Prerequisites

Ensure that you have the following installed and configured:

1. **Flask**: The web framework.
2. **Celery**: For asynchronous task processing.
3. **Redis**: Used as both the message broker and result backend for Celery.
4. **Python dependencies**: Install the required packages (e.g., `Flask`, `Celery`, `redis`) via `pip install -r requirements.txt`.

Base URL

All endpoints in the application are relative to the following base URL:

```
http://localhost:8080
```

Routes

---

1. `/get_all_projects`

- **Method(s)**: `GET`, `POST`
- **Description**: Returns a list of all projects stored in the base directory.
- **Response**:

```
{
  "tasks": ["project_id_1", "project_id_2", ...]
}
```

- **Error**:
  - 404 if the projects directory is not found.

Example Request:

```
curl http://localhost:8080/get_all_projects
```

---

2. `/create_new_project`

- **Method(s)**: `GET`, `POST`
- **Description**: Creates a new project directory with a unique ID. Returns the `project_id`.
- **Response**:

```
{
  "project_id": "new_project_id",
  "status": "success",
  "message": "Directory created at <path>"
}
```

Example Request:

```
curl -X POST http://localhost:8080/create_new_project
```

---

3. `/upload`

- **Method(s)**: `POST`
- **Description**: Uploads a file to a specific project. The uploaded file is saved as a raw CSV file and a Celery task for label encoding is triggered.
- **Request**: The request must include a `project_id` form field and a file form field (e.g., `file=@yourfile.csv`).
- **Response**:

```
{
  "message": "File encoding has started",
  "task_id": "task_id",
  "Project_id": "project_id"
}
```

Example Request:

```
curl -X POST http://localhost:8080/upload -F "project_id=test" -F "file=@sales_data.csv"
```

---

4. `/check-task/<task_id>`

- **Method(s)**: `GET`
- **Description**: Checks the status of a Celery task using the provided `task_id`. Returns the task's completion status.
- **Response**:
  - If the task is completed:

    ```
    {
      "status": "completed",
      "result": {"status": "Label encoding completed"}
    }
    ```

  - If the task is still running:

    ```
    {
      "status": "pending"
    }
    ```

Example Request:

```
curl http://localhost:8080/check-task/<task_id>
```

---

5. `/get_clusters`

- **Method(s)**: `POST`
- **Description**: Retrieves clusters based on the given project and path. Takes `project_id`, `path`, and `level` in the request body.
- **Request**:

  ```
  {
    "project_id": "project_id",
    "level": 3,
    "path": [1, 2, 1]
  }
  ```

- **Response**:

  ```
  {
    "full_path": "full/path/to/project",
    "project_id": "project_id",
    "clusters": ["cluster1", "cluster2", ...]
  }
  ```

Example Request:

```
curl -X POST http://localhost:8080/get_clusters \
-H "Content-Type: application/json" \
-d '{
    "project_id": "test",
    "level": 3,
    "path": [1, 2, 1]
}'
```

---

6. `/process`

- **Method(s)**: `POST`
- **Description**: Starts a sub-clustering process based on the provided `target_vars`, `target_var`, and `project_id`. It triggers an asynchronous Celery task.
- **Request**:

```
{
  "project_id": "project_id",
  "target_vars": ["var1", "var2"],
  "target_var": "target_variable",
  "level": 3,
  "path": [1, 2, 1]
}
```

- **Response**:

```
{
  "message": "File encoding has started",
  "task_id": "task_id",
  "Project_id": "project_id",
  "project_dir": "path/to/project/encoded_file.csv"
}
```

Example Request:

```
curl -X POST http://localhost:8080/process -H "Content-Type: application/json" \
-d '{
    "project_id": "test",
    "target_vars": ["reading_fee_paid", "Number_of_Months", "Coupon_Discount", "num_books"],
    "target_var": "amount_paid",
    "level": 0,
    "path": []
}'
```

General Notes:

- **Celery Background Tasks**: The application uses Celery to process tasks like label encoding (`async_label_encode_data`) and feature ranking (`async_optimised_feature_rank`) in the background. Celery stores the results of the tasks, which can be accessed using the task's unique ID.
- **Redis Configuration**: Ensure that Redis is running as it serves as the message broker and result backend for Celery.
- **Error Handling**: If any errors occur, they will be returned as a JSON response with a relevant error message and HTTP status code.

File Paths and Directories

The application works with directories that represent projects and their corresponding task results:

- Each **project** has its own directory structure.
- Files are uploaded under specific project directories and used for various processing tasks.
- The **task ID** returned by Celery is used to track the status of asynchronous tasks.

Running the Application

Make sure you have a Redis server running locally or on a remote server. Then, to start the Flask application with Celery:

1. Start Redis server:

   ```
   redis-server
   ```

2. Start Celery worker:

   ```
   celery -A celery_worker.celery worker --loglevel=info
   ```

3. Start Flask server:

   ```
   python your_flask_app.py
   ```