python

HomeWork

AI

1. فایل تمرین را در پنل خود آپلود کنید.
2. title فایل تمرین به صورت (نام تمرین+نام و نام خانوادگی) به انگلیسی باشد.
3. در صورتی که سوال و یا ابهامی دارید در گروه چت تلگرامی بپرسید.

۱. سیستم توصیه‌گر (Mini Recommendation System)

هدف: تقویت مفهوم توابع، پارامترها، شرط‌ها، حلقه‌ها، ساختار داده‌ای (list, dict) و طراحی ماژولار با یک مسأله‌ی ساده و مرتبط با AI (توصیه فیلم بر پایه شباهت کاربرها).

مهارت‌ها:

- تابع‌نویسی و آرگومان‌ها
- شرط‌ها و حلقه‌های تو در تو
- کار با دیکشنری و لیست

- خواندن/نوشتن فایل JSON

شرح وظایف:

موارد زیر را در یک فایل به نام recommender.py پیاده‌سازی کنید:

مرحله ۱: بارگذاری داده‌های کاربر

اگر فایل users.json وجود داشت آن را بخوانید؛ در غیر اینصورت از یک دیکشنری پیش‌فرض استفاده کنید مانند:

```
users = {
    "Ali": ["Inception", "Matrix"],
    "Sara": ["Titanic", "Inception"]
}
```

مرحله ۲: ورود نام کاربر (CLI / input):

از کاربر بخواهید نامش را وارد کند:

```
name = input("Enter your name: ").strip()
```

مرحله ۳: اگر کاربر جدید بود

به او اجازه دهید 2–3 فیلم محبوبش را وارد کند (هر فیلم با کاما جدا شود یا یکی‌یکی پرسیده شود) و آن را به دیتاست اضافه کنید.

مرحله ۴: محاسبه کاربر مشابه(best match)

- برای هر کاربر موجود، تعداد اشتراک (shared movies) را با کاربر فعلی محاسبه کنید.

- کاربری که بیشترین تعداد اشتراک را دارد «best match» است (در صورت تساوی، یکی را انتخاب کنید؛ می‌توانید روی بیشترین اشتراک و سپس اسم کاربر مرتب کنید).

- اگر هیچ اشتراکی نبود، پیام مناسب نمایش دهید.

مرحله ۵: پیشنهاد یک فیلم

- از best matchیک فیلم را پیشنهاد دهید که کاربر فعلی هنوز ندیده — یعنی از لیست best_matchانتخاب شود **و** در لیست کاربر فعلی نباشد.

- اگر best_matchهیچ فیلمی برای پیشنهاد نداشت، پیام مناسب نمایش دهید.

مرحله ۶: ذخیره دیتاست

- اگر دیتا تغییر کرد (کاربر جدید اضافه شد)، فایل users.jsonرا به‌روزرسانی و ذخیره کنید

ورودی / خروجی مورد انتظار:

- ورودی: نام کاربر، در صورت نیاز فهرست فیلم‌ها (برای کاربران جدید)

- خروجی: متن خوش‌آمدگویی، نام کاربر مشابه، تعداد اشتراک، فیلم پیشنهادی، و آپدیت فایل users.json

نکات پیاده‌سازی:

- پیاده‌سازی تابع similarity(user_a, user_b) -> intکه تعداد فیلم‌های مشترک را برمی‌گرداند.

- کد را به توابع منطقی تقسیم کنید:

```
load_users(), save_users(users), get_or_create_user(name),
find_best_match(name, users), recommend_for(name, users).
```

- از ساختار دیکشنری {str: List[str]} برای نگهداری کاربران استفاده کنید.
- هنگام مقایسهٔ نام فیلم‌ها، برای پاسخ‌پذیری بیشتر، می‌توانید مقایسه را case-insensitive انجام دهید (مثلاً با lower()).

موارد تحویل:

- recommender.py
- users.json

۲. شبیه‌ساز خط لوله AI ماژولار (Modular AI Pipeline Simulator)»

هدف: این تمرین با هدف تعمیق مفاهیم پیشرفته برنامه‌نویسی شیءگرا (OOP) طراحی شده است. شما یک خط لوله پردازش متن انعطاف‌پذیر و توسعه‌پذیر خواهید ساخت که در آن هر مرحله (Step) یک کامپوننت مستقل است. این تمرین بر مفاهیمی چون کلاس‌های پایه انتزاعی (Abstraction) و ارث‌بری (Inheritance) تمرکز دارد.

شرح وظایف:

شما باید کلاس‌های زیر را در فایلی به نام pipeline.py کامل کنید. طرح کلی کلاس‌ها و متدها را برای شما فراهم گردیده است. وظیفه شما، پیاده‌سازی منطق داخلی هر متد است.

۱. PipelineStep (کلاس پایه انتزاعی)

این کلاس از قبل برای شما تعریف شده است. این کلاس یک «قرارداد» است که تضمین می‌کند تمام مراحل خط لوله، متد process را خواهند داشت. نیازی به تغییر این کلاس ندارید.

```
# In pipeline.py
```

```
from abc import ABC, abstractmethod
from typing import Any, List

class PipelineStep(ABC):
    """
    An abstract base class representing a single step in a processing
pipeline.
    """
    @abstractmethod
    def process(self, data: Any) -> Any:
        """
        Processes the input data and returns the result.
        This method must be implemented by all concrete subclasses.
        """
        pass
```

۲. DataLoader (پیاده‌سازی مرحله اول)

این کلاس باید یک فایل متنی را بخواند و لیستی از خطوط آن را برگرداند.

وظیفه شما: منطق خواندن فایل و مدیریت خطاها (مانند FileNotFoundError) را در متد process پیاده‌سازی کنید.

```
# In pipeline.py

class DataLoader(PipelineStep):
    """
    Loads data from a text file, returning a list of lines.
    """
    def process(self, filepath: str) -> List[str]:
        """
        Reads a file from the given filepath and returns its lines as a
list of strings.
        Handles FileNotFoundError and other potential exceptions.
        """
        # TODO: Implement file reading logic here.
        # Use a try-except block to handle potential errors.
        # Remember to strip newline characters from each line.
        pass
```

۲. Preprocessor (مرحله پاک‌سازی متن):

این کلاس متن را پاک‌سازی می‌کند (تبدیل به حروف کوچک، حذف علائم نگارشی و فاصله‌های اضافی).

وظیفه شما: متد process را طوری پیاده‌سازی کنید که روی تک‌تک خطوط داده ورودی، عملیات پاک‌سازی را انجام دهد و لیست پاک‌شده را برگرداند.

```
# In pipeline.py

import re

class Preprocessor(PipelineStep):
    """
    Cleans a list of text strings by converting to lowercase, removing
punctuation,
    and stripping extra whitespace.
    """
    def __init__(self, punctuation_to_remove: str = r'[^\w\s]'):
        self.punctuation_pattern = re.compile(punctuation_to_remove)

    def process(self, data: List[str]) -> List[str]:
        """
        Applies cleaning steps to each string in the input list.
        """
        # TODO: Implement the cleaning logic.
        # For each text in the 'data' list, apply:
        # 1. Lowercasing
        # 2. Punctuation removal using self.punctuation_pattern
        # 3. Extra whitespace removal
        pass
```

۴. Analyzer (مرحله تحلیل پایه)

این کلاس آمار اولیه متن را محاسبه می‌کند.

وظیفه شما: منطق محاسبه آمار خواسته شده (تعداد کل خطوط، میانگین تعداد کلمات در هر خط، و تعداد کلمات منحصربه‌فرد) را در متد process پیاده‌سازی کنید. حواستان به حالت خاص فایل خالی (تقسیم بر صفر) باشد.

```
# In pipeline.py

class Analyzer(PipelineStep):
    """
    Analyzes the text data to compute basic statistics.
    """
    def process(self, data: List[str]) -> dict:
        """
        Calculates total lines, average words per line, and number of
unique words.
        """
        # TODO: Implement the analysis logic here.
        # Calculate the required statistics and return them in a
dictionary.
        # Handle the case where the input data is empty.
        # For unique words, using a set is a good approach.
        pass
```

نمونه خروجی این بخش میتواند به صورت زیر باشد:

```
{
  "total_lines": int,
  "avg_length": float,          # میانگین کلمات در هر خط
  "unique_words": int
}
```

۵. ReportGenerator (کلاس مسئول گزارش‌دهی)

این کلاس گزارش نهایی را در کنسول چاپ کرده یا در فایل ذخیره می‌کند.

وظیفه شما: متدهای print_to_console و save_to_file را برای نمایش و ذخیره‌سازی نتایج کامل کنید.

```
# In pipeline.py

class ReportGenerator:
    """
    Generates and outputs reports from the analysis statistics.
    """
    def print_to_console(self, stats: dict):
        """
        Prints the statistics in a formatted way to the console.
        """
        # TODO: Implement the console printing logic.
        # Loop through the stats dictionary and print each key-value
pair nicely.
        pass

    def save_to_file(self, stats: dict, filepath: str):
        """
        Saves the statistics in a formatted way to a text file.
        """
        # TODO: Implement the file writing logic.
        # Open the specified file and write the stats to it.
        pass
```

۶. AIPipeline (ارکستر خط لوله)

این کلاس مراحل مختلف را به ترتیب اجرا می‌کند.

وظیفه شما: منطق اجرای مراحل را در متد run پیاده‌سازی کنید. این متد باید خروجی هر مرحله را به عنوان ورودی به مرحله بعدی بدهد.

```
# In pipeline.py

class AIPipeline:
    """
    Orchestrates a series of pipeline steps to process data.
    """
```

```
    def __init__(self, *steps: PipelineStep):
        self.steps = steps

    def run(self, initial_input: Any) -> Any:
        """
        Executes all steps in the pipeline sequentially.
        """
        # TODO: Implement the pipeline execution logic.
        # You need to loop through self.steps and call the process()
method on each.
        # The output of one step becomes the input to the next.
        pass
```

فایل اجرایی نمونه (main.py)

برای تست کردن کد خود، یک فایل main.py بسازید و از الگوی زیر برای ساختن و اجرای خطوط لوله استفاده کنید.

```
# In main.py

from pipeline import (
    DataLoader,
    Preprocessor,
    Analyzer,
    ReportGenerator,
    AIPipeline
)

if __name__ == "__main__":
    # --- Component Definitions ---
    loader = DataLoader()
    preprocessor = Preprocessor()
    basic_analyzer = Analyzer()
    reporter = ReportGenerator()

    input_filepath = "sample_data.txt"

    # --- Pipeline  ---
    print("\n--- Running Pipeline ---")
    basic_pipeline = AIPipeline(loader, preprocessor, basic_analyzer)
    basic_results = basic_pipeline.run(input_filepath)
    reporter.print_to_console(basic_results)
```

تحویل

پروژه خود را با ساختار زیر تحویل دهید:

```
ai_pipeline_project/
│
├── pipeline.py         # Your completed implementation of the classes
├── main.py             # Script to run the pipelines
├── sample_data.txt      # A sample input file you created (at least 30
lines)
```