0, 0)$, tức là $|| \begin{bmatrix} x_t \\ y_t \end{bmatrix} || = \sqrt{x_t^2 + y_t^2}$.

Lưu ý: Hàm $f(x, y) = x^2 + 2y^2$ có độ cong lớn hơn theo hướng y (do hệ số $2y^2$), khiến tốc độ hội tụ theo y nhanh hơn theo x. Điều này giúp chúng ta hiểu tác động của hình dạng hàm mục tiêu lên GD.

Bài tập

1. **Tính toán thủ công:**
   - Chọn $(x_0, y_0) = (4, 2)$ và $\eta = 0.1$. Tính toán bằng tay 2 bước lặp đầu tiên của GD cho $f(x, y) = x^2 + 2y^2$. Ghi lại (x_1, y_1) và (x_2, y_2).
   - Tính giá trị hàm mục tiêu $f(x_t, y_t)$ tại các điểm đó.
2. **Lập trình:**
   - Viết chương trình thực hiện GD cho $f(x, y) = x^2 + 2y^2$.
   - Khởi tạo $(x_0, y_0) = (4, 2)$, $\eta = 0.1$, và lặp 100 lần hoặc cho đến khi $||\nabla f(x_t, y_t)|| < 10^{-6}$.
   - In ra (x_t, y_t) và $f(x_t, y_t)$ sau mỗi 10 lần lặp.
   - Vẽ quỹ đạo của (x_t, y_t) trên mặt phẳng 2D, cùng với đường đồng mức (contour plot) của $f(x, y)$ (Tương tự Day 19, phần này chúng ta có thể lấy trực tiếp code từ solution).
3. **Thử nghiệm nâng cao (optional):**
   - Chạy lại chương trình với $\eta = 0.01$ và $\eta = 0.4$. So sánh tốc độ hội tụ (số lần lặp để $|| \begin{bmatrix} x_t \\ y_t \end{bmatrix} || < 0.01$).
   - Thử với hàm mục tiêu khác: $f(x, y) = x^2 + y^2$. So sánh quỹ đạo GD với hàm ban đầu và rút ra nhận xét về tác động của hệ số $2y^2$.

* **Lưu ý:** Phần code solution sẽ là lời giải của phần lập trình, chúng ta có thể dựa vào code để tự kiểm tra lại kết quả tính toán thủ công, cũng như việc thay đổi η, rút ra kết luận, kinh nghiệm cá nhân có thể điền vào file colab làm bài tập, TA sẽ thực hiện comment trên kết luận rút ra đó.

Day 21 - Gradient Descent for Linear Regression

Đăng-Nhã Nguyễn

Lý thuyết

Bài tập này tập trung vào việc triển khai hồi quy tuyến tính, một mô hình học máy cơ bản, sử dụng phương pháp *Gradient Descent* (GD) để tối ưu hóa hàm mất mát. Chúng ta sẽ xây dựng một mô hình dự đoán giá trị liên tục dựa trên một biến đầu vào, sử dụng tập dữ liệu huấn luyện.

Hồi quy tuyến tính: Mục tiêu là tìm một hàm tuyến tính $h(x) = w_1 x + w_0$ để dự đoán giá trị y từ x, sao cho sai số giữa giá trị dự đoán và giá trị thực tế là nhỏ nhất. Trong đó:

- w_1: Hệ số góc (slope).
- w_0: Hệ số chặn (intercept).
- Vector trọng số: $\mathbf{w} = [w_1, w_0]^T$.

Tập dữ liệu: Giả sử ta có tập dữ liệu huấn luyện gồm n cặp $(\mathbf{x}_i, y_i)$, với $\mathbf{x}_i \in \mathbb{R}$ và $y_i \in \mathbb{R}$. Để tiện tính toán, ta biểu diễn $\mathbf{x}_i$ dưới dạng vector đặc trưng $\phi(\mathbf{x}_i) = [\mathbf{x}_i, 1]^T$. Khi đó:

$$h(\mathbf{x}_i) = \mathbf{w}^T \phi(\mathbf{x}_i) = w_1 \mathbf{x}_i + w_0.$$

Hàm mất mát: Hàm mất mát được sử dụng là trung bình bình phương sai số (*Mean Squared Error - MSE*):

$$L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{w}^T \phi(\mathbf{x}_i) - y_i \right)^2.$$

Mục tiêu là tìm $\mathbf{w}$ sao cho $L(\mathbf{w})$ nhỏ nhất.

Gradient Descent: Để tối ưu $L(\mathbf{w})$, ta sử dụng GD. Gradient của hàm mất mát là:

$$\nabla L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} 2 \left(\mathbf{w}^T \phi(\mathbf{x}_i) - y_i \right) \phi(\mathbf{x}_i).$$

Công thức cập nhật trọng số:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \cdot \nabla L(\mathbf{w}_t),$$

trong đó η là tốc độ học (*learning rate*).

Độ phủ: Độ phủ được đánh giá thông qua giá trị hàm mất mát $L(\mathbf{w})$ trên tập huấn luyện. Giá trị $L(\mathbf{w})$ nhỏ cho thấy mô hình dự đoán tốt, tức là đường thẳng $h(x)$ khớp tốt với dữ liệu.

Quá trình huấn luyện: Từ vector trọng số ban đầu $\mathbf{w}_0 = [0, 0]^T$, ta lặp lại quá trình cập nhật $\mathbf{w}$ trong một số lần lặp cố định (ví dụ, 2000 lần) hoặc cho đến khi gradient nhỏ ($||\nabla L(\mathbf{w})|| < \epsilon$).

Bài tập

1. **Lập trình:**

   - Tạo tập dữ liệu giả lập: Sinh 100 điểm dữ liệu (x_i, y_i) sao cho $y_i = 2x_i + 1 + \epsilon_i$, với ϵ_i là nhiễu ngẫu nhiên từ phân phối chuẩn $\mathcal{N}(0, 0.1)$.

```python
# Generate synthetic dataset
np.random.seed(42)  # For reproducibility
n = 100
x = np.linspace(0, 10, n)
epsilon = np.random.normal(0, 2, n)
y = 2 * x + 1 + epsilon
trainExample = list(zip(x, y))
plt.scatter(x,y)
```

   - Viết chương trình bằng numpy thực hiện hồi quy tuyến tính với GD:
     - Hàm $\phi(x) = [x, 1]^T$.
     - Hàm mất mát $L(\mathbf{w})$ theo công thức MSE.
     - Gradient $\nabla L(\mathbf{w})$ theo công thức trên.
     - GD với $\mathbf{w}_0 = [0, 0]^T$, $\eta = 0.01$, lặp 2000 lần.
     - In ra $\mathbf{w}$, $L(\mathbf{w})$, và $||\nabla L(\mathbf{w})||$ sau mỗi 200 lần lặp.
   - Vẽ đồ thị: Hiển thị các điểm dữ liệu (x_i, y_i) và đường thẳng $h(x) = w_1 x + w_0$ sau khi huấn luyện.

2. **Thử nghiệm nâng cao:**

   - Chạy lại chương trình với $\eta = 0.001$ và $\eta = 0.1$. So sánh giá trị $L(\mathbf{w})$ cuối cùng và tốc độ hội tụ (số lần lặp để $L(\mathbf{w}) < 0.05$, nếu đạt được).
   - Thử với tập dữ liệu không có nhiễu ($\epsilon_i = 0$). Nhận xét về $\mathbf{w}$ thu được so với giá trị thực $[2, 1]^T$.

* **Lưu ý:** Phần này có thể hơi khó vì có nhiều lý thuyết về toán khó hiểu, mọi người có thể lựa chọn đọc hiểu code thay vì tự code hẳn bài tập. File nộp bài tập có thể là file comment giải thích code solution.
Phần code này sẽ được giải thích và nhắc lại ở buổi live thứ 5.

Day 22 - Used Car Price Prediction - Simple Linear Regression

Đăng-Nhã Nguyễn

Lý thuyết

Dữ liệu xe cũ `used_car.csv`

Tệp dữ liệu `used_car.csv` chứa thông tin về các xe ô tô đã qua sử dụng, được sử dụng để xây dựng mô hình dự đoán giá xe. Dữ liệu có các đặc điểm sau:

- Số lượng hàng: 4009 samples
- Số lượng cột: 12 attributes
- Thông tin chi tiết về các cột:
  1. `brand`: Thương hiệu xe (kiểu `object`)
  2. `model`: Mẫu xe (kiểu `object`)
  3. `model_year`: Năm sản xuất (kiểu `int64`)
  4. `milage`: Số dặm đã đi (kiểu `object`)
  5. `fuel_type`: Loại nhiên liệu (kiểu `object`)
  6. `engine`: Thông tin động cơ (kiểu `object`)
  7. `transmission`: Hộp số (kiểu `object`)
  8. `ext_col`: Màu sắc ngoại thất (kiểu `object`)
  9. `int_col`: Màu sắc nội thất (kiểu `object`)
  10. `accident`: Tình trạng tai nạn (kiểu `object`)
  11. `clean_title`: Tình trạng giấy tờ (kiểu `object`)
  12. `price`: Giá xe (kiểu `object`)

Lý thuyết về Normalization

Normalization (chuẩn hóa dữ liệu) là quá trình biến đổi dữ liệu về một phạm vi chung, thường là [0, 1] hoặc [-1, 1], để đảm bảo các đặc trưng có cùng thang đo. Điều này giúp cải thiện hiệu suất của các mô hình học máy, đặc biệt là các mô hình dựa trên gradient như hồi quy tuyến tính. Các phương pháp chuẩn hóa phổ biến bao gồm:

- **Min-Max Scaling**: Biến đổi dữ liệu về phạm vi [0, 1] theo công thức:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- **Standardization**: Biến đổi dữ liệu để có trung bình bằng 0 và độ lệch chuẩn bằng 1:

$$x' = \frac{x - \mu}{\sigma}$$

  trong đó μ là trung bình và σ là độ lệch chuẩn.

Chuẩn hóa giúp mô hình hội tụ nhanh hơn và tránh hiện tượng một đặc trưng có ảnh hưởng không cân đối đến kết quả dự đoán.

Bài tập

Hãy thực hiện các bước sau để áp dụng mô hình hồi quy tuyến tính dự đoán giá xe cũ dựa trên tệp dữ liệu `used_car.csv` (sử dụng pandas và numpy để đọc dữ liệu va xây dựng mô hình). `matplotlib`.

1. **Tải dữ liệu**: Sử dụng lệnh sau để tải tệp `used_car.csv` từ Google Drive:

```
!gdown 1J1eghnm5bPJTFcBUwTDHZhOP19YGpD6D
```

2. **Đọc và in thông tin dữ liệu**: Sử dụng `pandas` để đọc tệp CSV và in ra thông tin cơ bản của dữ liệu (sử dụng hàm `info(), .columns`, ...).

3. **Tiền xử lý dữ liệu**:
   - Chọn đặc trưng đầu vào (`input`) là cột `model_year` và đầu ra (`output`) là cột `price`.
   - Chuyển đổi cột `price` từ kiểu `string` sang kiểu `int` để sử dụng trong mô hình.
   - Sử dụng thư viện `matplotlib` để vẽ biểu đồ phân tán (scatter plot) thể hiện mối quan hệ giữa `model_year` (x) và `price` (y). Tham khảo code từ ngày 21.
   - Chia dữ liệu thành tập train và test (80% dữ liệu train)

4. **Khai báo mô hình hồi quy tuyến tính**: Sử dụng mô hình hồi quy từ code solution ngày 21.

5. **Training**: Quá trình training, quan sát hàm loss, nếu hàm loss quá lớn dẫn tới các giá trị NaN, hãy áp dụng normalization cho input và output, sau đó thực hiện training lại.

6. **In ra hàm loss**: Sau khi huấn luyện mô hình, plot biểu đồ của hàm loss (code ngày 21).

7. **Thực hiện dự đoán**: Sử dụng mô hình đã huấn luyện để dự đoán giá xe dựa trên các giá trị `model_year`. In ra các trọng số (`weights`) của mô hình.

8. **Visualize đường thẳng dự đoán**: Plot biểu đồ `model_year` và `price` của tập test, sau đó vẽ thêm đường thẳng hồi quy tuyến tính dựa trên mô hình đã huấn luyện để thể hiện xu hướng dự đoán.

Day 23 - Features Engineering

Đăng-Nhã Nguyễn

Lý thuyết

Lý thuyết về Feature Engineering

Feature Engineering (kỹ thuật đặc trưng) là quá trình tạo ra và biến đổi các đặc trưng (features) từ dữ liệu thô để cải thiện hiệu suất của mô hình học máy. Một đặc trưng tốt giúp mô hình học được các mẫu (patterns) trong dữ liệu một cách hiệu quả hơn. Các kỹ thuật Feature Engineering phổ biến bao gồm:

- **Xử lý giá trị thiếu (Missing Data)**: Điền giá trị thiếu bằng trung bình, trung vị, mode hoặc xóa các cột/dòng có quá nhiều giá trị thiếu.
- **Chuyển đổi dữ liệu**: Biến đổi dữ liệu dạng chuỗi (string) thành số (numeric) hoặc mã hóa các biến phân loại (categorical variables) như Label Encoding hoặc One-Hot Encoding.
- **Tạo đặc trưng mới**: Tạo các đặc trưng mới từ các đặc trưng hiện có, ví dụ: tính tuổi xe từ năm sản xuất.
- **Chuẩn hóa và chuyển đổi thang đo**: Đảm bảo các đặc trưng có cùng thang đo, ví dụ: chuẩn hóa Min-Max hoặc Standardization.
- **Xử lý đặc trưng phân loại**: Chuyển các giá trị phân loại thành dạng số bằng cách ánh xạ (mapping) hoặc mã hóa.

Feature Engineering đòi hỏi sự hiểu biết về dữ liệu và bài toán để lựa chọn hoặc tạo ra các đặc trưng phù hợp, từ đó tăng độ chính xác và khả năng tổng quát hóa của mô hình.

Bài tập

Hãy thực hiện các bước sau để tiền xử lý và tạo đặc trưng từ tệp dữ liệu `used_car.csv` nhằm chuẩn bị cho bài toán dự đoán giá xe cũ (với nhiều features đầu vào).

1. **Tiền xử lý dữ liệu**:
   - Xử lý giá trị thiếu (missing data) cho tất cả các cột trong `used_car.csv`. Chúng ta có thể điền giá trị thiếu bằng mode cho các cột phân loại hoặc trung bình/trung vị cho cột số, tùy thuộc vào kiểu dữ liệu (ở đây chúng ta sẽ thực hiện xóa luôn các sample có giá trị `NaN`).
   - Xóa cột `clean_title` vì cột này có quá nhiều giá trị `NaN`.
2. **Xử lý nhãn (label)** `price`:

- Chuyển đổi cột `price` từ kiểu chuỗi (string) sang kiểu số để sử dụng làm output (`y`).
- Đảm bảo không có giá trị thiếu hoặc giá trị không hợp lệ trong cột `price`.

3. **Lựa chọn đặc trưng**:
   - Lựa chọn các đặc trưng sau làm đầu vào (`features`): `brand`, `model`, `milage`, `transmission`, `engine`, `int_col`, `ext_col`, `fuel_type`, `car_age`.
4. **Xử lý từng đặc trưng**:
   - **Chuyển `model_year` sang `car_age`**: Tính tuổi xe (`car_age`) bằng cách lấy năm hiện tại (2025) trừ đi giá trị trong cột `model_year`.
   - **Chuyển `milage` sang dạng số**: Cột milage đang ở dạng chuỗi, hãy chuyển đổi thành kiểu số (integer hoặc float) bằng cách xóa các ký tự không phải số (ví dụ: "miles") và chuyển đổi.
   - **Xử lý cột `engine`**: Sử dụng phương thức

     ```
     .str.extract(r'(\d+\.\d+)L')
     ```

     để trích xuất giá trị số (dung tích động cơ, ví dụ: 2.0 từ "2.0L") từ cột `engine`. Quá trình này sẽ xuất hiện giá trị thiếu (nếu có) nên hãy thực hiện check `NaN`.
   - **Xử lý các cột phân loại**: Đối với các cột `brand`, `model`, `transmission`, `int_col`, `ext_col`, `fuel_type`:
     - In ra các giá trị duy nhất (`unique`) của từng cột để hiểu dữ liệu.
     - Tạo từ điển (`dictionary`) ánh xạ các giá trị phân loại thành các số nguyên (ví dụ: {"Toyota": 0, "Honda": 1, ...}).
     - Thay thế các giá trị trong cột bằng các số tương ứng (sử dụng ánh xạ hoặc `map`).
5. **Chia dữ liệu thành tập huấn luyện và kiểm tra**:
   - Chia dữ liệu thành 80% tập huấn luyện (`train`) và 20% tập kiểm tra (`test`).
   - Đảm bảo các đặc trưng (`X`) và nhãn (`y`), in ra để kiểm tra.

Day 24 - Multiple Linear Regression

Đăng-Nhã Nguyễn

Lý thuyết

Hồi quy tuyến tính đa biến

Hồi quy tuyến tính đa biến (Multiple Linear Regression) là một mở rộng của hồi quy tuyến tính đơn biến, trong đó đầu vào $\mathbf{x}$ không chỉ là một đặc trưng (feature) mà là một vector chứa nhiều đặc trưng. Điều này cho phép mô hình dự đoán giá trị đầu ra y dựa trên sự kết hợp tuyến tính của nhiều biến đầu vào.
Trong hồi quy tuyến tính đơn biến, hàm dự đoán có dạng:

$$h(x) = w_0 + w_1 x$$

với x là một đặc trưng duy nhất, w_0 là bias, và w_1 là trọng số.
Trong hồi quy tuyến tính đa biến, đầu vào $\mathbf{x}$ là một vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, và hàm dự đoán được biểu diễn dưới dạng tích vô hướng:

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

trong đó:

- $\mathbf{w} = [w_0, w_1, \ldots, w_n]^T$: Vector trọng số (bao gồm bias w_0).
- $\mathbf{x} = [1, x_1, x_2, \ldots, x_n]^T$: Vector đặc trưng (thêm 1 để tương ứng với bias).

Sự khác biệt chính giữa hồi quy tuyến tính đơn biến và đa biến nằm ở số lượng đặc trưng:

- Hồi quy đơn biến chỉ sử dụng một đặc trưng, phù hợp với các bài toán đơn giản.
- Hồi quy đa biến sử dụng nhiều đặc trưng, cho phép mô hình học được các mối quan hệ phức tạp hơn giữa các đặc trưng và đầu ra.

Mục tiêu của mô hình là tìm vector trọng số $\mathbf{w}$ sao cho hàm mất mát (Mean Squared Error) được tối ưu hóa:

$$\text{Loss} = \frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}^{(i)}) - y^{(i)})^2$$

trong đó m là số lượng mẫu.

Bài tập

Hãy sử dụng dữ liệu đã được tiền xử lý từ ngày 23. Dựa vào code của ngày 21 (hồi quy tuyến tính đơn biến), chỉnh sửa mô hình để phù hợp với đầu vào nhiều đặc trưng.

Day 25 - Git/GitHub: Basic Git Command

Đăng-Nhã Nguyễn

Lý Thuyết:

Sau đây là các lệnh sẽ dùng cho phần bài tập

Lệnh	**Ý Nghĩa**
`git clone <URL>`	Sao chép repo từ GitHub về máy local.
`git add <file>`	Đưa file vào staging area.
`git status`	Kiểm tra trạng thái thay đổi.
`git commit -m "Message"`	Tạo commit.
`git log`	Xem lịch sử commit.
`git push -u origin main`	Đẩy code từ local lên GitHub.

Bài Tập

Bước 1: Tạo Tài Khoản GitHub

- Truy cập github.com → Đăng ký tài khoản (nếu chưa có).
- Xác thực email theo hướng dẫn.

Bước 2: Tạo Repository trên GitHub

1. Đăng nhập GitHub → Click **New Repository**.
2. Đặt tên repo (ví dụ: `my-first-repo`).
3. Chọn **Public** → Tích vào **Add a README.md** → Click **Create Repository**.

Bước 3: Thiết Lập Git Local

Mở Terminal/CMD và chạy: Sử dụng SSH key để kết nối với tài khoản github (theo video hướng dẫn)

Bước 4: Clone Repository về Máy

```
git clone https://github.com/username/my-first-repo.git
cd my-first-repo
```

Bước 5: Tạo File Cá Nhân

1. Tạo một file `abc.txt` hoặc thay đổi nội dung `README.md`:
2. Kiểm tra trạng thái:

```
git status
```

Kết quả mong đợi: File mới hiển thị ở `Untracked files`.

Bước 6: Thêm File vào Staging Area

```
git add README.md
git status  # File chuyển sang trạng thái "Changes to be committed"
```

Bước 7: Tạo Commit

```
git commit -m "Add readme file"
git log  # Xem commit vừa tạo
```

Bước 8: Push Code Lên GitHub

```
git push -u origin main
```

Bước 9: Kiểm Tra Trên GitHub

- Mở repo trên GitHub → Kiểm tra nội dung file `README.md` đã thay đổi.
- Vào tab **Commits** để xem lịch sử:

Day 26 - Advanced GitHub: Branch - Conflict Handling

Đăng-Nhã Nguyễn

Lý Thuyết

1. Branch (Nhánh)

Lệnh	Ý Nghĩa
`git branch`	Liệt kê các nhánh
`git checkout -b <tên-nhánh>`	Tạo nhánh mới và chuyển sang
`git merge <nhánh>`	Gộp nhánh vào nhánh hiện tại

2. Conflict (Xung đột)

Conflict xảy ra ở rất nhiều tình huống khi chúng ta merge branch, push code, tuy nhiên đơn giản hóa và hiểu rằng, conflict xảy ra khi nhiều người cùng chỉnh sửa cùng một file và Git không thể tự động gộp được.

3. Teamwork Tips

- Luôn bắt đầu từ `git pull` để cập nhật code mới nhất.
- Làm việc trên nhánh riêng (không sửa trực tiếp `main`): Luôn tạo một branch mới khi làm việc trên source code.
- Tạo Pull Request (GitHub) để review code khi đã hoàn thiện một tính năng.

Bài Tập

Bước 1: Tạo Nhánh và Làm Việc Độc Lập

1. Clone repo từ ngày 1:

```
git clone https://github.com/username/my-first-repo.git
cd my-first-repo
```

2. Tạo nhánh mới cho tính năng của bạn:

```
git checkout -b feature/add-README
```

3. Thêm thông tin vào file `README.md`:

```
"Skills: Git, LaTeX"
```

4. Commit thay đổi:

```
git add README.md
git commit -m "Add skills to README"
```

Bước 2: Giả Lập Conflict

1. Chuyển về nhánh `main`:

```
git checkout main
```

2. Sửa file `README.md` (Tình huống đồng đội sửa):

```
echo "Hobbies: Reading, Coding" >> README.md
git add README.md
git commit -m "Add hobbies"
```

3. Gộp nhánh `feature/add-README` vào `main`:

```
git merge feature/add-README
```

Kết quả: Xuất hiện conflict do sửa cùng file!

Bước 3: Giải Quyết Conflict

1. Mở file `README.md` trong trình soạn thảo, tìm các marker:

```
<<<<<<< HEAD
Hobbies: Reading, Coding
=======
Skills: Git, LaTeX
>>>>>>> feature/add-README
```

2. Sửa file để giữ cả 2 nội dung:

```
Hobbies: Reading, Coding
Skills: Git, LaTeX
```

3. Đánh dấu đã giải quyết xong:

```
git add README.md
git commit -m "Resolve merge conflict"
```

Bước 4: Làm Việc Nhóm với GitHub

1. Đẩy nhánh `main` lên GitHub:

```
git push origin main
```

2. Tạo Pull Request (PR):

   - Vào repo trên GitHub → Tab **Pull Requests** → **New PR**.
   - Chọn nhánh `feature/add-README` để merge vào `main`.
   - Thêm mô tả → Tạo PR → Đồng đội review code.

Day 27 - Matplotlib Visualization

Đăng-Nhã Nguyễn

Lý thuyết

1. Cấu trúc Matplotlib

- **Figure**: Là container cấp cao nhất, có thể chứa nhiều Axes

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,6))  # Figure 8x6 inches
```

- **Axes**: Vùng vẽ đồ thị (chứa title, labels, lines,...)

```
ax = fig.add_subplot(111)  # 1 hàng, 1 cột, vị trí 1
```

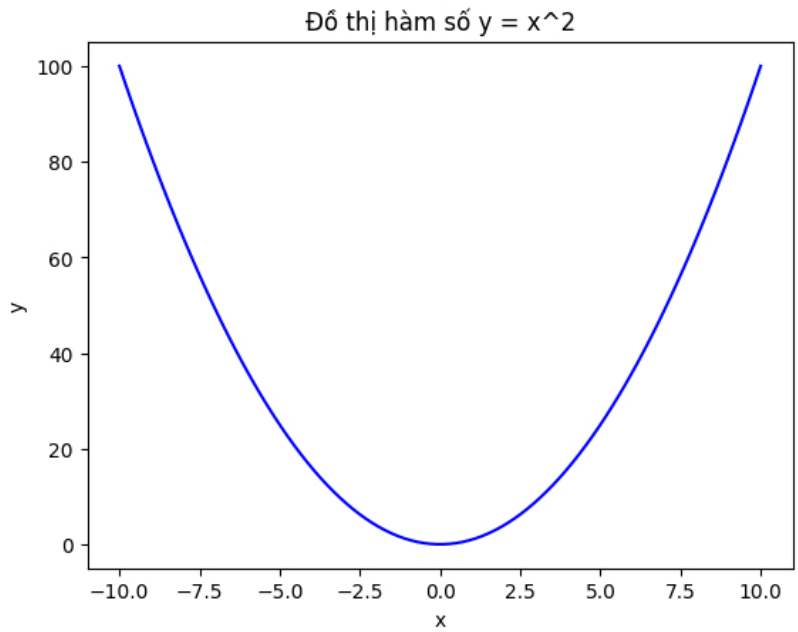
- **Axis**: Các trục tọa độ với tỷ lệ và ticks

```
ax.set_xlim(0, 10)  # Giới hạn trục x từ 0-10
ax.set_xticks([0, 5, 10])  # Đặt vị trí tick
```

Ví dụ tổng quát:

```
import matplotlib.pyplot as plt
import numpy as np
# Tạo dữ liệu
x = np.linspace(-10, 10, 100)
y = x**2

# Tạo Figure và Axes
fig, ax = plt.subplots()  # fig là Figure, ax là một Axes

# Vẽ đồ thị gồm 2 axis (Ox, Oy) lên Axes
ax.plot(x, y, label='y = x^2', color='blue')

# Thiết lập tiêu đề, nhãn trục qua Axes
ax.set_title("Đồ thị hàm số y = x^2")
ax.set_xlabel("x")  # Đây là XAxis (trục hoành)
ax.set_ylabel("y")  # Đây là YAxis (trục tung)

# Thêm lưới và chú thích
ax.grid(True)
ax.legend()

# Hiển thị Figure (bao gồm tất cả các Axes bên trong)
plt.show()
```

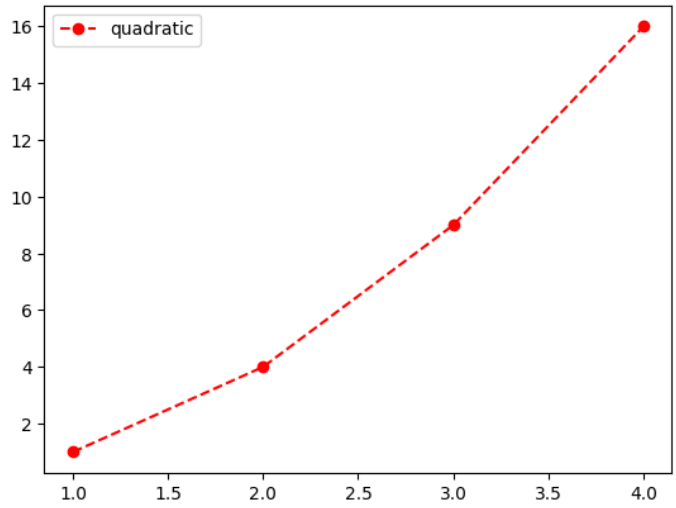
Kết quả:



2. Các loại đồ thị cơ bản

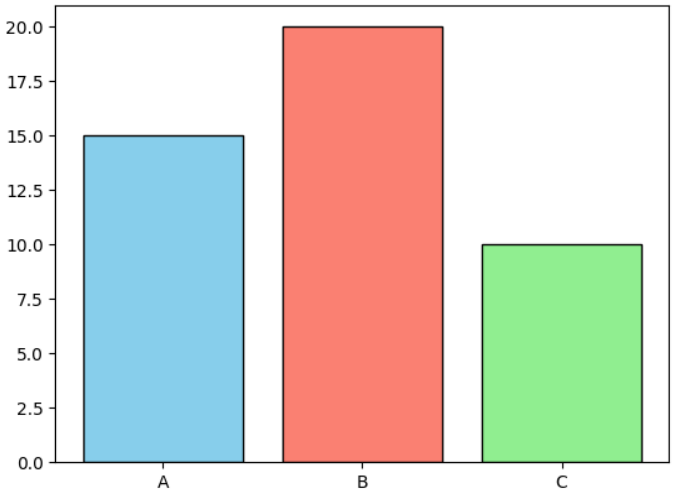
2.1 Đồ thị đường (Line plot)

```
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]
plt.plot(x, y,
         color='red',
         linestyle='--',
         marker='o',
         label='quadratic')
plt.legend()
```



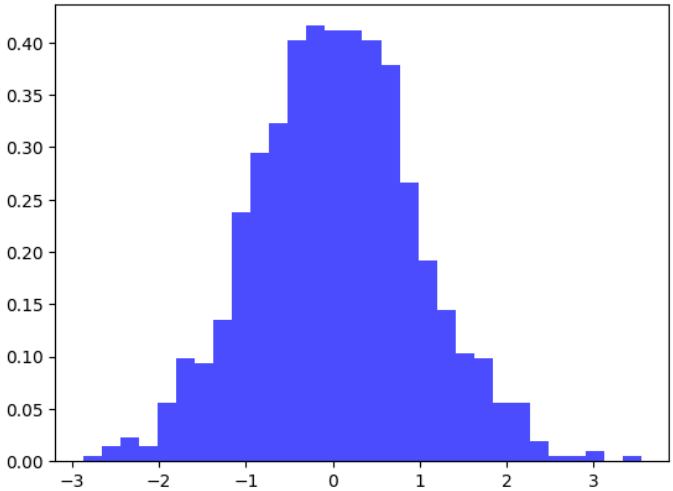
2.2 Biểu đồ cột (Bar chart)

```
categories = ['A', 'B', 'C']
values = [15, 20, 10]
plt.bar(categories, values,
        color=['skyblue', 'salmon', 'lightgreen'],
        edgecolor='black')
```



2.3 Histogram

```
data = np.random.normal(0, 1, 1000)
plt.hist(data, bins=30,
         density=True,
         alpha=0.7,
         color='blue')
```



Bài tập

Bài 1: Vẽ đồ thị cơ bản

Cho dữ liệu về nhiệt độ trong tuần:

```
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
temperatures = [22, 24, 25, 23, 26, 28, 27]
```

Hãy vẽ đồ thị đường biểu diễn nhiệt độ theo ngày với các yêu cầu:

- Đường màu đỏ, kiểu nét đứt
- Đánh dấu các điểm bằng hình tròn
- Thêm tiêu đề "Weekly Temperature"
- Thêm nhãn cho trục x (Day) và trục y (Temperature °C)
- Hiển thị lưới

Bài 2: Biểu đồ histogram

Sinh ngẫu nhiên một mảng 1000 số theo phân phối chuẩn (mean=0, std=1). Vẽ histogram với:

- 30 bins
- Màu xanh lá cây
- Đường viền màu đen

Bài 3: Subplots

Cho dữ liệu:

```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
```

Tạo một figure với 2 subplot (1 hàng, 2 cột):

- Subplot 1: Vẽ đồ thị hàm sin, màu xanh dương
- Subplot 2: Vẽ đồ thị hàm cos, màu đỏ
- Thêm tiêu đề cho từng subplot
- Điều chỉnh khoảng cách giữa các subplot

Bài 4: Bar plot kết hợp

Cho dữ liệu doanh thu 4 quý của 2 năm:

```
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
sales_2022 = [120, 135, 145, 160]
sales_2023 = [130, 148, 155, 170]
```

Vẽ biểu đồ cột kết hợp (grouped bar chart) so sánh doanh thu từng quý của 2 năm:

- Mỗi năm có màu khác nhau
- Thêm chú thích (legend)
- Thêm tiêu đề "Quarterly Sales Comparison"
- Thêm nhãn giá trị trên mỗi cột (optional)

Day 28 - Seaborn Visualization

Đăng-Nhã Nguyễn

Lý thuyết

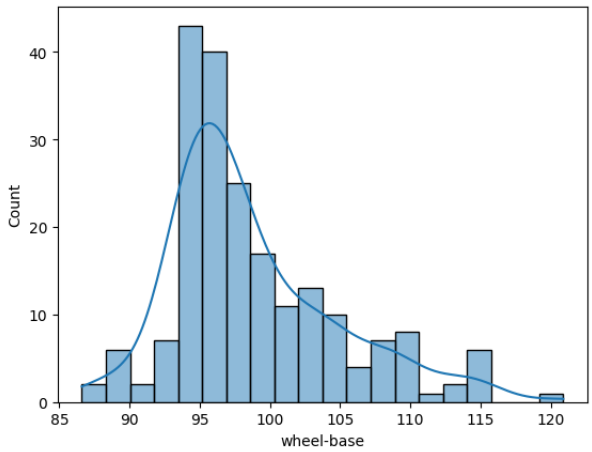
1. Seaborn

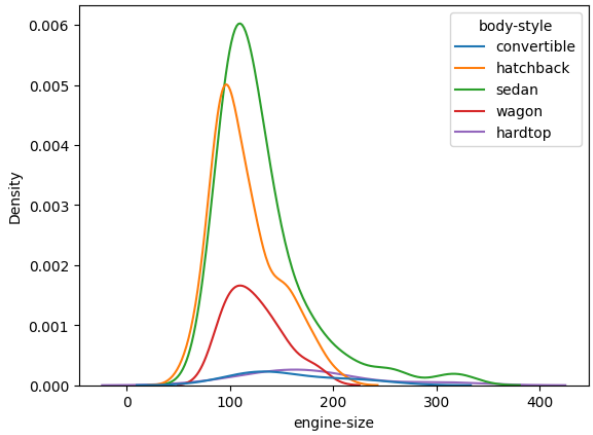
- Là thư viện visualization nâng cao xây dựng trên Matplotlib
- Giao diện đơn giản cho visualization thống kê (statistic)
- Tích hợp tốt với Pandas DataFrames

2. Các loại đồ thị phổ biến

2.1 Distribution Plots

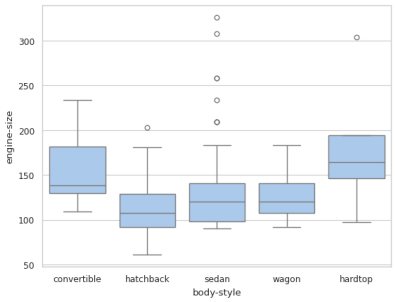
```
import seaborn as sns
sns.histplot(data=df, x='wheel-base', kde=True, bins=20)
sns.kdeplot(data=df, x='engine-size', hue='body-style')
```

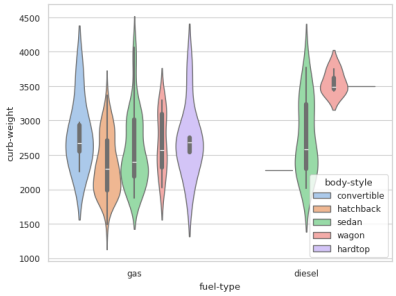


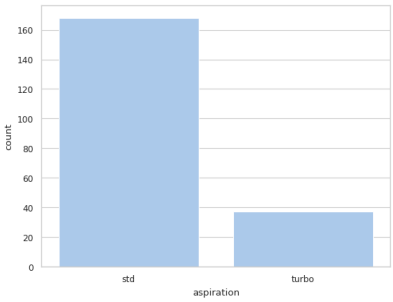


2.2 Categorical Plots

```
sns.boxplot(data=df, x='body-style', y='engine-size')
sns.violinplot(data=df, x='fuel-type', y='curb-weight', hue='body-style')
sns.countplot(data=df, x='aspiration')
```







3. Các tuỳ chỉnh khác

3.1 Figure Styles

```
sns.set_style("whitegrid")  # darkgrid, white, ticks
sns.set_context("paper")    # talk, poster
sns.set_palette("pastel")   # deep, muted, bright
```

3.2 FacetGrid

```
g = sns.FacetGrid(df, col="time", row="sex")
g.map(sns.scatterplot, "total_bill", "tip")
```

3.3 Heatmap

```
sns.heatmap(corr_matrix, annot=True,
            cmap="coolwarm", center=0)
```

Bài tập

Với mỗi bài tập, chúng ta cần đọc hiểu các dataset, dựa theo yêu cầu bài làm tìm đúng hàm seaborn phù hợp, đọc hiểu ý nghĩa của các thâm số, kết hợp matplotlib để vẽ các đồ thị được yêu cầu.

Bài 1: Phân bố dữ liệu

Cho dataset tips (có sẵn trong Seaborn):

```
tips = sns.load_dataset("tips")
```

Yêu cầu:

- Vẽ histogram phân bố tổng bill (total_bill) với KDE
- Phân tích phân bố theo thời điểm (time) bằng violinplot
- Thêm title và điều chỉnh màu sắc phù hợp

Bài 2: Phân tích tương quan

Cho dataset diamonds (có sẵn trong Seaborn):

```
diamonds = sns.load_dataset("diamonds")
```

Yêu cầu:

- Vẽ scatterplot carat vs price, hue theo cut
- Thêm regression line cho từng nhóm cut
- Điều chỉnh kích thước marker theo biến depth

Bài 3: Heatmap và FacetGrid

Cho dataset flights:

```
flights = sns.load_dataset("flights")
```

Yêu cầu:

- Chuyển dữ liệu thành dạng pivot table (year làm index, month làm columns)
- Vẽ heatmap thể hiện số lượng hành khách theo tháng/năm
- Sử dụng FacetGrid để phân tích trend theo năm

Day 29 - EDA Visualization

Đăng-Nhã Nguyễn

Lý thuyết:

1. Mục tiêu EDA Visualization

- Hiểu cấu trúc và phân phối dữ liệu
- Phát hiện outliers và missing values
- Khám phá mối quan hệ giữa các biến
- Chuẩn bị dữ liệu cho modeling

2. Các bước cơ bản

1. **Univariate Analysis**:

```
sns.histplot(data=df, x='age')  # Phân phối
sns.boxplot(data=df, y='income')  # Outliers
```

2. **Bivariate Analysis**:

```
sns.scatterplot(data=df, x='age', y='income')  # Quan hệ 2 biến
sns.heatmap(df.corr(), annot=True)  # Ma trận tương quan
```

3. **Multivariate Analysis**:

```
sns.pairplot(df, hue='target')  # Tương quan đa biến
sns.FacetGrid(df, col='category')  # Phân tích theo nhóm
```

3. Công cụ visualization

Mục đích	Matplotlib	Seaborn
Phân phối	hist, boxplot	histplot, kdeplot, boxplot
Quan hệ	scatter, bar	scatterplot, lineplot, barplot
Phân nhóm	-	violinplot, swarmplot

Bài tập: EDA Titanic Dataset

Dataset

```
import seaborn as sns
df = sns.load_dataset('titanic')  # Hoặc tải từ Kaggle
```

Yêu cầu EDA

Thực hiện các bước sau và trả lời câu hỏi:

1. **Hiểu dữ liệu cơ bản**:

```
df.info()
df.describe(include='all')
```

2. **Visualize missing data bằng heatmap**:

```
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
```

3. **Phân phối tuổi**:
   - Vẽ histogram + KDE cho biến 'age'
   - So sánh phân phối theo giới tính (hue='sex')
   - Phát hiện outliers bằng boxplot
4. **Tỷ lệ sống sót**:
   - Vẽ countplot cho biến 'survived'
   - Phân tích theo lớp vé (pclass) và giới tính
   - Sử dụng barplot với tỷ lệ
5. **Mối quan hệ fare - survival**:
   - Vẽ violinplot so sánh phân phối giá vé giữa người sống/chết
   - Tính correlation giữa fare và survival
6. **Multivariate analysis**:
   - Tạo pairplot cho các biến số (age, fare, parch)
   - Vẽ heatmap correlation matrix

Các hàm cần dùng - Gợi ý

```
# 1. Phân phối
sns.histplot(data=df, x='age', kde=True, hue='sex')
sns.boxplot(data=df, x='sex', y='age')

# 2. Tỷ lệ sống sót
sns.countplot(data=df, x='survived', hue='pclass')
sns.barplot(data=df, x='pclass', y='survived',
            estimator=lambda x: sum(x)/len(x)*100)

# 3. Phân tích đa biến
sns.pairplot(df[['age','fare','parch','survived']], hue='survived')
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

Day 30 - Pytorch Basic

Đăng-Nhã Nguyễn

Lý thuyết

- **PyTorch là gì?**
  - Thư viện tính toán số học tương tự NumPy nhưng hỗ trợ GPU
  - Cung cấp hệ thống tính gradient tự động (autograd)
  - Nền tảng cho deep learning
- **Tensor Creation**:
  - `torch.tensor(data)`: Tạo từ Python list/NumPy array
  - `torch.zeros/ones/empty(shape)`
  - `torch.rand/randn(shape)`: Uniform/Normal distribution
  - `torch.arange(start, end, step)`
- **Shape & Indexing**:
  - `.shape`: Thuộc tính kiểm tra kích thước
  - Indexing giống NumPy: `tensor[i,j]`, `tensor[:,1]`
  - Slicing: `tensor[1:3, 0:2]`
- **Tensor Math**:
  - Phép toán từng phần tử: `+, -, *, /`
  - Phép ma trận: `@` hoặc `torch.matmul()`
  - Broadcasting: Tự động mở rộng tensor cho phép toán
- **Reshaping**:
  - `.view(shape)`: Thay đổi kích thước (không copy dữ liệu)
  - `.reshape(shape)`: Tương tự view nhưng an toàn hơn
  - `.unsqueeze(dim)`: Thêm chiều có kích thước 1
  - `.squeeze()`: Bỏ các chiều có kích thước 1
- **Type & Device**:
  - `.dtype`: Kiểu dữ liệu (torch.float32, torch.int64,...)
  - `.to(device)`: Chuyển giữa CPU/GPU (`"cuda"`)
  - `.cpu()`: Chuyển về CPU

Bài tập

Tương tự với thư viện numpy đã học, hãy thực hiện các phép tạo, thao tác cơ bản với pytorch.

Bài 1: Tensor Creation & Manipulation

1. Tạo các tensor sau:
   - Tensor 2D kích thước (3,4) chứa số 7
   - Tensor 1D chứa số chẵn từ 0 đến 10
   - Tensor ngẫu nhiên kích thước (2,3) từ phân phối chuẩn
2. Cho tensor `A = torch.arange(12).reshape(3,4)`, thực hiện:
   - Lấy hàng đầu tiên
   - Lấy cột cuối cùng
   - Lấy phần tử ở hàng 1 cột 2
   - Lấy ma trận con gồm 2 hàng đầu, 2 cột cuối

Bài 2: Tensor Operations

Cho `A = torch.tensor([[1,2],[3,4]])`, `B = torch.tensor([[5],[6]])`

1. Thực hiện phép cộng A + B (broadcasting)
2. Tính tích element-wise của A với chính nó
3. Thực hiện phép nhân ma trận giữa A và B
4. Chuyển A thành tensor 4 chiều (1,1,2,2)

Bài 3: Reshape & Device

1. Cho tensor `C = torch.rand(6)`, biến đổi thành:
   - Tensor (6,1) dùng `view`
   - Tensor (2,3) dùng `reshape`
2. Kiểm tra xem GPU có khả dụng không (`torch.cuda.is_available()`)
   Lưu ý: Thay đổi runtime ở trên google colab sang GPU
3. Nếu có GPU:
   - Tạo tensor trên CPU rồi chuyển sang GPU
   - Tạo tensor trực tiếp trên GPU
   - Chuyển tensor về CPU

Day 31 - PyTorch Advanced

Đăng-Nhã Nguyễn

Lý thuyết

- **In-place Operations**
  - Thao tác trực tiếp trên bộ nhớ hiện tại, tiết kiệm bộ nhớ

```
x = torch.tensor([1., 2., 3.])
x.add_(5)  # Tương đương x = x + 5 nhưng không tạo bản copy
print(x)   # tensor([6., 7., 8.])
```

  - Các hàm có suffix _ đều là in-place:

```
x.mul_(2)     # Nhân tại chỗ
x.fill_(10)   # Gán tất cả giá trị thành 10
```

- **Reduction Operations**
  - Tính toán dọc theo các chiều chỉ định:

```
A = torch.tensor([[1,2], [3,4]])
print(A.sum(dim=0))   # Tổng theo cột: tensor([4, 6])
print(A.mean(dim=1))  # Trung bình theo hàng: tensor([1.5, 3.5])
```

  - Giữ nguyên số chiều với `keepdim=True`:

```
print(A.sum(dim=1, keepdim=True))  # tensor([[3], [7]])
```

- **Masking & Boolean Indexing**
  - Lọc phần tử dùng mask bool:

```
B = torch.rand(3,3)
mask = B > 0.5
print(B[mask])  # Xuất các giá trị > 0.5
```

  - `torch.where` để thay thế có điều kiện:

```
# Thay giá trị âm bằng 0
C = torch.tensor([-1, 2, -3])
print(torch.where(C > 0, C, torch.zeros_like(C)))  # [0, 2, 0]
```

- **Advanced Indexing**
  - `gather` để thu thập giá trị theo chỉ số:

```
data = torch.tensor([[1,2],[3,4]])
indices = torch.tensor([[0,0],[1,0]])
print(torch.gather(data, 1, indices))  # [[1,1], [4,3]]
```

- `scatter` để gán giá trị theo chỉ số:

```
out = torch.zeros(2,2)
src = torch.tensor([[1,2],[3,4]])
indices = torch.tensor([[0,1],[1,0]])
print(out.scatter(1, indices, src))  # [[1,2], [4,3]]
```

- **Batch Operations**
  - Nhân ma trận theo batch:

```
batch1 = torch.randn(10, 3, 4)  # Batch of 10 matrices
batch2 = torch.randn(10, 4, 5)
result = torch.bmm(batch1, batch2)  # Result shape: (10, 3, 5)
```

Bài tập thực hành

Bài 1: In-place Operations & Reductions

Cho tensor `A = torch.arange(12).reshape(3,4).float()`

1. Thực hiện các phép in-place:
   - Cộng thêm 5 vào tất cả phần tử
   - Nhân đôi giá trị các phần tử
2. Tính toán các giá trị reduction:
   - Tổng theo từng hàng (giữ nguyên chiều)
   - Giá trị lớn nhất theo từng cột
   - Chỉ số của các giá trị lớn nhất theo từng cột

Bài 2: Masking & Advanced Indexing

Cho tensor `B = torch.randn(5,5)`

1. Tạo mask để chọn các phần tử:
   - Lớn hơn 0.5
   - Nhỏ hơn -0.3 hoặc lớn hơn 0.7
2. Sử dụng `torch.where` để thay thế:
   - Các giá trị âm bằng 0
   - Các giá trị dương bằng 1
3. Sử dụng `torch.gather` để thu thập các phần tử theo chỉ số:
   - Chỉ số hàng: `[0,1,2,3,4]`
   - Chỉ số cột: `[4,3,2,1,0]`

Bài 3: Batch Operations & Memory

1. Cho batch matrices `X = torch.randn(3,2,2)`, `Y = torch.randn(3,2,2)`:
   - Thực hiện batch matrix multiplication
   - Tính tổng theo chiều batch
2. Thực hiện các thao tác quản lý bộ nhớ:
   - Kiểm tra contiguous của tensor X
   - Nếu không contiguous, tạo bản copy contiguous
   - Giải phóng bộ nhớ GPU cache (nếu dùng GPU)

Day 32 - Autograd & Simple Function Derivatives

Đăng-Nhã Nguyễn

Lý thuyết

Để hiểu và làm bài tập về Autograd và đạo hàm của các hàm đơn giản, cần nắm các kiến thức sau:

- **Autograd**: Là một công cụ tự động tính đạo hàm (automatic differentiation) được sử dụng trong các thư viện học sâu như PyTorch. Autograd theo dõi các phép toán trên các tensor và tính gradient một cách tự động thông qua quy tắc chuỗi (chain rule).
- **Quy tắc chuỗi**: Nếu $y = f(g(x))$, thì đạo hàm của y theo x là:

$$\frac{dy}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}.$$

- **Đạo hàm của các hàm cơ bản**:
  - Hàm số mũ: $\frac{d}{dx}(e^x) = e^x$.
  - Hàm lũy thừa: $\frac{d}{dx}(x^n) = nx^{n-1}$.
  - Hàm lượng giác: $\frac{d}{dx}(\sin x) = \cos x$, $\frac{d}{dx}(\cos x) = -\sin x$.
  - Hàm logarit: $\frac{d}{dx}(\ln x) = \frac{1}{x}$.
- **Gradient trong Machine Learning**: Gradient là vector chứa các đạo hàm riêng, được sử dụng để tối ưu hóa hàm mất mát trong các thuật toán như Gradient Descent.
- **Lưu ý khi sử dụng Autograd**:
  - Đảm bảo tensor có thuộc tính `requires_grad=True`.
  - Sử dụng `backward()` để tính gradient.
  - Xóa gradient sau mỗi bước tối ưu bằng `zero_grad()`.

Bài tập

Bài 1: Tính đạo hàm cơ bản

1. Cho $x = 3.0$ (đặt `requires_grad=True`), tính:
   - $y = x^2 + 3x + 1$, tìm $\frac{dy}{dx}$
   - $y = \sin(x) + e^x$, tìm $\frac{dy}{dx}$
2. Kiểm tra kết quả bằng tính toán tay

Bài 2: Đạo hàm nhiều biến

Cho $x = 2.0$, $y = 4.0$ (đều có `requires_grad=True`):

1. Tính $z = x^2y + y^3$
2. Tìm $\frac{\partial z}{\partial x}$ và $\frac{\partial z}{\partial y}$
3. Giải thích kết quả nhận được

Bài 3: Chain Rule

1. Cho hàm hợp $f(g(x))$ với:
   - $g(x) = x^3 + 2x$ (x = 1.0)
   - $f(u) = \ln(u)$
2. Tính $\frac{df}{dx}$ bằng PyTorch autograd
3. Kiểm tra bằng chain rule: $\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$

Day 33- Tensorflow Basic

Đăng-Nhã Nguyễn

Lý Thuyết

Tensor Basics

Tensor là cấu trúc dữ liệu cốt lõi trong TensorFlow, tương tự như mảng đa chiều trong NumPy. Mỗi tensor có các thuộc tính:

- `shape`: Kích thước các chiều
- `dtype`: Kiểu dữ liệu
- `rank`: Số chiều

```
import tensorflow as tf

# Tạo tensor từ list
tensor = tf.constant([[1, 2], [3, 4]])
print("Shape:", tensor.shape)
print("Dtype:", tensor.dtype)
print("Rank:", tensor.ndim)
```

Tensor Initialization

TensorFlow cung cấp nhiều cách khởi tạo tensor:

```
# Khởi tạo tensor zeros
zeros = tf.zeros([2, 3])

# Khởi tạo tensor ones
ones = tf.ones([3, 2])

# Khởi tạo tensor với giá trị ngẫu nhiên
random = tf.random.normal([2, 2], mean=0.0, stddev=1.0)
```

Type Casting

Chuyển đổi kiểu dữ liệu tensor:

```
float_tensor = tf.constant([1.0, 2.0])
int_tensor = tf.cast(float_tensor, tf.int32)
```

Indexing & Slicing

Truy cập phần tử tensor tương tự NumPy:

```
tensor = tf.constant([[1, 2, 3], [4, 5, 6]])

# Lấy phần tử tại hàng 1, cột 2
elem = tensor[1, 2]  # 6

# Lấy slice
slice = tensor[:, 1:]  # [[2, 3], [5, 6]]
```

Math Operations

Các phép toán element-wise:

```
a = tf.constant([1, 2, 3])
b = tf.constant([4, 5, 6])

# Cộng element-wise
add = tf.add(a, b)  # [5, 7, 9]

# Nhân element-wise
mul = tf.multiply(a, b)  # [4, 10, 18]
```

Linear Algebra Operations

Các phép toán đại số tuyến tính:

```
matrix_a = tf.constant([[1, 2], [3, 4]])
matrix_b = tf.constant([[5, 6], [7, 8]])

# Nhân ma trận
matmul = tf.matmul(matrix_a, matrix_b)

# Chuyển vị
transpose = tf.transpose(matrix_a)

# Nghịch đảo (nếu khả nghịch)
inverse = tf.linalg.inv(matrix_a)
```

Phần Bài Tập

Bài 1: Khởi tạo Tensor

Tạo các tensor sau:

- Tensor 3x4 chứa toàn giá trị 5
- Tensor 2x2x2 với giá trị ngẫu nhiên từ phân phối đều trong khoảng [0, 1)
- Tensor đường chéo chính từ list [1, 2, 3, 4]

Bài 2: Thao tác với Tensor

Cho tensor:

```
tensor = tf.constant([[1, 2, 3, 4],
                      [5, 6, 7, 8],
                      [9, 10, 11, 12]])
```

Thực hiện:

- Lấy hàng đầu và hàng cuối
- Lấy cột thứ 2
- Lấy ma trận con 2x2 từ góc trên bên phải
- Thay đổi shape thành 4x3

Bài 3: Operation

Cho 2 tensor:

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.constant([[5, 6], [7, 8]])
```

Tính:

- $2a + b^2$ (element-wise)
- Tích vô hướng của a và b
- Định thức của a
- Chuyển vị của b

Bài 4: Slicing

Giả sử chúng ta có tensor 3D biểu diễn ảnh RGB (shape: [height, width, 3]):

```
image = tf.random.uniform([256, 256, 3], maxval=255, dtype=tf.int32)
```

- Tách thành 3 tensor riêng biệt cho các kênh R, G, B
- Tính giá trị trung bình của từng kênh màu
- Chuyển ảnh sang grayscale bằng công thức: $0.299R + 0.587G + 0.114B$
- Cắt ảnh thành 4 ảnh nhỏ bằng nhau (mỗi ảnh 128x128)

Day 34 - Tensorflow Basic (2)

Đăng-Nhã Nguyễn

Lý thuyết

Hãy thực hiện chạy lại các ví dụ sau và quan sát output.

1. Các hàm TensorFlow phổ biến

a. `reshape`, `expand_dims`, `stack`, `concat`

```
import tensorflow as tf

a = tf.constant([1, 2, 3, 4, 5, 6])
reshaped = tf.reshape(a, (2, 3))  # [[1, 2, 3], [4, 5, 6]]

b = tf.expand_dims(reshaped, axis=0)  # shape: (1, 2, 3)

x = tf.constant([[1, 2]])
y = tf.constant([[3, 4]])
stacked = tf.stack([x, y], axis=0)  # shape: (2, 1, 2)

concatenated = tf.concat([x, y], axis=0)  # shape: (2, 2)
```

2. Ragged Tensors

- Dữ liệu không đều: mỗi hàng có độ dài khác nhau.

```
rt = tf.ragged.constant([[1, 2, 3], [4], [], [5, 6]])
print(rt[0])  # [1, 2, 3]
print(rt[3])  # [5, 6]
```

3. Sparse Tensors

- Dùng cho ma trận thưa (nhiều phần tử 0).

```
sparse = tf.sparse.SparseTensor(
    indices=[[0, 2], [2, 0]],
    values=[3, 4],
    dense_shape=[3, 3]
)
dense = tf.sparse.to_dense(sparse)
```

4. String Tensors

```
text = tf.constant("TensorFlow is great!")
words = tf.strings.split(text)  # ['TensorFlow', 'is', 'great!']
length = tf.strings.length(text)  # 20
replaced = tf.strings.regex_replace(text, "great", "awesome")
```

5. tf.Variable

```
var = tf.Variable([2.0, 4.0, 6.0])
var.assign_add([1.0, 1.0, 1.0])
var.assign([0.0, 0.0, 0.0])
```

Bài tập

Bài 1: Các phép biến đổi tensor cơ bản

a. Tạo một tensor 1D chứa 9 số nguyên. Reshape thành ma trận 3×3.

b. Dùng `expand_dims` để thêm chiều mới vào trước tiên.

c. Dùng `stack` hai tensor có cùng shape.

Bài 2: Làm việc với `concat` và `stack`

a. Tạo hai ma trận $A = [[1, 2], [3, 4]]$, $B = [[5, 6], [7, 8]]$.

b. Stack A và B theo trục mới.

c. Concatenate A và B theo chiều hàng và chiều cột.

Bài 3: Ragged Tensor thực tế

a. Tạo một ragged tensor chứa danh sách: $[[1, 2], [3], [], [4, 5, 6]]$.

b. Truy xuất phần tử thứ 2 và thứ 3.

c. Lặp qua từng hàng và in tổng giá trị.

Bài 4: Sparse Tensor

a. Tạo sparse tensor cho ma trận:

$$\begin{bmatrix} 0 & 0 & 10 \\ 0 & 0 & 0 \\ 5 & 0 & 0 \end{bmatrix}$$

b. Chuyển nó sang dạng dense.

c. Truy xuất phần tử ở (0,2) và (2,0).

Bài 5: Chuỗi và Biến

a. Tạo string tensor: `"Machine learning is fun!"`

b. Tách chuỗi thành các từ.

c. Thay từ `"fun"` thành `"powerful"`.

d. Tạo một `tf.Variable` khởi tạo bằng `[1.0, 2.0, 3.0]`, cộng thêm 2.0 cho mỗi phần tử.

Day 35 - Numpy, PyTorch & Tensorflow

Đăng-Nhã Nguyễn

Lý thuyết - So sánh cú pháp NumPy, PyTorch, TensorFlow

Hãy chạy lại các ví dụ dưới đây, đồng thời thử các phương thức khác của cả 3 thư viện để so sánh sự khác nhau giữa chúng.

1. Tạo và kiểm tra mảng / tensor

```
# NumPy
import numpy as np
a_np = np.array([[1, 2], [3, 4]])
print(a_np.shape)  # (2, 2)
```

```
# PyTorch
import torch
a_pt = torch.tensor([[1, 2], [3, 4]])
print(a_pt.shape)  # torch.Size([2, 2])
```

```
# TensorFlow
import tensorflow as tf
a_tf = tf.constant([[1, 2], [3, 4]])
print(a_tf.shape)  # (2, 2)
```

2. Broadcasting

Broadcasting là kỹ thuật cho phép áp dụng các phép tính toán trên các tensor (mảng đa chiều) không có cùng shape (số chiều).

```
# NumPy
a = np.array([1, 2, 3])
b = np.array([[10], [20], [30]])
print(a + b)
```

```
# PyTorch
a = torch.tensor([1, 2, 3])
b = torch.tensor([[10], [20], [30]])
print(a + b)
```

```
# TensorFlow
a = tf.constant([1, 2, 3])
b = tf.constant([[10], [20], [30]])
print(a + b)
```

3. Nhân ma trận

Cú pháp nhân ma trận giữa 2 tensor/mảng:

```
# NumPy
A = np.array([[1, 2], [3, 4]])
B = np.array([[5], [6]])
print(A @ B)
```

```
# PyTorch
A = torch.tensor([[1, 2], [3, 4]])
B = torch.tensor([[5], [6]])
print(A @ B)
```

```
# TensorFlow
A = tf.constant([[1, 2], [3, 4]])
B = tf.constant([[5], [6]])
print(tf.matmul(A, B))
```

4. Indexing (Truy xuất phần tử)

Truy xuất phần tử tại dòng 2, cột 1 (tức [1, 0]):

```
# Hãy khởi tạo tensor, array, constant bằng 1 mảng 2 chiều duy nhất, sau đó thực hiện slicing như sau
# NumPy
print(a_np[1, 0])
```

```
# PyTorch
print(a_pt[1, 0])
```

```
# TensorFlow
print(a_tf[1, 0])
```

5. Tính gradient (PyTorch và TensorFlow)

So sánh cách tính đạo hàm hàm số $f(x) = x^2$ tại $x = 2$.

PyTorch:

```
x = torch.tensor(2.0, requires_grad=True)
y = x ** 2
y.backward()
print(x.grad)  # 4.0
```

TensorFlow:

```
x = tf.Variable(2.0)
with tf.GradientTape() as tape:
    y = x ** 2
grad = tape.gradient(y, x)
print(grad)  # 4.0
```

Bài tập

Bài 1: Tạo tensor cơ bản

Tạo mảng sau trong NumPy, PyTorch và TensorFlow:

$$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$$

- In hình dạng (shape)
- Kiểu dữ liệu (dtype)
- GLấy ra giá trị tại dòng 2, cột 3 (indexing)

Bài 2: Truy xuất và thao tác cơ bản

Tạo ma trận 3x3 chứa các số từ 1 đến 9. Sau đó:

- Lấy phần tử dòng 2, cột 1.
- Tính tổng toàn bộ phần tử.
- Cộng thêm 5 cho toàn bộ mảng.

Bài 3: Nhân ma trận

Cho:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Thực hiện phép nhân $A \times B$ bằng NumPy, PyTorch và TensorFlow. In kết quả và hình dạng đầu ra.

Bài 4: Tính đạo hàm

Cho $x = 3.0$. Tính đạo hàm của:

$$f(x) = x^3 + 2x$$

bằng PyTorch và TensorFlow. So sánh cú pháp và kết quả.