Plan v2 — Full Updated Specification

PLAN V2 — UPDATED AND STRUCTURED DEVELOPMENT PLAN
==========

This document is a complete rewrite and expansion of Plan v1, incorporating all decisions, clarifications, and architectural commitments. It provides a full end to end blueprint for building a predictive NFL analytics engine capable of powering fantasy start/sit decisions, waiver recommendations, trade evaluations, and NFL game outcome predictions. The system will begin as a local Python application, later move toward a desktop frontend, and eventually incorporate Ollama for natural language reporting.

The plan is organized in chronological development order. Each section describes:
• What will be built
• How it will be built
• Where the data will come from
• How predictions will be trained
• Which statistics are tracked, and why
• How each component fits into the full prediction pipeline

No code is included in this document—this is the finalized architectural and methodological plan.

---
1. OVERVIEW AND HIGH LEVEL GOALS

The objective is to create a statistical engine that predicts individual NFL player statlines for upcoming games. From these statline predictions, fantasy scoring can be calculated, which directly supports:
• Weekly start/sit recommendations
• Waiver wire pickups
• Trade evaluations
• Rest-of-season projections
• NFL game score predictions
• Betting style spread projections

Prediction of *fantasy points* is NOT the primary model—our core

engine predicts *raw statlines*, because raw stats generalize to all outputs.

The tool will begin as a **local Python script**, backed by a **SQLite database**, using:
• Sleeper API (free) for weekly player statistics
• Optional nflfastR data for advanced metrics later

Once functioning, a desktop UI will be added. Ollama will later be integrated to generate readable, customized reports.

---
2. DATA SOURCES AND INGESTION PIPELINE

2.1 Sleeper API (Primary Source)

Sleeper provides free access to:
• Weekly player stats
• Player metadata (position, team, ID, name)
• League info (rosters, scoring rules, waiver pool)

These weekly stats include all major fantasy relevant categories such as:
- Passing: attempts, completions, yards, TD, INT, sacks
- Rushing: attempts, yards, TD, fumbles
- Receiving: targets, receptions, yards, TD
- Kicking: FG attempts, FG made, XP attempts, XP made
- Defense: sacks, INT, forced fumbles, TDs

These statistics will be pulled via a Python ingestion script and stored in a structured SQLite database.

2.2 nflfastR (Optional Advanced Metrics Source)

Used in later phases to enhance model accuracy:
• EPA (expected points added)
• Success rate
• Air yards
• Rush yards over expected
• Play-by-play context

These metrics are extremely predictive but not strictly necessary in Phase 1.

2.3 Other Data Sources (Influencing Factors)

Later enhancements include:
• Weather (OpenWeather API)
• Injuries (Sleeper injury reports)
• Trades / roster changes (Sleeper transactions)
• O-line and defensive depth chart injuries

Initially, influencing factors will be excluded from training until core prediction accuracy is established.

---
3. DATABASE DESIGN AND STRUCTURE

We will use **SQLite** for the initial implementation, as it is:
• Local
• Fast
• File-based
• Simple
• Zero configuration

The following tables will be created:

3.1 players
```
player_id (PK)
sleeper_id
name
position
team
archetype (added later)
```

3.2 teams
```
team_id (PK)
name
abbrev
conference
division
```

3.3 games
```
game_id (PK)
season
week
home_team_id
away_team_id
date
```

3.4 player_game_stats
One row per player per game. Includes all tracked statistics for all positions:
```
passing_attempts
completions
passing_yards
passing_tds
interceptions
sacks
rushing_attempts
rushing_yards
rushing_tds
receptions
targets
receiving_yards
receiving_tds
fumbles
```

3.5 team_game_stats_offense
```
total_yards
rushing_yards
passing_yards
rush_attempts
pass_attempts
rush_tds
pass_tds
points
red_zone_td_pct
```

3.6 team_game_stats_defense

Same fields as offense, but what the defense allowed.

3.7 influencing_factors
Added later:
```
game_id
player_id (nullable)
type (injury/weather/trade)
severity
weight
notes
```

---
4. STATISTICS TRACKED AND WHY THEY MATTER

Statistics are grouped by position, but all flow into predicted statlines.

4.1 Quarterback Stats
- Passing attempts, completions, completion %
- Passing yards, TDs, INTs
- Sacks taken
- Rushing attempts, yards, TDs
- Rushing fumbles

These determine fantasy scoring and feed team-level offensive predictions.

4.2 Running Back Stats
- Rushing attempts, yards, TDs
- Targets, receptions, receiving yards, receiving TDs
- Fumbles

These are the most volume-shaped position—predicting attempts is critical.

4.3 Wide Receiver / Tight End Stats
- Targets, receptions
- Receiving yards, TDs
- Rushing attempts (rare but included)
- Fumbles

Target share and defensive coverage matchup matter heavily.

4.4 Kicker Stats
• FG attempts
• FGs made
• Average FG distance
• XP attempts and %

These follow from predicted offensive drives.

4.5 Defense Stats
• Sacks
• Interceptions
• Forced fumbles
• Defensive TDs
• Points allowed

These map into fantasy scoring categories and game prediction accuracy.

4.6 Team Offense + Defense Stats
Used to contextualize matchups:
• Total offensive yards
• Pass attempts, rush attempts
• Red zone TD efficiency
• Opponent defensive yards allowed
• Opponent EPA allowed
• Opponent passing vs rushing splits

---
5. FEATURE ENGINEERING (THE CORE OF PREDICTION)

To predict upcoming statlines, we extract structured features:

5.1 Player Form Features
These are heavily inspired by the “previous 3 games” method in Plan v1:
• Season-to-date per-game averages
• Last 3 games exponential weighted average
• Short-term deviation from season baseline (how hot/cold they are)

This provides both short-term and mid-term trend signals.

5.2 Opponent Defense Features
For the defense the player is facing:
• Season defensive averages allowed to position
• Last 3 games allowed stats (recency-weighted)
• Matchup skew (how similar players performed vs this defense)

This integrates your “opponent context correction.”

5.3 Team Context Features
• Team offensive pace
• Pass rate / rush rate
• Red zone usage
• Quarterback mechanics for WR/TE/RB projections
• O-line strength indicators (later)

5.4 Influencing Factor Features (Later)
Examples:
• Weather: wind, rain, temperature
• Injuries: Offensive line missing starters, WR group weakened
• Trades & depth chart changes

Weights will be learned later.

---
6. BASELINE PREDICTION FORMULA

The first predictive layer uses a deterministic formula:

```
predicted_stat =
    α * (player_season_avg)
  + β * (player_last_3_weighted_avg)
  + γ * (defense_allowed_avg_to_position)
  + δ * (defense_last_3_allowed_weighted)
```

Where:
• α + β + γ + δ = 1
• The weights will be empirically tuned using historical backtesting

This guarantees:
• Recency is respected

• Long-term baselines are respected
• Defensive context is respected

---
7. TRAINING THE MACHINE LEARNING PREDICTOR

After building the baseline model, we train a machine learning correction layer.

7.1 Training Data
From the database, we build a table:

Each row = (player, game, actual stats, all engineered features)

7.2 Split Into Train/Test
• 70% training
• 15% validation
• 15% testing
• Stratified by position

7.3 Models Used
We will train multiple models:

1. **Ridge/Elastic Net Regression**
   Predicts statline residuals vs baseline.

2. **Gradient Boosting Model (LightGBM / XGBoost)**
   Handles nonlinear interactions between:
   - team offense
   - defense strength
   - pace
   - recent form
   - weather (later)

3. **Baseline model itself**
   Always included in ensemble.

7.4 Ensemble Blending
Final prediction:
```
final_prediction =
    w1 * baseline_prediction
  + w2 * regression_prediction
```

```
  + w3 * gradient_boost_prediction
```

Weights are selected based on performance on validation data.

7.5 Cross Validation
We evaluate:
• MAE (mean absolute error)
• RMSE
• Prediction interval calibration

7.6 Iterative Improvement
Once enough data is gathered:
• Add archetype classification
• Improve feature engineering
• Incorporate advanced metrics (EPA, success rate)
• Introduce influencing factor weights

---
8. OUTPUTS GENERATED FROM STAT PREDICTIONS

8.1 Fantasy Points
Using Sleeper league scoring settings, convert predicted stats to fantasy points.

8.2 Start vs Sit
Given two players A and B:
• Compare predicted fantasy points
• Estimate variance
• Compute boom/bust probability
• Recommend start/sit

8.3 Waiver Recommendations
For each waiver player:
• Predict statlines for all remaining games
• Sum fantasy points
• Rank players

8.4 Trade Evaluations
Sum predicted rest-of-season production for players on both sides.

8.5 NFL Game Predictions
Sum offensive player stat projections:

• Predicted TDs
• Projected field goals
• Projected team points

Generate:
• projected winner
• spread
• total points

---

9. USER INTERFACE & OLLAMA INTEGRATION (LATER)

9.1 Desktop Interface (Later Phase)
A small desktop UI will display:
• Start/Sit comparison
• Waiver recommendations
• Matchup overview
• Game predictions

9.2 Ollama Integration
Ollama will:
• Interpret the raw statistical predictions
• Identify surprising or meaningful influencing factors
• Generate human readable reports:
  - “Start Player X this week because…”
  - “Player Y is a top waiver pickup due to…”

Input to Ollama = structured JSON of predictions.

---

10. DEVELOPMENT TIMELINE

Phase 1 − Data ingestion + Database (Earliest)
• Build SQLite database
• Pull Sleeper API data
• Store first season of data

Phase 2 − Feature Engineering
• Implement form features
• Implement defensive context features

Phase 3 − Baseline Predictor

• Implement the weighted formula
• Evaluate against historical data

Phase 4 — ML Training
• Build residual model
• Build ensemble

Phase 5 — Fantasy & Game Outputs
• Convert stats → fantasy
• Build Start/Sit
• Build waivers/trade tools
• Build game prediction tool

Phase 6 — UI + Ollama
• Build GUI
• Build report generator

END OF PLAN V2
