NBA “Cover the Spread” Predictor — Detailed, Step-by-Step Build Plan

This guide expands the original outline into a full implementation plan with concrete tasks, code skeletons, acceptance criteria, and checklists. You can follow it top-to-bottom or pick sections as needed.

Phase 0 — Project Setup (½–1 day)

Goal: Clean, reproducible repo; local dev ready.

Folder Layout

```
spread-predictor/
├─ data/
│  ├─ raw/               # vendor & scraped CSVs/JSONs
│  ├─ interim/           # tmp artifacts during processing
│  └─ processed/         # clean Parquet/CSV (games, features)
├─ models/               # trained models, calibrators, metadata
├─ notebooks/            # ad-hoc analysis, EDA, reports
├─ reports/              # figs, md, pdfs
├─ src/
│  ├─ __init__.py
│  ├─ config.py          # paths, constants
│  ├─ utils.py           # shared helpers
│  ├─ data.py            # ingest/clean/label
│  ├─ features.py        # feature engineering
│  ├─ splits.py          # time-aware splits
│  ├─ metrics.py         # metrics incl. calibration
│  ├─ modeling.py        # model training/inference
│  ├─ calibrate.py       # probability calibration
│  ├─ backtest.py        # walk-forward, EV, bankroll
│  ├─ odds.py            # odds parsing, implied probs, vig removal
│  ├─ update_data.py     # refresh pipeline
│  └─ app.py             # Streamlit app
├─ tests/                # pytest unit tests
├─ .gitignore
├─ requirements.txt
├─ README.md
└─ pyproject.toml / setup.cfg (optional)
```

Environment

- Python 3.11+
- `pip install -U pandas numpy pyarrow scikit-learn matplotlib plotly streamlit joblib scipy statsmodels lightgbm xgboost shap tqdm python-dateutil pydantic typer rich pytest`
- Create `.env` if you'll use keys (not required initially).

Acceptance Criteria

- `python -c "import pandas, sklearn, lightgbm; print('ok')"` prints `ok`.
- `pytest -q` runs (even with 0 tests).

Phase 1 — Minimal Dataset (1–2 days)

Goal: Single table with `date, home_team, away_team, home_score, away_score, close_spread`.

Data Source

Start with a **static CSV** (Kaggle or similar). Drop it into `data/raw/odds_games.csv`.

Data Contract (Columns & Types)

- `date` (datetime, **naive** or UTC),
- `home_team`, `away_team` (string, canonical names),
- `home_score`, `away_score` (int),
- `close_spread` (float): **home spread** (negative means home favored). If source uses away spread or book perspective, normalize now.

Quick EDA (notebooks/01_explore_data.ipynb)

- Inspect date ranges, missing values, team name quirks.
- Count per-season games; spot duplicates.

Acceptance Criteria

- 10 random rows hand-checked for sanity.
- README has **column definitions** & quirks.

Phase 2 — Cleaning & Labeling (1–2 days)

Goal: `data/processed/games.parquet` with a `covered_home` label.

Tasks (`src/data.py`)

1. **Load & standardize team names**
2. Build `TEAM_MAP = {"L.A. Clippers": "LA Clippers", "Los Angeles Clippers": "LA Clippers", ...}`
3. Apply to `home_team`, `away_team`.
4. **Compute margins & label**
5. `home_margin = home_score - away_score`
6. `covered_home = int(home_margin + close_spread > 0)`
7. **De-duplication**
8. Dedup on `(date, home_team, away_team)` keeping the last by source priority.
9. **Missing handling**
10. Drop rows missing scores/spread; log counts.
11. **Save** to Parquet; write `dropped_rows.json` with reasons.

CLI Driver

- Optional: `typer` CLI in `data.py`: `python -m src.data --in data/raw/odds_games.csv --out data/processed/games.parquet`

Acceptance Criteria

- Print shape, NA counts; label balance ~45–55%.
- Parquet loads with correct dtypes.

Phase 3 — Feature Engineering v1 (2–4 days)

Goal: Leakage-safe rolling team features and matchup deltas.

Feature Principles

- **No future info:** rolling stats use **only prior games** for that team.
- **Symmetry:** create features for both teams, then compute **home − away** deltas.
- **Recency:** use 5-game and 10-game windows (configurable).

Candidate Inputs (if available in source)

- Team game-level: points for/against, margin, FG%, 3P%, FTA, ORtg/DRtg, pace, turnovers, rebounds.
- Context: rest days, home/away indicator for team, back-to-back (B2B), 3-in-4 days, travel proxy (optional later).

Implementation Steps (`src/features.py`)

1. Sort by `date` per team.
2. For each team, compute rolling means/stds for last N games **shifted by 1** (to exclude current game):

3. `pf_5`, `pa_5`, `margin_5`, `pf_10`, `pa_10`, `margin_10`, etc.
4. Compute **rest days**: `rest = (date - prev_date).days` (clip 0–7+; B2B = rest==1).
5. Join team features back to the game row twice: once as home, once as away (prefix `h_` / `a_`).
6. Build **delta features**: `d_pf_5 = h_pf_5 - a_pf_5`, etc. Keep raw `abs_spread = abs(close_spread)`.
7. Drop any rows with NaNs introduced by early-season windows (or keep but mark `games_played>=N`).
8. Output: `features.parquet` with `X` columns + `y=covered_home` + `meta` (date, teams, spread, scores).

Acceptance Criteria

- `X.shape[0]` ≈ `games - warmup`.
- No NaNs; feature counts between 20–60.

Phase 4 — Time-Aware Splits (½ day)

Goal: Train/val/test by **date** to avoid leakage.

Strategy (`src/splits.py`)

- Choose cutoffs (example):
- Train: up to 2021-06-30
- Val: 2021-07-01 to 2023-06-30
- Test: 2023-07-01+
- Provide helper to return boolean masks by `date`.

Acceptance Criteria

- No overlap; lengths sum to total rows.

Phase 5 — Baselines (1–2 days)

Goal: Beat simple heuristics.

Baseline Rules (`src/modeling.py`)

- **Favorite covers:** pick `covered_home=1` if `close_spread<0`, else `0`.
- **Underdog covers:** inverse of above.

Metrics (`src/metrics.py`)

- Accuracy, ROC-AUC, Brier score, Log loss.
- **Calibration curve**: bin predicted probs (or rule-based 0/1) and compute observed frequency.

Acceptance Criteria

- Print baseline metrics for val & test; snapshot to `reports/baselines.json`.

Phase 6 — First Model: Logistic Regression (1–2 days)

Goal: A fast, interpretable classifier with calibrated probabilities.

Steps (`src/modeling.py`)

1. Standardize features (Z-score) via `sklearn.pipeline.Pipeline`.
2. `LogisticRegression(penalty='l2', C∈{0.1,1,10}, class_weight='balanced'?)`.
3. Fit on **train**, tune C on **val** by log loss.
4. Evaluate on test; dump `model.joblib` and `feature_names.json`.

Acceptance Criteria

- Test log loss < baseline; accuracy ≥ baseline.

Phase 7 — Tree Model + Feature Importance (1–2 days)

Goal: Higher accuracy and insights.

Options

- `XGBoost` or `LightGBM` (faster). Start with LightGBM:

```
num_leaves: 31
max_depth: -1
learning_rate: 0.05
n_estimators: 1000 (early_stopping on val)
subsample: 0.8
colsample_bytree: 0.8
min_child_samples: 20
```

- Use `early_stopping_rounds=50` with a fixed validation set.

Deliverables

- `lgbm_model.txt` or `model_lgbm.joblib` + `importance_*.csv` (gain & permutation).

Acceptance Criteria

- Test log loss improves; plot top 20 features.

Phase 8 — Probability Calibration (½–1 day)

Goal: Probabilities that reflect reality (vital vs. odds).

Methods (`src/calibrate.py`)

- **Platt scaling** (`CalibratedClassifierCV(method='sigmoid')`) or **Isotonic** (more flexible, needs data).
- Fit calibration on **validation set predictions** only; freeze for test.
- Output `calibrator.joblib`.

Diagnostics (`src/metrics.py`)

- Reliability diagram; Expected Calibration Error (ECE).

Acceptance Criteria

- Brier score and ECE improve on val & test.

Phase 9 — Odds Math & EV (1 day)

Goal: Compute expected value vs. book odds.

American Odds → Implied Probability

- If **positive**: `p = 100 / (odds + 100)`
- If **negative**: `p = |odds| / (|odds| + 100)`

Remove Vig (2-way market)

Given book implied probs `p1, p2`: `k = p1 + p2`; **de-vigged**: `p1' = p1/k`, `p2' = p2/k`.

EV per $1 Stake

For price **-110** on your pick with model prob `q`: - Payout per $1 stake: `w = 100/110 = 0.9091` (profit if win) - Loss if lose: `-1` - `EV = q * w + (1-q) * (-1)`

Kelly Fraction (optional)

- Decimal odds `d = 1 + w` → `f* = (q*d - 1) / (d - 1)`; clip at 0–5%.

Acceptance Criteria

- Unit tests in `tests/test_odds.py` covering conversions & EV.

Phase 10 — Backtest & Policy (1–2 days)

Goal: Walk-forward simulation with bankroll & bet sizing.

Walk-Forward (rolling origin)

1. Choose seasons windows (e.g., train up to date *t*, predict next week, advance *t*).
2. For each frontier: train → calibrate on last month/season slice → predict next window.
3. Apply **bet selection rule**, e.g. place bet if `EV > 0.02` and `|q - p_book| > 0.05`.
4. Bankroll update with flat unit or fractional Kelly.

KPIs

- Win rate, CLV (closing line value; optional), ROI, max drawdown, turnover, #bets/week.
- Plot equity curve.

Acceptance Criteria

- `reports/backtest_summary.json` + figures in `reports/figs/`.

Phase 11 — Streamlit MVP (1–2 days)

Goal: Interactive prediction & EV check.

App Structure (`src/app.py`)

- **Inputs**: Select matchup/date from table, or manual team + spread + book odds.
- **Outputs**: Model prob (post-calibration), book implied & de-vigged prob, EV, recommended stake (Kelly α).
- **Panels**: Prediction | Performance (last 30 days) | About.
- Cache loaders with `@st.cache_data`.

Acceptance Criteria

- `streamlit run src/app.py` works; demo past games show sensible numbers.

Phase 12 — Polish & Explainability (2–4 days)

Goal: Credible dashboard and report.

Additions

- Calibration plot, confusion matrix at 0.5 & cost-sensitive thresholds.
- Feature importance (gain & permutation); optional SHAP summary plot.

- "How it works" page outlining data, features, splits, limitations.

Acceptance Criteria

- 3–5 minute live demo flows smoothly.

Phase 13 — Data Refresh & Hygiene (1–2 days)

Goal: One-command refresh for new games.

Script (`src/update_data.py`)

- Append latest `raw` CSV(s) → run `clean` → `features` → (optionally) **incremental** re-train or just predict upcoming.
- Idempotent: safe to run multiple times.

Acceptance Criteria

- `python -m src.update_data` completes and artifacts updated.

Phase 14 — Documentation & Final Report (1–2 days)

Goal: Reproducible write-up for grading.

`reports/final.md` Outline

- Problem & market framing (spread vs. moneyline; vig).
- Data pipeline (contracts, cleaning, labeling, features).
- Splits & rationale; leakage prevention.
- Models, tuning, calibration.
- Backtest policy & KPIs; sensitivity to thresholds.
- Limitations & next steps.

Acceptance Criteria

- Export to PDF with embedded plots; someone else can reproduce.

Stretch Roadmap (pick 1–3)

- **Injuries/travel** proxies (scrape or external API; add binary injury flags for stars; rolling "star absence" feature).
- **Player-level features**: On/off, RAPM, EPM (requires richer data; join to team-game table).
- **Hyperparameter tuning**: Optuna with **time-series CV** (blocked splits).

- **Model stacking**: LogReg on top of tree/logreg/elo features.

Concrete Code Skeletons

`src/config.py`

```
from pathlib import Path
ROOT = Path(__file__).resolve().parents[1]
DATA_RAW = ROOT / "data/raw"
DATA_PROCESSED = ROOT / "data/processed"
MODELS_DIR = ROOT / "models"
SEED = 42
```

`src/data.py`

```
from __future__ import annotations
import pandas as pd
from .config import DATA_RAW, DATA_PROCESSED

TEAM_MAP = {
    "L.A. Clippers": "LA Clippers",
    "Los Angeles Clippers": "LA Clippers",
    # ... fill out for all quirks
}

def load_raw(path: str | None = None) -> pd.DataFrame:
    path = path or (DATA_RAW / "odds_games.csv")
    df = pd.read_csv(path)
    df['date'] = pd.to_datetime(df['date']).dt.tz_localize(None)
    return df

def normalize_teams(df: pd.DataFrame) -> pd.DataFrame:
    for c in ['home_team','away_team']:
        df[c] = df[c].map(lambda x: TEAM_MAP.get(x, x))
    return df

def label_and_clean(df: pd.DataFrame) -> pd.DataFrame:
    df = df.dropna(subset=['home_score','away_score','close_spread']).copy()
    df['home_margin'] = df['home_score'] - df['away_score']
    df['covered_home'] = ((df['home_margin'] + df['close_spread']) >
0).astype(int)
    df = df.drop_duplicates(subset=['date','home_team','away_team'],
keep='last')
    return df
```

```
def run():
    df = load_raw()
    df = normalize_teams(df)
    df = label_and_clean(df)
    out = DATA_PROCESSED / "games.parquet"
    df.to_parquet(out, index=False)
    print(out, df.shape)

if __name__ == "__main__":
    run()
```

`src/features.py` **(rolling features)**

```
import pandas as pd
import numpy as np
from .config import DATA_PROCESSED

ROLLS = [5, 10]

def add_team_rolls(games: pd.DataFrame) -> pd.DataFrame:
    # long format for stacking home/away as 'team' rows
    home =
games[['date','home_team','home_score','away_score']].rename(columns={
        'home_team':'team', 'home_score':'pf', 'away_score':'pa'})
    away =
games[['date','away_team','away_score','home_score']].rename(columns={
        'away_team':'team', 'away_score':'pf', 'home_score':'pa'})
    long = pd.concat([home, away], ignore_index=True)
    long = long.sort_values(['team','date'])
    long['margin'] = long['pf'] - long['pa']
    long['prev_date'] = long.groupby('team')['date'].shift(1)
    long['rest'] = (long['date'] - long['prev_date']).dt.days.clip(lower=0)
    for n in ROLLS:
        for col in ['pf','pa','margin']:
            long[f'{col}_m{n}'] = long.groupby('team')
[col].shift(1).rolling(n).mean()
        long[f'games_played_m{n}'] = long.groupby('team')['pf'].cumcount()
    return long

def build_features(games: pd.DataFrame) -> pd.DataFrame:
    long = add_team_rolls(games)
    # split back into home/away features and merge
    h = long.rename(columns={c: f'h_{c}' for c in long.columns if c not in
['team','date']})
    a = long.rename(columns={c: f'a_{c}' for c in long.columns if c not in
```

```
['team','date']})
    df = games.merge(h, left_on=['home_team','date'], right_on=['team','date'])
              .merge(a, left_on=['away_team','date'], right_on=['team','date'],
suffixes=("",""))
    # deltas
    for base in
['pf_m5','pa_m5','margin_m5','pf_m10','pa_m10','margin_m10','rest']:
        df[f'd_{base}'] = df[f'h_{base}'] - df[f'a_{base}']
    # drop early-season NaNs
    mask = df[[c for c in df.columns if c.endswith('m5')]].notna().all(axis=1)
    df = df[mask].copy()
    return df
```

`src/splits.py`

```
import pandas as pd

def date_masks(dates: pd.Series, train_end: str, val_end: str):
    d = pd.to_datetime(dates)
    tr = d <= pd.Timestamp(train_end)
    va = (d > pd.Timestamp(train_end)) & (d <= pd.Timestamp(val_end))
    te = d > pd.Timestamp(val_end)
    return tr, va, te
```

`src/modeling.py` **(LogReg example)**

```
import joblib
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss


def fit_logreg(Xtr, ytr, Xva, yva, Cs=(0.1,1,10)):
    best = None
    best_ll = 1e9
    for C in Cs:
        pipe = Pipeline([
            ('scaler', StandardScaler()),
            ('clf', LogisticRegression(max_iter=200, C=C))
        ])
        pipe.fit(Xtr, ytr)
        p = pipe.predict_proba(Xva)[:,1]
        ll = log_loss(yva, p)
```

```
        if ll < best_ll:
            best_ll, best = ll, pipe
    return best

def save_model(pipe, path):
    joblib.dump(pipe, path)
```

`src/calibrate.py`

```
import joblib
from sklearn.calibration import CalibratedClassifierCV

def calibrate_on_val(model, Xva, yva, method='isotonic'):
    cal = CalibratedClassifierCV(model, method=method, cv='prefit')
    cal.fit(Xva, yva)
    return cal
```

`src/odds.py`

```
def amer_to_prob(odds: int) -> float:
    return 100/(odds+100) if odds>0 else abs(odds)/(abs(odds)+100)

def devig_twoway(p1: float, p2: float):
    k = p1 + p2
    return p1/k, p2/k

def ev_per_dollar(q: float, price: int) -> float:
    # q: model win prob for the selected side
    w = (100/abs(price)) if price<0 else (price/100)
    return q*w - (1-q)*1.0
```

`src/backtest.py` **(sketch)**

```
from dataclasses import dataclass
import pandas as pd

@dataclass
class Policy:
    min_ev: float = 0.02
    kelly_alpha: float = 0.25  # fraction of Kelly

# Assume we already have model probs q and book price (american)

def select_bets(df: pd.DataFrame, policy: Policy):
```

```
    sel = df.query('ev > @policy.min_ev').copy()
    # bet size via Kelly
    # decimal odds d = 1 + w, w per ev module
    return sel
```

Testing Checklist (minimum)

- `tests/test_data.py`: label correctness on crafted rows.
- `tests/test_features.py`: rolling uses only prior rows (check with a toy team schedule).
- `tests/test_odds.py`: conversions & EV formulas.
- `tests/test_splits.py`: masks produce no overlap.

Evaluation & Reporting

- Always report **val first**, then **test once**.
- Save: `reports/metrics_{model}_{date}.json`, and plots under `reports/figs/`.

Core plots: ROC, reliability, feature importance, equity curve (backtest).

Operational Notes

- **Random seeds**: fix for reproducibility; log versions.
- **Config**: centralize constants in `config.py`.
- **Logging**: use `rich` prints or `logging` to show dataset sizes, NA drops, and cutoffs.
- **Data contracts**: write them in README (schema + units + semantics).

Next-Step Script Order (copy/paste)

1. `python -m src.data`
2. Open `notebooks/02_check_games.ipynb` → sanity check.
3. `python - <<'PY'` (quick smoke for features)

```
from src.data import run as clean_run
from src.features import build_features
import pandas as pd

df = pd.read_parquet('data/processed/games.parquet')
fx = build_features(df)
```

```
fx.to_parquet('data/processed/features.parquet', index=False)
print(fx.shape, fx.columns[:10])
```

PY

4. Train baseline/logreg in `notebooks/03_train_logreg.ipynb` or scriptify.
5. Calibrate on val → save calibrator.
6. Backtest script → `reports/backtest_summary.json`.
7. `streamlit run src/app.py`.

Acceptance Gates (Go/No-Go)

- **Data**: clean Parquet; label sanity; team names unified.
- **Features**: no NaNs; leakage audit passed.
- **Model**: test log loss < baseline; Brier improved after calibration.
- **Backtest**: positive ROI with reasonable #bets; drawdown tolerable.
- **App**: loads in < 2s; shows EV and recommendation.

Appendix A — Leakage Audit Tips

- Ensure all rolling stats use `.shift(1)` before rolling.
- When merging team features back, **join on (team,date)** where team rows are pre-shifted.
- No season-end summary stats inside features.

Appendix B — Odds Edge Thresholds (Examples)

- Bet if `EV > 1.5%` **and** `|q - p_devig| > 4%`.
- Cap stake at 1–2% bankroll (fractional Kelly or flat).

Appendix C — Minimum Feature Set (If Data Is Sparse)

- `d_margin_m5`, `d_margin_m10`, `d_pf_m5`, `d_pa_m5`, `d_rest`, `abs_spread`, `close_spread`.

Appendix D — Streamlit UI Sketch

- Sidebar: select date/game or manual inputs.
- Main: probability card, EV card, small calibration chart, bet sizing.
- Tab: Performance (rolling accuracy last N days), About.

You can now copy these files into your repo and implement incrementally.