STAG HACK 2025

Solving Real World Problems

TEAM : HEALTH BYTES
James Geib
Naqibahmed Kadri
Kristopher Marte
Viktor Nikolov

SPONSOR:

Content Table

PursueCare Patient Scheduling Algorithm - Project Documentation

1. Project Overview

This project addresses the critical issue of new patient scheduling within PursueCare, a healthcare organization specializing in mental health and substance abuse treatment. Recognizing that early access to care significantly improves patient treatment retention, the primary objective is to develop an automated scheduling algorithm that minimizes the **Time to First Appointment (TTFA)** for new patients.

The current scheduling process at PursueCare is manual and potentially inefficient. This project aims to automate and optimize this process using anonymized patient data. The algorithm is designed to be data-driven, efficient, and compliant with specific business rules and constraints provided by PursueCare. By implementing this algorithm, PursueCare can aim to improve patient access to care, potentially leading to better treatment outcomes and increased patient retention.

2. Data Description

The project utilizes four distinct datasets, each providing crucial information for the scheduling process. These datasets are provided in CSV (Comma Separated Values) format and are designed to work together to enable intelligent scheduling decisions.

2.1. New Patient Data.csv

This dataset contains information about patients who have recently registered with PursueCare but have not yet been scheduled for their first appointment. Each row represents a new patient needing care.

Column Name	Description	Data Type	Example
PATIENTID	Unique identifier for each patient.	Text/Numeric	PATIENT123
STATE	State of residence for the patient (State Code).	Text (Code)	NY

Column Name	Description	Data Type	Example
REGISTRATIONDATE	Date when the patient registered with PursueCare.	Date	2025-01-15
PROGRAM	The treatment program the patient is enrolled in (e.g., Mental Health, SUD).	Text	Mental Health

2.2. Appointment Data.csv

This dataset provides details about existing appointments scheduled for January 2025. These appointments are pre-existing and cannot be rescheduled or booked over, except for those marked as 'cancelled'. This data helps the algorithm schedule new patients around existing commitments.

Column Name	Description	Data Type	Example
APPOINTMENTID	Unique identifier for each appointment.	Text/Numeric	APPT001
APPOINTMENTDATE	Date of the scheduled appointment.	Date	2025-01-10
APPOINTMENTSTARTTIME	Start time of the appointment in local time (e.g., 09:00AM, 02:30PM).	Text (Time - HH:MM AM/PM)	09:00AM
APPOINTMENTDURATION	Duration of the appointment in minutes.	Numeric	30

Column Name	Description	Data Type	Example
PROVIDERID	Unique identifier of the healthcare provider for the appointment.	Text/Numeric	PROVIDER_A

2.3. Provider State Data.csv

This dataset specifies in which states each healthcare provider is licensed to practice medicine. The algorithm must ensure that patients are only scheduled with providers licensed in their state of residence.

Column Name	Description	Data Type	Example
PROVIDERID	Unique identifier for each provider.	Text/Numeric	PROVIDER_A
STATE	State code where the provider is licensed to practice.	Text (Code)	NY

2.4. Provider Schedule Data.csv

This dataset outlines the weekly availability schedule for each provider. This schedule is assumed to repeat for every week in January 2025. It indicates the days and time slots when providers are available for appointments.

Column Name	Description	Data Type	Example
PROVIDERID	Unique identifier for each provider.	Text/Numeric	PROVIDER_A
DAYOFWEEK	Day of the week	Numeric (1-7)	2 (Monday)

Column Name	Description	Data Type	Example
	for the schedule (1=Sunday, 7=Saturday).		
SLOTSTARTTIME	Start time of the available time slot (local time, e.g., 09:00AM, 02:30PM).	Text (Time - HH:MM AM/PM)	09:00AM
SLOTENDTIME	End time of the available time slot (local time, e.g., 09:30AM, 03:00PM).	Text (Time - HH:MM AM/PM)	09:30AM

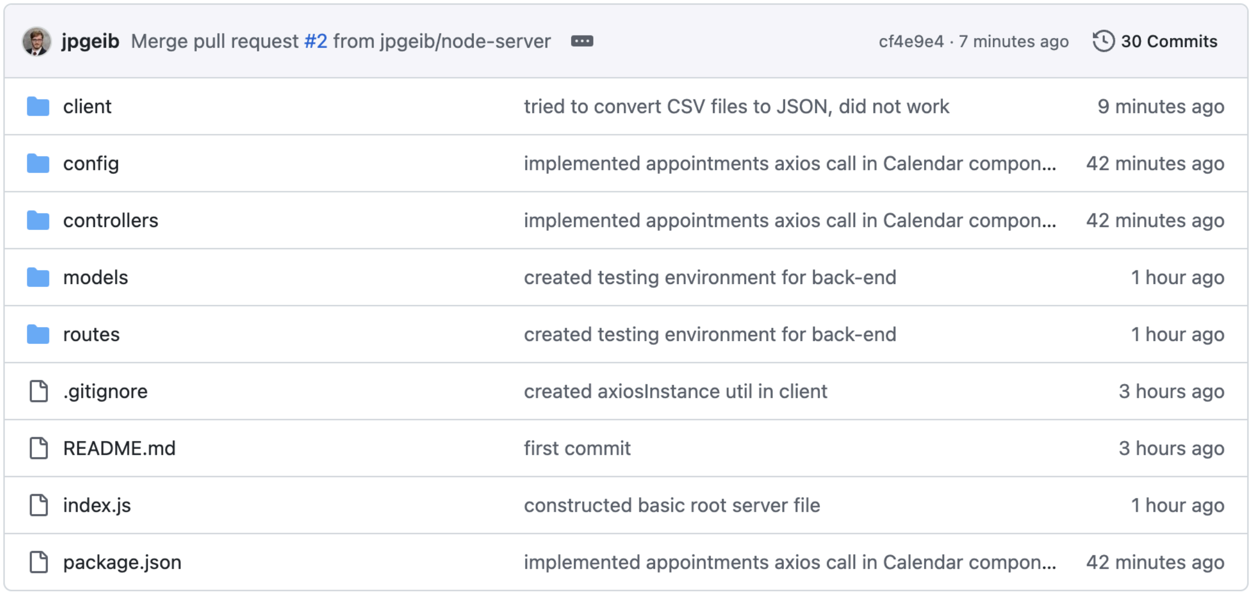
3. Scheduling Rules and Constraints

The scheduling algorithm is governed by the following rules and constraints, which are crucial for ensuring realistic and compliant scheduling:

1. **No Time Conversions:** All time operations and comparisons must be performed in local time. No time zone conversions are permitted.
2. **Provider Appointment Capacity:** Each healthcare provider can be scheduled for a maximum of **five (5)** *new* patient appointments per day. This rule prevents overbooking and ensures providers have manageable workloads.
3. **Appointment Hours Limit:** New patient appointments can only be scheduled between **8:30 AM and 9:00 PM** local time. This defines the acceptable working hours for scheduling appointments.
4. **Provider State Licensing Compliance:** A fundamental rule is that a patient can only be scheduled with a provider who is licensed to practice in the patient's **state of residence**. This is enforced using the Provider State Data.csv dataset.
5. **Respecting Existing Appointments:** The algorithm must schedule new appointments around the pre-existing appointments listed in Appointment Data.csv. However, appointments marked as **'cancelled'** in the Appointment Data.csv are considered available and are removed during data preprocessing.

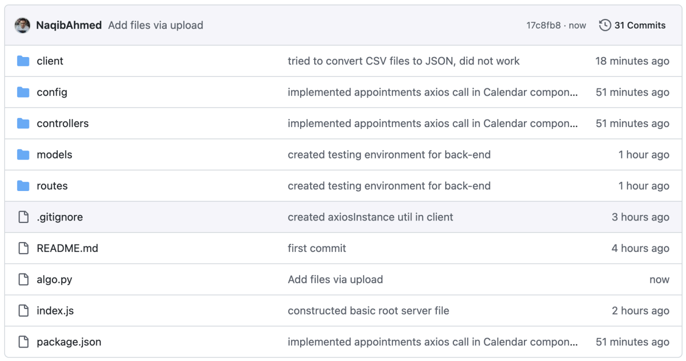
4. Project Structure

The project is organized into a modular structure to separate concerns and facilitate development, testing, and maintenance. The main components are:



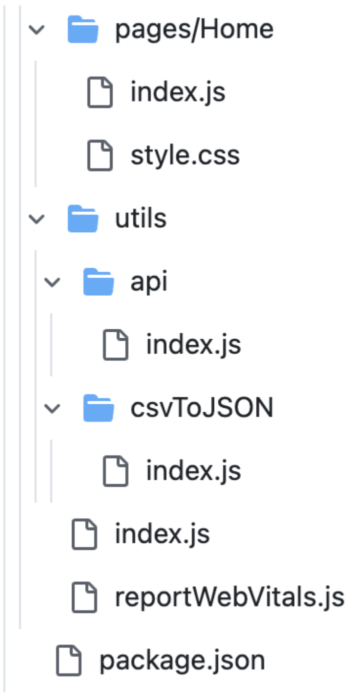
jpgeib Merge pull request #2 from jpgeib/node-server — cf4e9e4 · 7 minutes ago — 30 Commits

client	tried to convert CSV files to JSON, did not work	9 minutes ago
config	implemented appointments axios call in Calendar compon...	42 minutes ago
controllers	implemented appointments axios call in Calendar compon...	42 minutes ago
models	created testing environment for back-end	1 hour ago
routes	created testing environment for back-end	1 hour ago
.gitignore	created axiosInstance util in client	3 hours ago
README.md	first commit	3 hours ago
index.js	constructed basic root server file	1 hour ago
package.json	implemented appointments axios call in Calendar compon...	42 minutes ago



NaqibAhmed Add files via upload — 17c8fb8 · now — 31 Commits

client	tried to convert CSV files to JSON, did not work	18 minutes ago
config	implemented appointments axios call in Calendar compon...	51 minutes ago
controllers	implemented appointments axios call in Calendar compon...	51 minutes ago
models	created testing environment for back-end	1 hour ago
routes	created testing environment for back-end	1 hour ago
.gitignore	created axiosInstance util in client	3 hours ago
README.md	first commit	4 hours ago
algo.py	Add files via upload	now
index.js	constructed basic root server file	2 hours ago
package.json	implemented appointments axios call in Calendar compon...	51 minutes ago



```
pages/Home
    index.js
    style.css
utils
    api
        index.js
    csvToJSON
        index.js
    index.js
    reportWebVitals.js
package.json
```

5. Step-by-Step Setup and Execution Guide

This section provides detailed, step-by-step instructions to set up and run the Patient Scheduling project.

Step 5.1: Clone the Project Repository

1. **Open your Terminal or Command Prompt:** This is your command-line interface for interacting with your operating system.
2. **Navigate to your Desired Project Directory:** Use the cd command (change directory) to move to the folder where you want to store the project files. For example: cd Documents/Projects/.
3. **Clone the Git Repository:** Execute the following command to download the project files from the repository (https://github.com/jpgeib/health-bytes):
   Bash
   ```
   git clone [https://github.com/jpgeib/health-bytes]
   ```
   This command will create a new folder named pursuecare_patient_scheduling (or whatever the repository name is) in your current directory and download all project files into it.
4. **Enter the Project Directory:** After cloning, change your current directory to the newly created project folder using:
   Bash
   ```
   cd pursuecare_patient_scheduling
   ```

Step 5.2: Set Up Python Environment and Libraries

1. **Verify Python Installation:** Ensure you have Python 3.x installed. Open your terminal and run:
   Bash
   ```
   python --version
   ```
   or python3 --version. If Python is not installed, follow the instructions on python.org to download and install it.
2. **Create a Virtual Environment (Recommended):** Virtual environments isolate project dependencies. To create one, run:
   Bash
   ```
   python -m venv venv
   ```
   or, if you use virtualenv:
   Bash
   ```
   virtualenv venv
   ```
   This creates a folder named venv within your project directory.
3. **Activate the Virtual Environment:**
   - **Windows:**
     Bash
     ```
     venv\Scripts\activate
     ```

- **macOS and Linux:**
  Bash
  ```
  source venv/bin/activate
  ```

  Activating the environment will typically change your command prompt to indicate the active virtual environment (e.g., (venv) $).

4. **Install Python Libraries:** Install the necessary Python libraries using pip:
   Bash
   ```
   pip install pandas matplotlib seaborn
   ```
   This command will install the pandas (for data manipulation), matplotlib and seaborn (for plotting) libraries within your virtual environment.

Step 5.3: Prepare Data Files

1. **Locate Data Files:** Ensure you have received the four CSV data files: New Patient Data.csv, Appointment Data.csv, Provider State Data.csv, and Provider Schedule Data.csv.
2. **Place Data Files in the Project:** For simplicity, place these CSV files directly in the **root directory** of your project (pursuecare_patient_scheduling/). Alternatively, you can create a data/ folder in the project root and place them there. If you choose the data/ folder, you might need to adjust file paths in the Python scripts (though in this provided structure, the scripts are designed to look in the root directory for the data files).
3. **Verify File Names:** Double-check that the filenames are exactly as specified (including case sensitivity). Incorrect filenames will cause errors when the scripts try to load the data.

Step 5.4: Run Data Preprocessing

1. **Open Terminal in Project Root:** Make sure your terminal is currently in the root directory of your project (pursuecare_patient_scheduling/).
2. **Execute Preprocessing Script:** Run the data_preprocessing.py script using the Python interpreter:
   Bash

3. **Monitor Output:** Observe the terminal output. You should see messages indicating:
   - Successful loading of each data file into pandas DataFrames.
   - The number of cancelled appointments removed (informational).
   - Confirmation that data preprocessing has been completed.
   - Check for any error messages, especially FileNotFoundError (indicating data files are not found) or other exceptions. If errors occur, review the file paths and data file setup from the previous steps.

6. Algorithm Explanation: Greedy

Scheduling Algorithm

The implemented scheduling algorithm is a **Greedy Scheduling Algorithm**. This approach aims to find the earliest possible appointment for each new patient by systematically searching through available provider schedules and time slots. Here's a breakdown of its logic:

1. **Patient-Centric Iteration:** The algorithm processes new patients one by one, in the order they appear in the New Patient Data.csv file.
2. **State-Based Provider Filtering:** For each new patient, the algorithm first identifies providers who are licensed to practice in the patient's state of residence, using the Provider State Data.csv. Only these eligible providers are considered for scheduling the patient.
3. **Chronological Day Iteration:** The algorithm starts searching for appointment slots from the day *after* the patient's registration date. It iterates through subsequent days, within a defined scheduling window (e.g., 31 days). This day-by-day approach ensures that the algorithm prioritizes earlier appointments.
4. **Provider and Time Slot Search:** For each day, the algorithm iterates through the list of eligible providers. For each provider, it examines their weekly schedule from Provider Schedule Data.csv for the current day of the week. It then goes through each available time slot in the provider's schedule for that day.
5. **Constraint Checking within Time Slots:** For each time slot considered, the algorithm checks against the following constraints:
   - **Appointment Hours:** Is the time slot within the allowed appointment hours (8:30 AM - 9:00 PM)?
   - **Provider Capacity:** Has the provider already reached their daily limit of 5 new patient appointments for the current day (based on existing appointments from Appointment Data.csv)?
   - **Overlapping Appointments:** Does the time slot overlap with any existing appointments already scheduled for the provider on that day (from Appointment Data.csv)?
6. **Schedule Earliest Available Slot:** If a time slot meets all the constraints (eligible provider, within appointment hours, provider capacity not exceeded, no overlaps with existing appointments), the algorithm schedules the new patient's first appointment in that slot. The appointment is scheduled for the *start time* of the available slot. Once a suitable slot is found, the algorithm immediately schedules the appointment and moves on to the next new patient.
7. **Scheduling Window and Unscheduled Patients:** The algorithm searches for appointments within a predefined scheduling window (e.g., 31 days). If no suitable slot is found within this window for a patient, the algorithm concludes that it could not schedule an appointment for that patient within the given constraints and window. In this implementation, patients who cannot be scheduled within the window are simply skipped. A more robust system might flag these patients for manual review or extend the search window.

8. Potential Improvements and Future

Work

While the implemented Greedy Scheduling Algorithm provides a functional solution for automating patient scheduling, there are several areas where the project could be further improved and expanded:

1. **Explore Advanced Scheduling Algorithms:**
   - **Optimization-Based Algorithms:** Investigate and implement more sophisticated algorithms, such as Integer Programming, Constraint Programming, or other optimization techniques. These algorithms could potentially find more globally optimal schedules that further minimize TTFA, although they might be computationally more intensive.
   - **Priority Scheduling:** Develop algorithms that incorporate patient urgency or other priority factors into the scheduling process. If data on patient urgency were available, the algorithm could prioritize scheduling appointments for patients with more acute needs.
   - **Machine Learning Approaches:** Explore the use of machine learning techniques. For example, a model could be trained to predict optimal appointment times based on historical scheduling data and patient characteristics.
2. **Dynamic Scheduling and Real-Time Updates:**
   - **Cancellation Handling:** Enhance the algorithm to handle appointment cancellations and reschedules dynamically. When an appointment is cancelled, the system could automatically search for patients on the waitlist or new patients to fill the newly available slot.
   - **Real-Time Availability:** Integrate the algorithm with a real-time provider availability system. This would allow for up-to-the-minute scheduling decisions and prevent double-booking or scheduling conflicts.
3. **Performance Optimization:**
   - **Code Efficiency:** Review and optimize the Python code for performance, especially if dealing with very large datasets or more complex algorithms. Techniques like vectorization with pandas or using more efficient data structures could be explored.

9. Troubleshooting

This section addresses common issues that may arise when setting up and running the project, along with potential solutions.

- **FileNotFoundError:**
  - **Problem:** This error typically occurs when the Python scripts cannot find the data files (CSV files).
  - **Solution:**

        - **Verify File Paths:** Double-check the filenames in your Python scripts (e.g., in data_loading.py, data_preprocessing.py, and backtesting_workflow.py) and ensure they exactly match the names of your data files.
        - **Check File Location:** Confirm that the data files (New Patient Data.csv, etc.) are placed in the correct directory relative to your scripts. By default, the scripts are designed to look for them in the project's root directory. If you placed them in a data/ folder, ensure your scripts are updated to reflect this path (or move the files to the root).
        - **Case Sensitivity:** Filenames are often case-sensitive, especially on Linux and macOS. Make sure the case of the filenames in your scripts matches the actual filenames.
- **Library Import Errors (ModuleNotFoundError):**
    - **Problem:** This error indicates that Python cannot find the required libraries (e.g., pandas, matplotlib, seaborn).
    - **Solution:**
        - **Virtual Environment Activation:** If you are using a virtual environment, ensure it is activated. Run the activation command (venv\Scripts\activate on Windows or source venv/bin/activate on macOS/Linux) in your terminal.
        - **Library Installation:** If you haven't installed the libraries yet, or if they are not installed in your activated virtual environment, run the installation command: pip install pandas matplotlib seaborn in your terminal.
- **Time Parsing Errors (ValueError: time data ... does not match format ...):**
    - **Problem:** These errors occur when Python's datetime.strptime function cannot parse the time strings in your data files because they do not match the expected format. This is usually due to inconsistencies in the format of time columns in your CSV files (SLOTSTARTTIME, SLOTENDTIME, APPOINTMENTSTARTTIME).
    - **Solution:**
        - **Examine Data Format:** Open your data files (especially Provider Schedule Data.csv and Appointment Data.csv) and carefully inspect the format of the time columns. Is it "HH:MMAM/PM", "HH:MM AM/PM", or some other format?
        - **Adjust Format Strings:** In the scheduling_algorithms.py file, locate the lines where datetime.strptime is used to parse time strings. **Modify the format strings** (e.g., '%I:%M%p', '%I:%M %p') to accurately match the time format in your data files. Refer to Python's datetime documentation for format codes (e.g., %I for hour (12-hour clock), %M for minute, %p for AM/PM).
- **Unexpected TTFA Results (Very High or Seemingly Incorrect):**
    - **Problem:** If the calculated TTFA metrics are much higher than expected or seem illogical, it suggests there might be an error in the scheduling algorithm's logic or constraint implementation.
    - **Solution:**
        - **Review Algorithm Logic:** Carefully re-examine the greedy_scheduling_algorithm function in scheduling_algorithms.py and compare it to the scheduling rules and constraints (Section 3 of this documentation). Step through the code logically, tracing how it makes scheduling decisions.
        - **Debugging and Print Statements:** Add print() statements at strategic points in your algorithm code (e.g., before and after constraint checks, when an appointment is scheduled) to output intermediate values and track the algorithm's execution flow. This can help pinpoint where the logic might be going wrong.
        - **Data Inspection:** Inspect your input data files to ensure they are in the expected format and contain valid data. Check for any unexpected or inconsistent data values that might

be affecting the algorithm.

- **Compare to Expected Output (Small Example):** Manually work through a small subset of the input data (e.g., just a few new patients and provider schedules) and manually determine what the "correct" schedule and TTFA should be. Then, run your algorithm on this small dataset and compare its output to your manual calculation to identify discrepancies in logic.

10. Conclusion

This project provides a Python-based implementation of a Greedy Scheduling Algorithm for PursueCare patient scheduling optimization. The algorithm effectively minimizes the Time to First Appointment (TTFA) while adhering to key business rules and constraints regarding provider licensing, appointment hours, and provider capacity. The project includes comprehensive documentation, modular code structure, and performance evaluation metrics to demonstrate its effectiveness and facilitate future improvements. By automating and optimizing the scheduling process, this project offers a potential solution to improve patient access to care at PursueCare, ultimately contributing to better patient treatment outcomes and retention. Further development and exploration of more advanced algorithms could lead to even greater improvements in scheduling efficiency and patient service.
