Package ‘preventr’

January 27, 2025

Title An Implementation of the PREVENT and Pooled Cohort Equations

Version 0.11.0

Description Implements the American Heart Association Predicting
Risk of cardiovascular disease EVENTs (PREVENT) equations from Khan
SS, Matsushita K, Sang Y, and colleagues (2023)
<doi:10.1161/CIRCULATIONAHA.123.067626>, with optional comparison
with their de facto predecessor, the Pooled Cohort Equations from the
American Heart Association and American College of Cardiology (2013)
<doi:10.1161/01.cir.0000437741.48606.98> and the revision to the Pooled
Cohort Equations from Yadlowsky and colleagues (2018)
<doi:10.7326/M17-3011>.

License MIT + file LICENSE

URL `https://martingmayer.com/preventr`,
`https://github.com/martingmayer/preventr`,
`https://martingmayer.shinyapps.io/prevent-equations/`

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

Imports dplyr

Suggests data.table, devtools, knitr, purrr, rmarkdown, testthat (>=
3.0.0), utils

Config/testthat/edition 3

Config/Needs/website rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Martin Mayer [aut, cre, cph] (<`https://orcid.org/0000-0003-3210-2274`>)

Maintainer Martin Mayer <`mmayer@ebsco.com`>

Repository CRAN

Date/Publication 2025-01-26 23:20:01 UTC

Contents

---

app *Navigate to browser-based Shiny implementation of PREVENT equations*

Description

This function opens a browser window (using the user's default browser) and navigates to the Shiny app located at:

`https://martingmayer.shinyapps.io/prevent-equations`

Easier-to-remember URLs:

- https://tiny.cc/prevent-equations
- https://tiny.cc/preventequations

The app includes risk visualization and several options for customizing the output.

Usage

```
app(...)
```

Arguments

...	Not used. Reserved for future use.

Value

Returns `NULL` invisibly after opening app in your default browser.

Examples

```
app()
```

`estimate_risk` *Estimate risk of cardiovascular events using the American Heart Association (AHA) Predicting Risk of cardiovascular disease EVENTs (PREVENT) equations, with optional comparison to their* de facto *predecessor, the Pooled Cohort Equations (PCEs) from the AHA and American College of Cardiology (ACC)*

Description

`estimate_risk()` and `est_risk()` are the same function, with the latter being a function synonym for those who favor syntactical brevity.

Estimation via the PREVENT equations includes both 10- and 30-year risk of 5 events:

- Total cardiovascular disease (CVD), which includes atherosclerotic CVD (ASCVD) and heart failure as defined below
- ASCVD, which includes coronary heart disease (CHD) and stroke as defined below
- Heart failure (often abbreviated HF, but not herein)
- CHD, which includes nonfatal myocardial infarction (MI) and fatal CHD
- Stroke

Estimation via the PCEs includes 10-year risk of ASCVD. The title of the function focuses on the "official" version of the PCEs from the AHA/ACC, but this function permits estimation via the revised PCEs released by Yadlowsky and colleagues in 2018. Further details are in the "Arguments" section.

See also the README for this package, which goes into additional detail about the PREVENT equations (site, GitHub).

Usage

```
estimate_risk(
  age,
  sex,
  sbp,
  bp_tx,
  total_c,
  hdl_c,
  statin,
  dm,
  smoking,
  egfr,
  bmi,
  hba1c = NULL,
  uacr = NULL,
  zip = NULL,
  model = NULL,
  time = "both",
  chol_unit = "mg/dL",
  optional_strict = FALSE,
  quiet = is.data.frame(use_dat),
  collapse = is.data.frame(use_dat),
  use_dat = NULL,
  add_to_dat = is.data.frame(use_dat),
  progress = is.data.frame(use_dat)
)

est_risk(
  age,
  sex,
  sbp,
  bp_tx,
  total_c,
  hdl_c,
  statin,
  dm,
  smoking,
  egfr,
  bmi,
  hba1c = NULL,
  uacr = NULL,
  zip = NULL,
  model = NULL,
  time = "both",
  chol_unit = "mg/dL",
  optional_strict = FALSE,
  quiet = is.data.frame(use_dat),
  collapse = is.data.frame(use_dat),
  use_dat = NULL,
  add_to_dat = is.data.frame(use_dat),
  progress = is.data.frame(use_dat)
)
```

Arguments

`age`	Numeric (required predictor variable): Age in years, from 30-79. Note the PCEs have a lower age limit of 40, so for ages 30-39, the function will only provide estimates for the PREVENT equations, irrespective of whether a user also requests estimation via the PCEs via the `model` argument (more precisely, the function will still carry out the estimation from the PCEs, but will return `NA`).
`sex`	Character (required predictor variable): Either `"female"` or `"male"` (`"f"` and `"m"` are accepted abbreviations).
`sbp`	Numeric (required predictor variable): Systolic blood pressure (SBP) in mmHg, from 90-180; see the "Details" section for more information about the upper bound of the range.

`bp_tx`	Logical or numeric equivalent (required predictor variable): Whether the person is on blood pressure treatment, either `TRUE` or `FALSE` (1 or 0 are accepted as alternative input).
`total_c`	Numeric (required predictor variable): Total cholesterol in mg/dL or mmol/L (see `chol_unit` argument), from 130-320 (for `chol_unit = "mg/dL"`) or 3.36-8.28 (for `chol_unit = "mmol/L"`).
`hdl_c`	Numeric (required predictor variable): High-density lipoprotein cholesterol (HDL-C) in mg/dL or mmol/L (see `chol_unit` argument), from 20-100 (for `chol_unit = "mg/dL"`) or 0.52-2.59 (for `chol_unit = "mmol/L"`).
`statin`	Logical or numeric equivalent (required predictor variable): Whether the person is taking a statin, either `TRUE` or `FALSE` (1 or 0 are accepted as alternative input).
`dm`	Logical or numeric equivalent (required predictor variable): Whether the person has diabetes mellitus (DM), either `TRUE` or `FALSE` (1 or 0 are accepted as alternative input).
`smoking`	Logical or numeric equivalent (required predictor variable): Whether the person is currently smoking (which PREVENT defines as cigarette use within the last 30 days), either `TRUE` or `FALSE` (1 or 0 are accepted as alternative input).
`egfr`	Numeric or call (required predictor variable): Estimated glomerular filtration rate (eGFR) in mL/min/1.73m2, entered either as a numeric from 15-140 or as a call to `calc_egfr()` or synonyms, as described in the "Details" section.
`bmi`	Numeric or call (required predictor variable): Body mass index (BMI) in kg/m2, entered either as a numeric from 18.5-39.9 or as a call to `calc_bmi()` or synonyms, as described in the "Details" section.
`hba1c`	Numeric (optional predictor variable): Glycated hemoglobin (HbA1c) in %, from 4.5-15; see the "Details" section for more information about the lower bound of the range.
`uacr`	Numeric (optional predictor variable): Urine albumin-to-creatinine ratio (UACR) in mg/g, from 0.1-25000.
`zip`	Character (optional predictor variable): ZIP code of the person's residence, used to estimate the Social Deprivation Index (SDI); see the "Details" section for more information.
`model`	Character or list (optional behavior variable): • If character, the PREVENT model to use, one of `NULL` (the default), `"base"` (the base model), `"hba1c"` (the base model adding HbA1c), `"uacr"` (the base model adding UACR), `"sdi"` (the base model adding SDI), or `"full"` (the base model adding HbA1c, UACR, and SDI). If `NULL`, the model will be determined by algorithm specified in the "Details" section, and this is the intended argument for most users. The ability to specify mainly exists for specific use cases (e.g., research purposes). • If passing a list, the list must have the following elements unless otherwise specified (any other elements in the list will be ignored): – `main_model` (character): The PREVENT model to use, following the same requirements specified for when `model` is character. This element is required only if the user desires to specify which PREVENT model to use. It can otherwise be omitted, in which case the function will

set `main_model` to `NULL` (which has the same impact as when `model` is `NULL`).

- `other_models` (character): The PCEs to use, one of `"pce_orig"` (for the original PCEs released by the ACC/AHA in 2013), `"pce_rev"` (for the revised PCEs released by Yadlowsky and colleagues in 2018, but not officially endorsed by ACC/AHA), or `"pce_both"` (for both).
- `race_eth` (character): The race and ethnicity of the person, which is required by the PCEs. One of `"Black"` (for non-Hispanic Black), `"White"` (for non-Hispanic White), or `"Other"` (`"B"`, `"W"`, or `"O"` are accepted as alternative input, as are lowercase versions of the full word or its first-letter abbreviation). See the "Details" section for further discussion.

`time` Character or numeric (optional behavior variable): Whether to estimate risk over 10 or 30 years, one of `"both"` (character; the default); `10` (numeric), `"10"` (character), or `"10yr"` (character); or `30` (numeric), `"30"` (character), or `"30yr"` (character). Two additional things to note:

- If a user requests estimation over a 30-year time horizon, but the user also requests estimation via the PCEs, a 10-year time horizon will automatically be added, as the PCEs only estimate 10-year risk (see the "Value" section for more information).
- If estimating over 30 years when age > 59, a warning will accompany the results regarding the reliability of the estimation (see the "Value" section for more information).

`chol_unit` Character (optional behavior variable): The unit of measurement for `total_c` and `hdl_c`, either `"mg/dL"` (the default) or `"mmol/L"` (`"mg"` and `"mmol"` are accepted abbreviations).

`optional_strict`

Logical (optional behavior variable): Whether to enforce strictness on optional predictor variables, either `TRUE` or `FALSE` (the default). The argument itself is strict, so 1 or 0 are *not* accepted (in contrast with predictor variables expecting logical input), and moreover, anything other than `TRUE` will be treated as `FALSE`. If `FALSE`, the function will discard invalid optional predictor variables but still allow the model to run. If `TRUE`, optional predictor variables entered (if any) must be valid for the function to return risk estimates. See the "Value" section for more information.

`quiet` Logical (optional behavior variable): Whether to suppress messages and warnings in the console, either `TRUE` or `FALSE`; this argument is strict, so 1 or 0 are *not* accepted (in contrast with predictor variables expecting logical input), and moreover, anything other than `TRUE` will be treated as `FALSE`. The default is `FALSE` when `use_dat` is *not* a data frame and `TRUE` when `use_dat` *is* a data frame. Having `quiet = FALSE` when `use_dat` is a data frame could result in a fairly noisy console, and the information contained in console-printed messages and warnings regarding model selection and input problems will already be in the return data frame via columns `model` and `input_problems`. However, if `use_dat` receives something other than a data frame or a data frame with zero rows, it will always warn the user, independent of the `quiet` argument.

`collapse` Logical (optional behavior variable): Whether to collapse the output into a single data frame if applicable, either `TRUE` or `FALSE`; this argument is strict, so 1 or 0 are *not* accepted (in contrast with predictor variables expecting logical input), and moreover, anything other than `TRUE` will be treated as `FALSE`. The default is `FALSE` when `use_dat` is *not* a data frame (this ensures backward compatibility) and `TRUE` when `use_dat` *is* a data frame. More precisely, however, although I have specified the default as `is.data.frame(use_dat)` for clarity in behavior, this argument is actually just ignored when a user passes a data frame to `use_dat`. See the description of the `use_dat` argument and the "Value" section for more information.

`use_dat` data frame via base R's data.frame or data frame extension via tibble or data.table (optional behavior variable): Whether to use a data frame provided by the user, either `NULL` (the default) or a data frame. More precisely, anything other than passing a data frame to `use_dat` has no impact. Passing a data frame to `use_dat` modifies the behavior of the function in the following manner:

- The function expects each row in the data frame to represent a candidate for risk estimation.
- Predictor variables must be present in the data frame passed to `use_dat`. Optional predictor variables remain optional, though (for example, there is no requirement for HbA1c data to be in the data frame *per se*, but if the user wishes to use HbA1c data as part of predicting risk with the PREVENT equations, those data must be represented in a column in the data frame). Any given argument for a predictor variable may be omitted, in which case the function expects the data frame to have a column with the name of the omitted argument. For example, if `age` is omitted from the function call, the function expects a column named `age` in the data frame passed to `use_dat`; it (of course) furthermore expects column `age` to contain data adhering to specifications set forth for the `age` argument (for example, an age of 20 is still considered unacceptable). Alternatively, the user may pass a column name for any predictor variable. Continuing with the previous example, if the column containing age data were instead named `years_old`, the user could pass either `age = years_old` or `age = "years_old"` to the function call, and values for `age` would be extracted from the `years_old` column.
- Optional behavior variables may *either* be in the data frame passed to `use_dat` in a column with the same name as the argument *or* passed to the function call as usual. If an optional behavior variable is omitted from the call when a user passes a data frame to `use_dat`, the function will first look for a column with the name of the optional behavior variable in the data frame; if it does not find such a column, it will use the default behavior for the optional behavior variable. If the user includes an argument for an optional behavior variable in the call, the function will always use this, irrespective of any column in the data frame that might share the same name. Additionally, the following arguments are not passable via the data frame: `collapse` (ignored when `use_dat` is a data frame), `use_dat` (this would be self-referential), `add_to_dat` (again, essentially self-referential), and `progress` (this applies to the entire call when `use_dat` is a data frame). As an example, suppose a user wishes to specify the model(s) to use. The user may either include a column in the data frame named `model` and omit the `model`

argument from the call or pass the desired behavior to the `model` argument. If a column named `model` exists in the data frame passed to `use_dat` *and* the user passes something to the `model` argument, the function will use the argument. Including an optional behavior variable in the data frame activates row-by-row behavior alteration of the function. If the user wants to alter the default behavior of the function, but wishes to do so in the same manner across all rows, it may be easier to forego having optional behavior variables in the data frame and instead pass the optional behavior variables via function arguments (though this is not strictly required; one can achieve the same behavior by having the same value repeated across all rows of the column for the optional behavior variable).

- The `collapse` argument is ignored (results will always be returned as a data frame when `use_dat` is a data frame).

See `vignette("using-data-frame")` for further discussion and examples.

`add_to_dat` Logical (optional behavior variable): Whether to add the output to the data frame passed to `use_dat`, either `TRUE` (the default) or `FALSE`. This argument is only considered when `use_dat` is a data frame. This argument is strict, so 1 or 0 are *not* accepted (in contrast with predictor variables expecting logical input), and moreover, anything other than `TRUE` will be treated as `FALSE`. See the "Value" section for more information.

See `vignette("using-data-frame")` for further discussion and examples.

`progress` Logical (optional behavior variable): Whether to display a progress bar during computation, either `TRUE` or `FALSE`. This argument is only considered when `use_dat` is a data frame, when it defaults to `TRUE`. This argument is strict, so 1 or 0 are *not* accepted (in contrast with predictor variables expecting logical input), and moreover, anything other than `TRUE` will be treated as `FALSE`. This argument is independent of the `quiet` argument. It requires the `utils` package, which is part of the R distribution (i.e., outside of atypical scenarios, you should not need to install the `utils` package yourself).

Details

Why is the upper limit of the SBP range 180 mmHg?:

Some may notice the upper limit is set to 180 mmHg here, whereas the PREVENT equations technically permit up to 200 mmHg. The Pooled Cohort Equations (PCEs) do this as well. I have restricted to 180 mmHg, as SBP beyond 180 mmHg constitutes hypertensive urgency (per AHA's own definitions), and irrespective of the debate surrounding labels like hypertensive urgency and emergency, it would seem clinically unreasonable to engage with the PREVENT equations when someone has more pressing matters to address (better blood pressure control *per se*).

Why is the lower limit of the HbA1c 4.5%?:

Some may notice the lower limit is set to 4.5% here, whereas the PREVENT equations technically permit down to 3%. I have restricted to 4.5%, as HbA1c of 3% is neither realistic nor safe for a person. For example, using the HbA1c to estimated average glucose (eAG) converter from the American Diabetes Association (https://professional.diabetes.org/glucose_calc), a HbA1c of 3% corresponds to an eAG of 39 mg/dL (2.2 mmol/L).

Entering eGFR and BMI as a call rather than a numeric value:

The `eGFR` and `bmi` arguments can be entered as numeric values or as calls to `calc_egfr()` and `calc_bmi()`, respectively. They both have synonyms as well:

- Synonyms for `calc_egfr()` are `calculate_egfr()`, `calc_ckd_epi()`, and `calculate_ckd_epi()`, with the latter two synonyms reflecting the calculation is from the CKD-EPI equations (the reparameterized version without race, which is also what the PREVENT equations use).
- The synonym for `calc_bmi()` is `calculate_bmi()`.

These convenience functions add value where a person might have the necessary components to calculate the respective parameter but do not have handy the parameter itself.

The syntax for these convenience functions is as follows:

- `calc_egfr(cr, units = "mg/dL", age, sex, quiet = FALSE)`
  - `cr` is the creatinine in whatever units are specified by `units`.
  - `units` is the unit of measurement for `cr`, either `"mg/dL"` or `"umol/L"`, with `"mg"` and `"umol"` being accepted abbreviations.
  - `age` is the age of the person, but there is no need to enter this, as the function will extract this from the `age` argument of `estimate_risk()`; in fact, any argument entered here will be ignored in favor of the `age` argument of `estimate_risk()`.
  - `sex` is the sex of the person, but there is no need to enter this, as the function will extract this from the `sex` argument of `estimate_risk()`; in fact, any argument entered here will be ignored in favor of the `sex` argument of `estimate_risk()`.
  - `quiet` is a logical indicating whether to suppress the warning about use outside of `estimate_risk()`.
  - An example call would be `calc_egfr(1.2)` (because `units` defaults to `"mg/dL"`) or `calc_egfr(88, "umol")`.
- `calc_bmi(weight, height, units = "nonmetric", quiet = FALSE)`
  - `weight` is the weight in pounds if `units = "nonmetric"` or kilograms if `units = "metric"`.
  - `height` is the height in inches if `units = "nonmetric"` or centimeters if `units = "metric"`.
  - `units` is the unit of measurement for `weight` and `height`, either `"nonmetric"` or `"metric"`.
  - `quiet` is a logical indicating whether to suppress the warning about use outside of `estimate_risk()`.
  - An example call would be `calc_bmi(150, 70)` (because `units` defaults to `"nonmetric"`) or `calc_bmi(68, 178, "metric")`.

What is the Social Deprivation Index (SDI)?:

Read more from the Robert Graham Center's page on the SDI (https://www.graham-center.org/maps-data-tools/social-deprivation-index.html)

Model selection when `model = NULL`:

If `model = NULL`, the model will be determined by the following algorithm:

- If no optional predictor variables (HbA1c, UACR, zip code) are entered, or only invalid optional variables are entered and `optional_strict = FALSE`: The base model
- If one of the optional predictor variables is entered, or two or more optional predictor variables are entered but only one is valid and `optional_strict = FALSE`: The base model adding that variable (e.g., if HbA1c is entered and no other optional predictor variables are entered, the base model adding HbA1c; if HbA1c and UACR are entered, but HbA1c is invalid and `optional_strict = FALSE`, the base model adding UACR)

- If two or more of the optional predictor variables are entered, or all three optional variables are entered but one is invalid and `optional_strict = FALSE`: The full model (the PREVENT equations include a term for optional predictor variables being missing, so if one of the optional predictor variables is missing in this scenario, it is treated as such within the full model)

What if SDI is not available for a zip code?:

Some zip codes do not have SDI data available, and the PREVENT equations include a term for SDI being missing. As such, if a user enters a valid zip code but no SDI data are available, the user will be notified (unless `quiet = TRUE`), and the tool will then implement the missing term as part of predicting risk whenever the full model is used, but SDI will otherwise be removed from prediction. Specifically, the following models will predict risk in the situation where the user enters a valid zip code, but no SDI data are available:

- If the user does not enter a valid HbA1c or UACR: The base model.
- If the user enters valid HbA1c and UACR: The full model (treating SDI as missing).
- If the user enters a valid HbA1c: The base model adding HbA1c.
- If the user enters a valid UACR: The base model adding UACR.

Race and/or ethnicity in predictive models:

The use of race and/or ethnicity in predictive (also called prognostic) models is, in a word, problematic. It is problematic for a few reasons, and fortunately, this has received much-needed attention in recent years. The PCEs require this input as specified in the "Arguments" section of the documentation. If you would like to read a bit more about this issue, see here.

Why is the risk estimation higher with the PCEs?:

The PCEs are known to overestimate risk. Indeed, this was a key motivation for Yadlowsky and colleagues to develop the revised PCEs, and was also a key motivation for development of the PREVENT equations.

Why don't you export the PCEs or the convenience functions for BMI and eGFR?:

These are not exported for two main reasons:

- With specific regard to the PCEs, they are not the focal point of this package, but they are of potential comparative interest.
- With regard to all these functions, I (of course) tested them for accuracy and intended behavior, but they are implemented primarily for internal package use or as part of estimating risk with `estimate_risk()` or `est_risk()`. For example, although they implement at least basic checks of input, some of the input checking and handling is delegated to other processes that are invoked when using these functions in the aforementioned ways. To give more concrete examples, if invoking these functions outside the context of `estimate_risk()` or `est_risk()`, although implementation of the PCEs checks input validity, it just returns `NA` with no messaging if it finds a problem. The functions for BMI and eGFR also implement checks for input validity (such as numeric inputs needing to be a number greater than 0), but they do not reject extreme numeric values (aside from the `age` input for eGFR, which implements some further restriction on age). Again, however, the calculations have certainly been tested for accuracy, so for users who are confident (1) they understand the cautions described here and (2) in the fidelity of their input for the functions, they can use them judiciously outside of `estimate_risk()` or `est_risk()` (via `preventr:::<function>`).

Value

Basic information about the return:

`estimate_risk()` will always return either (1) a list of length 2, with each list element having a single data frame or (2) a single data frame. All references herein to a data frame being returned are for a data frame as a tibble (see the tibble package for more detail) unless `use_dat` receives a data frame, in which case the return data frame will be of the same type passed to `use_dat` to ensure type-stability.

Whether the return is a list of data frames or a single data frame is determined by:

- whether the risk estimation is occurring over a single time horizon
- the value of the `collapse` argument
- whether the user has passed a data frame to the `use_dat` argument.

When all of the following conditions are met, the function will return a list of length 2, with each item in the list being a single data frame containing the 10-year and 30-year estimates, in that order:

- the user did *not* pass a data frame to `use_dat`
- `collapse = FALSE`
- either (1) `time = "both"` or (2) `time = "30yr"` *and* the user requests estimation with the PCEs via the `model` argument (thus adding a 10-year time horizon, as the PCEs only estimate risk at 10 years).

In all other scenarios, the function will return a single data frame. Note this includes scenarios where `collapse` will have no impact, namely when:

- the user passes a data frame to `use_dat` (passing a data frame to `use_dat` will always result in a data frame being returned to the user)
- the estimation occurs over one time horizon, namely if (1) `time = "30yr"` and the user does not request estimation with the PCEs or (2) `time = "10yr"`.

The data frame will have the following columns:

- `total_cvd`: The estimated risk of a total CVD event (column type: double)
- `ascvd`: The estimated risk of an ASCVD event (column type: double)
- `heart_failure`: The estimated risk of a HF event (column type: double)
- `chd`: The estimated risk of a CHD event (column type: double)
- `stroke`: The estimated risk of a stroke event (column type: double)
- `model`: The PREVENT or PCE model used (column type: character)
- `over_years`: The time horizon for the risk estimate (column type: integer)
- `input_problems`: Semicolon-separated vector of length one delineating any input problems (column type: character)

In addition, when `use_dat` is a data frame, the return data frame will also have the following composition:

- A column named `preventr_id` (column type: integer) that acts as a unique identifier for each row in the data frame passed to `use_dat`. This column will always be the first column in the returned data frame. The values of `preventr_id` are simply the row numbers of the data frame passed to `use_dat`. So, for example, if a row has `preventr_id` equal to 5, this means it is based on the input present in row 5 of the data frame passed to `use_dat`.

- If `add_to_dat` = `TRUE`, the returned data frame will include the columns in `use_dat`. So, the composition of the return data frame will be: `preventr_id` column + columns from `use_dat` + risk estimation columns. In addition, for a given row in the `use_dat` data frame with `preventr_id` *x* (hereafter, "row *x*"), if *n* represents the number of models requested for row *x*, then row *x* will be replicated *n* times in the output to accommodate reporting the different model outputs for that row. Note also *n* is determined by what the function receives for both the `model` and `time` arguments (because, for example, if `model` = `"base"` and `time` = `"both"`, this is a request for 2 models). For those familiar with joins, the expansion described here is simply the result of a left join of the data frame passed to `use_dat` with the data frame returned by `estimate_risk()` (using `preventr_id` as the key). For those not familiar with joins, if the above does not seem clear, the vignette about using data frames (`vignette("using-data-frame")`) should help.
- If `add_to_dat` = `FALSE`, the returned data frame will *not* include the columns in `use_dat`, so the composition of the return data frame will be: `preventr_id` column + risk estimation columns. The replication behavior described for when `add_to_dat` = `TRUE` will still occur. For this reason, the `preventr_id` column is perhaps especially important when `add_to_dat` = `FALSE`, as it provides a mechanism to associate the results with the original data frame.
- If the user passes a data frame with a column named `model` (see the argument specifications for `use_dat` for further detail), the function will rename this column to `model_input` in the return data frame to prevent name conflicts, because the return data frame will also have the column `model` based on the risk estimation output.

When valid input parameters exist for all required predictor variables:

The risk estimate columns are all of type double, and they are presented as a proportion rounded to 3 decimal places. Halves are rounded up to align with what many people likely expect, but this is in contrast to base R's default rounding behavior (it is a perfectly reasonable default, but perhaps somewhat unexpected for people who are not familiar with different standards/conventions for rounding; see `round()` for further detail).

The `model` column will be of type character, taking one of the following values: `"base"`, `"hba1c"`, `"uacr"`, `"sdi"`, or `"full"`. If opting in for comparison to the PCEs, `model` for those estimates will be one of `"pce_orig"` or `"pce_rev"`.

The `over_years` column will be of type integer, either 10 or 30.

If `optional_strict` = `TRUE`, the above will only hold if the optional predictor variables that are entered (if any) are valid; if any optional predictor variables are entered but are invalid, the function will behave in the same manner as when invalid input parameters exist for one or more required variables.

When invalid input parameters exist for one or more required predictor variable(s):

The function will issue a warning about the problematic variables, unless `quiet` = `FALSE`. A data frame will be returned with the following characteristics:

- All risk estimates will be set to `NA_real_`
- The `model` column will state "none"
- The `over_years` column will be set to `NA_integer_`
- The `input_problems` column will contain a character vector of length 1 delineating the problematic variable(s); if multiple problematic variables exist, they will be separated by semicolons

When invalid input parameters exist for one or more optional predictor variable(s):

When `optional_strict = TRUE`*:*
The function will behave similarly to when invalid input parameters exist for one or more required variables, with the `input_problems` column delineating the problematic variables

When `optional_strict = FALSE`*:*
The function will issue a warning about the problematic variables, unless `quiet = FALSE`. The problematic optional variables will then be functionally discarded and the PREVENT equations still run, in accordance with the specifications detailed in the "Details" section regarding model selection. A data frame will be returned with the following characteristics:

- All estimates will be returned as specified in the valid input parameters section, as will the `model` and `over_years` columns
- The `input_problems` column will contain a character vector of length 1 delineating the problematic variables (because optional predictor variables are allowed to be empty, any input that is functionally empty or missing (such as `NULL`, `numeric(0)`, `NA`, etc.) will not be considered problematic and thus not populate in the `input_problems` column)

When estimating 30-year risk and age > 59:
The function advises 30-year risk prediction for people > 59 years is questionable via two warnings:

- in the console (that can be suppressed by setting `quiet = TRUE`)
- in the column `input_problems` of the return tibble (`quiet` has no impact here)

The special case of the `zip` argument:
The above rule for optional predictor variables applies to the `zip` argument as well, but with the additional reminder that there are valid zip codes that do not have an SDI score. This is importantly different from an invalid input for zip. See the "Details" section for more information about how this is handled, but users should *not* expect anything to populate in the `input_problems` column if the zip is valid, regardless of whether that zip has an SDI score. As will be clear from the "Details" section, users will be able to determine when a zip code does not have an SDI score based on the model that was used.

What about the PCEs?:
Within the broader context of the function itself, the PCEs are treated as optional. Thus, as long as there is valid input for the PREVENT equations, the function will run, returning risk estimates from the PREVENT equations. Note, however, that valid input for the PREVENT equations requires valid input for the `model` argument. Thus, if the `model` argument is invalid or malformed (i.e., not adherent to the specifications delineated for that argument), the function will behave as described for the circumstance when invalid input exists for one or more required predictor variables.

If a list containing elements `other_models` and `race_eth` is passed to argument `model`, then within the sub-context of running additional models for comparison, the elements `other_models` and `race_eth` are required. Thus, if either `other_models` or `race_eth` is invalid, the returned row(s) within the data frame will function comparably to what is described for the circumstance when invalid input exists for one or more required predictor variables for the PREVENT equations. For example, suppose someone enters valid input for the PREVENT equations and passes the following argument to `model`: `list(other_models = "pce_both", race_eth = NA)`. The function would then run, returning risk estimates for the PREVENT equations, but the user would be notified of the invalid input for argument `race_eth` within the argument `model` in the console (unless quiet = TRUE); furthermore, the return data frame for the 10-year time horizon would contain two rows dedicated to the PCEs (given other_models = "pce_both", a valid argument), but each row would behave in the manner described for the PREVENT equations when one or more required predictor variable is invalid. That is, each row dedicated to the PCEs would consist of NAs (of the appropriate type) for each column, aside from the column model, which would say "none", and the column input_problems, which would specify there was erroneous input for the argument race_eth. Likewise, if other_models were instead "pce_orig", "pce_rev", or an invalid input, there would only be one row dedicated to the PCEs, because in the first two cases, the user entered a valid argument specifying interest in only one of the two options for the PCEs, and in the third case, the user entered invalid input for the options for the PCEs (thus becoming functionally similar to a situation if someone gave invalid input for the model argument).

Lastly, note the risk estimation columns total_cvd, heart_failure, chd, and stroke will always be NA_real_, because the PCEs only estimate the risk of ASCVD.

Combining output into a single data frame:

Depending on the arguments to the function, the output may be a list of data frames, one for each time horizon, (see the subsection "Basic information about the return" within the "Value" section). The argument collapse supports collapsing these into a single data frame, but it is also easy to do outside of this package, e.g.:

```
res_dplyr <- dplyr::bind_rows(res)       # Combine in dplyr
res_dt <- data.table::rbindlist(res)     # Combine in data.table
res_base_r <- do.call(rbind, res)        # Combine in base R

# These all yield the same tabular output, but the attributes vary
# (e.g., the classes will obviously differ)

all.equal(res_dplyr, res_dt, check.attributes = FALSE)       # TRUE
all.equal(res_base_r, res_dplyr, check.attributes = FALSE)   # TRUE
```

Type-stability of return when passing a data frame to use_dat:

Importantly, the function maintains type-stability of the data frame it receives via the use_dat argument, meaning passing a data.frame will yield a data.frame, passing a tibble will yield a tibble, and passing a data.table will yield a data.table. See vignette("using-data-frame") for more information.

Examples

```
# Example with all required predictor variables (example from Table S25
# in the supplemental PDF appendix of the PREVENT equations article)
#
# Optional predictor variables are all omitted (and thus take their default).
# `model` is also omitted (and thus takes its default, with the function
# selecting # the model based on the algorithm specified in the "Details"
# section).
# `time` is also omitted (and thus takes its default, with the function
# returning estimates for both 10- and 30-year risk as specified in the
# "Value" section).
#
```

```
# Expect the base model to run given absence of optional predictor variables
res <- estimate_risk(
  age = 50,
  sex = "female",    # or "f"
  sbp = 160,
  bp_tx = TRUE,      # or 1
  total_c = 200,     # default unit is "mg/dL"
  hdl_c = 45,        # default unit is "mg/dL"
  statin = FALSE,    # or 0
  dm = TRUE,         # or 1
  smoking = FALSE,   # or 0
  egfr = 90,
  bmi = 35
)

# Based on Table S25, expect the 10-year risk for `total_cvd` to be 0.147,
# and based on the supplemental Excel file, also expect:
# 10-year risks: `ascvd`, 0.092; `heart_failure`, 0.081;
# `chd`, 0.044; `stroke`, 0.054
# 30-year risks: `total_cvd`, 0.53; `ascvd`, 0.354; `heart_failure`, 0.39;
# `chd`, 0.198; `stroke`, 0.221
res

# Example with HbA1c
# (also changing required predictor variables & limiting to 10-year results)
estimate_risk(
  age = 66,
  sex = "male",      # or "m"
  sbp = 148,
  bp_tx = FALSE,
  total_c = 188,
  hdl_c = 52,
  statin = TRUE,
  dm = TRUE,
  smoking = TRUE,
  egfr = 67,
  bmi = 30,
  hba1c = 7.5,
  time = "10yr"      # only 10-year results will show
)

# Example with UACR (limited to 30-year results)
estimate_risk(
  age = 66,
  sex = "female",
  sbp = 148,
  bp_tx = FALSE,
  total_c = 188,
  hdl_c = 52,
  statin = TRUE,
  dm = TRUE,
  smoking = TRUE,
  egfr = 67,
  bmi = 30,
  uacr = 750,
  time = "30yr"      # only 30-year results will show
)

# The remaining examples will all be limited to 10-year results unless
# otherwise specified

# Example with SDI with valid zip code with SDI data available
estimate_risk(
  age = 66,
  sex = "female",
  sbp = 148,
  bp_tx = FALSE,
  total_c = 188,
  hdl_c = 52,
  statin = TRUE,
  dm = TRUE,
  smoking = TRUE,
  egfr = 67,
  bmi = 30,
  zip = "59043",   # Lame Deer, MT (selected randomly)
  time = 10        # Note numeric 10 (not "10yr"),
                   # just to show the option of entering this way
)

# Example with SDI with valid zip code without SDI data available
# (base model will be used)
estimate_risk(
  age = 66,
  sex = "male",
  sbp = 148,
  bp_tx = FALSE,
  total_c = 188,
  hdl_c = 52,
  statin = TRUE,
  dm = TRUE,
  smoking = TRUE,
  egfr = 67,
  bmi = 30,
  zip = "00738",   # Fajardo, PR
  time = 10
)

# Example with full model (even though zip does not have available SDI, full
# model used given availability of HbA1c and UACR; because zip is valid,
# column `input_problems` will be NA)
estimate_risk(
  age = 66,
  sex = "female",
  sbp = 148,
  bp_tx = FALSE,
  total_c = 188,
  hdl_c = 52,
  statin = TRUE,
  dm = TRUE,
  smoking = TRUE,
  egfr = 67,
  bmi = 30,
  hba1c = 9,
  uacr = 75,
  zip = "00738",
  time = "10yr"
)

# Example with full model (zip has SDI data available, UACR is valid, but
# HbA1c is not; column `input_problems` will specify problem with `hba1c`,
# but full model will still run given availability of the other two optional
# predictor variables)
estimate_risk(
  age = 66,
  sex = "male",
  sbp = 148,
  bp_tx = FALSE,
  total_c = 188,
  hdl_c = 52,
  statin = TRUE,
  dm = TRUE,
  smoking = TRUE,
  egfr = 67,
  bmi = 30,
  hba1c = 20,
  uacr = 75,
  zip = "59043",
  time = "10yr"
)

# Example of using the convenience functions `calc_bmi()` and `calc_egfr()`
res_convenience_fxs <- estimate_risk(
  age = 50,
  sex = "female",
  sbp = 130,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = calc_egfr(1), # units unspecified, so treated as 1 mg/dL; eGFR = 69
  bmi = calc_bmi(70, 150, "metric"), # weight in kg, height in cm; BMI = 31.1
  time = "10yr",
  quiet = TRUE
)

res_direct_entry <- estimate_risk(
  age = 50,
  sex = "female",
  sbp = 130,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 69,
  bmi = 31.1,
  time = "10yr",
  quiet = TRUE
)

identical(res_convenience_fxs, res_direct_entry)

# Example of using `model` argument to compare results from PREVENT equations
# to both versions of the PCEs
estimate_risk(
  age = 50,
  sex = "female",
  sbp = 130,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 90,
  bmi = 35,
  hba1c = 8,
  uacr = 30,
  time = "10yr",
  model = list(other_models = "pce_both", race_eth = "Black")
  # Note omission of element `main_model` within the list is okay, and the
  # element will then be treated as NULL (and thus model selection here
  # will be "full" given availability of valid HbA1c and UACR)
)

# Essentially a repeat of example immediately above, but now will specify
# `main_model` as "hba1c" and limit `other_models` to the revised PCEs
estimate_risk(
  age = 50,
  sex = "female",
  sbp = 130,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 90,
  bmi = 35,
  hba1c = 8,
  uacr = 30,
  time = "10yr",
  model = list(main_model = "hba1c", other_models = "pce_rev", race_eth = "Black")
)

# Because the PCEs only give 10-year estimates, if a user specifies an
# interest in a 30-year time horizon but also expresses interest in
# comparison with the the PCEs, a 10-year time horizon must be added for the
# PCEs, but this will not automatically result in estimation of 10-year risk
# for the PREVENT equations.
estimate_risk(
  age = 50,
  sex = "female",
  sbp = 130,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 90,
  bmi = 35,
  time = "30yr",
  model = list(other_models = "pce_both", race_eth = "Black")
)

# Repeat of above, but setting `collapse = TRUE`
res_collapsed <- estimate_risk(
  age = 50,
  sex = "female",
  sbp = 130,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 90,
  bmi = 35,
  time = "30yr",
  model = list(other_models = "pce_both", race_eth = "Black"),
  collapse = TRUE
)

res_collapsed

# Can also accomplish this after the fact, as documented in "Combining output
# into a single data frame" within the "Value" section
res_uncollapsed <- estimate_risk(
  age = 50,
  sex = "female",
  sbp = 130,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 90,
  bmi = 35,
  time = "30yr",
  model = list(other_models = "pce_both", race_eth = "Black")
)

all.equal(
  do.call(rbind, res_uncollapsed),
  res_collapsed,
  check.attributes = FALSE
)
# Can also accomplish with `dplyr` and `data.table`, as detailed in the
# "Combining output into a single data frame" subsection of the "Value"
# section


# Passing a data frame to argument `use_dat`
if(interactive()) {
  vignette("using-data-frame")
}

# Expect table of NAs due to invalid input for `age` and `sbp`, and column
# `input_problems` to contain explanations about problems with `age` and `sbp`
res <- estimate_risk(
  age = 8675309,
  sex = "female",
  sbp = 112358,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 90,
  bmi = 35,
  time = "10yr"
)

res

# Quiet version of the above example
res <- estimate_risk(
  age = 8675309,
  sex = "female",
  sbp = 112358,
  bp_tx = TRUE,
  total_c = 200,
  hdl_c = 45,
  statin = FALSE,
  dm = TRUE,
  smoking = FALSE,
  egfr = 90,
  bmi = 35,
  time = "10yr",
  quiet = TRUE       # Suppresses messages, but not column `input_problems`
)

res

# If only invalid input is for PCEs, PREVENT equations will still run
for(time in c("10yr", "30yr", "both")) {
  cat(paste0("\n", "----- `time = \"", time, "\"` -----", "\n"))
  print(
    estimate_risk(
      age = 38,                  # This age is okay for the PREVENT
      sex = "female",            # equations, but not for the PCEs
      sbp = 144,
      bp_tx = TRUE,
      total_c = 200,
      hdl_c = 45,
      statin = FALSE,
      dm = TRUE,
      smoking = FALSE,
      egfr = 90,
      bmi = 35,
      time = time,
      model = list(
        other_models = "pce_both",
        race_eth = NA              # Invalid `race_eth` for PCEs
      ),
      quiet = TRUE
    )
  )
}

# Note `input_problems` column is semicolon-separated, but it is easy to
# print as separate lines with `gsub()` and `cat()`, e.g.:
cat(gsub("; ", "\n", res$input_problems))

res$input_problems |> gsub(pattern = "; ", replacement = "\n", x = _) |> cat()
# ... and could, of course, also use the `magrittr` pipe `%>%` if desired
```

Index