Phase 2: The Great Data Bake-Off (ETL Pipeline)

Our Goal:

To take our raw, lumpy ingredients (the CSV files) and follow a recipe (our script) to bake a beautiful, multi-layered cake (our database). This whole process is called **ETL: Extract, Transform, Load.**

The "Why":

You wouldn't bake a cake by just throwing flour, eggs, and a live chicken into the oven. You have to prepare your ingredients! Data is the same. Raw data is often messy, inconsistent, and has more missing pieces than a toddler's jigsaw puzzle. Our ETL script will be our automated recipe to clean everything up, so we only have to write it once.

Step 2.1: Get Your Ingredients (The 'Extract' Part)

- **The Goal:** To find and download our raw datasets.
- **Your Mission, Should You Choose to Accept It:**
    1. Fire up your web browser and search for **"TCI-ICRISAT District Level Data India"**. You'll likely find this on Kaggle. It's a popular dataset for exactly this kind of project.
    2. Download the main files. You're looking for two specific types:
        - One with crop data (look for columns like District, State, Year, Crop, Area, Production, Yield).
        - One with weather data (look for columns like District, Year, and monthly rainfall).
    3. **Crucial Step:** Once downloaded, find the files in your `Downloads` folder using **File Explorer** (`Win + E`). Rename them to something simple and predictable. Let's use:
        - `crop_production_raw.csv`
        - `monthly_rainfall_raw.csv`
    4. Move these two newly-renamed files into the `data/` folder inside your `crop_yield_project` directory. You can do this by dragging and dropping them in File Explorer or using your terminal:
        - **On Windows (PowerShell):**
        - PowerShell

```
Move-Item "C:\Users\YourUser\Downloads\original_crop_file_name.csv"
"C:\path\to\crop_yield_project\data\crop_production_raw.csv"
Move-Item "C:\Users\YourUser\Downloads\original_rainfall_file_name.csv"
"C:\path\to\crop_yield_project\data\monthly_rainfall_raw.csv"
```

-
- *(Replace `YourUser` and `original_file_name.csv` with your actual details and file names. Adjust the project path accordingly.)*
- **On Windows (Legacy CMD):**
- DOS

```
move "C:\Users\YourUser\Downloads\original_crop_file_name.csv"
"C:\path\to\crop_yield_project\data\crop_production_raw.csv"
move "C:\Users\YourUser\Downloads\original_rainfall_file_name.csv"
"C:\path\to\crop_yield_project\data\monthly_rainfall_raw.csv"
```

-
- *(Replace `YourUser` and `original_file_name.csv` with your actual details and file names. Adjust the project path accordingly.)*
- **On WSL (Linux/Bash - assuming your Downloads folder is accessible via `/mnt/c/Users/YourUser/Downloads`):**
- Bash

```
mv /mnt/c/Users/YourUser/Downloads/original_crop_file_name.csv
/path/to/your/crop_yield_project/data/crop_production_raw.csv
mv /mnt/c/Users/YourUser/Downloads/original_rainfall_file_name.csv
/path/to/your/crop_yield_project/data/monthly_rainfall_raw.csv
```

-
- *(Replace `YourUser`, `original_file_name.csv`, and `/path/to/your/crop_yield_project` with your actual details.)*

Step 2.2: Writing the Recipe (The 'Transform' & 'Load' Parts)

- **The Goal:** To write the Python code in `scripts/etl.py` that will automatically clean our data and load it into our `agriculture.db`.
- Open `scripts/etl.py` in Notepad++. You can do this by navigating to `crop_yield_project/scripts/` in File Explorer, right-clicking `etl.py`, and choosing "Edit with Notepad++". We're going to build this recipe step-by-step.

Part A: The Setup (Your *Mise en Place*)

- **Note:** First, let's get our tools and ingredients ready by defining all our paths at the top of the script. This makes the code much easier to read and modify later.

- Python

```
# scripts/etl.py

import pandas as pd     # Our master chef for handling data tables
import sqlite3          # The manager of our SQLite database
import os               # The helpful assistant who knows all the file paths

# --- 1. DEFINE FILE PATHS ---
# Let's not hardcode paths. That's like writing a recipe that only works
# in one specific kitchen. os.path.join builds paths that work on any OS.
# When running the script, ensure your terminal's current directory is 'crop_yield_project'.
CWD = os.getcwd()  # This gets the path to our main 'crop_yield_project' folder
DB_PATH = os.path.join(CWD, 'data', 'agriculture.db')
CROP_DATA_PATH = os.path.join(CWD, 'data', 'crop_production_raw.csv')
RAINFALL_DATA_PATH = os.path.join(CWD, 'data', 'monthly_rainfall_raw.csv')
```

-
-
- **What this code does:** We're just telling our script where everything is. By using `os.path.join`, our script is now portable. It doesn't care if you're on Windows (`C:\Users\...`) or Linux (`/home/...`), it just works. Using `data` as a separate argument to `os.path.join` (e.g., `os.path.join(CWD, 'data', 'agriculture.db')`) is a slightly more robust way to handle path construction compared to concatenating strings like `'data/agriculture.db'`.

Part B: The Cleaning Function (Washing the Vegetables)

- **Note:** Next, we'll write a function whose only job is to clean up any messy data table (DataFrame) we give it. This is great practice because we can reuse it for multiple files.
- Python

```
def clean_data(df):
    """A reusable function to clean our dataframes. It's the dishwasher of our script."""

    # First, let's fix those messy column names.
    # 'District Name' becomes 'district_name'. Much tidier.
    df.columns = df.columns.str.lower().str.replace(' ', '_')

    # Now, let's hunt for missing values (NaNs), the ghosts in our data machine.
    # A simple strategy: if a number is missing, we'll assume it's 0.
    # If text is missing, we'll label it 'Unknown'.
    for col in df.columns:
        if pd.api.types.is_numeric_dtype(df[col]):
            df[col].fillna(0, inplace=True)
```

```
        else:
            # For text columns, we'll also strip any sneaky whitespace from the edges.
            df[col] = df[col].astype(str).str.strip()
            df[col].fillna('Unknown', inplace=True)

    return df
```

-
-
- **What this code does:** This function is our workhorse. It takes a DataFrame, standardizes the column names to be nice and Python-friendly, and then fills in any empty cells so our database doesn't have holes in it.

Part C: The Main Recipe (`main` function)

- **Note:** This is where we put all the steps together in order. We'll add this code to the *bottom* of our `etl.py` file. It will call our helper functions and execute the full Extract, Transform, and Load process.
- Python

```
def main():
    """This is the main recipe that runs our entire ETL pipeline."""
    print("--- Starting The Great Data Bake-Off! ---")

    # --- EXTRACT ---
    print("Step 1: Reading the raw, lumpy ingredients (CSVs)...")
    try:
        crop_df = pd.read_csv(CROP_DATA_PATH)
        rainfall_df = pd.read_csv(RAINFALL_DATA_PATH)
        print("Successfully loaded CSVs. They smell... raw.")
    except FileNotFoundError as e:
        print(f"🚨 HEY! I couldn't find a file. Error: {e}")
        print("Did you rename the CSVs and put them in the 'data/' folder? Go check!")
        return # Stop the script if we can't find the files.

    # --- TRANSFORM ---
    print("Step 2: Cleaning, chopping, and mixing (Transforming Data)...")
    crop_df_clean = clean_data(crop_df.copy())
    rainfall_df_clean = clean_data(rainfall_df.copy())
    print("Data is now sparkling clean. You could eat off it.")

    # --- LOAD ---
    print("Step 3: Putting the cake in the oven (Loading to Database)...")
    try:
        # 'with' is a magic word in Python that handles opening and closing things for us.
        with sqlite3.connect(DB_PATH) as conn:
```

```
            # We're telling pandas to take our clean DataFrame and dump it into an SQL table.
            # if_exists='replace' is like smashing your old cake to bake a new one.
            # Great for development, TERRIFYING in production. Use with caution!
            crop_df_clean.to_sql('crop_production', conn, if_exists='replace', index=False)
            rainfall_df_clean.to_sql('monthly_rainfall', conn, if_exists='replace', index=False)
            print("Ding! Cake is ready. Data loaded into the database.")
    except Exception as e:
        print(f"🔥 Whoops, something went wrong while baking. Error: {e}")
        return

    print("--- ETL Pipeline Complete. Bon Appétit! ---")

# This little piece of magic makes sure main() only runs when you execute this file directly.
# It's like the 'On' button for our script.
if __name__ == '__main__':
    main()
```

-
-

Step 2.3: Running the Script & Admiring Your Cake

1. **Open your terminal:**
    - **On Windows (PowerShell/CMD):** Open a new PowerShell or Command Prompt window.
    - **On WSL (Linux/Bash):** Open your WSL distribution's terminal (e.g., Ubuntu).
2. **Navigate to your project's root directory (`crop_yield_project`):**
    - **On Windows (PowerShell / Legacy CMD):**
    - PowerShell

```
cd C:\path\to\crop_yield_project
```

    -
    - *(Replace `C:\path\to\crop_yield_project` with the actual path).*
    - **On WSL (Linux/Bash):**
    - Bash

```
cd /path/to/your/crop_yield_project
```

    -
    - *(If your project is on the Windows file system, remember to navigate via `/mnt/c/Users/YourUser/Documents/crop_yield_project`.)*
3. **Activate your virtual environment:** (Crucial!)

- **On Windows (PowerShell):**
- PowerShell

```
.\venv\Scripts\Activate.ps1
```

-
-
- **On Windows (Legacy CMD):**
- DOS

```
.\venv\Scripts\activate.bat
```

-
-
- **On WSL (Linux/Bash):**
- Bash

```
source venv/bin/activate
```

-
-
- Ensure your terminal prompt now shows `(venv)` at the beginning.

4. **Run the script:**
5. Bash

```
python scripts/etl.py
```

6.
7.
8. Watch the `print` statements fly by. It should tell you it completed successfully.
9. **The Moment of Truth (Viewing your data):**
   - Since Notepad++ doesn't have an integrated SQLite Explorer, you'll need your standalone **"DB Browser for SQLite"** tool.
   - Open "DB Browser for SQLite".
   - Click "Open Database" and navigate to `your_crop_yield_project_folder/data/agriculture.db`.
   - Behold! On the left panel, you should now see two tables: `crop_production` and `monthly_rainfall`.
   - Click on a table name, then click the "Browse Data" tab to see your beautiful, clean data, neatly organized in rows and columns.

---

4. Debugging - When Your Cake is a Bit... Flat

Here are common issues and their solutions, applicable to both Windows native terminals and WSL.

- **Error: `FileNotFoundError`**
    - **What it means:** "I went to the pantry for the flour, but it wasn't there!" The script couldn't find one of the raw CSV files or the database file it expected.
    - **The Fix:**
        1. **Check the `data/` folder:** Use **File Explorer (`Win + E`)** to navigate to `your_crop_yield_project_folder/data/`. Are your `crop_production_raw.csv` and `monthly_rainfall_raw.csv` files physically present there?
        2. **Spelling and Path:** Did you spell their names *exactly* right in the `etl.py` script, including the `.csv` part? Is the `DB_PATH` correctly pointing to `data/agriculture.db`?
        3. **Terminal Current Directory:** Double-check that you are running the `python scripts/etl.py` command from the *root* of your `crop_yield_project` folder in your terminal. If you are in a different directory, `os.getcwd()` will resolve to the wrong path.
- **Error: `KeyError` or `AttributeError` during the `clean_data` step.**
    - **What it means:** "The recipe says to crack an egg, but you gave me a potato." The script expected a column with a certain name (e.g., 'District Name'), but it found something else or couldn't perform an operation because a column was missing or of an unexpected type.
    - **The Fix:**
        1. **Inspect Raw CSVs:** Open your `crop_production_raw.csv` and `monthly_rainfall_raw.csv` files in Notepad++. Look closely at the very first row (the header row).
        2. **Compare Column Names:** Are the column names what you expect? For instance, does your crop file have a column named "District Name" or something similar? The `clean_data` function tries to standardize names, but if they are vastly different, you might need to manually rename them or adjust the cleaning logic. For example, if your column is `Dist_Name` instead of `District Name`, the `replace(' ', '_')` part might not work as intended.
        3. **Add `df.rename`:** If a crucial column has a completely different name, you might need to add a line like `df.rename(columns={'bizarre_name': 'good_name'}, inplace=True)` in your `main()` function *before* calling `clean_data` on that specific DataFrame.
- **Problem: The script ran, but the data in the database looks weird.**
    - **What it means:** Something went wrong in the mixing bowl. The data was loaded, but the transformations might not have worked as expected.
    - **The Fix:** Become a data detective! Add `print()` statements in your `etl.py` script to see what the data looks like at each stage of the transformation process.

- Python

```
# In etl.py, inside main()
# ... (after reading CSVs)
print("\n--- Raw Crop Data Head (First 5 rows) ---")
print(crop_df.head()) # See the first 5 rows of raw crop data
print("\n--- Raw Rainfall Data Head (First 5 rows) ---")
print(rainfall_df.head()) # See the first 5 rows of raw rainfall data

# ... (after calling clean_data)
crop_df_clean = clean_data(crop_df.copy())
rainfall_df_clean = clean_data(rainfall_df.copy())
print("\n--- Cleaned Crop Data Head (First 5 rows) ---")
print(crop_df_clean.head()) # See the first 5 rows of clean crop data
print("\n--- Cleaned Rainfall Data Head (First 5 rows) ---")
print(rainfall_df_clean.head()) # See the first 5 rows of clean rainfall data
# ...
```

-
-
- Save `etl.py` in Notepad++ after adding these print statements.
- Run the script again from your activated terminal (`python scripts/etl.py`).
- By comparing the "before" and "after" printouts in your terminal, you can pinpoint exactly where your transformation went sideways.

You've officially conquered the hardest part of many data science projects. You now have a repeatable, automated pipeline to turn messy data into a clean, queryable database. Now, we're ready for the fun part: analysis and modeling!
