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

Here's the revised summary, reflecting our latest progress:

**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 (campsites, events), and saves this information to a facilities\_data.json file in the json\_output directory.
  + create\_dataframes.py: This script loads the raw JSON data from the file and transforms it into multiple pandas DataFrames and outputs 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 global time.sleep(rate\_limit\_delay) to introduce a delay between API calls to limit calls to the main API, and a separate counter and time delay within the fetch\_ridb\_related\_data to respect the related calls limits. We have also added timeouts for all API calls.
* **Data Normalization and Duplication Issues:**
  + **Problem:** The initial dataset had repeating, nested data structures, like PERMITTEDEQUIPMENT and ATTRIBUTES and would create duplicate campsite IDs if not handled correctly.
  + **Solution:** We created a separate dataframe for permitted\_equipment\_df and campsite\_attributes\_df to flatten this data and remove the nested columns in the main campsites\_df table.
* **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.
* **Incomplete Data**:
  + **Problem**: Not all of the necessary data was being returned from the API.
  + **Solution:** We included the parameter full=true so that the /facilities API endpoint now returns the complete dataset that is required.
* **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 and how long the processes were taking.
* **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" and "json\_output".
* **Incorrect Looping:**
  + **Problem:** The loops had an issue where they were not terminating correctly and the script could seem like it was getting stuck.
  + **Solution:** Corrected the loop logic to make sure they are now working correctly with a while loop inside of a for loop.
* **Unnecessary Imports:**
  + **Problem:** There were unneeded import statements in the create\_dataframes.py code that were not needed.
  + **Solution:** We removed all of these imports and removed any calls to functions that used those import statements.
* **Missing Imports:**
  + **Problem:** The code would error out because the import time was missing from a script and would throw a runtime error.
  + **Solution:** We added the necessary import to the script so that it would no longer throw that error.
* **Incorrect Data Checks**:
  + **Problem:** There were checks being done to make sure data existed before performing actions that were causing valid responses to be misinterpreted as errors and exiting the loop prematurely.
  + **Solution:** We updated the error checks so that they were more precise and specific, to ensure we were not skipping over valid records.
* **Missing Address Details:**
  + **Problem:** The initial API response lacked address details, even though an address field was present in the response.
  + **Solution:** We added logic to pull the address data directly from the nested list that is present in the facilities data. We also created a separate dataframe to maintain data normalization.
* **Duplicated CampsiteID:**
  + **Problem:** The original code would cause duplicate CampsiteIDs when merging the permittedequipment and attributes back into the main campsites\_df table.
  + **Solution:** The logic was updated to create a separate data frame and no longer append those values to the main dataset.

**5. Current State:**

* **Data Extraction:** Your fetch\_and\_save\_data.py now reliably fetches all facilities data with the related activities, campsites, and events from the Recreation.gov API for multiple states. It also includes the address, and saves this information to a JSON file. It is also more robust due to added error handling, rate limiting, and includes time stamps so that it is easy to determine how long the script is running.
* **Data Transformation:** Your create\_dataframes.py loads the JSON and processes that data, transforming it into a set of normalized DataFrames and outputs them to a series of CSV files. It also uses try, except and conditional logic to make sure that the program is running well even if data is missing from the responses.
* **Output Files:** The data is now structured with separate CSV files, all of which are outputted to the csv\_output folder, ready for ingestion into your SQL database.

**Key Takeaways:**

* You have built a modular and well-designed ETL process.
* You are capturing all of the necessary data and transforming it into a usable format.
* You have been iteratively designing and implementing your code and have become very proficient at data engineering.