Let's Start: Step-by-Step Setup for Windows with WSL

This guide will walk you through setting up your project environment, specifically tailored for a Windows operating system. We'll provide commands for native Windows command-line interfaces (PowerShell and Legacy CMD) and for the Windows Subsystem for Linux (WSL). We'll also include useful Windows keyboard shortcuts for navigation.

Windows Navigation Shortcuts (General Use)

These shortcuts can significantly speed up your workflow on Windows:

- **File Explorer:**
  - `Win + E`: Open File Explorer.
  - `Alt + D`: Select the address bar.
  - `Alt + Left Arrow`: Go back to the previous folder.
  - `Alt + Right Arrow`: Go forward to the next folder.
  - `Ctrl + Shift + N`: Create a new folder.
  - `F2`: Rename selected item.
  - `Ctrl + C / Ctrl + V / Ctrl + X`: Copy / Paste / Cut.
  - `Delete`: Delete selected item.
- **Window Management:**
  - `Win + Left Arrow / Right Arrow`: Snap window to left/right half of the screen.
  - `Win + Up Arrow / Down Arrow`: Maximize / Restore Down / Minimize window.
  - `Alt + Tab`: Switch between open windows.
  - `Win + Tab`: Open Task View.
  - `Win + D`: Show/hide desktop.
- **General Productivity:**
  - `Ctrl + Z`: Undo.
  - `Ctrl + Y`: Redo.
  - `Win + S` or `Win + Q`: Open Windows Search.
  - `Win + R`: Open Run dialog.
  - `Ctrl + Shift + Esc`: Open Task Manager.
  - `F5` (in File Explorer/Browser): Refresh.

Step 1: Create Your Project's "House"

We need a dedicated folder for our project.

1. Open your computer's main terminal. On Windows, you have a choice:
   - **PowerShell:** Search for "PowerShell" in the Start Menu.
   - **Command Prompt (CMD):** Search for "CMD" in the Start Menu.
   - **WSL (Ubuntu, Debian, etc.):** Search for your installed Linux distribution (e.g., "Ubuntu") in the Start Menu.
2. Navigate to a place you like to store projects, such as your 'Documents' folder.

    - **PowerShell / Legacy CMD (Windows):** `cd C:\Users\YourUser\Documents` (Replace `YourUser` with your actual Windows username).
    - **WSL (Linux/Bash):** `cd ~/Documents` or `cd /home/YourLinuxUser/Documents` (If your Documents folder is linked, otherwise use `cd /mnt/c/Users/YourUser/Documents` to access your Windows Documents from WSL).
3. Run these commands one by one to create your project structure:
4. Bash

```
# This command creates our main project folder

# For Windows (PowerShell / Legacy CMD):
mkdir "crop yield project"

# For WSL (Linux/Bash):
mkdir "crop yield project"

# This command moves you inside that new folder

# For Windows (PowerShell / Legacy CMD):
cd "crop yield project"

# For WSL (Linux/Bash):
cd "crop yield project"

# This creates the "rooms" inside our house for organization

# For Windows (PowerShell / Legacy CMD):
mkdir data scripts notebooks

# For WSL (Linux/Bash):
mkdir data scripts notebooks
```

5.
    - `data/` is for all data files.
    - `scripts/` is for reusable Python code.
    - `notebooks/` is for experiments and analysis.

Step 2: Open the Project in Notepad++

Notepad++ is a robust text editor. It doesn't have an integrated project explorer in the same way an IDE like VS Code does, but you can manage your files effectively.

1. Use **File Explorer (`Win + E`)** to navigate to your `crop yield project` folder.

2. You can open individual files within Notepad++ as you create them. Alternatively, you can drag the entire crop yield project folder into the Notepad++ interface. While Notepad++ won't display a traditional "file explorer" panel, dragging the folder will open all current files within it in separate tabs, allowing you to quickly switch between them. You will primarily use your operating system's file explorer for navigating the directory structure.

Step 3: Create a "Bubble" for Our Project (The Virtual Environment)

This is a crucial step for avoiding conflicts between project dependencies. We'll create a self-contained Python "bubble" (venv) that holds all the specific packages for this project only.

1. Open your *main computer terminal* again (the one you used in Step 1). Ensure you are inside the crop yield project directory.
2. Run this command to create the environment named venv:
3. Bash

```
# For Windows (PowerShell / Legacy CMD) / WSL (Linux/Bash):
python3 -m venv venv
```

4.
    - **Note for Windows:** On some Windows systems, python might be used instead of python3. If python3 fails, try python -m venv venv.
5. You'll observe a new venv folder appearing inside your crop yield project directory. Now, you need to "step inside" this bubble to activate it:
    - **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
```

-
-

6. Once activated, your terminal prompt will change to include `(venv)` at the beginning (e.g., `(venv) C:\Users\YourUser\Documents\crop yield project>` or `(venv) youruser@yourmachine:~/Documents/crop yield project$`). This visually confirms that the bubble is active.

Step 4: Create the Initial Files

Let's create the empty files we'll be working with. These commands should be run in your *active (venv) terminal*.

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

```
New-Item -ItemType File -Path ".gitignore"
New-Item -ItemType File -Path "requirements.txt"
New-Item -ItemType File -Path "scripts/etl.py"
New-Item -ItemType File -Path "scripts/dashboard.py"
New-Item -ItemType File -Path "notebooks/01_Data_Exploration.ipynb"
```

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

```
type nul > .gitignore
type nul > requirements.txt
type nul > scripts\etl.py
type nul > scripts\dashboard.py
type nul > notebooks\01_Data_Exploration.ipynb
```

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

```
touch .gitignore
touch requirements.txt
touch scripts/etl.py
touch scripts/dashboard.py
touch notebooks/01_Data_Exploration.ipynb
```

-

-

Step 5: Install Our Tools (Dependencies)

Now we'll install the Python libraries we need. The best practice is to list them in `requirements.txt` first.

1. Open the `requirements.txt` file in Notepad++ (you can drag and drop it from File Explorer or use File > Open). Paste the following content:

```
# Core libraries for data manipulation and database interaction
pandas
sqlalchemy
# For exploratory notebooks
jupyterlab
notebook
# For downloading data
requests
```

2.
3.
4. Save the `requirements.txt` file.
5. In your *active (venv) terminal*, run this command to install everything listed in that file:
6. Bash

```
# For Windows (PowerShell / Legacy CMD) / WSL (Linux/Bash):
pip install -r requirements.txt
```

7.
8.

Best Practice: Later, if you need a new package (like `streamlit`), you should:

1. Run `pip install streamlit` in your *active (venv) terminal*.
2. **Immediately** run `pip freeze > requirements.txt` in the same terminal to update your `requirements.txt` file with the new package and its exact version. This ensures others (or your future self) can replicate your environment.

Step 6: Set Up the Database

We'll use SQLite, which is just a single file. Our Python script will create and manage it later, but we can create the empty file now.

1. In your *active (venv) terminal*, create the empty database file:
2. Bash

```
# For Windows (PowerShell):
New-Item -ItemType File -Path "data\agriculture.db"

# For Windows (Legacy CMD):
type nul > data\agriculture.db

# For WSL (Linux/Bash):
touch data/agriculture.db
```

3.
4.
5. **To see the data inside Notepad++:** Notepad++ is primarily a text editor and does not have built-in database Browse capabilities. To view SQLite database contents, you will need a separate, dedicated tool.
    - **Recommendation:** Use a standalone SQLite browser like **"DB Browser for SQLite"**. This free, open-source tool is available for Windows, macOS, and Linux. Once installed, you can use it to open your `data/agriculture.db` file and view its tables and data.

How to Debug if Things Go Wrong

Here are common issues and their solutions, applicable to Windows environments, including WSL.

- **Problem:** `command not found: python3` **or** `pip`
    - **Cause:** Python isn't installed correctly on your system, or it's not accessible via your system's PATH environment variable. In WSL, this might mean Python isn't installed within your Linux distribution.
    - **Fix:**
        1. **For Windows (PowerShell/CMD):** Reinstall Python from the official website (python.org). During installation, make sure to check the box that says "Add Python to PATH."
        2. **For WSL (Linux/Bash):** Open your WSL terminal and install Python: `sudo apt update && sudo apt install python3 python3-pip`.
- **Problem:** `ModuleNotFoundError: No module named 'pandas'`
    - **Cause:** This is the most common error. It means you're trying to run your Python script using the wrong Python interpreter. Your `venv` "bubble" is likely not active in the terminal where you're executing the script.
    - **Fix:**
        1. **Check your terminal:** Does it prominently display `(venv)` at the start of the prompt? If not, run the appropriate activation command for your environment (refer to Step 3, point 4) to activate the virtual environment.

2. **Execute script from active terminal:** When running your Python script (e.g., `python scripts/etl.py`), ensure you are running it from the *same terminal window* where `(venv)` is active.
3. **Reinstall dependencies:** As a precautionary measure, run `pip install -r requirements.txt` again in your *active (venv) terminal* to ensure all necessary packages are installed within that specific environment.

- **Problem: `Permission Denied` when activating `venv` on Windows (PowerShell).**
  - **Cause:** Your Windows system's execution policy is blocking PowerShell scripts from running. This is a security feature.
  - **Fix:**
    1. Open **PowerShell as an Administrator**. To do this, search for "PowerShell" in the Start Menu, right-click on "Windows PowerShell," and select "Run as administrator."
    2. In the administrative PowerShell window, run the following command:
    3. PowerShell

Set-ExecutionPolicy Unrestricted

    4.
    5. You might be prompted to confirm; type `Y` and press Enter.
    6. Close the administrative PowerShell window.
    7. Now, try activating your `venv` again in your *regular (non-administrative) PowerShell terminal*.