最終課題レポート

概要

本リポジトリでは、Python プログラミング課題に取り組む受講生が自分のコードを提出すると、自動でアドバイスを返す Web アプリケーションを実装しました。講座の宿題に対して "正解／不正解" だけではなく、どこを直せば良いかを学習者が考えられるようなヒントを提示することが目的です。

目的

- GCI 講座の宿題で、受講生がつまずいた箇所を素早く把握し、自力で改善できるよう支援する
- 添削者の負担を軽減しつつ、教育的なフィードバックを提供する
- チート行為を防止し、学習意欲を高める

システム構成

- **フロントエンド:** Next.js 15 + TypeScript
- **バックエンド:** FastAPI / SQLAlchemy
- **コード実行:** Docker コンテナによるサンドボックス
- **AI アドバイス生成:** LLM (Google GenAI) を利用
- **データベース:** SQLite

工夫した点

1. **安全なコード実行**
    - Docker コンテナを用いることで、提出コードによるシステムへの影響を最小化しました。
    - Python スクリプトだけでなく Jupyter Notebook 形式の提出にも対応し、ノートブックから Python コードを抽出して実行します。
2. **自動正解判定**
    - 正解コードと提出コードの出力を比較し、自動で正解かどうかを判定するようにしました。
    - コード内の`!pip install PACKAGE_NAME`というコードを自動で認識し、その外部ライブラリを自動でインストールすることで、外部ライブラリを用いたコードの正解かどうかもテストできるようにしました。
3. **答えを教えないアドバイス**
    - プロンプトで「正解を直接教えない」ことを明示し、学習者がヒントから自分で考えられるようにしています。
    - その他アドバイスのポイントを細かく指定し、また、正解かどうかの結果、課題の情報、提出者のコード等必要な情報を含めることで、より良い出力が得られるようにプロンプトを工夫しました。

    プロンプト

    ```
    【判定結果】
    {"正解です！素晴らしい！" if is_correct else "不正解です。"}
    ```

```
あなたは、Pythonプログラミングを学ぶ初学者をサポートする親切なAIアシスタントです。
以下の情報に基づいて、学習者が自分で間違いに気づき、解決できるようになるためのヒントやアドバイスを生成してください。

【重要】
- **絶対にコードの正解そのものを直接教えてはいけません。**
- 指摘は具体的かつ建設的に行い、学習者のモチベーションを維持するよう努めてください。
- 難しい専門用語は避け、分かりやすい言葉で説明してください。
- アドバイスは日本語でお願いします。
{"- 正解の場合は、コードの改善点や別の解き方などを提案してください。" if is_correct else ""}

【課題情報】
タイトル: {problem_title}
問題文:
{problem_description}

【学習者の提出コード】

{user_code}


【コードの実行結果】
標準出力:
{execution_stdout if execution_stdout else "なし"}
標準エラー:
{execution_stderr if execution_stderr else "なし"}

【アドバイスのポイント】
1.  **エラーがある場合:**
    - エラーメッセージ（{execution_stderr}）が何を意味するのか、考えられる原因は何かを優しく説明してください。
    - エラーが発生している箇所を特定するためのデバッグ方法（例: print文の挿入箇所など）を提案してください。
2.  **エラーがないが期待通りに動作しない場合（または改善点がある場合):**
    - コードのロジックで改善できる点や、より効率的な書き方があれば示唆してください。
    - 変数名やコメントの付け方など、読みやすいコードにするための一般的なアドバイスも適宜含めてください。
    -（もし正解コードが提供されていれば、それを直接見せるのではなく、学習者のコードとの違いからヒントを得られるような問いかけをしてください）
3.  **よくある間違いの指摘:**
    - 初学者が陥りやすい間違いのパターンに合致する場合は、それとなく教えてあげてください。
      （例: for文の範囲、インデックスエラー、無限ループの可能性など）
```

4. **CI/CD とテスト**
    - Docker イメージのビルドと API の動作を GitHub Actions で自動テストできるよう構成しています。
5. **コストと性能を両立させたモデル選定**

- モデルとしてはGemini2.0 Flashを採用しました。オープンソースのモデルも検討しましたが、日本語性能とコーディング性能を両立したものが少なく、またローカルで実装せずinference providerを使用すると無料枠を超えた場合に料金がかかってしまうので、無料枠が大きくかつ性能、コストも抑えたGoogle AI StudioのAPIを採用しました。

6. **管理画面の作成**

   - 運営が使用する課題の管理画面も作成し、タイトル、問題の説明、正解コード、標準入力（必要な場合）を入力することで、課題を作成することができるようにしました。jupyter notebookやpythonのコードをアップロードするだけで正解コードを登録できるようにしました。

使い方

1. `docker-compose up --build` を実行してフロントエンド (http://localhost:3000) とバックエンド API (http://localhost:8000) を起動します。
2. (http://localhost:3000)にアクセスします。
3. 受講生はブラウザ上のエディタでコードを提出すると、サンドボックスで実行され、結果をもとに AI がアドバイスを生成します。

課題提出画面の実際の挙動

正解です！

おめでとうございます！正しい答えです。

実行結果

正常に実行されました

31284ms

終了コード: 0

出力結果:

stdout

```
東京都
神奈川県
大阪府
愛知県
埼玉県
```

システムメッセージ:
コードの実行が完了しました

お手本の実行結果

正解コードの実行結果です

25976ms

出力結果:

stdout

```
東京都
神奈川県
大阪府
愛知県
埼玉県
```

解説・振り返り

正解です！さらなる理解のための解説です

正解おめでとうございます！素晴らしい出来です！
今回のコードは、データ分析の基本的な流れ（データの読み込み、加工、抽出、出力）をしっかりと捉えられていて、とてもよく書けていますね。
さらに理解を深めるために、いくつか改善点と別の解き方を提案させてください。

改善点:

ライブラリのインストール: `!pip install pandas` は、すでにpandasがインストールされている環境では不要な場合があります。課題文の指示ではありますが、毎回実行する必要があるわけではないことを覚えておくと良いでしょう。

別の解き方:

DataFrameを使わずに解く: 今回はpandasのDataFrameを使って解きましたが、DataFrameを使わずに、辞書とリストの操作だけで同じ結果を得ることも可能です。例えば、`zip`関数を使って都道府県名と人口を組み合わせたリストを作成し、`sorted`関数で人口の多い順にソートしてから上位5件を抽出する、という方法も考えられます。DataFrameを使わないことで、より基本的なデータ構造の理解を深めることができます。
例えばこんな感じです。

```
population_data = {
```

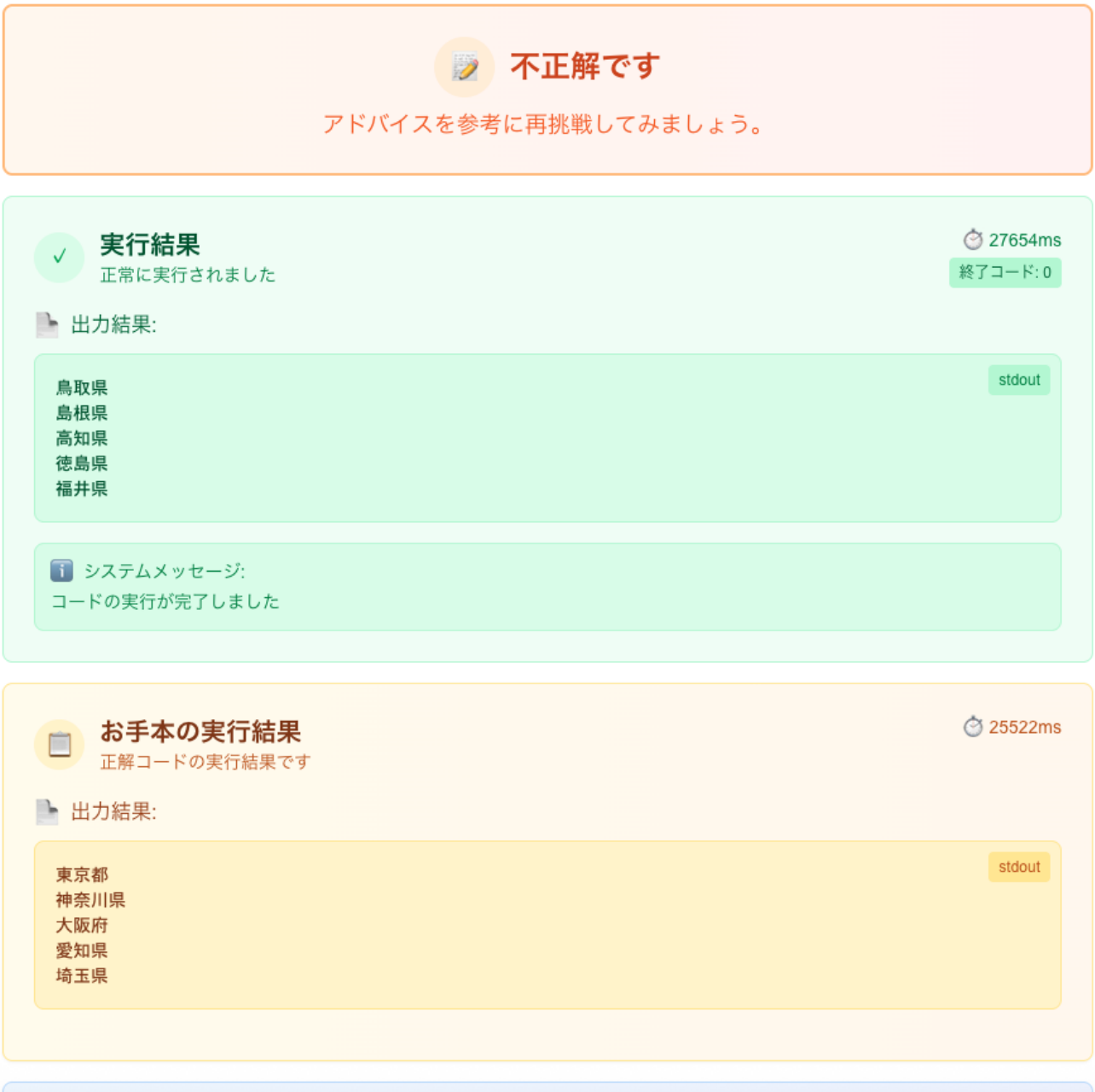
```
    '都道府県': ['北海道', '青森県', ...], # データは省略
    '人口': [5113334, 1204622, ...]
}

# 都道府県名と人口を組み合わせたリストを作成
prefecture_population = list(zip(population_data['都道府県'], population_data['人口']))

# 人口の多い順にソート
sorted_population = sorted(prefecture_population, key=lambda x: x[1], reverse=True)

# 上位5件を抽出して出力
for prefecture, population in sorted_population[:5]:
    print(prefecture)
```

この方法を試してみることで、DataFrameの便利さを改めて実感できるかもしれませんし、Pythonの基本的なデータ構造の理解が深まると思います。ぜひチャレンジしてみてください！



今後の課題

- Bedrock など複数のモデルを切り替えられるよう改良し、コスト試算を自動化する
- 本番運用に向けたセキュリティ設定の強化