CSV Manager Project Report

Table of Contents

CSV Manager Project Report **1**

Introduction *2*

Application Features *2*

Core Features 2

System Architecture *3*

Components 3

Database Design 3

Processing Logic *3*

Uploading a CSV 3

Adding Data 4

Visualization 4

Key Design Decisions *4*

Limitations *4*

Conclusion *5*

Introduction

The CSV Manager is a full-stack Django application designed for managing, processing, and visualizing CSV files. The primary focus of the project is to showcase dynamic processing of data, user interaction with uploaded files, and real-time visualization. The project highlights the capability of handling CSV files generically, making it adaptable for different use cases.

The application is deployed at https://django-csv-app.onrender.com. However, since Render's free tier does not support Redis, the background processing functionality is unavailable in the live environment. Static pages, such as the home, remain accessible. For setup instructions and a detailed demo, visit the GitHub repository or download the demo video.

Application Features

Core Features

1. **Upload CSV Files**: Users can upload CSV files, which are stored in their original form for future processing.
2. **Background Processing**: CSV files are processed asynchronously to avoid timeouts for large files. During processing:
   - Headers are normalized by removing special characters, replacing spaces with underscores, and converting to lowercase.
   - The schema (data type inference) is determined for each column using a robust logic.
   - All integer and float columns are halved (a demonstration logic to showcase processing). This can be replaced with domain-specific logic.
3. **Add Data to CSV**: Users can append rows to a CSV. Changes are saved in a database table (CSVChanges) and processed asynchronously to create a derived CSV.
4. **Derived CSVs**: Each modification or processing step results in a derived version of the CSV. These are grouped under their parent uploaded CSV for better organization.
5. **Delete CSV**: Uploaded and derived CSVs can be deleted.
6. **Data Visualization**: The system supports both univariate and multivariate visualizations using Chart.js. While not all visualizations are meaningful, this feature demonstrates real-time chart rendering capabilities.
7. **Schema Inference**: Schema inference dynamically determines the data type of each column. Supported types include string, integer, float, date, and datetime.

System Architecture

Components

1. **Frontend**: Built using Django templates with Bootstrap for modern UI/UX.
2. **Backend**:
   - Django serves as the core framework for routing and data processing.
   - Celery handles asynchronous tasks like processing uploaded CSVs and applying changes.
   - Redis (for background task queuing) is required for full functionality.
3. **Database**:
   - SQLite is used as the primary database for simplicity.
   - Models:
     - UploadedCSV: Stores uploaded files and metadata.
     - DerivedCSV: Stores processed versions of uploaded files.
     - CSVChanges: Tracks modifications applied to CSVs.
4. **Visualization**:
   - Powered by Chart.js.
   - Provides options for bar charts, line charts, scatter plots, and pie charts.
   - Displays only the last 400 records for a better experience.

Database Design

- **UploadedCSV**:
  - Fields: name, content, schema, status, failure_reason, created_at.
  - The schema is stored as JSON and inferred upon upload.
  - Statuses: unprocessed, processing, processed, failed_processing.
- **DerivedCSV**:
  - Fields: parent (FK to UploadedCSV), content, created_at.
  - Each derived CSV links back to its parent for traceability.
- **CSVChanges**:
  - Fields: content_type, object_id, data, status, created_at.
  - Tracks changes applied to any CSV (uploaded or derived).

Processing Logic

Uploading a CSV

1. **File Validation**:
   - Ensures the file is a valid CSV.
   - Saves the raw content to the database.

2. **Background Processing**:
   - Normalizes headers.
   - Infers schema.
   - Applies the demo processing logic (divides integer and float columns by 2).
   - Saves the processed data as a derived CSV.

Adding Data

- Users append rows using dynamic input fields generated from the inferred schema.
- Changes are saved to the CSVChanges table.
- A background job processes these changes to avoid system timeouts.

Visualization

- Users can visualize data dynamically:
  - **Univariate**: Select a single column to visualize.
  - **Multivariate**: Select X and Y axes for scatter, line, or bar charts.
- While the system supports various charts, it's primarily designed to showcase real-time graph updates rather than domain-specific logic.

Key Design Decisions

1. **Generic Behavior**:
   - The system dynamically handles any CSV structure, making it adaptable for various datasets. (But primary testing is done on the air quality dataset)
2. **Asynchronous Processing**:
   - Background jobs ensure scalability and prevent UI blocking.
3. **Schema Inference**:
   - Uses Pandas for robust type inference.
   - Handles edge cases (e.g., mixed data types in columns).
4. **Versioning**:
   - Derived CSVs ensure original data integrity while supporting modifications.

Limitations

- **Redis Dependency**:
  - The live deployment lacks Redis support, disabling background jobs.
- **Visualization Limitations**:
  - Some combinations of charts and data don't produce meaningful results.
  - Visualization is focused on demonstrating graphing capabilities.

Conclusion

The CSV Manager demonstrates a versatile, modular, and scalable approach to handling CSV files. While it uses simple logic (e.g., dividing values by 2) for demonstration, the architecture is flexible enough to adapt to complex processing workflows. For setup instructions, refer to the GitHub repository.