cfbplotR · cfb4th · cfbseedR :: CHEAT SHEET

Three companion packages to cfbfastR: **cfbplotR** puts CFB team logos, wordmarks, headshots and colours into ggplot2 and gt (the CFB twin of nflplotR, built on ggpath); **cfb4th** scores 4th-down decisions (twin of nfl4th); **cfbseedR** computes standings, CFP seeds and season simulations (twin of nflseedR). Page 1: cfbplotR. Page 2: cfb4th + cfbseedR and the Python mirror in `sportsdataverse.cfb`.

cfbplotR v0.1.0

R ≥ 4.1 · ggplot2 ≥ 4.0 · ggpath ≥ 1.1
cfbplotR.sportsdataverse.org

CFBPLOTR · LOGOS IN GGPLOT2 & GT

INSTALL & LOAD

```
pak::pak("sportsdataverse/cfbplotR")  # GitHub only
library(cfbplotR); library(ggplot2)
```

Team names are matched against `valid_team_names()`; cfbfastR `home_team` / `school` / `pos_team` strings work directly. Images are downloaded once and cached (`.cfbplotR_clear_cache()`).

LOGOS, WORDMARKS & HEADSHOTS

Geom	Required aes
`geom_cfb_logos()`	`x y team`
`geom_cfb_wordmarks()`	`x y team`
`geom_cfb_headshots()`	`x y player_id` (ESPN id)
`geom_from_path()`	`x y path` (any image; ggpath)

Optional aes: `alpha colour angle hjust vjust width height` (npc; typical `width = 0.075`, `height = 0.1`); `colour = "b/w"` greyscales. ggproto: `GeomCFBlogo GeomCFBwordmark GeomCFBheads GeomFromPath GeomRefLines`.

```
df <- data.frame(team = valid_team_names()[1:16],
                 a = rnorm(16), b = rnorm(16))
ggplot(df, aes(x = a, y = b)) +
  geom_cfb_logos(aes(team = team), width = 0.075) +
  geom_mean_lines(aes(x0 = a, y0 = b)) +
  theme_minimal()
# headshots: ESPN athlete ids
geom_cfb_headshots(aes(player_id = athlete_id), height = .2)
```

`geom_mean_lines()` / `geom_median_lines()` (ggpath) draw reference lines at the mean / median of `x0`, `y0`.

TITLES & PREVIEW

```
p + ggtitle_image(title_image = "Utah", title = "Utes",
                  image_side = "left", image_height = 40) +
  theme_title_image()
ggpreview(p, width = 7, height = 5, units = "in")
```

`ggtitle_image()` takes a team name (logo auto-resolved) or any image URL; `ggpreview()` renders at exact output size (`asp dpi device bg` as in `ggsave`).

COLOUR & FILL SCALES

```
scale_color_cfb(alt_colors = NULL, values = NULL, ...,
                na.value = "grey50", guide = NULL,
                alpha = NA)
scale_fill_cfb(...)        # same args; scale_colour_cfb alias
```

Maps team names to primary colours from `logo_ref`; list teams in `alt_colors` to use their secondary colour instead.

```
ggplot(df, aes(x = team, y = b)) +
  geom_col(aes(colour = team, fill = team), width = .5) +
  scale_color_cfb(alt_colors = df$team) +
  scale_fill_cfb(alpha = 0.4)
```

LOGO AXES

Two routes to image tick labels:

Route	Calls
theme element (ggpath)	`theme(axis.text.x = element_cfb_logo())` · `element_cfb_wordmark()` · `element_cfb_headshot()` · generic `element_path()` `element_raster()`
scale + markdown theme	`scale_x_cfb()` / `scale_y_cfb()` (logos) · `scale_x_cfb_headshots()` / `scale_y_cfb_headshots()` + `theme_x_cfb()` / `theme_y_cfb()` (needs ggtext, gridtext > 0.1.4)

```
# route 1 — element
ggplot(df, aes(x = team, y = b)) + geom_col() +
  theme(axis.text.x = element_cfb_logo(size = 1))
# route 2 — scale (size = logo px)
ggplot(df, aes(y = team, x = b)) + geom_col() +
  scale_y_cfb(size = 12) + theme_y_cfb()
```

`element_cfb_*()` args: `alpha colour hjust vjust size` (size in cm).

GT TABLE HELPERS

Function	Does
`gt_fmt_cfb_logo(gt, columns, height = 30)`	team name cells → logos
`gt_fmt_cfb_wordmark(gt, columns, height)`	cells → wordmarks
`gt_fmt_cfb_headshot(gt, columns, height)`	ESPN id / headshot URL → headshot
`gt_cfb_cols_label(gt, ..., height)`	column *labels* → logos
`gt_merge_stack_team_color(gt, col1, col2, team_col)`	stack two cols; bottom in team colour

```
library(gt)
team_df |>
  dplyr::transmute(logo = school, school, mascot,
                   conference) |>
  head(8) |> gt() |>
  gt_merge_stack_team_color(school, mascot, school) |>
  gt_fmt_cfb_logo(columns = c("logo", "conference"))
```

TEAM UTILITIES

Function	Returns
`valid_team_names(division = "FBS")`	names; also `P5 G5 FCS DII DIII Conference` (conference logos work everywhere)
`clean_school_names(x, keep_non_matches = TRUE)`	canonical school name (1,100+ aliases)
`clean_team_abbrs(x, keep_non_matches)`	canonical abbreviation
`cfb_team_factor(teams)`	factor limited to valid names
`add_athlete_id_col(df, name_col, team_col = NULL, headshot_urls = FALSE)`	joins ESPN `athlete_id` from cfbfastR-data rosters
`.cfbplotR_clear_cache()`	drop cached images

```
clean_school_names(c("utah", "Hawaii", "UTSA", "SLC"))
add_athlete_id_col(qbs, player_name, team_col = team)
```

PREMADE PLOT: TEAM TIERS

```
tiers <- data.frame(
  tier_no = c(1, 1, 2, 2, 3),
  team    = c("Georgia", "Ohio State", "Oregon",
              "Texas", "Utah"))
cfb_team_tiers(tiers,
  title = "CFB Team Tiers, 2024 as of Week 6",
  tier_desc = c("1" = "Playoff", "2" = "Very Good",
                "3" = "Medium"),
  no_line_below_tier = 2, devel = FALSE)
```

Needs `tier_no` + `team`; `presort`, `alpha`, `width`, `caption`, `subtitle`; `devel = TRUE` skips image rendering while iterating. Suggests `sjmisc`.

WITH CFBFASTR DATA

```
library(cfbfastR)
pbp <- load_cfb_pbp(2024)
off <- pbp |> dplyr::filter(rush == 1 | pass == 1) |>
  dplyr::group_by(team = pos_team) |>
  dplyr::summarise(epa = mean(EPA, na.rm = TRUE),
                   sr  = mean(EPA > 0, na.rm = TRUE))
ggplot(off, aes(x = sr, y = epa)) +
  geom_mean_lines(aes(x0 = sr, y0 = epa)) +
  geom_cfb_logos(aes(team = team), width = 0.05) +
  labs(x = "Success rate", y = "EPA / play")
```

Team strings from `load_cfb_pbp()`, `cfbd_game_info()`, `cfbd_team_info()` match `valid_team_names()`; run `clean_school_names()` on other sources (ESPN, 247, hand-typed). Headshots: `cfbd_team_roster()` ships `athlete_id` + `headshot_url`.

SIBLINGS

nflplotR (NFL original, same API) · mlbplotR · ggpath (backend) · cfbfastR (data) · cfb4th / cfbseedR (page 2) · Python: `sportsdataverse.cfb` loaders feed the same team strings.

cfb4th · cfbseedR :: 4th-down decisions · standings, seeds & simulations

nflverse originals: **nfl4th** · **nflseedR** · data: **cfbfastR** · Python mirror: `sportsdataverse.cfb` (`cfb_fourth_down`, `cfb_standings`, `cfb_simulations`)

CFB4TH V0.1.2 · R ≥ 3.5 · CFB4TH.SPORTSDATAVERSE.ORG

INSTALL & WORKFLOW

```
devtools::install_github("sportsdataverse/cfb4th")
library(cfb4th)   # imports cfbfastR ≥ 1.4, xgboost ≥ 2.0
```

Five exports: `add_4th_probs()`, `load_4th_pbp()`, `get_4th_plays()`, `make_table_data()`, `%>%`. Models (EP, WP, FG, first-down) are bundled in `sysdata.rda`; no API key needed for `add_4th_probs()`. `load_4th_pbp()` / `get_4th_plays()` call cfbfastR's CFBD endpoints, so set `CFBD_API_KEY` once (`usethis::edit_r_environ()`).

ADD_4TH_PROBS(DF)

Input: one row per 4th-down situation. A cfbfastR pbp frame (has `down`, filtered to 4th down automatically) or a hand-built tibble with these columns:

Game	Situation
`home away pos_team def_pos_team spread over_under`	`half period TimeSecsRem adj_TimeSecsRem down distance yards_to_goal pos_score_diff_start pos_team_receives_2H_kickoff pos_team_timeouts_rem_before def_pos_team_timeouts_rem_before`

`spread` is from the home team's view (Utah −7 = home favoured by 7); `TimeSecsRem` = seconds left in the half, `adj_TimeSecsRem` = in the game.

Columns added

Column	Meaning
`go_boost`	WP gain (pct points) from going vs best of FG / punt
`first_down_prob`	P(convert) if going
`wp_fail` · `wp_succeed`	WP after a failed / successful try
`go_wp`	expected WP when going
`fg_make_prob`	P(make)
`miss_fg_wp` · `make_fg_wp`	WP after miss / make
`fg_wp`	expected WP when kicking
`punt_wp`	expected WP when punting (NA if not sensible)

Recommendation = the choice with max WP; `go_boost > 0` means "go".

ONE PLAY, END TO END

```
play <- tibble::tibble(
  home = "Utah", away = "BYU",
  pos_team = "Utah", def_pos_team = "BYU",
  spread = -7, over_under = 55,
  half = 2, period = 3,
  TimeSecsRem = 900, adj_TimeSecsRem = 900,
  down = 4, distance = 4, yards_to_goal = 40,
  pos_score_diff_start = 7,
  pos_team_receives_2H_kickoff = 1,
  pos_team_timeouts_rem_before = 3,
  def_pos_team_timeouts_rem_before = 3)
probs <- add_4th_probs(play)
make_table_data(probs) |> gt::gt()
```

`make_table_data()` takes ONE scored play and returns a 3-row tibble (`choice choice_prob success_prob fail_wp success_wp`; WP in %), sorted best first: "Go for it" / "Field goal attempt" / "Punt".

SEASON & GAME LOADERS

```
pbp4 <- load_4th_pbp(2024)        # seasons ≥ 2014
pbp4 |> dplyr::filter(go_boost > 2, go == 0) |>
  dplyr::count(pos_team, sort = TRUE)  # timid teams

game  <- cfbfastR::cfbd_game_info(2019, team = "Utah",
                                  week = 3)
plays <- get_4th_plays(game) |>
add_4th_probs()
```

`load_4th_pbp(seasons)` = `cfbfastR::load_cfb_pbp()` + consensus `spread` / `over_under` from `cfbd_betting_lines()` + `add_4th_probs()` + `go` (100 went / 0 kicked / NA penalty-timeout). Slow (a minute or two per season). `get_4th_plays(df)` needs `game_id home_team away_team` and returns the ESPN 4th downs with `desc type_text index`.

PYTHON MIRROR

```
from sportsdataverse.cfb.cfb_fourth_down
import (
    get_4th_down_probs, get_go_wp, get_fg_wp,
get_punt_wp)
out = get_4th_down_probs(pbp_df)   # pandas in / out
```

Same models and column names; in sdv-py the fourth-down surface is also applied default-on inside `CFBPlayProcess`.

CFBSEEDR V0.2.0 · R ≥ 4.1 · CFBSEEDR.SPORTSDATAVERSE.ORG

INSTALL & GAMES SCHEMA

```
remotes::install_github("sportsdataverse/cfbseedR")
library(cfbseedR)
sched <- cfbfastR::load_cfb_schedules(2024)
games <- cfb_games_from_schedule(sched)
teams <- cfbfastR::cfbd_team_info(year = 2024) |>
  dplyr::transmute(team = school, conference)
```

games col	Meaning
`season` / `sim`	season or simulation id
`game_type`	`REG` · `CONF_CHAMP` (notes mention "championship") · `POST`
`week home_team away_team`	names must match `teams$team`
`result`	home margin; `NA` = to be simulated
`neutral`	0/1 (optional)

`teams`: `team conference` + optional `division` "FBS"/"FCS" (Big 12 FCS-win cap), `conf_division` "East"/"West" (division format - champs take `conf_rank` 1-2, e.g. the 2026 Sun Belt), `postseason_eligible` (FALSE keeps a team out of the title-game ranks). "FBS Independents" / NA = independent (no conference rank).

STANDINGS & CFP SEEDS

```
st <- cfb_standings(games, teams,
        tiebreaker_depth = "SOS",  # PRE-SOV POINTS RANDOM
        playoff_seeds = 12, rankings = NULL,
        tiebreaker_data = NULL, verbosity = "MIN")
attr(st, "tiebreak_notes")
cfb_playoff_seeds(st, rankings = cfp, playoff_seeds = 12)
```

Returns one row per (`sim`, `team`): `games wins losses ties win_pct pd conf_games conf_wins conf_losses conf_ties conf_pct conf_pd sov sos conf_rank conf_champ` (+ `seed`). `sov` / `sos` are conference-REG-scoped; CONF_CHAMP games count for the overall record and decide `conf_champ`, not the conference record.

Seeds = straight seeding (no bump-up) with a season-keyed `autobid`. `"2026"` (default): ACC/Big 12/Big Ten/SEC champions in **regardless of rank**, the highest-ranked **Group-of-6 team** (champion or not), and Notre Dame when ranked inside the field. `"2025"`: the 5 highest-ranked champions. Without `rankings`: `win_pct sov sos pd team`.

PYTHON MIRROR

```
from sportsdataverse.cfb import
(cfb_standings,
    cfb_playoff_seeds,
cfb_games_from_schedule,
    cfb_simulations, cfb_compute_results)
```

SEASON SIMULATION

```
games$result[games$week >= 10] <- NA   # unplayed
set.seed(4)
sim <- cfb_simulations(games, teams,
         simulations = 10000, playoff_seeds = 12,
         sim_include = "POST",  # or "REG"
         rankings = cfp, autobid = "2026",
         tiebreaker_data = list(cfp_rankings = cfp))
sim$overall   # wins conf_champ playoff seed1 won_natty
sim$team_wins; sim$game_summary; sim$standings
```

Returns a `cfbseedR_simulation` list: `standings` (+ `seed`, `exit`: 0 missed, r = out in round r, max+1 = champion), `games` (incl. generated playoff games), `overall`, `team_wins`, `game_summary`, `sim_params`. Bracket = fixed 12-team CFP (1 v 8/9, 4 v 5/12, 3 v 6/11, 2 v 7/10), first round at higher seed, no reseeding; runs sequentially.

COMPUTE_RESULTS HOOK

```
my_results <- function(teams, games, week_num, ...) {
  # fill games$result where week == week_num & is.na
  list(teams = teams, games = games)
}
simulations_verify_fct(my_results)          # TRUE
cfb_simulations(games, teams,
                compute_results = my_results, ...)
```

Default `cfbseedR_compute_results()` = nflseedR's dynamic ELO (home +20, ×1.2 postseason, margin sd 13; pass `elo =` named vector to seed ratings). Contract: keep all rows/cols, fill exactly this week's NA results, no ties outside REG.

CFB TIEBREAKERS

Conference ranks by `conf_pct`; ties use each conference's **official procedure**, registered as **season-scoped epochs** — SEC, Big Ten, Big 12, ACC, MAC, American, CUSA, Mountain West and Sun Belt. The **ACC's all-new 2026 policy** (h2h → SportSource SSR → draw, plus the alternate-game-count candidate pool: teams matching the leaders' wins *or* losses join the tie) applies from 2026; earlier seasons keep the 2024 cascade. Pass a `season` id to pick the epoch. Rungs: multi-team h2h, common opponents (+ descending), opponents' conf win%, SEC capped margin 42/48, Big 12 capped wins, `div_pct`, `cfp_ranked_final_week` (ranked tied team advances only if it *won* its final conference game), metric composite, APR, coin toss. Feed externals via `tiebreaker_data`: `analytics_ratings`, `cfp_rankings`, `apr`. The re-formed **Pac-12** has published no procedure → generic fallback. Skipped rungs land in `attr(, "tiebreak_notes")`.

cfbplotR · cfb4th · cfbseedR :: CHEAT SHEET

Three companion packages to **cfbfastR**: **cfbplotR** puts CFB team logos, wordmarks, headshots and colours into ggplot2 and gt (the CFB twin of nflplotR, built on ggpath); **cfb4th** scores 4th-down decisions (twin of nfl4th); **cfbseedR** computes standings, CFP seeds and season simulations (twin of nflseedR). Page 1: cfbplotR. Page 2: cfb4th + cfbseedR and the Python mirror in `sportsdataverse.cfb`.

cfbplotR v0.1.0

R ≥ 4.1 · ggplot2 ≥ 4.0 · ggpath ≥ 1.1
cfbplotR.sportsdataverse.org

CFBPLOTR · LOGOS IN GGPLOT2 & GT

INSTALL & LOAD

```
pak::pak("sportsdataverse/cfbplotR")  # GitHub only
library(cfbplotR); library(ggplot2)
```

Team names are matched against `valid_team_names()`; cfbfastR `home_team` / `school` / `pos_team` strings work directly. Images are downloaded once and cached (`.cfbplotR_clear_cache()`).

LOGOS, WORDMARKS & HEADSHOTS

Geom	Required aes
`geom_cfb_logos()`	`x y team`
`geom_cfb_wordmarks()`	`x y team`
`geom_cfb_headshots()`	`x y player_id` (ESPN id)
`geom_from_path()`	`x y path` (any image; ggpath)

Optional aes: `alpha colour angle hjust vjust width height` (npc; typical `width = 0.075`, `height = 0.1`); `colour = "b/w"` greyscales. ggproto: `GeomCFBlogo GeomCFBwordmark GeomCFBheads GeomFromPath GeomRefLines`.

```
df <- data.frame(team = valid_team_names()[1:16],
                 a = rnorm(16), b = rnorm(16))
ggplot(df, aes(x = a, y = b)) +
  geom_cfb_logos(aes(team = team), width = 0.075) +
  geom_mean_lines(aes(x0 = a, y0 = b)) +
  theme_minimal()
# headshots: ESPN athlete ids
geom_cfb_headshots(aes(player_id = athlete_id), height = .2)
```

`geom_mean_lines()` / `geom_median_lines()` (ggpath) draw reference lines at the mean / median of `x0`, `y0`.

TITLES & PREVIEW

```
p + ggtitle_image(title_image = "Utah", title = "Utes",
                  image_side = "left", image_height = 40) +
  theme_title_image()
ggpreview(p, width = 7, height = 5, units = "in")
```

`ggtitle_image()` takes a team name (logo auto-resolved) or any image URL; `ggpreview()` renders at exact output size (`asp dpi device bg` as in `ggsave`).

COLOUR & FILL SCALES

```
scale_color_cfb(alt_colors = NULL, values = NULL, ...,
                na.value = "grey50", guide = NULL,
                alpha = NA)
scale_fill_cfb(...)        # same args; scale_colour_cfb alias
```

Maps team names to primary colours from `logo_ref`; list teams in `alt_colors` to use their secondary colour instead.

```
ggplot(df, aes(x = team, y = b)) +
  geom_col(aes(colour = team, fill = team), width = .5) +
  scale_color_cfb(alt_colors = df$team) +
  scale_fill_cfb(alpha = 0.4)
```

LOGO AXES

Two routes to image tick labels:

Route	Calls
theme element (ggpath)	`theme(axis.text.x = element_cfb_logo())` · `element_cfb_wordmark()` · `element_cfb_headshot()` · generic `element_path()` `element_raster()`
scale + markdown theme	`scale_x_cfb()` / `scale_y_cfb()` (logos) · `scale_x_cfb_headshots()` / `scale_y_cfb_headshots()` + `theme_x_cfb()` / `theme_y_cfb()` (needs ggtext, gridtext > 0.1.4)

```
# route 1 — element
ggplot(df, aes(x = team, y = b)) + geom_col() +
  theme(axis.text.x = element_cfb_logo(size = 1))
# route 2 — scale (size = logo px)
ggplot(df, aes(y = team, x = b)) + geom_col() +
  scale_y_cfb(size = 12) + theme_y_cfb()
```

`element_cfb_*()` args: `alpha colour hjust vjust size` (size in cm).

GT TABLE HELPERS

Function	Does
`gt_fmt_cfb_logo(gt, columns, height = 30)`	team name cells → logos
`gt_fmt_cfb_wordmark(gt, columns, height)`	cells → wordmarks
`gt_fmt_cfb_headshot(gt, columns, height)`	ESPN id / headshot URL → headshot
`gt_cfb_cols_label(gt, ..., height)`	column *labels* → logos
`gt_merge_stack_team_color(gt, col1, col2, team_col)`	stack two cols; bottom in team colour

```
library(gt)
team_df |>
  dplyr::transmute(logo = school, school, mascot,
                   conference) |>
  head(8) |> gt() |>
  gt_merge_stack_team_color(school, mascot, school) |>
  gt_fmt_cfb_logo(columns = c("logo", "conference"))
```

TEAM UTILITIES

Function	Returns
`valid_team_names(division = "FBS")`	names; also `P5 G5 FCS DII DIII Conference` (conference logos work everywhere)
`clean_school_names(x, keep_non_matches = TRUE)`	canonical school name (1,100+ aliases)
`clean_team_abbrs(x, keep_non_matches)`	canonical abbreviation
`cfb_team_factor(teams)`	factor limited to valid names
`add_athlete_id_col(df, name_col, team_col = NULL, headshot_urls = FALSE)`	joins ESPN `athlete_id` from cfbfastR-data rosters
`.cfbplotR_clear_cache()`	drop cached images

```
clean_school_names(c("utah", "Hawaii", "UTSA", "SLC"))
add_athlete_id_col(qbs, player_name, team_col = team)
```

PREMADE PLOT: TEAM TIERS

```
tiers <- data.frame(
  tier_no = c(1, 1, 2, 2, 3),
  team    = c("Georgia", "Ohio State", "Oregon",
              "Texas", "Utah"))
cfb_team_tiers(tiers,
  title = "CFB Team Tiers, 2024 as of Week 6",
  tier_desc = c("1" = "Playoff", "2" = "Very Good",
                "3" = "Medium"),
  no_line_below_tier = 2, devel = FALSE)
```

Needs `tier_no` + `team`; `presort`, `alpha`, `width`, `caption`, `subtitle`; `devel = TRUE` skips image rendering while iterating. Suggests `sjmisc`.

WITH CFBFASTR DATA

```
library(cfbfastR)
pbp <- load_cfb_pbp(2024)
off <- pbp |> dplyr::filter(rush == 1 | pass == 1) |>
  dplyr::group_by(team = pos_team) |>
  dplyr::summarise(epa = mean(EPA, na.rm = TRUE),
                   sr  = mean(EPA > 0, na.rm = TRUE))
ggplot(off, aes(x = sr, y = epa)) +
  geom_mean_lines(aes(x0 = sr, y0 = epa)) +
  geom_cfb_logos(aes(team = team), width = 0.05) +
  labs(x = "Success rate", y = "EPA / play")
```

Team strings from `load_cfb_pbp()`, `cfbd_game_info()`, `cfbd_team_info()` match `valid_team_names()`; run `clean_school_names()` on other sources (ESPN, 247, hand-typed). Headshots: `cfbd_team_roster()` ships `athlete_id` + `headshot_url`.

SIBLINGS

nflplotR (NFL original, same API) · **mlbplotR** · **ggpath** (backend) · **cfbfastR** (data) · **cfb4th** / **cfbseedR** (page 2) · Python: `sportsdataverse.cfb` loaders feed the same team strings.

cfb4th · cfbseedR :: 4th-down decisions · standings, seeds & simulations

nflverse originals: **nfl4th** · **nflseedR** · data: **cfbfastR** · Python mirror: `sportsdataverse.cfb` (`cfb_fourth_down`, `cfb_standings`, `cfb_simulations`)

CFB4TH V0.1.2 · R ≥ 3.5 · CFB4TH.SPORTSDATAVERSE.ORG

INSTALL & WORKFLOW

```
devtools::install_github("sportsdataverse/cfb4th")
library(cfb4th)   # imports cfbfastR ≥ 1.4, xgboost ≥ 2.0
```

Five exports: `add_4th_probs()`, `load_4th_pbp()`, `get_4th_plays()`, `make_table_data()`, `%>%`. Models (EP, WP, FG, first-down) are bundled in `sysdata.rda`; no API key needed for `add_4th_probs()`. `load_4th_pbp()` / `get_4th_plays()` call cfbfastR's CFBD endpoints, so set `CFBD_API_KEY` once (`usethis::edit_r_environ()`).

ADD_4TH_PROBS(DF)

Input: one row per 4th-down situation. A cfbfastR pbp frame (has `down`, filtered to 4th down automatically) or a hand-built tibble with these columns:

Game	Situation
`home away pos_team def_pos_team spread over_under`	`half period TimeSecsRem adj_TimeSecsRem down distance yards_to_goal pos_score_diff_start pos_team_receives_2H_kickoff pos_team_timeouts_rem_before def_pos_team_timeouts_rem_before`

`spread` is from the home team's view (Utah −7 = home favoured by 7); `TimeSecsRem` = seconds left in the half, `adj_TimeSecsRem` = in the game.

Columns added

Column	Meaning
`go_boost`	WP gain (pct points) from going vs best of FG / punt
`first_down_prob`	P(convert) if going
`wp_fail` · `wp_succeed`	WP after a failed / successful try
`go_wp`	expected WP when going
`fg_make_prob`	P(make)
`miss_fg_wp` · `make_fg_wp`	WP after miss / make
`fg_wp`	expected WP when kicking
`punt_wp`	expected WP when punting (NA if not sensible)

Recommendation = the choice with max WP; `go_boost > 0` means "go".

ONE PLAY, END TO END

```
play <- tibble::tibble(
  home = "Utah", away = "BYU",
  pos_team = "Utah", def_pos_team = "BYU",
  spread = -7, over_under = 55,
  half = 2, period = 3,
  TimeSecsRem = 900, adj_TimeSecsRem = 900,
  down = 4, distance = 4, yards_to_goal = 40,
  pos_score_diff_start = 7,
  pos_team_receives_2H_kickoff = 1,
  pos_team_timeouts_rem_before = 3,
  def_pos_team_timeouts_rem_before = 3)
probs <- add_4th_probs(play)
make_table_data(probs) |> gt::gt()
```

`make_table_data()` takes ONE scored play and returns a 3-row tibble (`choice choice_prob success_prob fail_wp success_wp`; WP in %), sorted best first: "Go for it" / "Field goal attempt" / "Punt".

SEASON & GAME LOADERS

```
pbp4 <- load_4th_pbp(2024)        # seasons ≥ 2014
pbp4 |> dplyr::filter(go_boost > 2, go == 0) |>
  dplyr::count(pos_team, sort = TRUE)  # timid teams

game  <- cfbfastR::cfbd_game_info(2019, team = "Utah",
                                  week = 3)
plays <- get_4th_plays(game) |> add_4th_probs()
```

`load_4th_pbp(seasons)` = `cfbfastR::load_cfb_pbp()` + consensus `spread` / `over_under` from `cfbd_betting_lines()` + `add_4th_probs()` + `go` (100 went / 0 kicked / NA penalty-timeout). Slow (a minute or two per season). `get_4th_plays(df)` needs `game_id home_team away_team` and returns the ESPN 4th downs with `desc type_text index`.

PYTHON MIRROR

```
from sportsdataverse.cfb.cfb_fourth_down import (
    get_4th_down_probs, get_go_wp, get_fg_wp, get_punt_wp)
out = get_4th_down_probs(pbp_df)   # pandas in / out
```

Same models and column names; in sdv-py the fourth-down surface is also applied default-on inside `CFBPlayProcess`.

CFBSEEDR V0.2.0 · R ≥ 4.1 · CFBSEEDR.SPORTSDATAVERSE.ORG

INSTALL & GAMES SCHEMA

```
remotes::install_github("sportsdataverse/cfbseedR")
library(cfbseedR)
sched <- cfbfastR::load_cfb_schedules(2024)
games <- cfb_games_from_schedule(sched)
teams <- cfbfastR::cfbd_team_info(year = 2024) |>
  dplyr::transmute(team = school, conference)
```

games col	Meaning
`season` / `sim`	season or simulation id
`game_type`	`REG` · `CONF_CHAMP` (notes mention "championship") · `POST`
`week home_team away_team`	names must match `teams$team`
`result`	home margin; `NA` = to be simulated
`neutral`	0/1 (optional)

`teams`: `team conference` + optional `division` "FBS"/"FCS" (Big 12 FCS-win cap), `conf_division` "East"/"West" (division format - champs take `conf_rank` 1-2, e.g. the 2026 Sun Belt), `postseason_eligible` (FALSE keeps a team out of the title-game ranks). "FBS Independents" / NA = independent (no conference rank).

STANDINGS & CFP SEEDS

```
st <- cfb_standings(games, teams,
        tiebreaker_depth = "SOS",  # PRE-SOV POINTS RANDOM
        playoff_seeds = 12, rankings = NULL,
        tiebreaker_data = NULL, verbosity = "MIN")
attr(st, "tiebreak_notes")
cfb_playoff_seeds(st, rankings = cfp, playoff_seeds = 12)
```

Returns one row per (`sim`, `team`): `games wins losses ties win_pct pd conf_games conf_wins conf_losses conf_ties conf_pct conf_pd sov sos conf_rank conf_champ` (+ `seed`). `sov` / `sos` are conference-REG-scoped; CONF_CHAMP games count for the overall record and decide `conf_champ`, not the conference record.

Seeds = straight seeding (no bump-up) with a season-keyed `autobid`. `"2026"` (default): ACC/Big 12/Big Ten/SEC champions in **regardless of rank**, the highest-ranked **Group-of-6 team** (champion or not), and Notre Dame when ranked inside the field. `"2025"`: the 5 highest-ranked champions. Without `rankings`: `win_pct sov sos pd team`.

PYTHON MIRROR

```
from sportsdataverse.cfb import (cfb_standings,
    cfb_playoff_seeds, cfb_games_from_schedule,
    cfb_simulations, cfb_compute_results)
```

SEASON SIMULATION

```
games$result[games$week >= 10] <- NA   # unplayed
set.seed(4)
sim <- cfb_simulations(games, teams,
         simulations = 10000, playoff_seeds = 12,
         sim_include = "POST",  # or "REG"
         rankings = cfp, autobid = "2026",
         tiebreaker_data = list(cfp_rankings = cfp))
sim$overall   # wins conf_champ playoff seed1 won_natty
sim$team_wins; sim$game_summary; sim$standings
```

Returns a `cfbseedR_simulation` list: `standings` (+ `seed`, `exit`: 0 missed, r = out in round r, max+1 = champion), `games` (incl. generated playoff games), `overall`, `team_wins`, `game_summary`, `sim_params`. Bracket = fixed 12-team CFP (1 v 8/9, 4 v 5/12, 3 v 6/11, 2 v 7/10), first round at higher seed, no reseeding; runs sequentially.

COMPUTE_RESULTS HOOK

```
my_results <- function(teams, games, week_num, ...) {
  # fill games$result where week == week_num & is.na
  list(teams = teams, games = games)
}
simulations_verify_fct(my_results)          # TRUE
cfb_simulations(games, teams,
                compute_results = my_results, ...)
```

Default `cfbseedR_compute_results()` = nflseedR's dynamic ELO (home +20, ×1.2 postseason, margin sd 13; pass `elo =` named vector to seed ratings). Contract: keep all rows/cols, fill exactly this week's NA results, no ties outside REG.

CFB TIEBREAKERS

Conference ranks by `conf_pct`; ties use each conference's **official procedure**, registered as **season-scoped epochs** — SEC, Big Ten, Big 12, ACC, MAC, American, CUSA, Mountain West and Sun Belt. The **ACC's all-new 2026 policy** (h2h → SportSource SSR → draw, plus the alternate-game-count candidate pool: teams matching the leaders' wins *or* losses join the tie) applies from 2026; earlier seasons keep the 2024 cascade. Pass a `season` id to pick the epoch. Rungs: multi-team h2h, common opponents (+ descending), opponents' conf win%, SEC capped margin 42/48, Big 12 capped wins, `div_pct`, `cfp_ranked_final_week` (ranked tied team advances only if it *won* its final conference game), metric composite, APR, coin toss. Feed externals via `tiebreaker_data`: `analytics_ratings`, `cfp_rankings`, `apr`. The re-formed **Pac-12** has published no procedure → generic fallback. Skipped rungs land in `attr(, "tiebreak_notes")`.