Phase 1 v1 — Data Ingestion & Database Setup

1. Purpose of Phase 1

In Phase 1, we are doing the minimum required “plumbing” so that our project has a solid foundation.

Right now, the prediction logic, machine learning, and fancy UI do not matter if we do not have clean, reliable data stored in one place. Phase 1 is about:

- Choosing a local database (we will use SQLite).
- Designing the database structure (tables and columns).
- Writing a small Python program that connects to the Sleeper API and downloads NFL game and player stats.
- Saving that data into the database in a clean and consistent way.
- Verifying that the data looks correct with some simple checks.

By the end of Phase 1, you will have a single .db file on your computer that contains historical NFL data from Sleeper in a structure that is ready for modeling and predictions later. This phase is more about “data engineering” than “data science.”

2. High-Level Architecture of Phase 1

At a high level, Phase 1 involves three moving pieces:

1) Your computer (Windows) — where you will:
   - Install Python
   - Create a virtual environment for the project
   - Run Python scripts
   - Store the SQLite database file

2) The Sleeper API (on the internet) — a public HTTP-based API that returns JSON data about NFL players, games, and stats.

3) The SQLite database file — a local file (for example, fantasy_nfl.db) that stores all of the data in tables.

Flow of data:

Sleeper API (JSON over HTTPS) → Python script (parses JSON) → SQLite DB (tables + rows)

Everything runs locally: you run the Python script, it talks to Sleeper, gets the data, and writes into your local database file.

3. Tools You Need Installed

Before you start, make sure you have:

1) Python 3.x installed on Windows
   • Go to python.org and download the latest Python 3 for Windows.
   • During installation, check the box “Add Python to PATH” if it appears.

2) A code editor or IDE
   • VS Code is a good choice (free, easy to use).

3) Basic command line access
   • You will use Command Prompt or PowerShell to:
     - Create a project folder
     - Create and activate a virtual environment
     - Run your Python scripts

4) Internet access
   • Required so your Python script can talk to the Sleeper API and download data.

4. Project Folder and Virtual Environment

We will keep everything for this project in one main folder. For example:

C:\Users\<YourUser>\fantasy_predictor\

Inside this folder, we will keep:
• The virtual environment (venv)
• Your Python scripts (e.g., ingest_sleeper_data.py)
• The SQLite database file (e.g., fantasy_nfl.db)

Step-by-step:

1) Create the folder in File Explorer or via command line:
   • mkdir C:\Users\<YourUser>\fantasy_predictor

2) Open a terminal (Command Prompt or PowerShell) in that folder.

3) Create a virtual environment:
   • python -m venv venv

4) Activate the virtual environment:
   • On PowerShell: .\venv\Scripts\Activate.ps1
   • On Command Prompt: .\venv\Scripts\activate

5) Install needed Python packages:
   • pip install requests

Note: SQLite is built into Python via the sqlite3 module, so you don't need to install anything extra for it.

5. Why We Chose SQLite for Phase 1

SQLite is a database engine that stores everything in a single file (for example, fantasy_nfl.db). It's perfect for this project phase because:

- It requires no server — it's just a file on disk.
- Python comes with sqlite3 built in.
- It is fast and reliable enough for the data sizes we're dealing with.
- It is easy to back up and move (copy the .db file).

Later, if you move to a web app or need multi-user access, you can migrate the same schema to PostgreSQL or another server-based database. But for now, SQLite keeps things simple and moves you forward quickly without DevOps complexity.

6. Database Schema — Tables and Their Roles

We will define a minimal set of tables that can support stat prediction later. You can think of each table as an Excel sheet with columns (fields) and rows (records).

6.1. players table

This table stores basic information about each NFL player.

Columns (initial idea):

- player_id (PRIMARY KEY, integer) — auto-increment ID in our database.
- sleeper_id (text) — the unique ID from Sleeper's system.
- name (text) — player's full name.
- position (text) — e.g., QB, RB, WR, TE, K, DEF.
- team (text) — team abbreviation (e.g., KC, BUF).

Why we need it: predictions are made per player, so we want a stable way to reference each player across games.

6.2. teams table

This table stores information about teams.

Columns:

- team_id (PRIMARY KEY, integer) — auto-increment ID.
- abbrev (text) — e.g., KC, BUF, DAL.
- name (text) — Kansas City Chiefs, etc.

Why we need it: teams are used for offense and defense stats, and to identify matchups.

6.3. games table
One row per NFL game.

Columns:
• game_id (PRIMARY KEY, integer) − auto-increment ID.
• sleeper_game_id (text) − the game ID from Sleeper (if available).
• season (integer) − e.g., 2023.
• week (integer) − 1–18.
• home_team_id (integer, FOREIGN KEY to teams.team_id).
• away_team_id (integer, FOREIGN KEY to teams.team_id).
• game_date (text or datetime) − date/time of the game.

Why we need it: predictions are “for week X vs team Y.” Games structure everything.

6.4. player_game_stats table
This is the main stats table: one row per (player, game). This will include all the per-position stats we care about (even if some will be null/zero depending on position).

Example columns:
• id (PRIMARY KEY)
• player_id (FOREIGN KEY to players.player_id)
• game_id (FOREIGN KEY to games.game_id)
• team_id (FOREIGN KEY to teams.team_id)
• opponent_team_id (FOREIGN KEY to teams.team_id)

Stat columns (examples; can expand as needed):
Passing:
• pass_attempts
• pass_completions
• pass_yards
• pass_tds
• interceptions
• sacks

Rushing:
• rush_attempts
• rush_yards
• rush_tds
• rush_fumbles

Receiving:
• targets
• receptions
• rec_yards
• rec_tds

Kicking:
• fg_attempts
• fg_made
• xp_attempts
• xp_made

Defense (for team defenses, or defensive players if needed):
• def_sacks
• def_ints
• def_forced_fumbles
• def_fumble_recoveries
• def_td
• points_allowed

Why we need it: this is the core training data for our prediction model. Each row describes what a player actually did in a specific game.

6.5. team_game_stats_offense & team_game_stats_defense (optional in Phase 1)
You can derive these by aggregating player_game_stats or by pulling team-level stats from future APIs. For Phase 1, you can skip them or create them as empty shells that we fill later.

The key point: get players, games, and player_game_stats right first. We can always add more tables as Phase 2 and Phase 3 need them.

7. Understanding the Sleeper API at a High Level

The Sleeper API is a REST API. That just means:

• You send an HTTP request to a specific URL.
• The server responds with data in JSON format.
• JSON is basically a text-based representation of nested lists and dictionaries (key-value pairs).

For example, a GET request might look like:
• https://api.sleeper.app/v1/some/endpoint

In Python, we'll use the requests library to:
• Send an HTTP GET request.
• Parse the JSON response.
• Loop over the objects we get back and write them into our SQLite tables.

You do NOT have to fully understand HTTP or JSON deeply at first. You just need to know:
• "requests.get(url)" gets the data.
• "response.json()" converts it into Python objects (lists/dicts).
• You then read the keys from those dicts and insert them into the database.

We will look up the specific Sleeper endpoints we need (for example, stats by week, players, etc.) when we implement the script, but conceptually this is how it works.

8. Step-by-Step Plan for Phase 1

Here is the concrete, ordered list of tasks you will complete.

Step 1 – Create the project folder and virtual environment
• Make a folder for the project (e.g., C:\Users\You\fantasy_predictor).
• In a terminal, navigate to that folder.
• Run: python -m venv venv
• Activate the venv.
• Run: pip install requests

Step 2 – Create the SQLite database
• In your Python script, you will:
  - import sqlite3
  - connect to a database file (sqlite3.connect("fantasy_nfl.db"))
  - create a cursor (conn.cursor())
  - execute SQL CREATE TABLE statements for the tables defined above.
• Run the script once to create the database and tables.
• You can confirm the file fantasy_nfl.db exists in your project folder.

Step 3 – Write a basic Sleeper API test script
• Create a Python file (e.g., test_sleeper.py).
• Use requests.get(...) to hit a simple Sleeper endpoint (like /players or a sample stats endpoint).
• Print the JSON to make sure:
  - Your internet access works.

- Sleeper is responding.
- Your virtual environment and requests package are working.

Step 4 — Write the data ingestion script
• Create a main ingestion script (e.g., ingest_sleeper_data.py).
• High-level logic:
1) Connect to SQLite with sqlite3.connect("fantasy_nfl.db").
2) Pull player data from Sleeper, insert/update records in players table.
3) Pull game/weekly stats for a specific season and week range.
4) For each stat line, find or create the related player, team, and game records.
5) Insert a row into player_game_stats with the mapped stats.

Key concept: you must map whatever field names Sleeper uses (like passing_yards, rushing_yards, etc.) into your own column names in player_game_stats. This is "normalization."

Step 5 — Data validation
• After ingesting data for a test season/week, open a Python shell and run:
- SELECT COUNT(*) FROM players;
- SELECT COUNT(*) FROM player_game_stats;
- A few SELECT * LIMIT 10 queries to see real data.
• Check a few known players to confirm their stats look correct compared to Sleeper or a public stats page.

Step 6 — Rinse and repeat for more seasons/weeks
• Once the pipeline works for a small sample (like one season, a few weeks), expand it to all weeks in that season.
• Optionally loop through multiple seasons.

By the end of Step 6, your fantasy_nfl.db will contain enough historical data to begin Phase 2 (feature engineering and baseline prediction formulas).

9. How Phase 1 Sets Up Later Training

Even though Phase 1 does NOT include machine learning, everything we are doing here is in service of training in later phases. Specifically:

• Each row in player_game_stats will become a "training example."
• Features (inputs) such as:
- player's recent games
- opponent defense strength
- team context

will all be computed from these tables.
• The “label” (what we are trying to predict) is the actual stat value in each row (e.g., rushing_yards, pass_tds, targets).

If the data is messy or missing, training quality will suffer badly. That’s why Phase 1 must focus on clean, consistent ingestion and normalization:

• Clear IDs (player_id, game_id, team_id).
• Correct mapping between players, teams, and games.
• Correct stat values for each player in each game.

Once Phase 1 is done, you will be able to query your database like:

• “Give me all games for Player X in 2023.”
• “Give me all statlines for RBs vs Team Y’s defense.”
• “Give me last 3 games of stats for this player before Week N.”

These queries are exactly what we’ll need in Phase 2 and Phase 3 to build the prediction features and to train the regression/ensemble models.

10. Summary – What Success Looks Like for Phase 1

You will know Phase 1 is successfully completed when all of the following are true:

1) You have a local project folder with:
   • A working Python virtual environment
   • One or more ingestion scripts
   • A fantasy_nfl.db file (or similar SQLite db)

2) The SQLite database has the expected tables:
   • players
   • teams
   • games
   • player_game_stats

3) You can run a small Python script to:
   • Connect to the database
   • Run SELECT queries
   • Print out real player-game rows with plausible stats

4) You can confidently say:
   “If I want to know what Player X did in Week Y of Season Z, I can get that from my database without touching an API.”

At this point, your system has a “memory” of real historical games stored locally. This is the foundation that will power all future steps: feature engineering, model training, fantasy point prediction, start/sit logic, waiver recommendations, and game outcome predictions.