Package 'fcipSupplementalLab'

January 22, 2026

Type Package

Title Research Framework for Supplemental Crop Insurance in the FCIP

Version 0.0.0.9000

Author Francis Tsiboe [aut, cre] (<https://orcid.org/0000-0001-5984-1072>)

Maintainer Francis Tsiboe <ftsiboe@hotmail.com>

Creator Francis Tsiboe

Description Provides a research framework for analyzing supplemental crop insurance products in the United States Federal Crop Insurance Program (FCIP). The package supports reproducible workflows to evaluate adoption and demand, actuarial performance and program soundness, fiscal exposure, risk reduction and income transfer, and basis risk and coverage quality. Functions emphasize transparent data preparation, diagnostic summaries, and modular analysis components suitable for reports and policy briefs.

License GPL-3 + file LICENSE

URL https://github.com/ftsiboe/fcipSupplementalLab

BugReports https://github.com/ftsiboe/fcipSupplementalLab/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports data.table, rfcip, stringr, urbnmapr, matrixStats, ggplot2

Remotes github::dylan-turner25/rfcip, github::UrbanInstitute/urbnmapr, github::dylan-turner25/rfsa

Suggests dplyr, tidyr, knitr, rmarkdown, mockery, withr, testthat (>= 3.0.0), piggyback, purrr, readr, devtools

LazyData true

Contents

`aggregate_expected_outcomes` *Aggregate and winsorize expected outcomes (year-level)*

Description

Loads all per-task expected outcome files for a given year, aggregates them, winsorizes key relative metrics within groups (5th-95th percentiles), and saves a single cleaned file.

Usage

```
aggregate_expected_outcomes(
  year,
  expected_directory = NULL,
  output_directory = NULL,
  study_environment,
 agent_identifiers = c("commodity_year", "state_code", "county_code", "type_code"),
  disaggregate = NULL
)
```

Arguments

`year`	Integer. Year to aggregate.
`expected_directory`	Character or NULL. Directory containing per-task `expected_*.rds` files for the year. If NULL, uses `file.path(study_environment$wd$dir_expected, year)`.
`output_directory`	Character or NULL. Directory to write the aggregated file. If NULL, uses `study_environment$wd$dir_expected`.

`study_environment`
: List. Must provide `$wd$dir_expected` (and is used to resolve defaults when directories are NULL).

`agent_identifiers`
: Character vector. Grouping keys used for by-group winsorization (default: `c("commodity_year","s`

`disaggregate`
: Character or NULL. Optional extra grouping column (e.g., `"combination"`). If provided but missing, it is created as `"ALL"`.

Details

Reads all `.rds` files under `expected_directory`, binds them, computes 5th and 95th percentiles for `Relmean`, `Relsd`, `Relcv`, `Rellapv`, `Rellrpv`, `Relnlapv`, `Relnlrpv`, `Relvar` within each group, caps values to that range, and writes `expected_<year>.rds` to `output_directory`.

Value

Invisibly returns the path to the saved file.

`build_agent_simulation_data` *Build agent simulation panel*

Description

Read cleaned agent-level simulation data for a crop year, unnest per-draw outcomes, filter to the requested draw(s), compute county-level expected yields, and add per-row revenue.

Usage

```
build_agent_simulation_data(
  year,
  sim,
  agents_directory = "data/cleaned_agents_data"
)
```

Arguments

`year`
: Integer. Crop year.

`sim`
: Integer vector. Draw number(s) to keep.

`agents_directory`
: Character. Directory containing cleaned agent data. Default: `"data/cleaned_agents_data"`.

Details

The function:

1. Loads `cleaned_agents_data_<year>.rds` from `agents_directory`.
2. Unnests draw pools: number, farm yield/price, and county yield/price.
3. Filters to `sim` (matching `rma_draw_number`).
4. Renames simulated fields to canonical names and floors negative county yields at zero.
5. Computes a planted-acre-weighted `expected_county_yield`.
6. Computes row-level `revenue` = `actual_farm_yield` * `actual_price` * `planted_acres`.

Value

A data.table containing all original columns plus:

- `expected_county_yield`
- `final_county_yield`
- `harvest_price`
- `revenue`

`build_supplemental_adoption_dynamics` *Build County-Level Supplemental Insurance Eligibility, Offering, and Adoption Dynamics*

Description

Builds a county-year-commodity panel describing **(i) eligibility**, **(ii) offering availability**, and **(iii) adoption intensity** for selected FCIP supplemental insurance products. The function combines:

- a cleaned SOB/TPU-derived dataset containing base insured acres and (optionally) supplemental adoption measures (either as shares or as acre totals), and
- RMA Actuarial Data Master (ADM) insurance-offer records from `A00030_InsuranceOffer` to determine whether a supplemental product is eligible and/or offered in each county-year-commodity.

Usage

```
build_supplemental_adoption_dynamics(sob, supplemental_stubs = NULL)
```

Arguments

`sob`	A data.frame or `data.table` containing (at minimum) `commodity_year`, `state_code`, `county_code`, `commodity_code`, `insured_acres`, and the supplemental measure columns referenced by `supplemental_stubs[[*]]$measure`. Measures may be shares (ending in `"_share"`) or acreage totals (e.g., `stax_area`, `mp_area`).
`supplemental_stubs`	Optional named list defining which supplemental products to include and how to interpret them. Each element must be a list with: • `measure`: column name in `sob` to use (share or acres) • `offering_codes`: insurance plan codes indicating the product is offered in ADM • `eligibile_codes`: insurance plan codes indicating the county is eligible in ADM If `NULL`, a default set is constructed for SCO, STAX, MP, ECO (90/95), HIP-WI, PACE, and FIP-SI.

Details

For each entry in supplemental_stubs, the function performs the following steps:

1. **Input validation:** checks that sob contains the required identifiers and the requested supplemental measure column (e.g., sco_share or stax_area).
2. **Compute adoption acres from SOB/TPU:**
   - If the measure name contains "_share", it computes adoption_acres = insured_acres * measure (interpreting the measure as a share of base insured acres).
   - Otherwise, it treats the measure as an acreage quantity and sets adoption_acres = measure and then sets insured_acres = adoption_acres (so the "insured_acres" field in the stub-specific output reflects the relevant acreage universe for that product).

   It then aggregates adoption acres and insured acres to commodity_year by state_code by county_code by commodity_code and restricts to commodity_year >= 2015.
3. **Construct ADM availability flags:** for each year present in sob, downloads ADM A00030_InsuranceOffer via rfcip::get_adm_data(), filters to the union of eligibility and offering plan codes for the stub, and creates two binary county-level indicators:
   - eligibile: equals 1 if any record exists with a plan in eligibile_codes
   - offered: equals 1 if any record exists with a plan in offering_codes

   (Indicators are collapsed using a max() rule within each county-year-commodity.)
4. **Build a complete county shell:** creates a complete county shell using urbnmapr::get_urbn_map("counties") and crosses all counties with the set of (commodity_year, commodity_code) pairs present in ADM so that counties with zero insured/adopted acres are retained in the final panel.
5. **Join and finalize:** left-joins ADM availability to the county shell, then joins adoption totals from SOB/TPU. Missing numeric fields are set to 0. The result is then aggregated deterministically to one row per county-year-commodity and labeled with supplemental_plan (the stub name) and a reconstructed county_fips.

The returned object stacks all stub-specific panels into one long table with a supplemental_plan identifier.

Value

A data.table with one row per commodity_year by state_code by county_code bycommodity_code by supplemental_plan containing:

- eligibile: 0/1 indicator for eligibility in ADM
- supplemental_offered: 0/1 indicator for offering availability in ADM
- insured_acres: aggregated acreage universe used for the stub
- supplemental_acres: aggregated adoption acres (derived from shares or acres)
- county_fips: 5-digit county FIPS (character)
- supplemental_plan: stub name (character)

clean_rma_sco_and_eco_adm

Build SCO/ECO/Area ADM table for a given year (adds SCO88/SCO90)

Description

Downloads yearly ADM fragments from GitHub Releases for *Supplemental SCO*, *Supplemental ECO*, and *Area* plans, aggregates key parameters by common grouping keys, linearly interpolates SCO rates to 88% and 90% (using AYP and, for years >= 2021, ECO anchors), and returns the cleaned, stacked table.

Usage

```
clean_rma_sco_and_eco_adm(year)
```

Arguments

`year`	Integer. commodity year (e.g., 2022).

Value

A data.table containing original SCO/ECO/Area ADM rows plus synthesized **SCO88** (`insurance_plan_code + 10`) and **SCO90** (`insurance_plan_code + 20`) rows with non-invalid `base_rate`.

Note

Requires internet access. Missing plan files for a year are skipped silently.

clean_rma_sobtpu

Clean and aggregate RMA Summary of Business (SOB-TPU) data

Description

Retrieves RMA SOB-TPU records for requested years, combining **live** years (last 5 years, fetched via `rfcip::get_sob_data()`) with **stable** years (downloaded from a prebuilt `.rds` release), then filters, harmonizes insurance plan codes, coverage levels, and unit structure codes, and returns an analysis-ready `data.table` aggregated to common keys.

Usage

```
clean_rma_sobtpu(
  years = as.numeric(format(Sys.Date(), "%Y")) - 1,
  insurance_plan = NULL,
  acres_only = TRUE,
  addon_only = TRUE,
  harmonize_insurance_plan_code = TRUE,
  harmonize_coverage_level_percent = TRUE,
  harmonize_unit_structure_code = TRUE
)
```

Arguments

years	Integer vector of commodity years.
insurance_plan	Optional integer vector of harmonized plan codes to keep after harmonization (1=YP, 2=RP, 3=RP-HPE). If NULL, keep all.
acres_only	Logical; keep only acres-level records. Default TRUE.
addon_only	Logical; exclude CAT (coverage_type_code == "C"). Default TRUE.
harmonize_insurance_plan_code	Logical; recode plans to (1,2,3). Default TRUE.
harmonize_coverage_level_percent	Logical; normalize coverage levels to decimal in 0.50 to 0.95 at 0.05 steps. Default TRUE.
harmonize_unit_structure_code	Logical; recode unit structure to (OU, BU, and EU). Default TRUE.

Value

A data.table aggregated to the keys with columns: insured_acres, endorsed_acres, liability_amount, total_premium_amount, subsidy_amount, indemnity_amount.

---

compute_base_policy_outcomes

Compute base-policy outcomes

Description

Vectorized **data.table** implementation of base-policy guarantees, acres/liability, premium pieces (total/subsidy/producer), and indemnity, plus a tidy column subset for downstream joins.

Usage

```
compute_base_policy_outcomes(cleaned_agents_data)
```

Arguments

cleaned_agents_data	A data.frame or data.table with the required columns (see error message if any are missing).

Details

Requires a set of core inputs (e.g., yields, prices, coverages, acres) and returns the standard monetary outputs for each policy row. Price risk is handled via a new_insurance_guarantee that depends on plan code.

Value

A data.table with key fields and outputs: insured_acres, liability, total_premium, subsidy_amount, producer_premium, indemnity, revenue, and supporting fields such as harvest_price, expected_county_yield, final_county_yield, new_insurance_guarantee, projected_price.

compute_expected_outcomes *Compute expected outcomes and risk metrics from simulation outputs*

Description

Joins cleaned agent records to simulation files, then computes expected (mean/sd) revenues, downside-risk measures (loss-side residual moments), relative improvements with insurance, and insurance performance statistics. Writes a single `.rds` result file and returns its path (invisibly).

Usage

```
compute_expected_outcomes(
  year,
  task_id,
  agents_directory = "data/cleaned_agents_data",
  simulation_directory = NULL,
  output_directory = NULL,
  study_environment,
 agent_identifiers = c("commodity_year", "state_code", "county_code", "commodity_code",
    "type_code", "practice_code", "unit_structure_code", "insurance_plan_code",
    "coverage_level_percent", "insured_acres"),
  disaggregate = NULL
)
```

Arguments

`year`	Integer (scalar). Analysis year (used to resolve input/output paths).
`task_id`	Integer or integer vector. Pseudo-task partition(s) to keep; the function cycles a 1..500 index over agent rows and filters to these values.
`agents_directory`	Character. Directory containing `cleaned_agents_data_<year>.rds`.
`simulation_directory`	Character or `NULL`. Directory with simulation `.rds` files; default is `file.path(study_environment$` `year)`.
`output_directory`	Character or `NULL`. Directory to write results; default is `file.path(study_environment$wd$dir_ex` `year)`.
`study_environment`	List. Must include `wd$dir_sim` and `wd$dir_expected` if the corresponding directory arguments are `NULL`.
`agent_identifiers`	Character vector. Columns that identify agent units and define aggregation groups (used for joins and by); default includes year, location, crop, unit structure, plan, coverage, and acres.
`disaggregate`	Character or `NULL`. Optional extra column to disaggregate by (for example, `"combination"`). If provided but missing after the join, the column is created and set to `"ALL"`.

Details

Pipeline

1. Load agent data and keep only `agent_identifiers`; coerce to `data.table`.
2. Assign a pseudo `task` (cycles 1..500), then filter to `task_id`.
3. Guardrails:
   - Stop if no simulation files are found.
   - Stop if the combined join yields zero rows.
   - Validate required numeric columns: `revenue`, `indemnity`, `producer_premium`, `liability`, `total_premium`, `subsidy_amount`.
   - Use `safe_div()` to avoid `Inf`/`NaN` on zero or non-finite denominators.
4. Compute revenues (floored at 0): `Revenue` and `Revenue_Inc` (= revenue + indemnity
   - producer premium).
5. By `uid` (=`agent_identifiers` plus `disaggregate` if provided), compute means, sds, residual-based downside measures (loss-only squared residuals and their frequency), and derived statistics (variance, CV, LAPV, LRPV, normalized forms).
6. Compute **relative** metrics (insured vs. uninsured ratios): `Relmean`, `Relsd`, `Relcv`, `Rellapv`, `Rellrpv`, `Relnlapv`, `Relnlrpv`, `Relvar`. Base `Revenue*` statistics are dropped before the final merge to keep results compact.
7. Aggregate insurance performance by group: mean `liability`, `total_premium`, `subsidy_amount`, `producer_premium`, `indemnity`, premium and LCR rates (`Simrate`, `SimrateP`, `Simsuby`, `Simlcr`), and **group sums** for `lr_indemnity` and `lr_premium`. Merge with the relative metrics.

Join note The join uses `data[simdt, on = <keys>, nomatch = 0]`, i.e., it returns rows aligned to the simulation table entries that match the agent keys.

Value

Invisibly returns the saved file path (`expected_<year>_<task-range>.rds`).

`compute_supplemental_current`
Aggregate supplemental results for the current environment

Description

Scale selected SCO/ECO factors by base-policy weights (`sco`, `eco90`, `eco95`), aggregate by policy keys, append base outcomes, and label the rollup as `"Basic+CURRENT"`.

Usage

```
compute_supplemental_current(base_policy_data, supplemental_factors)
```

Arguments

`base_policy_data`
: data.table. Base-policy outcomes (contains keys, weights, and monetary fields).

`supplemental_factors`
: data.table. Supplemental outcomes from `compute_supplemental_factors` including `sup`.

Value

A data.table aggregated by policy keys with: `revenue`, `liability`, `total_premium`, `subsidy_amount`, `producer_premium`, `indemnity`, and `combination`.

`compute_supplemental_factors` *Compute supplemental policy factors (SCO/ECO)*

Description

Compute shallow-loss protection, premiums, and indemnities for one SCO/ECO endorsement offering, aligning plan families and joining ADM rating inputs.

Usage

```
compute_supplemental_factors(base_policy, adm, plan, subsidy, trigger)
```

Arguments

`base_policy`	data.table. Base-policy rows (keys, yields, prices, liability, etc.).
`adm`	data.table. Rating inputs with `base_rate` and join keys.
`plan`	Integer. Plan code in the offering (e.g., 31-33, 51-53, 87-89).
`subsidy`	Numeric. Subsidy factor (e.g., `0.65`, `0.80`, `0.44`).
`trigger`	Numeric. Coverage trigger level (e.g., `0.86`, `0.90`, `0.95`).

Details

Handles plan families via offsets (31-33, 41-43, 51-53, 87-89). For plans 87-89 (ECO), the `coverage_level_percent` for ADM is matched to the `trigger` (with a small tolerance), and the subsidy factor special-case is applied for underlying plan code 1. Emits a standard `sup` label like `"SCO8665"` or `"ECO9544"`.

Value

A data.table with columns: `commodity_year`, `state_code`, `county_code`, `commodity_code`, `type_code`, `practice_code`, `unit_structure_code`, `insurance_plan_code`, `coverage_level_percent`, `liability`, `total_premium`, `subsidy_amount`, `producer_premium`, `indemnity`, `sup`.

`compute_supplemental_full` *Aggregate supplemental full-participation results*

Description

Given selected `sup` labels, sum their monetary fields, append base outcomes, and produce a final rollup by policy keys with a descriptive `combination` label.

Usage

```
compute_supplemental_full(
  base_policy_data,
  supplemental_factors,
  supplemental_pick
)
```

Arguments

`base_policy_data`
: data.table. Base-policy outcomes.

`supplemental_factors`
: data.table. Results from `compute_supplemental_factors`.

`supplemental_pick`
: Character vector of `sup` labels to include.

Details

The function self-filters `supplemental_factors` to the provided `supplemental_pick` (after dropping empties), aggregates within keys, appends base outcomes, and re-aggregates.

Value

A data.table aggregated by the policy keys with: `revenue`, `liability`, `total_premium`, `subsidy_amount`, `producer_premium`, `indemnity`, and `combination`.

`compute_supplemental_incremental` *Compute incremental supplemental results at an adoption rate*

Description

Build an incremental scenario by scaling `SCO8665` supplemental dollars by a user-specified adoption rate, aggregating by keys, and appending base outcomes.

Usage

```
compute_supplemental_incremental(
  base_policy_data,
  supplemental_factors,
  adoption_rate
)
```

Arguments

base_policy_data	data.table. Base-policy outcomes.
supplemental_factors	data.table. Output from compute_supplemental_factors filtered to sup == "SCO8665".
adoption_rate	Numeric. Percentage (e.g., 10 for 10\ scale incremental supplemental amounts.

Value

A data.table aggregated by the policy keys with: revenue, liability, total_premium, subsidy_amount, producer_premium, indemnity, and combination.

---

dispatcher_supplemental_simulation

Dispatcher: simulate supplemental outcomes for one draw

Description

Orchestrate the full supplemental simulation workflow for a given crop year and draw: build the agent panel, compute base-policy results, generate supplemental factors, assemble *Current*, *Full*, and *Incremental* scenarios, and write the combined results to disk.

Usage

```
dispatcher_supplemental_simulation(
  sim,
  year,
  agents_directory = "data/cleaned_agents_data",
 cleaned_rma_sco_and_eco_adm_file_path = "data/cleaned_rma_sco_and_eco_adm.rds",
  output_directory = NULL
)
```

Arguments

sim	Integer. Draw number used in data building and the filename.
year	Integer. Crop year.
agents_directory	Character. Directory for cleaned agents data.
cleaned_rma_sco_and_eco_adm_file_path	Character. Path to RDS of SCO/ECO ADM with join keys and base_rate. Default: "data/cleaned_rma_sco_and_eco_adm.rds".
output_directory	Character or NULL. Where to write results; see Details for default behavior.

Details

The pipeline:

1. `build_agent_simulation_data` to construct the panel.
2. `compute_base_policy_outcomes` for base outcomes.
3. `study_scenarios` to enumerate offerings/mixes.
4. Load SCO/ECO ADM; filter to `commodity_year == year`; average `base_rate` by key; drop invalid/zero rates.
5. Loop offerings through `compute_supplemental_factors`.
6. Build scenarios:
   - *Current*: `compute_supplemental_current`.
   - *Full*: `compute_supplemental_full`.
   - *Incremental*: `compute_supplemental_incremental`.
7. Aggregate base-only results, `rbind` all scenarios, and save as `simXXX.rds` in `output_directory`.

If `output_directory` is `NULL`, it defaults to `file.path(study_environment$wd$dir_sim, year)` (ensure `study_environment$wd$dir_sim` exists in the calling environment).

Value

Invisibly writes `simXXX.rds` to `output_directory`.

`ers_theme` *ERS Theme*

Description

ERS Theme

Usage

```
ers_theme()
```

Source

coppied from https://github.com/USDA-REE-ERS/MTED-Theme on 08/01/2025

Examples

```
ggplot2::ggplot() + ers_theme()
```

`farm_performance_metrics` *Farm performance metrics by scenario and disaggregate*

Description

Load `expected_<year>.rds`, derive outcome variables, compute deltas vs. baselines, trim extremes using quantile limits, aggregate (weighted mean/median) by requested disaggregates, and save a summarized `.rds`. Returns the saved path invisibly.

Usage

```
farm_performance_metrics(
  year,
 agent_identifiers = c("commodity_year", "state_code", "county_code", "commodity_code",
    "type_code", "practice_code", "unit_structure_code", "insurance_plan_code",
    "coverage_level_percent"),
 outcome_list = c("its", "Iits", "rrs1", "rrs2", "rrs3", "Irrs1", "Irrs2", "Irrs3",
   "sner1", "sner2", "sner3", "Simrate", "SimrateP", "Simsuby", "Simlcr", "rrp1",
    "rrp2", "rrp3", "itp"),
  combo,
  weight_variable = NULL,
  expected_directory = NULL,
  draw = NULL,
  draw_list_file_path = NULL,
  disaggregates = NULL,
  output_file_path = NULL,
  distributional_limits = c(0.05, 0.95)
)
```

Arguments

`year`	Policy year used to locate `expected_<year>.rds`.
`agent_identifiers`	Character vector of ID columns for grouping prior to long-pivot and averaging.
`outcome_list`	Character vector of outcome columns to reshape and aggregate.
`combo`	Target scenario (e.g., `"Basic+CURRENT"`, `"Basic+SCO8665"`, or another).
`weight_variable`	`NULL` for equal weights (=1) or a character name of a numeric weight column.
`expected_directory`	Directory containing `expected_<year>.rds`.
`draw`	Optional draw identifier used for filtering and filename tag.
`draw_list_file_path`	Optional path to an RDS (named list) with the draw table; required if `draw` is not `NULL`.
`disaggregates`	Optional character vector of additional disaggregate columns (alongside `"FCIP"`).
`output_file_path`	Output file path

distributional_limits
: Numeric length-2 vector of lower/upper probabilities (e.g., c(0.05, 0.95)); must satisfy 0 < p1 < p2 < 1.

Details

Steps:

1. Filter rows to combination %in% {"Basic+CURRENT", combo, "Basic+SCO8665"}.
2. Create derived metrics: rrs1/2/3, its, flags Irrs*/Iits, sner*, percent/level transforms (rrp*, itp), and scale Sim* by 100.
3. Reshape to long on outcome_list, drop non-finite values, average within identifiers (agent_identifiers, and weight_variable if provided), scenario, variable.
4. Join baselines: if combo != "Basic+CURRENT", add "Basic+CURRENT" as base00; if combo = {"Basic+SCO8665", add "Basic+SCO8665" as base01. Compute chglvl00/01 and chgpct00/01 (guard divide-by-zero).
5. Build labels PLAN, RPYP, COV, STRUCT.
6. Compute trimming limits per (variable, combination, state_code, IRR, commodity_code) using distributional_limits (default c(0.05,0.95)), require n greater or equal to 20, and cap to *T columns.
7. For each of c("FCIP", disaggregates), compute weighted mean and weighted median of raw and trimmed metrics; stack results and write output.

Value

Invisibly returns the character path of the saved .rds.

Required columns

All agent_identifiers, plus: combination, state_code, county_code, commodity_code, type_code, practice_code, IRR, Relcv, Relnlrpv, Relnlapv, Relmean, Simrate, SimrateP, Simsuby, Simlcr, coverage_level_percent, unit_structure_code, insurance_plan_code. If weight_variable is not NULL, that column must exist and be numeric.

Note

Baseline joins use nomatch = 0 by design, so rows missing in the baseline are dropped before delta computation. Change to nomatch = NA if you prefer to retain such rows with NA deltas.

See Also

data.table::data.table, data.table::melt, matrixStats::weightedMedian

`fcipSupplementalLab_controls` *Create a control list of adjustment factors for FCIP-related packages*

Description

Create a control list of adjustment factors for FCIP-related packages

Usage

```
fcipSupplementalLab_controls()
```

Value

A named list of control parameters, ready to be passed to other simulation functions.

`get_fcip_agents` *Build FCIP record-level dataset for a commodity year from calibration artifacts and RMA reference tables*

Description

Downloads year-specific calibration artifacts from GitHub (revenue draws, calibrated yields, and compressed projected prices), restricts revenue-draw records to insurance pools present in `relevant_adm`, joins SOB/TPU reference records from `relevant_sob` using explicit and validated keys, computes observed premium-rate and subsidy-share measures, attaches yield and price fields, filters invalid records, and returns a streamlined `data.table`.

Usage

```
get_fcip_agents(
  year,
  relevant_adm,
  relevant_sob,
  keep_variables = NULL,
  temporary_dir = tempdir()
)
```

Arguments

`year`	Integer. Commodity year to process (e.g., 2015).
`relevant_adm`	data.table. Pre-filtered administrative table defining the set of relevant insurance pools. Must contain at least `state_code`, `county_code`, `commodity_code`, `type_code`, and `practice_code`.
`relevant_sob`	data.table. Pre-filtered SOB/TPU-style table used to construct `producer_id` and to join reference records into the revenue-draw data. Must contain the fields required to build `producer_id`: `state_code`, `county_code`, `commodity_code`, `type_code`, `practice_code`, `unit_structure_code`, `insurance_plan_code`, `coverage_type_code`, `coverage_level_percent`. If present, `commodity_year` may also be used in joins.

keep_variables Character vector of additional column names to retain (if present after all joins and filtering). Default is NULL.

temporary_dir Character. Directory used to store downloaded calibration artifacts. Defaults to tempdir(). The directory will be created if it does not exist.

Details

The function expects calibration artifacts to be available as GitHub release assets with the following structure:

- Repository ftsiboe/rfcipCalibrate, tag revenue_draw: revenue_draw_<year>.rds
- Repository ftsiboe/rfcipCalibrate, tag calibrated_yield: calibrated_yield_<year>.rds
- Repository ftsiboe/rfcipCalcPass, tag adm_compressed: <year>_A00810_Price.rds

Revenue-draw records are first restricted to insurance pools observed in relevant_adm. SOB/TPU records are then joined **into** the revenue-draw data (inner join), ensuring the unit of observation remains the revenue-draw / policy-unit record.

Observed ratios are computed using NA-safe denominators:

- observed_premium_rate = total_premium_amount / liability_amount (rounded to 8 decimals)
- observed_subsidy_percent = subsidy_amount / total_premium_amount (rounded to 3 decimals)

Denominators that are non-finite, NA, or non-positive yield NA_real_.

Projected prices are aggregated to the mean by commodity_year, state_code, county_code, commodity_code, type_code, and practice_code, and are left-joined into the output so no additional rows are created.

The function filters out records with non-finite calibrated_yield values. Records with missing observed ratios are retained.

Convenience columns are added for downstream FCIP pipelines: planted_acres = insured_acres, and price_election, insured_share, and damage_area_rate are set to 1.

Value

A data.table containing one row per retained FCIP record, including identifying keys, selected calibration and draw fields, observed premium and subsidy measures, projected price, and any variables listed in keep_variables that exist. Returns NULL if an error occurs.

Side effects and requirements

- Downloads external files using piggyback::pb_download().
- Reads and writes temporary RDS files in temporary_dir.
- Requires the calibration artifacts to be accessible via GitHub releases.

`get_study_releases` *Download all assets from a GitHub release with rate limiting*

Description

Downloads all files attached to a specified GitHub release tag while **throttling requests** to avoid GitHub rate limits and abuse protection. This helper is designed for releases containing many or large assets (e.g., `.rds` outputs generated on HPC systems).

Usage

```
get_study_releases(
  owner,
  repository,
  release_tag,
  output_directory = NULL,
  github_token = NULL,
  sleep_seconds = 3,
  max_rounds = 3
)
```

Arguments

`owner`	Character string giving the GitHub repository owner (e.g., `"ftsiboe"`).
`repository`	Character string giving the GitHub repository name (e.g., `"indexDesignWindows"`).
`release_tag`	Character string specifying the GitHub release tag whose assets should be downloaded.
`output_directory`	Optional character string specifying the local directory where release assets should be saved. Defaults to `data-raw/releases/{release_tag}`.
`github_token`	Optional GitHub personal access token (PAT). Passed to **piggyback** via `.token`. Strongly recommended.
`sleep_seconds`	Numeric scalar giving the number of seconds to pause between individual file downloads. Increasing this value reduces the likelihood of triggering GitHub rate limits.
`max_rounds`	Integer giving the maximum number of retry rounds. Each round attempts to download any files still missing locally.

Details

The function downloads assets incrementally, pauses between requests, and retries failed downloads across multiple rounds. Already-downloaded files are skipped, allowing the function to safely resume after interruptions or rate-limit errors.

The function:

1. Constructs a default output directory (`data-raw/releases/{release_tag}`) if none is supplied.
2. Queries GitHub once to obtain the list of release assets.
3. Downloads assets **one at a time** using **piggyback**.

4. Pauses for `sleep_seconds` between downloads to reduce request bursts.
5. Retries failed or missing downloads for up to `max_rounds`.
6. Skips files that already exist locally.

This approach is especially useful when GitHub returns repeated `HTTP 403 (Forbidden)` errors during bulk downloads.

Authentication via a GitHub personal access token (PAT) is strongly recommended, even for public repositories.

Value

Invisibly returns `NULL`. Files are downloaded for their side effects.

`get_supplemental_adoption` *Build a Base-Policy Panel with Supplemental Adoption Measures from RMA SOB-TPU*

Description

Constructs an analysis-ready panel of **base insurance outcomes** and appends measures of **supplemental insurance adoption** and/or **supplemental acreage totals** for a user-specified set of FCIP supplemental products.

The function first aggregates base (individual-based) insurance outcomes from `sob_full`, harmonizing plan code 90 to base plan 1 where applicable. It then sequentially augments this base panel with supplemental adoption measures computed via `get_supplemental_shares()` and with direct acreage aggregates for selected area-based products.

Usage

```
get_supplemental_adoption(
  sob_full,
  supplemental_codes = c(31:33, 87:89, 35:36, 16:17, 67:69, 26:28, 37, 38),
  disaggregates = NULL
)
```

Arguments

`sob_full`	A `data.table` of cleaned SOB-TPU records containing both base and supplemental policies. Must include insured acres, endorsed acres, financial totals, and standard FCIP identifier columns.
`supplemental_codes`	Integer vector of supplemental insurance plan codes to include. Defaults to a comprehensive set of FCIP supplemental products.
`disaggregates`	Optional character vector defining the primary analysis grain for adoption measures. Defaults to commodity-year by county by commodity/type/practice.

Details

Output content depends on which supplemental plan codes are supplied:

- **SCO (31-33):** Adds `sco_share`, measuring the share of base insured acres stacked with SCO.
- **ECO (87-89):** Adds coverage-specific adoption measures `eco90_share` and `eco95_share`.
- **PACE (26-28):** Adds `pace_share`.
- **HIP-WI (37):** Adds `hipwi_share` (not disaggregated by base plan or coverage level).
- **FIP-SI (38):** Adds `fipsi_share`.
- **MP (16-17):** Adds `mp_area`, representing aggregated acres insured under Margin Protection.
- **STAX (35-36):** Adds `stax_area`, representing aggregated acres insured under STAX.

Adoption shares are based on `endorsed_acres` in the numerator and base-policy `insured_acres` in the denominator. Area-based products (MP and STAX) are returned as acreage totals rather than shares.

After all merges, the function enforces basic bounds:

- Non-finite values are set to 0
- Negative values are truncated to 0
- Share variables are capped at 1

Value

A `data.table` containing base insured acreage and financial totals at the base-policy aggregation grain, augmented with one or more supplemental adoption variables (ending in `"_share"`) and/or supplemental acreage totals (ending in `"_area"`).

---

get_supplemental_shares

Compute Supplemental Product Shares Relative to a Base Insured-Acres Denominator

---

Description

Computes **supplemental adoption measures** by expressing supplemental acres as a fraction of **base insured acres** from a companion base dataset. This function is a low-level helper used by `calculate_supplemental_adoption()` to construct product-level uptake variables such as `sco_share`, `pace_share`, `hipwi_share`, or coverage-specific measures like `eco90_share` and `eco95_share`.

Usage

```
get_supplemental_shares(
  sob_base,
  sob_full,
  supplemental_codes,
  supplemental_name,
  disaggregates = NULL,
  base_anchor = NULL,
  track_base_plan = TRUE,
  track_base_coverage_level = TRUE,
  split_by_coverage_level = FALSE
)
```

Arguments

sob_base	A data.table containing **base-policy outcomes** used as the denominator. Must include insured_acres and the identifier columns needed to form merge keys.
sob_full	A data.table containing the **full SOB-TPU extract** (base and supplemental records). Must include endorsed_acres, insurance_plan_code, and the identifier columns referenced in disaggregates.
supplemental_codes	Integer vector of SOB insurance plan codes identifying the supplemental product(s) whose adoption is to be measured.
supplemental_name	Character scalar used to construct output column names. The suffix "_share" is appended internally.
disaggregates	Optional character vector defining the primary aggregation grain. Defaults to c("commodity_year","state_code","county_code","commodity_code", "type_code","prac
base_anchor	Optional integer used to re-anchor stacked plan codes to base plan codes.
track_base_plan	Logical. If TRUE, adoption shares are computed separately by base insurance plan.
track_base_coverage_level	Logical. If TRUE, adoption shares are computed separately by base coverage level.
split_by_coverage_level	Logical. If TRUE, returns separate share columns by coverage level percent.

Details

The function implements the following steps:

1. **Supplemental aggregation (numerator):** Filters sob_full to supplemental_codes and aggregates endorsed_acres to form supplemental_area at the requested disaggregation level. Depending on user settings, aggregation may additionally track the base plan (insurance_plan_code) and/or the base coverage level (coverage_level_percent).
2. **Plan-code anchoring (optional):** If base_anchor is supplied, insurance_plan_code is shifted by subtracting the anchor value. This is typically used to map stacked-plan codes back onto their underlying base plans (e.g., SCO codes 31-33 -> base plans 1-3 by subtracting 30).
3. **Base aggregation (denominator):** Aggregates sob_base to the intersection of keys shared with the supplemental aggregation and computes total insured_acres for use as the denominator.
4. **Share construction:** Computes supplemental_share = supplemental_area / insured_acres when insured_acres > 0; otherwise assigns NA.
5. **De-duplication and reshaping:** If multiple rows exist for the same identifier set and coverage label, the function collapses duplicates by taking the **mean** share within each identifier group before reshaping to wide format.

If split_by_coverage_level = TRUE, output columns are labeled by coverage level percent (rounded) and named <supplemental_name><CL>_share (e.g., eco90_share). In this case, coverage_level_percent is removed from merge keys so that shares vary only along the remaining disaggregation dimensions.

All non-finite share values (NA, Inf, -Inf, NaN) are coerced to **0** in the final output.

Value

A data.frame (resulting from tidyr reshaping) containing the requested identifiers and one or more wide columns ending in "_share".

---

setup_environment *Setup Project Environment*

Description

Initializes the working environment for a project by creating required directories, setting useful global options, and fixing the random seed.

Usage

```
setup_environment(
  year_beg = 2001,
  year_end = as.numeric(format(Sys.Date(), "%Y")),
  seed = 1980632,
  project_name,
 local_directories = list(file.path("data-raw", "output"), file.path("data-raw",
    "scripts"), file.path("data")),
  fastscratch_root = NULL,
  fastscratch_directories = NULL
)
```

Arguments

year_beg	Integer. Beginning year of the analysis (default: 2001).
year_end	Integer. Ending year of the analysis (default: current system year).
seed	Integer. Random seed for reproducibility (default: 1980632).
project_name	Character. Project name (required). Used to build fast-scratch directory paths.
local_directories	List of project-local directories to create (default: list("data-raw/output", "data-raw/scripts", "data")).
fastscratch_root	Optional character. Root directory for fast-scratch files. If NULL, it is set automatically: • Windows: "C:/fastscratch" • Linux/macOS: "/fastscratch/<username>"
fastscratch_directories	List of fast-scratch subdirectories (relative to <fastscratch_root>/<project_name>) to create. If NULL, no fast-scratch subdirectories are created and wd is returned as an empty list.

Details

The function ensures the requested directories exist, creating them if necessary. Directory keys in the returned `wd` list are the basenames of the provided `fastscratch_directories`.

It also sets the following options:

- `options(scipen = 999)` (turns off scientific notation)
- `options(future.globals.maxSize = 8 * 1024^3)` (~8 GiB)
- `options(dplyr.summarise.inform = FALSE)` (quiet **dplyr**)

Finally, the random number generator is seeded with the provided `seed`.

Value

A list with:

wd Named list of created fast-scratch directories. Empty if `fastscratch_directories = NULL`.

year_beg Starting year (integer).

year_end Ending year (integer).

seed Seed value used for RNG.

`study_scenarios` *Build study scenarios (SCO/ECO offerings and mixes)*

Description

Define the endorsement offerings (plan family - trigger - subsidy - label) and the full-participation SCO/ECO mixes to evaluate for a given year.

Usage

```
study_scenarios(year)
```

Arguments

`year`	Integer. Crop year used to determine available ECO variants.

Details

For years `>= 2021`, ECO 90/44 and 95/44 variants are added and the participation set is expanded accordingly. Offerings create `sup` labels such as `"SCO8665"`, `"SCO9080"`, `"ECO9044"`, `"ECO9544"`.

Value

A named list with:

- `offerings`: data.table of `insurance_plan_code`, `Trigger`, `plan`, `Subsidy_factor`.
- `full_participation`: data.table of SCO/ECO label combinations to test (columns `sco`, `eco`).

Index