NBA Rolling Metrics Final Essay

Kaleb Coleman

Introduction

NBA home-court advantage varies across seasons, and single-game box scores are often too noisy to distinguish true team form from short-term randomness. This project uses NBA game data as a realistic setting to practice end-to-end statistical modeling and data engineering, with an emphasis on feature construction, leakage control, and model evaluation. Starting from raw team box score data, I construct pre-game rolling performance metrics that summarize only prior information available before each game. This problem mirrors real-world forecasting settings where only historical information is available at decision time, and improper feature construction can overstate performance.

Using regular-season NBA data from 2002–2025, pulled via `hoopR::load_nba_team_box`, I align home and away team rows at the game level and build possession-based rolling features over 3-, 5-, and 10-game windows. These rolling metrics include net rating, effective field goal percentage, turnover percentage, and rebounding measures. All features are explicitly lagged to ensure that no post-game information leaks into the predictors.

The analysis focuses on two core questions: (1) which rolling matchup differentials (home minus away) are most strongly associated with the probability that the home team wins, and (2) how much predictive improvement these rolling metrics provide beyond a majority-class baseline. To address these questions, I fit one interpretable inferential model—a logistic regression with estimated odds ratios and confidence intervals—and a set of penalized predictive models using ridge and elastic net regression applied to the full rolling feature set. Model performance is evaluated using a final-season holdout, with resampling results summarized to assess stability rather than relying on a single train–test split.

All figures and tables correspond directly to the modeling pipeline, with attention to data coverage, missingness arising from early-season games, and the construction of rolling windows. The goal is not to maximize predictive accuracy, but to demonstrate principled feature engineering and evaluation in a time-ordered sports analytics setting.

Data Evaluation (Materials)

Data source and scope

Team box scores are pulled directly from the NBA Stats API via the `hoopR` package (`hoopR::load_nba_team_box`). I restrict to regular-season games (season_type = 2) across 2002–2025, keep only team rows with valid NBA IDs (<= 30), and recode home/away indicators so each observation represents a paired home–away matchup at the game level. Basic renaming makes the raw box-score columns consistent before pairing opponents for possession and efficiency calculations.

Variables, types, and descriptions

Table 1: Row/column counts and season coverage.

Dataset	Rows	Columns	Seasons
Raw team box (team_box_raw)	57,378	57	2002-2025
Modeling table (games_combined)	28,671	118	2002-2025

Table 2: Key modeling variables with descriptions and types

Name	Description	Type
home_win	1 if home team wins, 0 otherwise	binary
matchup_net_roll3	Home net rating roll3 minus away net rating roll3	numeric
matchup_net_roll5	Home net rating roll5 minus away net rating roll5	numeric
matchup_net_roll10	Home net rating roll10 minus away net rating roll10	numeric
matchup_ortg_drtg_roll5	Home ortg roll5 minus away drtg roll5	numeric
efg_roll5_diff	Home eFG% roll5 minus away eFG% roll5	numeric
orb_roll5_diff	Home offensive reb% roll5 minus away offensive reb% roll5	numeric
orb_roll5_ratio	Ratio of home to away offensive reb% roll5	numeric
tov_roll5_diff	Home turnover% roll5 minus away turnover% roll5	numeric
tov_roll5_ratio	Ratio of home to away turnover% roll5	numeric
Elo_Diff_season	Season-reset Elo differential (home minus away)	numeric
Elo_Diff_alltime	All-time Elo differential (home minus away)	numeric

The coverage table summarizes how many rows/columns are in the raw team box scores versus the modeling table and which seasons are included. The variable table lists the core predictors and their types. All matchup variables are constructed using only lagged team information available prior to the game date.

Missingness and handling of early-season games

Lagged rolling features are undefined for each team's first game of a season, so the only missing values come from those initial rows. I dropped those rows for modeling to avoid inventing prior performance; imputation was not pursued because these missing values arise from the absence of prior games rather than measurement error.

Table 3: Rolling features have missing values only for first-team games; these rows were dropped.

Min missing	Max missing
337	407

This missingness summary confirms that NA values appear only because rolling windows are undefined for each team's first game of a season.

Methods (Modeling)

Data engineering pipeline

Raw team box scores were cleaned and aligned so each game has a home and away row with consistent columns. I derived possession-based efficiencies and created rolling windows that are lagged by one game to avoid leakage. Matchup features are computed as home minus away (or ratios) to reflect the pre-game edge. Early-season rows with undefined rolling windows were removed to keep the feature set fully observed for modeling.

The pipeline is implemented in `R/AdvancedStats.R` via `run_advanced_stats()`, which pulls the API data, constructs rolling features, and saves the modeling objects to `data/nba_games_modeling.RData`. This keeps the feature engineering step reproducible and separate from modeling.

Software and packages

Core data manipulation and plotting used `dplyr`, `tidyr`, `readr`, `magrittr`, and `ggplot2`. Rolling feature creation and scaling relied on base R and `scales`, while model fitting used `glm` (base R), `glmnet` (ridge/elastic net), and `pROC` (AUC). Results tables were formatted with `broom` and `knitr`.

Inferential model

I fit a logistic regression on pre-game matchup differentials - net rating, eFG%, and turnover% - using five-game lagged rolling averages for both teams. Logistic regression provides a natural probabilistic interpretation for binary outcomes while remaining interpretable under correlated predictors. Coefficients were translated into odds ratios with 95% confidence intervals. Because the rolling windows are lagged (no current-game data), predictors stay free of leakage. Feature distributions by win/loss were examined descriptively to confirm expected directional relationships prior to model fitting.

Predictive models and evaluation

Predictive benchmarking used ridge logistic regression and elastic net (alpha tuned over a small grid) on the rolling feature set. Models trained on seasons up to 2024 and evaluated on 2025 as a holdout. Cross-validation was used only for model selection and stability assessment, while all reported performance is based on the untouched final-season holdout. Elo-style ratings were tested as an add-on but did not materially change accuracy, so they are not emphasized in the results.

I also bootstrapped holdout accuracy for the ridge model at the default 0.5 cutoff and at a tuned threshold. This provides a sense of variability around the holdout estimate and whether a different classification threshold meaningfully improves accuracy. Bootstrapping emphasizes uncertainty in single-season evaluation rather than overstating point estimates.

Analysis Results

Model estimates and predictive performance

Table 4: Holdout-season performance (train: seasons < 2025; test: 2025).

Feature set	Model	Accuracy	AUC
full	Elastic net	0.657	0.714
full	Ridge logistic	0.662	0.714
full	Random Forest	0.645	0.693
inferential	Logistic (lean)	0.643	0.677
none	Majority baseline	0.553	NA

The table compares only the lean inferential logistic and two penalized models on the full feature set. The inferential logistic row (feature set = inferential) has holdout accuracy around 0.64 - within a few points of the full ridge/elastic - so it is competitive while remaining interpretable. Ridge and elastic net use all rolling features (including Elo if present) to maximize predictive power. Accuracy and AUC together show modest lift over the baseline, with ridge and elastic essentially tied.

Table 5: Cross-validation summary (training seasons only).

model	feature_set	cv_accuracy	cv_auc
Elastic net	full	0.661	0.708
Ridge logistic	full	0.661	0.708
Random Forest	full	0.659	0.699
Logistic (lean)	full	0.628	0.649

Table 6: Bootstrapped holdout accuracy (ridge model).

model	threshold	type	holdout_accuracy	mean_accuracy	ci_low	ci_high
Ridge logistic	0.50	default	0.662	0.663	0.634	0.691
Ridge logistic	0.55	tuned	0.670	0.670	0.642	0.696

Model performance is summarized numerically above; a ROC diagnostic is shown in the Supplementary Figures section. The ROC curve sits only modestly above the diagonal, indicating limited but real discrimination (consistent with mid-0.60s AUC). This reinforces that the model captures signal but remains far from highly confident classification on single-game outcomes.

The bootstrap results show a best tuned-threshold holdout accuracy around 0.67, which is a modest but consistent lift over the default 0.5 cutoff. Although the absolute improvement over the majority baseline is modest, the lift is meaningful given the inherent randomness of single NBA games and the strict pre-game information constraint.

Odds ratios for matchup differentials

Table 7: Logistic regression odds ratios for rolling matchup features.

Term	Odds ratio	CI low	CI high	p-value
matchup_net_roll5	1.013	1.009	1.018	0.000
matchup_ortg_drtg_roll5	1.008	1.004	1.013	0.000
orb_roll5_diff	0.948	0.573	1.569	0.837
tov_roll5_diff	1.837	0.610	5.530	0.279
efg_roll5_diff	0.788	0.312	1.993	0.616
Elo_Diff_alltime	1.003	1.003	1.003	0.000
Elo_Diff_season	1.002	1.002	1.003	0.000

Net rating differentials exhibit the strongest and most precisely estimated association with home-win probability among the examined features; turnover differential is in the expected direction but smaller.

Discussion

The five-game matchup differentials (net rating, eFG%, turnover%) behaved as expected: better recent shooting and efficiency increased home-win odds, and sloppy play eroded them. These findings address the first research question by showing that recent efficiency-based differentials, particularly net rating, are most strongly associated with win probability. The lean logistic nearly matched the penalized models, indicating most signal was captured by a small set of five-game matchup features. Ridge and elastic net added a slight lift and smoother calibration but still achieved mid-0.60s AUC, consistent with noisy single-game outcomes. The relatively modest performance highlights how easily leakage could inflate results if lagging were not enforced, reinforcing the importance of time-aware feature construction. Limitations: early-season NAs were dropped, not imputed; no rest/travel controls; the quick Elo was noisy and offered little gain; cross-validation summarized model performance on training seasons while the final holdout remained the primary metric.

Conclusions

Five-game rolling matchup differentials provided the best balance of recency and stability. Net rating and offensive rebounding were the clearest drivers of home-win odds, with turnover differential in the expected direction. Penalized models improved modestly over the majority baseline but remained bounded by game-level volatility, while the lean logistic stayed interpretable and close in accuracy. Future work could add rest/travel effects, richer interactions, and better probability calibration, and should revisit window length as new seasons arrive.

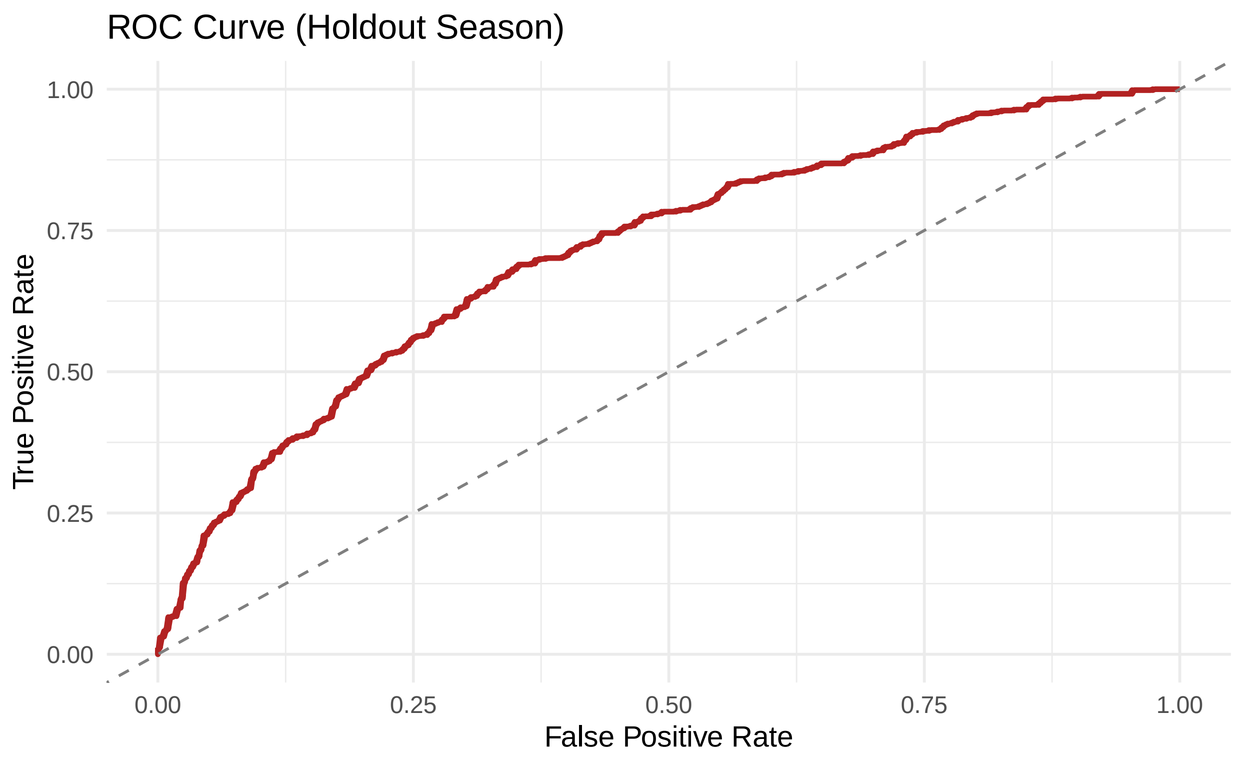

Figure 1: ROC curve for ridge logistic model (holdout season).

Supplementary Figures

Citations

- SportsDataverse. (2024). *hoopR: Access to NBA data from stats.nba.com* (R package). https://CRAN.R-project.org/package=hoopR
- National Basketball Association. (2024). *NBA Stats API*. https://stats.nba.com/
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2nd ed.). Springer. https://www.statlearning.com/

Appendix: Code Listings

AdvancedStats.R

```
# --- advancedStats.R ---------------------------------------------------------
# Builds per-team-per-game advanced stats + season-to-date (lagged) features.
# Produces:
#   - model_df            : per-team/per-game advanced stats
#   - model_df_sdt        : per-team/per-game with season-to-date (lag-1) features
#   - home_df_sdt         : home-team-only rows with SDT features (1 row per game)
#   - model_df_roll       : per-team/per-game with rolling averages (no leakage)
#   - home_df_roll        : home-team-only rolling averages
#   - games_combined      : one row per game, home & away features for modeling
```

```r
suppressPackageStartupMessages({
  library(hoopR)
  library(dplyr)
  library(tidyr)
  library(lubridate)
  library(slider)
})

#' Run advanced stats pipeline
#'
#' Pulls NBA team box scores, constructs rolling features, and saves
#' modeling objects to an RData file.
#'
#' @param seasons Integer vector of seasons to include.
#' @param output_file Output path for the saved RData file.
#' @export
run_advanced_stats <- function(seasons = 2002:2025,
                               output_file = "data/nba_games_modeling.RData") {
  # -------- PARAMETERS --------
  # Expand historical coverage (will increase download time/size; requires network).
  SEASONS <- seasons
  OUTPUT_FILE <- output_file

  # -------- LOAD RAW TEAM BOX --------
  team_box_raw <- load_nba_team_box(seasons = SEASONS) %>%
    filter(season_type == 2)   # 2 = Regular Season

  # -------- MINIMAL COLUMNS & RENAME --------
  tb <- team_box_raw %>%
    transmute(
      game_id, season,
      game_date = as.Date(game_date),
      team_id,
      team = team_abbreviation,
      team_name,
      side = if_else(tolower(team_home_away) == "home", "home", "away"),
      pts = team_score,
      fgm = field_goals_made, fga = field_goals_attempted,
      fg3m = three_point_field_goals_made, fg3a = three_point_field_goals_attempted,
      ftm = free_throws_made, fta = free_throws_attempted,
      orb = offensive_rebounds, drb = defensive_rebounds, trb = total_rebounds,
      ast = assists, stl = steals, blk = blocks, tov = turnovers
    ) %>%
    filter(!is.na(team_id) & team_id <= 30)

  # -------- ADD OPPONENT STATS (PAIR HOME/AWAY) --------
  opp <- tb %>%
    transmute(
      game_id, opp_side = side,
      opp_team_id = team_id, opp_team = team,
      opp_pts = pts, opp_fga = fga, opp_fta = fta,
      opp_orb = orb, opp_drb = drb, opp_trb = trb, opp_tov = tov,
      opp_fgm = fgm, opp_fg3m = fg3m, opp_fg3a = fg3a
    )

  df <- tb %>%
    mutate(desired_opp_side = if_else(side == "home", "away", "home")) %>%
    left_join(opp, by = c("game_id" = "game_id", "desired_opp_side" = "opp_side")) %>%
    # -------- ADVANCED STATS (same-game) --------
    mutate(
      efg   = (fgm + 0.5*fg3m) / pmax(fga, 1),
      ts    = pts / (2 * pmax(fga + 0.44*fta, 1)),
      ftr   = fta / pmax(fga, 1),
      x3par = fg3a / pmax(fga, 1),

      poss_team = fga + 0.4*fta - orb + tov,
      poss_opp  = opp_fga + 0.4*opp_fta - opp_orb + opp_tov,
      poss      = 0.5 * (poss_team + poss_opp),

      ortg = 100 * pts     / pmax(poss, 1),
      drtg = 100 * opp_pts / pmax(poss, 1),
      net_rating = ortg - drtg,

      orb_perc = orb / pmax(orb + opp_drb, 1),
      drb_perc = drb / pmax(drb + opp_orb, 1),
      tov_perc = tov / pmax(poss, 1),

      margin = pts - opp_pts,
      win = as.integer(margin > 0),
      is_home = as.integer(side == "home")
    )

  # -------- MODEL TABLE (same-game; useful for EDA only) --------
  model_df <- df %>%
    select(
      game_id, season, game_date, team_id, team, team_name, is_home, win, margin,
      pts, opp_pts,
      efg, ts, ftr, x3par,
      ortg, drtg, net_rating,
      orb_perc, drb_perc, tov_perc,
      ast, trb, stl, blk, tov,
      fgm, fga, fg3m, fg3a, ftm, fta
    )
```

```r
# -------- ROLLING MEANS (lagged, no leakage) --------
roll_features <- c(
  "efg", "ts", "ftr", "x3par",
  "ortg", "drtg", "net_rating",
  "orb_perc", "drb_perc", "tov_perc",
  "margin", "pts", "opp_pts"
)

roll_windows <- c(3, 5, 10)
roll_pattern <- paste0("roll(", paste(roll_windows, collapse = "|"), ")$")

lagged_mean <- function(x) {
  x <- head(x, -1)
  if (!length(x)) {
    return(NA_real_)
  }
  mean(x, na.rm = TRUE)
}

df_roll <- df %>%
  arrange(season, team, game_date) %>%
  group_by(season, team) %>%
  mutate(games_played = dplyr::lag(row_number(), default = 0))

for (window in roll_windows) {
  df_roll <- df_roll %>%
    mutate(
      across(
        all_of(roll_features),
        ~ slider::slide_dbl(
          .x,
          lagged_mean,
          .before = window,
          .complete = FALSE
        ),
        .names = paste0("{.col}_roll", window)
      )
    )
}

df_roll <- df_roll %>% ungroup()

model_df_roll <- df_roll %>%
  select(
    game_id, season, game_date, team_id, team, team_name, is_home, win, margin,
    games_played,
    pts, opp_pts,
    efg, ts, ftr, x3par,
    ortg, drtg, net_rating,
    orb_perc, drb_perc, tov_perc,
    ast, trb, stl, blk, tov,
    fgm, fga, fg3m, fg3a, ftm, fta,
    matches(roll_pattern)
  )

home_df_roll <- model_df_roll %>% filter(is_home == 1)
away_df_roll <- model_df_roll %>% filter(is_home == 0)

valid_team <- function(team_id) {
  !is.na(team_id) & team_id <= 30
}

compute_elo_core <- function(df, k = 20, baseline = 1500, hfa = 100, reset_each_season = FALS
  required_cols <- c("game_id", "game_date", "season", "home_team", "away_team", "home_win")
  missing <- setdiff(required_cols, names(df))
  if (length(missing)) {
    stop("Missing columns for Elo: ", paste(missing, collapse = ", "))
  }

  df <- df %>%
    select(all_of(required_cols)) %>%
    arrange(game_date, game_id)

  split_list <- if (reset_each_season) split(df, df$season) else list(all = df)
  results <- vector("list", length(split_list))
  names(results) <- names(split_list)

  for (s in seq_along(split_list)) {
    chunk <- split_list[[s]]
    teams <- unique(c(chunk$home_team, chunk$away_team))
    ratings <- setNames(rep(baseline, length(teams)), teams)
    out <- chunk %>% mutate(Elo_H = NA_real_, Elo_A = NA_real_)

    for (i in seq_len(nrow(chunk))) {
      h <- chunk$home_team[i]
      a <- chunk$away_team[i]
      win_h <- as.logical(chunk$home_win[i])

      out$Elo_H[i] <- ratings[h]
      out$Elo_A[i] <- ratings[a]

      r_h <- ratings[h] + hfa
      r_a <- ratings[a]
      e_h <- 1 / (1 + 10^((r_a - r_h) / 400))
      s_h <- if (isTRUE(win_h)) 1 else 0

      ratings[h] <- ratings[h] + k * (s_h - e_h)
      ratings[a] <- ratings[a] + k * ((1 - s_h) - (1 - e_h))
    }

    results[[s]] <- out %>% mutate(Elo_Diff = Elo_H - Elo_A)
  }

  bind_rows(results)
}

home_roll_wide <- home_df_roll %>%
  filter(valid_team(team_id)) %>%
  select(-is_home) %>%
  rename_with(~ paste0("home_", .), -c(game_id, season, game_date, win, margin)) %>%
  rename(home_win = win, home_margin = margin)

away_roll_wide <- away_df_roll %>%
  filter(valid_team(team_id)) %>%
  select(-is_home) %>%
  rename_with(~ paste0("away_", .), -c(game_id, season, game_date, win, margin)) %>%
  rename(away_win = win, away_margin = margin)

games_combined_full <- home_roll_wide %>%
  inner_join(away_roll_wide, by = c("game_id", "season", "game_date"), suffix = c("", "")) %>
  filter(valid_team(home_team_id), valid_team(away_team_id))

safe_ratio <- function(num, denom) {
  ifelse(is.na(denom) | abs(denom) < 1e-6, NA_real_, num / denom)
}

games_combined <- games_combined_full %>%
  select(
    game_id, season, game_date,
    home_team_id, home_team, home_team_name,
    away_team_id, away_team, away_team_name,
    home_win,
    matches(paste0("^home_.*_", roll_pattern)),
    matches(paste0("^away_.*_", roll_pattern))
  )

for (window in roll_windows) {
  suf <- paste0("_roll", window)
  games_combined[[paste0("matchup_ortg_drtg", suf)]] <- games_combined[[paste0("home_ortg", s
  games_combined[[paste0("matchup_net", suf)]] <- games_combined[[paste0("home_net_rating", s
  games_combined[[paste0("orb", suf, "_diff")]] <- games_combined[[paste0("home_orb_perc", su
  games_combined[[paste0("orb", suf, "_ratio")]] <- safe_ratio(games_combined[[paste0("home_c

  games_combined[[paste0("drb", suf, "_diff")]] <- games_combined[[paste0("home_drb_perc", su
  games_combined[[paste0("drb", suf, "_ratio")]] <- safe_ratio(games_combined[[paste0("home_c

  games_combined[[paste0("tov", suf, "_diff")]] <- games_combined[[paste0("home_tov_perc", su
  games_combined[[paste0("tov", suf, "_ratio")]] <- safe_ratio(games_combined[[paste0("home_t
}

# -------- ELO RATINGS (season-reset and all-time) --------
elo_base <- games_combined %>%
  select(game_id, game_date, season, home_team, away_team, home_win)

elo_season <- compute_elo_core(elo_base, reset_each_season = TRUE) %>%
  rename(Elo_H_season = Elo_H, Elo_A_season = Elo_A, Elo_Diff_season = Elo_Diff)

elo_alltime <- compute_elo_core(elo_base, reset_each_season = FALSE) %>%
  rename(Elo_H_alltime = Elo_H, Elo_A_alltime = Elo_A, Elo_Diff_alltime = Elo_Diff)

games_combined <- games_combined %>%
  left_join(elo_season %>% select(game_id, Elo_H_season, Elo_A_season, Elo_Diff_season), by =
  left_join(elo_alltime %>% select(game_id, Elo_H_alltime, Elo_A_alltime, Elo_Diff_alltime),

# Predictor-only version for downstream use without leakage targets
games_combined_features <- games_combined %>%
  select(-home_win)

# -------- SEASON-TO-DATE (lagged) FEATURES (NO LEAKAGE) --------
model_df_sdt <- df %>%
  arrange(season, team, game_date) %>%
  group_by(season, team) %>%
  mutate(
    gp          = row_number(),
    efg_sdt     = lag(cummean(efg)),
    ts_sdt      = lag(cummean(ts)),
    ortg_sdt    = lag(cummean(ortg)),
    drtg_sdt    = lag(cummean(drtg)),
    net_sdt     = ortg_sdt - drtg_sdt,
    orb_sdt     = lag(cummean(orb_perc)),
    drb_sdt     = lag(cummean(drb_perc)),
    tov_sdt     = lag(cummean(tov_perc))
  ) %>%
  ungroup() %>%
  select(
    game_id, season, game_date, team_id, team, team_name, is_home, win,
    gp,
    efg_sdt, ts_sdt, ortg_sdt, drtg_sdt, net_sdt, orb_sdt, drb_sdt, tov_sdt
  )

# -------- PERSIST MODELING OBJECTS --------
model_objects <- c(
  "SEASONS",
  "team_box_raw",
  "games_combined",
  "games_combined_full",
  "games_combined_features",
  "model_df_sdt"
)

save(list = model_objects, file = OUTPUT_FILE, compress = "xz")
message("Saved modeling data to ", OUTPUT_FILE)
}

if (identical(Sys.getenv("RUN_ADVANCED_STATS"), "true")) {
run_advanced_stats()
}
```

EloPlot.R

```
# --- EloPlot.R ---------------------------------------------------------------
# Quick Elo trajectory plot for selected teams.
# Reads nba_games_modeling.RData (with Elo columns from AdvancedStats) and
# saves outputs/elo_trends.png.

suppressPackageStartupMessages({
  library(dplyr)
  library(ggplot2)
  library(tidyr)
})

#' Run Elo plot
#'
#' Generates a season-reset Elo trajectory plot for selected teams.
#'
#' @param plot_teams Character vector of team abbreviations.
#' @export
run_elo_plot <- function(plot_teams = c("PHX", "OKC", "LAL", "BOS", "GSW", "MIA")) {
  if (!file.exists("data/nba_games_modeling.RData")) {
    stop("data/nba_games_modeling.RData not found. Run R/AdvancedStats.R first.")
  }

  load("data/nba_games_modeling.RData")
  if (!all(c("Elo_H_season", "Elo_A_season") %in% names(games_combined))) {
    stop("Elo columns not found in games_combined. Re-run R/AdvancedStats.R to add Elo.")
  }

  elo_long <- games_combined %>%
    transmute(
      game_date = as.Date(game_date),
      season,
      team = as.character(home_team),
      elo = Elo_H_season
    ) %>%
    bind_rows(
      games_combined %>%
        transmute(
          game_date = as.Date(game_date),
          season,
          team = as.character(away_team),
          elo = Elo_A_season
        )
    ) %>%
    filter(!is.na(team), !is.na(elo))

  elo_plot <- elo_long %>%
    filter(team %in% plot_teams) %>%
    ggplot(aes(x = game_date, y = elo, group = interaction(team, season))) +
    geom_line(color = "steelblue", linewidth = 0.9, alpha = 0.8) +
    facet_wrap(~ team, ncol = 3, scales = "free_y") +
    labs(title = "Season-reset Elo trajectories (selected teams)", x = "Date", y = "Elo (seaso
    theme_minimal(base_size = 11)

  dir.create("outputs", showWarnings = FALSE)
  ggsave("outputs/elo_trends.png", elo_plot, width = 10, height = 6, dpi = 300)
  message("Saved outputs/elo_trends.png")
}

if (identical(Sys.getenv("RUN_ELO_PLOT"), "true")) {
  run_elo_plot()
}
```

InferentialModel.R

```
# --- InferentialModel.R -------------------------------------------------------
# Fits logistic regression on rolling matchup differentials and saves odds ratios.
# Requires nba_games_modeling.RData (created by R/AdvancedStats.R).

suppressPackageStartupMessages({
  library(dplyr)
```

```r
  library(readr)
  library(broom)
})

#' Run inferential model
#'
#' Fits the logistic regression on matchup differentials and saves
#' the fitted model and odds-ratio table to outputs/.
#'
#' @export
run_inferential_model <- function() {

if (!file.exists("data/nba_games_modeling.RData")) {
  stop("data/nba_games_modeling.RData not found. Run R/AdvancedStats.R first.")
}

load("data/nba_games_modeling.RData")

# Prefer Elo-augmented data if present
if (file.exists("outputs/games_with_elo.rds")) {
  games_df <- readRDS("outputs/games_with_elo.rds")
} else {
  games_df <- games_combined
}

games_model <- games_df %>%
  mutate(season_start = if (is.numeric(season)) as.integer(season) else as.integer(substr(seas

  feat <- c(
    "matchup_net_roll5", "matchup_ortg_drtg_roll5",
    "orb_roll5_diff", "tov_roll5_diff",
    "efg_roll5_diff"
  )
# Add an Elo differential if available for interpretability
elo_opts <- c("Elo_Diff_alltime", "Elo_Diff_season")
elo_to_use <- elo_opts[elo_opts %in% names(games_model)]
feat <- unique(c(feat, elo_to_use))

  model_df <- games_model %>%
    mutate(efg_roll5_diff = home_efg_roll5 - away_efg_roll5) %>%
    select(home_win, season_start, all_of(feat)) %>%
  mutate(across(all_of(feat), as.numeric)) %>%
  filter(if_all(all_of(feat), ~ is.finite(.)))

# Train on all seasons prior to the most recent (hold out latest for prediction/plots)
holdout_season <- max(model_df$season_start, na.rm = TRUE)
train <- filter(model_df, season_start < holdout_season)
```

```r
logit_fit <- glm(home_win ~ . - season_start, data = train, family = binomial())

or_table <- broom::tidy(logit_fit, conf.int = TRUE, exponentiate = TRUE) %>%
  filter(term != "(Intercept)") %>%
  transmute(
    term,
    odds_ratio = estimate,
    ci_low = conf.low,
    ci_high = conf.high,
    p_value = p.value
  )

dir.create("outputs", showWarnings = FALSE)
saveRDS(logit_fit, "outputs/logit_fit.rds")
write_csv(or_table, "outputs/logit_odds_ratios.csv")

  message("Saved inferential results to outputs/logit_fit.rds and outputs/logit_odds_ratios.cs
  print(or_table)

}

if (identical(Sys.getenv("RUN_INFERENTIAL_MODEL"), "true")) {
  run_inferential_model()
}
```

Load_data.R

```r
library(hoopR)
library(dplyr)
library(tidyr)

# Basic team box-score snapshot for reference; renamed to avoid clashes with modeling objects.
team_box_raw <- load_nba_team_box(seasons = 2020:2025) %>%
  filter(season_type == 2)

home_stats_raw <- filter(team_box_raw, team_home_away == "home") %>%
  select(game_id, season, home_team = team_name, home_abbrev = team_abbreviation,
         home_points = team_score, home_FG_pct = field_goal_pct,
         home_3P_pct = three_point_field_goal_pct, home_reb = total_rebounds,
         home_tov = turnovers)

away_stats_raw <- filter(team_box_raw, team_home_away == "away") %>%
  select(game_id, away_team = team_name, away_abbrev = team_abbreviation,
         away_points = team_score, away_FG_pct = field_goal_pct,
         away_3P_pct = three_point_field_goal_pct, away_reb = total_rebounds,
         away_tov = turnovers)
```

```
# Reference join; kept separate from advanced modeling tables.
games_combined_raw <- inner_join(home_stats_raw, away_stats_raw, by = "game_id")
```

PredictiveModels.R

```
# --- PredictiveModels.R -----------------------------------------------------
# Predictive modeling entrypoint with optional CV and bootstrap.
# Outputs:
#   outputs/predictive_results.rds
#   outputs/predictive_metrics.csv
#   outputs/predictive_summary_table.csv
# Optional:
#   outputs/predictive_cv_metrics.csv
#   outputs/threshold_grid_accuracy.csv
#   outputs/threshold_bootstrap_summary.csv

suppressPackageStartupMessages({
  library(dplyr)
  library(readr)
  library(pROC)
  library(glmnet)
})

# Basic progress indicator without extra dependencies.
progress_init <- function(total_steps) {
  if (!interactive()) return(NULL)
  utils::txtProgressBar(min = 0, max = total_steps, style = 3)
}

progress_step <- function(pb, step) {
  if (!is.null(pb)) utils::setTxtProgressBar(pb, step)
}

progress_close <- function(pb) {
  if (!is.null(pb)) close(pb)
}

#' Run predictive models
#'
#' Fits predictive models, writes holdout metrics, and optionally runs CV
#' and bootstrap diagnostics.
#'
#' @param feature_mode "full" or "fast".
#' @param run_cv Logical; run k-fold CV on training seasons.
#' @param run_bootstrap Logical; bootstrap holdout accuracy for ridge.
#' @param include_rf Logical; include random forest benchmark.
```

```r
#' @param cv_folds Number of CV folds.
#' @param alpha_grid Elastic-net alpha grid.
#' @export
run_predictive_models <- function(feature_mode = "full",
                                  run_cv = FALSE,
                                  run_bootstrap = FALSE,
                                  include_rf = FALSE,
                                  cv_folds = 5,
                                  alpha_grid = c(0.25, 0.5, 0.75)) {
  # ---- Controls ----
  FEATURE_MODE <- feature_mode   # "full" or "fast"
  RUN_CV <- run_cv
  RUN_BOOTSTRAP <- run_bootstrap
  INCLUDE_RF <- include_rf
  CV_FOLDS <- cv_folds
  ALPHA_GRID <- alpha_grid

if (!file.exists("data/nba_games_modeling.RData")) {
  stop("data/nba_games_modeling.RData not found. Run R/AdvancedStats.R first.")
}

set.seed(42)
load("data/nba_games_modeling.RData")

# Prefer Elo-augmented data if present
if (file.exists("outputs/games_with_elo.rds")) {
  games_df <- readRDS("outputs/games_with_elo.rds")
} else {
  games_df <- games_combined
}

games_model <- games_df %>%
  mutate(
    season_start = if (is.numeric(season)) as.integer(season) else as.integer(substr(season, 1
    efg_roll5_diff = home_efg_roll5 - away_efg_roll5
  )

team_cols <- grep("^(home|away)_.*_roll(3|5|10)$", names(games_model), value = TRUE)
matchup_cols <- grep("^matchup_", names(games_model), value = TRUE)
diff_cols <- grep("_diff$", names(games_model), value = TRUE)
ratio_cols <- grep("_ratio$", names(games_model), value = TRUE)
elo_cols <- grep("^Elo_", names(games_model), value = TRUE)

feat_all <- sort(unique(c(team_cols, matchup_cols, diff_cols, ratio_cols, elo_cols, "efg_roll5_
feat_fast <- sort(unique(c(matchup_cols, diff_cols, ratio_cols, elo_cols, "efg_roll5_diff")))

feat_main <- if (FEATURE_MODE == "fast") feat_fast else feat_all
```

```r
model_df <- games_model %>%
  select(home_win, season_start, all_of(feat_main)) %>%
  mutate(across(all_of(feat_main), as.numeric)) %>%
  filter(if_all(all_of(feat_main), ~ is.finite(.)))

holdout_season <- max(model_df$season_start, na.rm = TRUE)
train <- filter(model_df, season_start < holdout_season)
test  <- filter(model_df, season_start == holdout_season)

yte <- test$home_win
baseline_acc <- max(mean(yte == 1), mean(yte == 0))

infer_feats <- c("matchup_net_roll5", "matchup_ortg_drtg_roll5",
                 "orb_roll5_diff", "tov_roll5_diff",
                 "efg_roll5_diff")

calc_metrics <- function(prob, truth) {
  tibble(
    accuracy = mean((prob >= 0.5) == truth),
    auc = as.numeric(pROC::auc(truth, prob))
  )
}

fit_ridge <- function(train_df, test_df) {
  xtr <- model.matrix(home_win ~ ., data = train_df)[, -1, drop = FALSE]
  xte <- model.matrix(home_win ~ ., data = test_df)[, -1, drop = FALSE]
  cv_fit <- cv.glmnet(xtr, train_df$home_win, family = "binomial", alpha = 0)
  prob <- as.numeric(predict(cv_fit, xte, s = "lambda.min", type = "response"))
  list(prob = prob, model = cv_fit)
}

fit_elastic <- function(train_df, test_df, alpha_grid = ALPHA_GRID) {
  xtr <- model.matrix(home_win ~ ., data = train_df)[, -1, drop = FALSE]
  xte <- model.matrix(home_win ~ ., data = test_df)[, -1, drop = FALSE]
  best <- NULL
  best_cvm <- Inf
  best_alpha <- NA_real_
  for (a in alpha_grid) {
    cv_fit <- cv.glmnet(xtr, train_df$home_win, family = "binomial", alpha = a)
    m <- min(cv_fit$cvm, na.rm = TRUE)
    if (m < best_cvm) {
      best_cvm <- m
      best <- cv_fit
      best_alpha <- a
    }
  }
  prob <- as.numeric(predict(best, xte, s = "lambda.min", type = "response"))
  list(prob = prob, model = best, alpha = best_alpha)
}

fit_rf <- function(train_df, test_df) {
  if (!INCLUDE_RF) return(NULL)
  if (!requireNamespace("randomForest", quietly = TRUE)) {
    stop("randomForest package is required when INCLUDE_RF = TRUE")
  }
  train_df <- train_df %>% mutate(home_win = factor(home_win, levels = c(0, 1)))
  test_df <- test_df %>% mutate(home_win = factor(home_win, levels = c(0, 1)))
  rf <- randomForest::randomForest(home_win ~ ., data = train_df)
  prob <- predict(rf, newdata = test_df, type = "prob")[, "1"]
  list(prob = as.numeric(prob), model = rf)
}

# ---- Fit models ----
total_steps <- 5L + as.integer(RUN_CV) + as.integer(RUN_BOOTSTRAP)
pb <- progress_init(total_steps)
step_id <- 0L

logit_fit <- glm(home_win ~ ., data = train %>% select(home_win, all_of(infer_feats)), family =
logit_prob <- predict(logit_fit, newdata = test %>% select(all_of(infer_feats)), type = "respon
step_id <- step_id + 1L; progress_step(pb, step_id)

full_train <- train %>% select(home_win, all_of(feat_main))
full_test  <- test %>% select(home_win, all_of(feat_main))

ridge_full <- fit_ridge(full_train, full_test)
step_id <- step_id + 1L; progress_step(pb, step_id)
elastic_full <- fit_elastic(full_train, full_test)
step_id <- step_id + 1L; progress_step(pb, step_id)
rf_full <- fit_rf(full_train, full_test)
step_id <- step_id + 1L; progress_step(pb, step_id)

metrics <- bind_rows(
  tibble(
    model = c("Majority baseline", "Logistic (lean)"),
    feature_set = c("none", "inferential"),
    accuracy = c(baseline_acc, mean((logit_prob >= 0.5) == yte)),
    auc = c(NA_real_, as.numeric(pROC::auc(yte, logit_prob)))
  ),
  tibble(
    model = c("Ridge logistic", "Elastic net"),
    feature_set = rep(FEATURE_MODE, 2),
    accuracy = c(mean((ridge_full$prob >= 0.5) == yte),
                 mean((elastic_full$prob >= 0.5) == yte)),
    auc = c(as.numeric(pROC::auc(yte, ridge_full$prob)),
```

```r
            as.numeric(pROC::auc(yte, elastic_full$prob)))
  )
)

if (!is.null(rf_full)) {
  metrics <- bind_rows(
    metrics,
    tibble(
      model = "Random Forest",
      feature_set = FEATURE_MODE,
      accuracy = mean((rf_full$prob >= 0.5) == yte),
      auc = as.numeric(pROC::auc(yte, rf_full$prob))
    )
  )
}

calib <- tibble(
  pred = ridge_full$prob,
  truth = yte
) %>%
  mutate(bin = ntile(pred, 10)) %>%
  group_by(bin) %>%
  summarise(pred_mean = mean(pred), obs = mean(truth), .groups = "drop")

summary_table <- metrics %>%
  mutate(holdout_season = holdout_season) %>%
  arrange(desc(auc))

dir.create("outputs", showWarnings = FALSE)

saveRDS(
  list(
    metrics = metrics,
    calib = calib,
    yte = yte,
    ridge_prob = ridge_full$prob,
    ridge_prob_list = list(all_features = ridge_full$prob),
    elastic_prob = elastic_full$prob,
    holdout_season = holdout_season,
    elastic_alpha = elastic_full$alpha
  ),
  "outputs/predictive_results.rds"
)
write_csv(metrics, "outputs/predictive_metrics.csv")
write_csv(summary_table, "outputs/predictive_summary_table.csv")

message("Holdout season: ", holdout_season)
message("Wrote outputs/predictive_metrics.csv and outputs/predictive_summary_table.csv")
print(summary_table)
step_id <- step_id + 1L; progress_step(pb, step_id)

# ---- Optional: CV ----
if (RUN_CV) {
  if (!requireNamespace("caret", quietly = TRUE)) {
    stop("caret package is required when RUN_CV = TRUE")
  }
  folds <- caret::createFolds(train$home_win, k = CV_FOLDS, list = TRUE, returnTrain = FALSE)
  cv_sets <- vector("list", length(folds))
  f_id <- 1L
  for (va_idx in folds) {
    tr_idx <- setdiff(seq_len(nrow(train)), va_idx)
    tr_dat <- train[tr_idx, , drop = FALSE]
    va_dat <- train[va_idx, , drop = FALSE]

    logit_cv <- glm(home_win ~ ., data = tr_dat %>% select(home_win, all_of(infer_feats)), fami
    logit_cv_prob <- predict(logit_cv, newdata = va_dat %>% select(all_of(infer_feats)), type =

    ridge_cv <- fit_ridge(tr_dat %>% select(home_win, all_of(feat_main)),
                          va_dat %>% select(home_win, all_of(feat_main)))
    elastic_cv <- fit_elastic(tr_dat %>% select(home_win, all_of(feat_main)),
                              va_dat %>% select(home_win, all_of(feat_main)))

    fold_metrics <- bind_rows(
      calc_metrics(logit_cv_prob, va_dat$home_win) %>% mutate(model = "Logistic (lean)"),
      calc_metrics(ridge_cv$prob, va_dat$home_win) %>% mutate(model = "Ridge logistic"),
      calc_metrics(elastic_cv$prob, va_dat$home_win) %>% mutate(model = "Elastic net")
    ) %>%
      mutate(feature_set = FEATURE_MODE, fold = f_id, .before = 1)

    if (INCLUDE_RF) {
      rf_cv <- fit_rf(tr_dat %>% select(home_win, all_of(feat_main)),
                      va_dat %>% select(home_win, all_of(feat_main)))
      fold_metrics <- bind_rows(
        fold_metrics,
        calc_metrics(rf_cv$prob, va_dat$home_win) %>%
          mutate(model = "Random Forest", feature_set = FEATURE_MODE, fold = f_id, .before = 1
      )
    }

    cv_sets[[f_id]] <- fold_metrics
    f_id <- f_id + 1L
  }

  cv_metrics <- bind_rows(cv_sets) %>%
    group_by(model, feature_set) %>%
    summarise(cv_accuracy = mean(accuracy), cv_auc = mean(auc), .groups = "drop")

  write_csv(cv_metrics, "outputs/predictive_cv_metrics.csv")
  message("Wrote outputs/predictive_cv_metrics.csv (k = ", CV_FOLDS, ")")
  print(cv_metrics)
  step_id <- step_id + 1L; progress_step(pb, step_id)
}

# ---- Optional: bootstrap ----
if (RUN_BOOTSTRAP) {
  threshold_grid <- seq(0.3, 0.7, by = 0.01)
  acc_tbl <- tibble(
    threshold = threshold_grid,
    accuracy = vapply(threshold_grid, function(t) mean((ridge_full$prob >= t) == yte), numeric
  )

  best_row <- acc_tbl %>% slice_max(accuracy, n = 1, with_ties = FALSE)
  best_thresh <- best_row$threshold

  boot_acc <- function(thresh, n_boot = 1000) {
    n <- length(yte)
    replicate(n_boot, {
      idx <- sample.int(n, n, replace = TRUE)
      mean((ridge_full$prob[idx] >= thresh) == yte[idx])
    })
  }

  boot_default <- boot_acc(0.5)
  boot_tuned   <- boot_acc(best_thresh)

  boot_summary <- tibble(
    model = "Ridge logistic",
    threshold = c(0.5, best_thresh),
    type = c("default", "tuned"),
    holdout_accuracy = c(mean((ridge_full$prob >= 0.5) == yte), best_row$accuracy),
    mean_accuracy = c(mean(boot_default), mean(boot_tuned)),
    ci_low = c(quantile(boot_default, 0.025), quantile(boot_tuned, 0.025)),
    ci_high = c(quantile(boot_default, 0.975), quantile(boot_tuned, 0.975))
  )

  write_csv(acc_tbl, "outputs/threshold_grid_accuracy.csv")
  write_csv(boot_summary, "outputs/threshold_bootstrap_summary.csv")
  message("Wrote outputs/threshold_grid_accuracy.csv and outputs/threshold_bootstrap_summary.c
  print(boot_summary)
  step_id <- step_id + 1L; progress_step(pb, step_id)
}
```

```r
progress_close(pb)

}

if (identical(Sys.getenv("RUN_PREDICTIVE_MODELS"), "true")) {
  run_predictive_models()
}
```