**Our Data Engineering Journey: A Summary (Updated)**

Here's a revised chronological overview of our development process, highlighting the key steps, code changes, and reasoning behind them:

**1. Initial Exploration and Prototyping (Jupyter Notebook):**

* **Goal:** To understand the Recreation.gov API, fetch data, and perform basic transformations.
* **Approach:** Started with a Jupyter Notebook (recGovDFbuild.ipynb) for interactive exploration. This allowed for:
  + Quick API calls using the requests library.
  + Immediate inspection of data as DataFrames using pandas.
  + Iterative development of data transformation logic.
* **Initial Code:**
  + Basic functions fetch\_ridb\_data to pull data and create\_dataframe to create pandas dataframes.
* **Output:** Basic dataframe information and previews of the data.

**2. Transition to Python Scripts:**

* **Goal:** To move from a more flexible jupyter notebook, to more robust and structured production-ready python scripts.
* **Approach:** We moved from a single jupyter notebook file to the following two python scripts: fetch\_and\_save\_data.py to pull data and save it as a JSON file, and create\_dataframes.py to read the local file and create the DataFrames.
* **Files created:**
  + fetch\_and\_save\_data.py: This script now handles all API calls for fetching raw data (facilities) *and* related data (activities, campsites, events), and saves this information in the facilities\_data.json file inside the json\_output directory.
  + create\_dataframes.py: This script loads the raw JSON data from the file and transforms it into multiple pandas DataFrames, outputting them to CSV files in the csv\_output directory.
* **Output:** The first script outputs a single JSON file, and the second one outputs various different CSV files and prints their data to the console.

**3. Key Problems and Fixes Implemented:**

* **API Key Management:**
  + **Problem:** Hardcoded API keys directly in the code, which was a security risk.
  + **Solution:** Moved the API key to a separate config/api\_keys.py file and set up the .gitignore file to make sure that file was not being tracked.
* **Generalizing the Fetch Method:**
  + **Problem:** The initial fetch\_ridb\_data was specific to facilities, making it difficult to reuse for activities, campsites etc.
  + **Solution:** Introduced an endpoint parameter to the function to allow for more dynamic fetching of multiple different API endpoints. We also added a states parameter to allow for multiple different states to be pulled.
* **Rate Limiting:**
  + **Problem:** Initially, the script made API calls too quickly, potentially getting blocked by the server.
  + **Solution:** Implemented a time.sleep(rate\_limit\_delay) call, and timeouts on the request.get() function, as well as a separate logic to limit the number of calls to the /facilities/{facilityId}/... endpoints.
* **Data Normalization and Duplication Issues:**
  + **Problem:** The initial dataset had repeating nested data structures, like PERMITTEDEQUIPMENT and ATTRIBUTES. Also the campsites dataframe was having duplicate CampsiteID's due to the way we were joining data together.
  + **Solution:** We added a permitted\_equipment\_data list and campsite\_attribute\_db list to flatten this data, we also created a dedicated permitted\_equipment\_df dataframe to remove duplicate CampsiteIDs. The nested PERMITTEDEQUIPMENT and ATTRIBUTES columns were dropped from the main campsite dataframe.
* **Data Loading**
  + **Problem:** The code was not efficiently using the JSON data and was making too many API calls, therefore was having issues with performance.
  + **Solution:** Refactored code to remove API calls from create\_dataframes.py and have it only deal with local files, then included API calls for related data in fetch\_and\_save\_data.py so that it fetches all data (including related data) at one time and outputs that to a single json file.
* **Improved Output:**
  + **Problem:** Initially, the output made it difficult to track progress, especially when pulling a lot of data.
  + **Solution:** Introduced a progress indicator with timestamps (... Time) while fetching the data to provide visual feedback that the script is running.
* **Timeouts:**
* **Problem:** The code was not handling timeouts during API calls.
* **Solution:** The code was updated to handle timeouts for the API calls.
* **File Output Path:**
  + **Problem:** Initially, the file output was saved relative to the location of the script, making it difficult to control where the files were saved.
  + **Solution:** We modified the code to create an output path relative to the root of your project using the syntax "../csv\_output" for the output folder.
* **Incorrect Looping**
  + **Problem** We had incorrect logic in our for loop that would cause the script to stop prematurely.
  + **Solution** We updated the logic to correctly iterate through the different states when pulling data from the API.
* **Unnecessary Imports:**
  + **Problem:** There were unneeded imports that were causing some confusion, such as importing the API key into the create\_dataframes.py code when it was not needed.
  + **Solution:** We removed the unnecessary import statements from the create\_dataframes.py code.
* **create\_dataframe handling empty data**:
  + **Problem:** The create\_dataframe function would return an error if no data was provided, which would interrupt processing.
  + **Solution:** The create\_dataframe function has been updated to now return an empty dataframe when there is no data, which will prevent these errors.
* **Incorrect process\_facilities\_data call**:
* **Problem:** The process\_facilities\_data function was being called with more parameters than it was expecting, which was causing an error.
* **Solution:** Corrected the function to call the process\_facilities\_data with the proper arguments.
* **Duplicate Campsite ID's**
  + **Problem**: The campsite ID was being duplicated by the code due to incorrect merging.
  + **Solution:** We removed the merge operations and now the Campsites table and the new PermittedEquipment, and CampsiteAttributes tables, exist as separate dataframes and can be loaded to separate tables.

**5. Current State:**

* **Data Extraction:** Your fetch\_and\_save\_data.py now reliably fetches all facility data with associated activities, campsites, and events from the Recreation.gov API for multiple states, saving the data to a local facilities\_data.json file. It is also more robust due to added error handling and logging, and it handles rate limiting much better than previous versions.
* **Data Transformation:** Your create\_dataframes.py loads the JSON and performs data transformation steps, creating separate DataFrames with properly structured data and relationships using a foreign key. It now handles nested data in a more efficient way by creating additional dataframes for permitted\_equipment, and campsite\_attributes.
* **Output Files:** The code outputs your data to local JSON files and CSV files for easy loading into a database.

**Key Takeaways:**

* Successfully separated data extraction and transformation into distinct scripts.
* Created a process to manage data normalization and can better structure your tables with your new data set.
* Have a more robust and reliable code that handles most errors and rate limiting and can now efficiently pull data from the API to disk.
* The codebase is now organized and has a very clean separation of concerns.