S3 Classes for CCMPP Input Data Structures

# Introduction

This document presents a general framework for a set of S3 classes that will be used for objects that are inputs and outputs to CCMPP functions.

## Context

CCMPP requires as input a collection of data by age, sex, and period:

1. Population counts at the baseline year (or “jump-off” population).
2. Period(?) fertility rates in each period of the projection.
3. Period(?) sex ratios at birth in each period of the projection.
4. Cohort(?) survival ratios in each period of the projection.
5. Period(?) net migration in each period of the projection, preferably as counts.

There may also be

1. Population counts at some years in the interval of projection.

CCMPP produces as output

* Projected population counts at each period, according to the step size.

Other quantities of interest that can be derived from the inputs and outputs include (all possibly age- sex- period-specific):

* Birth counts in each period of the projection.
* (Period, Cohort ?) Death counts in each period of the projection.
* Period(?) mortality rates in each period of the projection.
* Other?

Other systems will be interacting with the *R* code performing CCMPP operations, such as systems that prepare the input and systems that read and process the output.

## Approach

*R*’s S3 object oriented programming system is used to formally encode input and output object structures as *classes*. Functions are defined that validate an object as a member of one (or more) of the classes by checking its structure matches the class definitions. For example, the class for fertility rates specifies that all values be non-negative. Objects passed into the CCMPP functions will be validated once at the top of the function to ensure that the input object matches the specification the function expects. This approach reduces the chance of failures later on in the function and of undetected errors in computations.

To achieve this:

1. A complete description of the object class is encoded in a series of *R* functions that *create* and *validate* objects as members of the class.
2. Special functions that take objects of the class as their primary input are defined as *generics* with *methods* for the class. This provides a *minimal* amount of safety but makes a clear statement to the user that the object expected should conform to expectations.
3. Functions *validate* the input object before doing any calculations. This should prevent any invalid inputs making their way into the function calculations potentially causing undetected errors. *Subsequent* calculation errors due to the function’s own operations are the responsibility of the function, not the class.

For further reading on *R*’s object-oriented programming system, including S3, see (Wickham ([2019](#ref-wickham_advanced_2019)), Part III).

## Purpose and Limitations

The purpose of the S3 object classes is to specify the structure a set of inputs to CCMPP functions requires to be valid. They are not intended to be classes for a data bases or any other use-case.

## Justification

Having specifications for valid data structures makes development and maintenance of R code easier, and makes the code less susceptible to errors and failures. It makes it easier for users to generate inputs since they know exactly what is required, and for them to work with the outputs since they know exactly what the outputs will look like. Complete specification of input and output objects are particularly important for interoperability between systems in a chained operation, where one system may generate inputs to the CCMPP functions and another processes the outputs.

S3 programming promotes transparency and rigor as the specifications of the objects are defined in *R* code. Each object of a given class will have attached to it meta data describing its characteristics with respect to the class. These meta data are stored in the objects’ *attributes*. Constructor and validator functions for, respectively, creating objects and confirming they have valid structure, encode a series of rules which define the structure of the class.

Creating object classes abstracts the object type from the details. A user-interface will be developed to perform key operations on objects of a class independently of the class’s implementation. It is not relevant to the user, nor should it affect the visible behaviour of the function, whether the object uses data frames, matrices, or lists to store its data. These underlying details can be changed without affecting the user interface and, thereby, breaking existing user code.

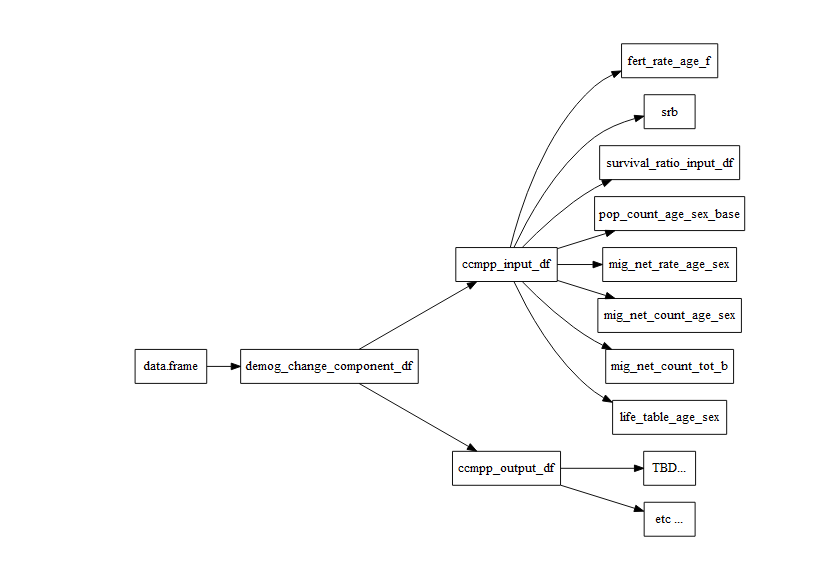
Functions that operate on objects with classes can be written such that different *methods* are available for different classes of objects. This means that the same function call when applied to objects of different classes can produce different outputs. This is another useful form of abstraction; the user interface is separated from the details of the implementation. In *R* it also allows new behaviours for new classes to be quickly added to existing functions.

This system provides programming efficiencies, and can be extended over time in a modular fashion, due to *inheritance*. Where appropriate, class definitions will be nested such that objects inherit the attributes of their ‘parents’. This means that specifications of ‘child’ classes need only include the additional attributes not held by their parents. Moreover, functions with a method defined for a parent will automatically apply that method to objects of a child class, unless a specific method for the child class is defined.

# S3 Classes for CCMPP Input and Output

The classes defined and their inheritance relations are shown in Figure @ref(fig:S3-class-hierarchy-diagram). The base class is demog\_change\_component\_df (“df” for “data frame”). All other classes are *descendants*. The descendants’ specifications include all those of demog\_change\_component\_df plus additions.

An additional class, ccmpp\_input\_list, is not shown in Figure @ref(fig:S3-class-hierarchy-diagram). This is a class for a list of all the required data frames for a complete CCMPP run. It is described in Section @ref(sec:ccmpp-inut-list-defn).



Hierarchy of S3 classes for CCMPP input and output objects

## Base Class: demog\_change\_component\_df

The class was designed with reference to data(wpp\_input\_example) which is a list data frames of components of demographic change down by time, age, and sex. The most general of these data frames is wpp\_input\_example$life\_table\_age\_sex which contains life table parameters by indicator, time, sex, and age group.

head(wpp\_input\_example$life\_table\_age\_sex)  
#> indicator time\_start time\_span sex age\_start age\_span value  
#> 1 lt\_nMx 1950 1 male 0 1 0.046796179  
#> 2 lt\_nMx 1950 1 male 1 1 0.002608884  
#> 3 lt\_nMx 1950 1 male 2 1 0.002154391  
#> 4 lt\_nMx 1950 1 male 3 1 0.001784905  
#> 5 lt\_nMx 1950 1 male 4 1 0.001489286  
#> 6 lt\_nMx 1950 1 male 5 1 0.001257002

The class demog\_change\_component\_df was designed to accommodate all of the data frames in wpp\_input\_example. It has two main components: **attributes** (meta-data) and a **data component**.

### Attributes

demog\_change\_component\_df inherits from *R*’s standard data.frame class (Figure @ref(fig:S3-class-hierarchy-diagram)). Therefore, it has all the attributes of data frames plus two new attributes, dimensions and value\_type.

“dimensions”

This attribute indicates which dimensions are present in the object. Each object must have at least one dimension of the possible four:

1. indicator
2. time
3. sex
4. age

“value\_type”

This attribute specifies the type of data in the value column (see below). There are seven value types:

1. count
2. rate
3. ratio
4. proportion
5. percentage
6. real
7. categorical

The data frames in wpp\_input\_example have various subsets of the dimensions. For example, wpp\_input\_example$srb has only the “time” dimension, the fertility rate component has only the “time” and “age” dimensions. Specific subclasses for these data frames are defined to formalize their characteristics. The “value\_type” “real” is a catch-all for numeric values that do not fall under any other type. It is also used when the value column contains a mixture of the other types, as is the case for wpp\_input\_example$life\_table\_age\_sex.

### Data Component

The data component of demog\_change\_component\_dfs is a standard *R* data frame. Therefore, demog\_change\_component\_df inherits from the class data.frame and any functions applied to demog\_change\_component\_dfs that do not have specifically defined methods (see below) will behave as if they were given a data frame.

The data frame must contain a column called value to hold the parameter values. Additional columns depend on the dimensions of the object, as detailed in Table @ref(tab:data-frame-columns-by-dimension).

(#tab:data-frame-columns-by-dimension) Required columns by dimension for demog\_change\_component\_df objects.

|  |  |  |
| --- | --- | --- |
| Dimension | Columns Required | Description |
| indicator | * indicator | Character values labeling the indicator (no restrictions) |
| time | * time\_start * time\_span | Numeric values indicating the start and span of the time period |
| sex | * sex | Character values in c("female", "male", "both") |
| age | * age\_start * age\_span | Numeric values indicating the start and span of the age period |

**NOTE** We could add age\_open\_ended to indicate which row is the open-ended age group.

### Constraints

The following constraints must be satisfied for an object to be a valid member of the class:

1. Exactly the columns required for the “dimensions” must all be present; none can be absent and no additional columns can be present.
2. The value column must conform to the “value\_type” attribute.
3. The data frame has exactly one finite and non-missing value per indicator age\_start sex time\_start combination .
4. The only values allowed in the sex column are “both”, “female”, and “male”.

## Class ccmpp\_input\_df

The class ccmpp\_input\_df is a subclass of demog\_change\_component\_df. It has all the features of its parent plus the following.

### Attributes

As well as the “dimensions” and “value\_type” attributes from its parent, the ccmpp\_input\_df class has attributes

“age\_span”

The common age\_span value. If “age” is a dimension it must be a finite, non-missing scalar. Otherwise it is equal to the zero-length double().

“time\_span”

The common time\_span value. If “time” is a dimension it must be a finite, non-missing scalar. Otherwise it is equal to the zero-length double().

The age\_spans and time\_spans must all be equal to a single common value for the CCMPP to work, therefore they are conceptually better represented as attributes.

### Data Component

No structural differences. Although age\_span and time\_span are attributes the columns are retained in the data frame.

### Constraints

The ccmpp\_input\_df has some additional constraints to ensure the objects are valid CCMPP inputs.

1. age\_span and time\_span are equal to each other and apply to all ages and times.
2. For every unique indicator time\_start sex combination there is an age\_start that equals 0; i.e., the age breakdown must start at age 0.
3. The rows must be sorted by indicator, time, sex, age, varying slowest to fastest in that order.

## Subclasses of ccmpp\_input\_df

Classes for the individual data frame components of the CCMPP lists are all subclasses of ccmpp\_input\_df. Apart from fert\_rate\_age\_f they have no additional attributes; none have additional data columns. All have pre-specified dimensions and value\_types as described in Table @ref(tab:ccmpp-input-subclass-restrictions).

The class fert\_rate\_age\_f gets the additional attribute:

“non\_zero\_fert\_ages”

A list of the values of age\_start marking the reproductive age groups. Functions can take advantage of this attribute to set fertility to zero in all other age groups regardless of the actual values in the value column.

(#tab:ccmpp-input-subclass-restrictions) Additional attributes and pre-specified dimensions and values types of subclasses of ccmpp\_input\_df.

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Additional Attributes | Dimensions | Value type |
| fert\_rate\_age\_f | “non\_zero\_fert\_ages” | c(“time”, “age”) | “rate” |
| srb |  | c(“time”) | “ratio” |
| survival\_ratio\_input\_df |  | c(“time”, “sex”, “age”) | “proportion” |
| mig\_net\_rate\_age\_sex |  | c(“time”, “sex”, “age”) | “rate” |
| mig\_net\_count\_age\_sex |  | c(“time”, “sex”, “age”) | “count” |
| mig\_net\_count\_tot\_b |  | c(“time”) | “count” |
| mig\_parameter |  | c(“time”) | “categorical” |
| pop\_count\_age\_sex\_base |  | c(“time”, “sex”, “age”) | “count” |
| life\_table\_age\_sex |  | c(“indicator”, “time”, “sex”, “age”) | “real” |

# Role in CCMPP Workflow

## ccmpp\_input\_list

ccmppWPP\_workflow\_one\_country\_variant is the workhorse function for performing the CCMPP on a single country. It takes a single argument, a list of data frame, each containing data for a component of demographic change (population at baseline, fertility, etc.). The class ccmpp\_input\_list is a list of objects with the classes listed in Table @ref(tab:ccmpp-input-subclass-restrictions). It specifies exactly the inputs needed for a complete run of the CCMPP. It acquires all the attributes of its constituents:

* “age\_span”
* “time\_span”
* “non\_zero\_fert\_ages”

## Validating Input

A single line is added to ccmppWPP\_workflow\_one\_country\_variant to enact the validation:

ccmppWPP\_workflow\_one\_country\_variant <- function(wpp\_input) {  
  
 wpp\_input <- as\_ccmpp\_input\_list(wpp\_input) #<<<<<<<< ADDED: VALIDATE INPUTS  
  
 # extract objects needed for ccmpp  
 ccmpp\_input <- data\_parse\_ccmpp\_input(indata = wpp\_input)  
  
 ... etc.

# Construction and Validation

Functions to construct objects of the various classes, and validate existing objects of unconfirmed class, are provided.

## Main Constructor Functions

A user-level constructor is provided for each class which takes as input the base data frame object plus arguments specifying the dimensions and value types. The names of these functions are the class names.

demog\_change\_example\_df <-  
 demog\_change\_component\_df(x = wpp\_input\_example$mig\_net\_count\_age\_sex,  
 dimensions = c("time", "sex", "age"),  
 value\_type = "count")

The class of the object is contained in the “class” attribute accessed with base *R* function class()

class(demog\_change\_example\_df)  
#> [1] "demog\_change\_component\_df" "data.frame"

Note that the data.frame is the second element of the class, indicating that objects of class demog\_change\_component\_df explicitly inherit from the base class data.frame.

Constructing objects of the other classes is done in the same way, e.g.:

ccmpp\_input\_example\_df <-  
 ccmpp\_input\_df(x = wpp\_input\_example$mig\_net\_count\_age\_sex,  
 dimensions = c("time", "sex", "age"),  
 value\_type = "count")  
  
fert\_rate\_input\_example\_df <-  
 fert\_rate\_age\_f(x = wpp\_input\_example$fert\_rate\_age\_f,  
 non\_zero\_fert\_ages = 15:45)  
#> 'non\_zero\_fert\_ages' set to '15, 16, 17, ... , 43, 44, 45

To make a ccmpp\_input\_list that binds together all the inputs needed for a projection, pass all the constituent parts as arguments to the constructor:

ccmpp\_input\_list\_example <-  
 ccmpp\_input\_list(pop\_count\_age\_sex\_base = wpp\_input\_example$pop\_count\_age\_sex\_base,  
 life\_table\_age\_sex = wpp\_input\_example$life\_table\_age\_sex,  
 fert\_rate\_age\_f = wpp\_input\_example$fert\_rate\_age\_f,  
 srb = wpp\_input\_example$srb,  
 mig\_net\_count\_age\_sex = wpp\_input\_example$mig\_net\_count\_age\_sex,  
 mig\_net\_rate\_age\_sex = wpp\_input\_example$mig\_net\_rate\_age\_sex,  
 mig\_net\_count\_tot\_b = wpp\_input\_example$mig\_net\_count\_tot\_b,  
 mig\_parameter = wpp\_input\_example$mig\_parameter)  
#> 'non\_zero\_fert\_ages' set to '12, 13, 14, ... , 51, 52, 53

### Construction Conveniences

The constructor functions check that the data frame being supplied is valid as an input for the class being created. The following operations are also done as conveniences. Note that these are hierarchical; everything that is listed under a function applies to all those beneath it.

demog\_change\_component\_df

* Superfluous columns will be discarded; only the columns needed for the specified “dimensions” and the value column are retained.
* If the dimensions argument is omitted the constructor functions will attempt to guess the “dimensions” attribute from the columns in the data frame.

## Dimensions can be guessed from the columns in the data frame  
demog\_change\_example\_df <-  
 demog\_change\_component\_df(x = wpp\_input\_example$mig\_net\_count\_age\_sex)  
#> Argument 'dimensions' is 'NULL'; setting 'dimensions' to 'time, sex, age' based on column names of 'x'.  
#> Argument 'value\_type' is 'NULL'; setting 'value\_type' to 'real'.

ccmpp\_input\_df

* Pre-sorting is not required; the data frame will be sorted appropriately.
* The “time\_span” and “age\_span” attributes will be set using the corresponding columns in the data frame, which will be checked for validity.

fert\_rate\_age\_f

* An attempt will be made to guess the attribute “non\_zero\_fert\_ages” if it is not supplied as an argument.

### User Responsibilities

The following are the responsibility of the user and will *not* be done automatically . Note that these are hierarchical; everything that is listed under a function applies to all those beneath it.

demog\_change\_component\_df

* There must be only one row per unique indicator time\_start sex combination. Use dplyr or similar before passing to demog\_change\_component\_df().
* There cannot be any missing values; CCMPP won’t work with them.
* The sex column (if present) can only have values “both”, “female”, or “male”. Variations such as “Both”, “MALE”, etc. are not valid.

ccmpp\_input\_df

* Ensuring that the age breakdown starts at age 0 for every unique indicator time\_start sex combination.

## Validation

The function validate\_ccmpp\_object() is available to validate any object as a member of the class. As per convention, if the object is valid it is returned; if not, an error is signalled:

x <- validate\_ccmpp\_object(x = demog\_change\_example\_df)  
class(x)  
#> [1] "demog\_change\_component\_df" "data.frame"  
   
## Not run:  
## y <- validate\_ccmpp\_object(x = wpp\_input\_example$mig\_net\_count\_age\_sex)  
## > Error in validate\_ccmpp\_object.default(x = wpp\_input\_example$mig\_net\_count\_age\_sex) :  
## 'x' is not an object with a valid CCMPP object class ...

validate\_ccmpp\_object() is an S3 generic function; it has methods for all the classes and subclasses associated with the demog\_change\_component\_df base class.

You can always check if an existing object inherits from a particular class using the appropriate is\_ function. These functions only check the “class” attribute; they do not check that the underlying data components are valid.

is\_demog\_change\_component\_df(demog\_change\_example\_df)  
#> [1] TRUE  
is\_demog\_change\_component\_df(ccmpp\_input\_example\_df) #inheritance  
#> [1] TRUE  
  
is\_ccmpp\_input\_df(demog\_change\_example\_df) #inheritance does not go backwards  
#> [1] FALSE  
is\_ccmpp\_input\_df(ccmpp\_input\_example\_df)   
#> [1] TRUE

## Object Visualization

### print

print and summary are two common functions for examining objects. Special methods have been written to make these useful for the demog\_change\_component\_df class. The print method takes inspiration from the method for tibbles. It prints a short summary of the object and then the first few rows of the data. By default, the first six rows are printed:

demog\_change\_example\_df  
#> # A 'demog\_change\_component\_df' with 14,140 rows.  
#> # dimensions = 'time, sex, age'.  
#> # value\_type = 'real'.  
#> # 'sex' has levels: female, male.  
#> time\_start sex age\_start time\_span age\_span value  
#> 1950 male 0 1 1 -1906.5913  
#> 1950 male 1 1 1 -772.8935  
#> 1950 male 2 1 1 52.2876  
#> 1950 male 3 1 1 765.6268  
#> 1950 male 4 1 1 1291.9252  
#> 1950 male 5 1 1 1651.7646  
#> # ... etc.

To show more rows, call print explicitly and use the argument n:

print(demog\_change\_example\_df, n = 10)  
#> # A 'demog\_change\_component\_df' with 14,140 rows.  
#> # dimensions = 'time, sex, age'.  
#> # value\_type = 'real'.  
#> # 'sex' has levels: female, male.  
#> time\_start sex age\_start time\_span age\_span value  
#> 1950 male 0 1 1 -1906.5913  
#> 1950 male 1 1 1 -772.8935  
#> 1950 male 2 1 1 52.2876  
#> 1950 male 3 1 1 765.6268  
#> 1950 male 4 1 1 1291.9252  
#> 1950 male 5 1 1 1651.7646  
#> 1950 male 6 1 1 1865.7268  
#> 1950 male 7 1 1 1954.3936  
#> 1950 male 8 1 1 1938.3468  
#> 1950 male 9 1 1 1838.1682  
#> # ... etc.

Objects of class ccmpp\_input\_df must have a common value for age\_span and time\_span. The print method emphasizes this by showing only the first row. In subsequent rows the entries are shown as “.”. The values are also printed in the summary above the table.

ccmpp\_input\_example\_df  
#> # A 'ccmpp\_input\_df' with 14,140 rows.  
#> # dimensions = 'time, sex, age'.  
#> # value\_type = 'count', age\_span = '1', time\_span = '1'.  
#> # 'sex' has levels: female, male.  
#> time\_start sex age\_start time\_span age\_span value  
#> 1950 male 0 1 1 -1906.5913  
#> 1950 male 1 . . -772.8935  
#> 1950 male 2 . . 52.2876  
#> 1950 male 3 . . 765.6268  
#> 1950 male 4 . . 1291.9252  
#> 1950 male 5 . . 1651.7646  
#> # ... etc.

The print method for objects of class fert\_rate\_age\_f attempts to emphasize the “non\_zero\_fert\_ages” attribute:

fert\_rate\_input\_example\_df  
#> # A 'fert\_rate\_age\_f' with 7,070 rows.  
#> # dimensions = 'time, age'.  
#> # value\_type = 'rate', age\_span = '1', time\_span = '1'.  
#> # non\_zero\_fert\_ages = '15, 16, ... 44, 45'.  
#> time\_start age\_start time\_span age\_span value  
#> 1950 0 1 1 [zero]  
#> 1950 1 . . [zero]  
#> 1950 2 . . [zero]  
#> . . . . .  
#> 1950 15 . . 0.00489907203  
#> 1950 16 . . 0.01217600016  
#> 1950 17 . . 0.03258414286  
#> # ... etc.

Note that the special formatting is only for vizualization; the actual values for age\_span, time\_span and value are not altered in the data components.

fert\_rate\_input\_example\_df$value[1:10]  
#> [1] 0 0 0 0 0 0 0 0 0 0

### summary

The method for summary is based on the data.frame method, but gives some additional summary information:

summary(demog\_change\_example\_df)  
#> dimensions: time, sex, age  
#> time: range = [1950, 2019]  
#> age: range = [0, 100]  
#> sex: levels = female, male  
#> value\_type: real  
#> ------------------------------------------------------------  
#> table:  
#> time\_start sex age\_start time\_span age\_span  
#> Min. :1950 Length:14140 Min. : 0 Min. :1 Min. :1   
#> 1st Qu.:1967 Class :character 1st Qu.: 25 1st Qu.:1 1st Qu.:1   
#> Median :1984 Mode :character Median : 50 Median :1 Median :1   
#> Mean :1984 Mean : 50 Mean :1 Mean :1   
#> 3rd Qu.:2002 3rd Qu.: 75 3rd Qu.:1 3rd Qu.:1   
#> Max. :2019 Max. :100 Max. :1 Max. :1   
#> value   
#> Min. :-1918.91   
#> 1st Qu.: 40.51   
#> Median : 410.62   
#> Mean : 744.91   
#> 3rd Qu.: 1201.08   
#> Max. : 6661.00

# Using and Manipulating Objects

## Object Queries

Utility functions are provided for querying various attributes and characteristics of objects inheriting from demog\_change\_component\_df.

### Dimensions Present

To get the names of the dimensions of an object use demog\_change\_component\_dimensions[[1]](#footnote-50)

demog\_change\_component\_dimensions(demog\_change\_example\_df)  
#> [1] "time" "sex" "age"

Each dimension also has an associated “is\_by\_[dim]” function to test whether “[dim]” is present:

is\_by\_age(demog\_change\_example\_df)  
#> [1] TRUE  
is\_by\_indicator(demog\_change\_example\_df)  
#> [1] FALSE

### Dimension Values

To get the values of specific dimensions functions named for the dimensions are provided, e.g., ages, times. These return the unique values of the requested dimension.

ages(demog\_change\_example\_df)  
#> [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  
#> [19] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35  
#> [37] 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53  
#> [55] 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71  
#> [73] 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89  
#> [91] 90 91 92 93 94 95 96 97 98 99 100  
  
## NOT RUN:  
## indicators(demog\_change\_example\_df)  
## Error in indicators.demog\_change\_component\_df(demog\_change\_example\_df) :   
## 'indicator' is not a dimension of 'x'.

### Other Attributes

Functions value\_type, age\_span, time\_span, non\_zero\_fert\_ages are also available to access those attributes for classes that have them.

## Methods for Standard Generic Functions

The following *methods* are provided for *R*’s existing generic functions.

### Subsetting

If an object inheriting from the demog\_change\_component\_df class is subsetted using any of “[” , “$”, “[[” the result looses the class. A warning will be issued if used interactively:

## Extract one column:  
z <- demog\_change\_example\_df[, "age\_start"]  
#> Warning in `[.demog\_change\_component\_df`(demog\_change\_example\_df, ,  
#> "age\_start"): Subsetting a 'demog\_change\_component\_df' will not preserve the  
#> class or attributes. See '?subset\_time' and friends for an alternative approach.  
class(z)  
#> [1] "numeric"

## Extract a sub-data frame  
z <- demog\_change\_example\_df[demog\_change\_example\_df$time\_start == 1950, ]  
#> Warning in `[.demog\_change\_component\_df`(demog\_change\_example\_df,  
#> demog\_change\_example\_df$time\_start == : Subsetting a 'demog\_change\_component\_df'  
#> will not preserve the class or attributes. See '?subset\_time' and friends for an  
#> alternative approach.  
class(z)  
#> [1] "data.frame"

The warning is only issued if “[” (or “$”, or “[[”) are called at the top level. E.g.,

## Same effect as `[` above but no warning  
z <- subset(demog\_change\_example\_df, time\_start == 1950)  
class(z)  
#> [1] "data.frame"

To create a valid subset, re-make the object as a member of the class (suppressWarnings() is optional):

z <-  
 demog\_change\_component\_df(  
 suppressWarnings(  
 demog\_change\_example\_df[demog\_change\_example\_df$time\_start == 1950, ]),  
 age\_span = attr(demog\_change\_example\_df, "age\_span"),  
 time\_span = attr(demog\_change\_example\_df, "time\_span"))  
#> Argument 'dimensions' is 'NULL'; setting 'dimensions' to 'time, sex, age' based on column names of 'x'.  
#> Argument 'value\_type' is 'NULL'; setting 'value\_type' to 'real'.  
class(z)  
#> [1] "demog\_change\_component\_df" "data.frame"

Alternatively, one of the special subset functions could be used (see Section @ref(new-subset-functions)).

### Subset-Replacing

If a column in an object inheriting from demog\_change\_component\_df is replaced using any of “[<-” , “$<-”, “[[<-” the class will, again, be dropped.

z <- demog\_change\_example\_df  
z[, "age\_start"] <- z$age\_start  
#> Warning in `[<-.demog\_change\_component\_df`(`\*tmp\*`, , "age\_start", value =  
#> c(0, : Replacing elements in a 'demog\_change\_component\_df' will not preserve the  
#> class or attributes.  
class(z)  
#> [1] "data.frame"

### Combining

Combining objects inheriting from demog\_change\_component\_df via base::rbind will also cause the class to be dropped. As above, pass the result back to the relevant constructor function to create a new classed object after the bind.

### Coercion to Data Frames

A method has been written for base::as.data.frame that also ensures the class is dropped.

z <- as.data.frame(demog\_change\_example\_df)  
#> Warning in as.data.frame.demog\_change\_component\_df(demog\_change\_example\_df): The  
#> result of the coercion will not inherit from class 'demog\_change\_component\_df'  
#> and will not have any attributes specific to that class.  
class(z)  
#> [1] "data.frame"

### New Utility Functions for Subsetting

New functions are provided to subset objects inheriting from demog\_change\_component\_df by one or more of their dimensions, and return an object that still inherits from the class. They are named systematically as follows subset\_[dimension], where “[dimension]” is one of the valid dimensions. The first argument is the object to subset, the second is a vector giving the values of the dimension to be retained in the subset. For example, to subset on time use subset\_time:

x <- subset\_time(demog\_change\_example\_df,  
 times = c("1990", "1991"))  
class(x)  
#> [1] "demog\_change\_component\_df" "data.frame"  
summary(x)  
#> dimensions: time, sex, age  
#> time: range = [1990, 1991]  
#> age: range = [0, 100]  
#> sex: levels = female, male  
#> value\_type: real  
#> ------------------------------------------------------------  
#> table:  
#> time\_start sex age\_start time\_span age\_span  
#> Min. :1990 Length:404 Min. : 0 Min. :1 Min. :1   
#> 1st Qu.:1990 Class :character 1st Qu.: 25 1st Qu.:1 1st Qu.:1   
#> Median :1990 Mode :character Median : 50 Median :1 Median :1   
#> Mean :1990 Mean : 50 Mean :1 Mean :1   
#> 3rd Qu.:1991 3rd Qu.: 75 3rd Qu.:1 3rd Qu.:1   
#> Max. :1991 Max. :100 Max. :1 Max. :1   
#> value   
#> Min. :-193.86   
#> 1st Qu.: 37.55   
#> Median : 544.97   
#> Mean : 748.84   
#> 3rd Qu.:1329.46   
#> Max. :2520.15

To subset on multiple dimensions use the respective functions in a pipe:

library(magrittr)  
x <- subset\_time(demog\_change\_example\_df,  
 times = c("1990", "1991")) %>%  
 subset\_sex("male")  
class(x)  
#> [1] "demog\_change\_component\_df" "data.frame"  
summary(x)  
#> dimensions: time, sex, age  
#> time: range = [1990, 1991]  
#> age: range = [0, 100]  
#> sex: levels = male  
#> value\_type: real  
#> ------------------------------------------------------------  
#> table:  
#> time\_start sex age\_start time\_span age\_span  
#> Min. :1990 Length:202 Min. : 0 Min. :1 Min. :1   
#> 1st Qu.:1990 Class :character 1st Qu.: 25 1st Qu.:1 1st Qu.:1   
#> Median :1990 Mode :character Median : 50 Median :1 Median :1   
#> Mean :1990 Mean : 50 Mean :1 Mean :1   
#> 3rd Qu.:1991 3rd Qu.: 75 3rd Qu.:1 3rd Qu.:1   
#> Max. :1991 Max. :100 Max. :1 Max. :1   
#> value   
#> Min. :-135.56   
#> 1st Qu.: 56.11   
#> Median : 514.73   
#> Mean : 716.94   
#> 3rd Qu.:1309.58   
#> Max. :2520.15

A subset that is not a valid class object will not be created and an error will be signalled:

## NOT RUN:  
## x <- subset\_age(ccmpp\_input\_example\_df, age = 5)  
## Error in validate\_ccmpp\_object.ccmpp\_input\_df(new\_ccmpp\_input\_df(li$df, :   
## 'age\_start' does not start at '0' for each time \* sex combination.

# What Next?

1. Get feedback on the general set-up, names of things, operation, etc.
2. Decide if the output list would benefit from having classes defined like the input list. This would be straightforward given the infrastructure already created.
3. Complete documentation files for functions and classes (e.g., ?ccmpp\_input\_list).
4. Consider other methods that would be useful. E.g., would a plot method be useful to vizualize the inputs? This would allow quick and easier creation of a standard set of vizualizations to check the inputs/outputs.

# References

Wickham, Hadley. 2019. *Advanced R*. 2nd ed. Boca Raton, Florida: CRC Press. <https://adv-r.hadley.nz/>.

1. This name is probably too long. [↑](#footnote-ref-50)