dfidx

Yves Croissant

2024/08/22

In some situations, series from a data frame have a natural two-dimensional (tabular) representation because each observation can be uniquely characterized by a combination of two indexes. Two major cases of this situations in applied econometrics are:

- panel data, where the same individuals are observed for several time periods,
- random utility models, where each observation describes the features of an alternative among a set of alternatives for a given choice situation.

The idea of **dfidx** is to keep in the same object the data and the information about its structure. A `dfidx` is a data frame with an `idx` column, which is a data frame that contains the series that defines the indexes.

This vignette supersede the preceding vignette of the **dfidx** package by showing the advantages of creating `dfidx` objects from a tibble and not from an ordinary data frame.[1]. It also introduces a new vector interface to define the indexes.

1 Basic use of the `dfidx` function

The `dfidx` package is loaded using:

```
library(dfidx)
```

We also attach the **dplyr** package because we'll use throughout this vignette tibbles and not ordinary data frames and we'll show how **dplyr**'s verbs can be used with `dfidx` objects thanks to appropriate methods.

```
library(dplyr)
```

[1]The advantage of attaching the **dplyr** package (Wickham et al. 2023) is that the **magrittr**'s pipe (Bache and Wickham 2022) and functions from the **tibble** package (Müller and Wickham 2023) are exported

To illustrate the features of **dfidx**, we'll use the `munnell` data set (Munnell 1990) that is used in Baltagi (2013)'s famous book and is part of the **plm** package as `Produc`. It contains several economic series for American states from 1970 to 1986. We've added to the initial data set a `president` series which indicates the name of the American president in power for the given year.

```
munnell
```

```
# A tibble: 816 x 12
  state    year region     president publiccap highway water utilities
  <chr>   <int> <chr>      <chr>         <dbl>   <dbl> <dbl>     <dbl>
1 Alabama  1970 East-Sout~ Nixon        15033.   7326. 1656.     6051.
2 Alabama  1971 East-Sout~ Nixon        15502.   7526. 1721.     6255.
3 Alabama  1972 East-Sout~ Nixon        15972.   7765. 1765.     6442.
# i 813 more rows
# i 4 more variables: privatecap <dbl>, gsp <int>, labor <dbl>,
#   unemp <dbl>
```

The two indexes are `state` and `year` and both are nested in another variable: `state` in `region` and `year` in `president`. A `dfidx` object is created with the `dfidx` function: the first argument should be a data frame (or a tibble) and the second argument `idx` is used to indicate the indexes. As, in the `munnell` data set, the first two columns contain the two indexes, the `idx` argument is not mandatory and a `dfidx` can be obtained from the `munnell` tibble simply by using:

```
munnell %>% dfidx
```

```
# A tibble: 816 x 11
# Index:    48 (state) x 17 (year)
# Balanced: yes
  idx          region      president publiccap highway water utilities
  <idx>        <chr>       <chr>         <dbl>   <dbl> <dbl>     <dbl>
1 Alabama:1970 East-South~ Nixon        15033.   7326. 1656.     6051.
2 Alabama:1971 East-South~ Nixon        15502.   7526. 1721.     6255.
3 Alabama:1972 East-South~ Nixon        15972.   7765. 1765.     6442.
# i 813 more rows
# i 4 more variables: privatecap <dbl>, gsp <int>, labor <dbl>,
#   unemp <dbl>
```

The resulting object is of class `dfidx` and is a tibble with an `idx` column, which is a tibble containing the two indexes. Note that the two indexes are now longer standalone series in the resulting tibble, because the default value of the `drop.index` argument is `TRUE`. The header of the tibble indicates the names and the cardinal of the two indexes. It also indicated whether the data set is balanced ie, in this panel data context, whether all the states are observed for the same set of years (which is the case for the `munnell` data set). The `idx` column can be retrieved using the `idx` function:

```
munnell %>% dfidx %>% idx
```

```
# A tibble: 816 x 2
  state   year
  <chr>   <fct>
1 Alabama 1970
2 Alabama 1971
3 Alabama 1972
# i 813 more rows
```

If the first two columns don't contain the indexes, the `idx` argument should be set. If the observations are ordered first by the first index and then by the second one and if the data set is *balanced*, `idx` can be an integer, the number of distinct values of the first index:

```
munnell %>% dfidx(48)
```

```
# A tibble: 816 x 13
# Index:    48 (id1) x 17 (id2)
# Balanced: yes
  idx   state    year region          president publiccap highway water
  <idx> <chr>   <int> <chr>           <chr>         <dbl>   <dbl> <dbl>
1 1:1   Alabama  1970 East-South Ce~ Nixon        15033.   7326. 1656.
2 1:2   Alabama  1971 East-South Ce~ Nixon        15502.   7526. 1721.
3 1:3   Alabama  1972 East-South Ce~ Nixon        15972.   7765. 1765.
# i 813 more rows
# i 5 more variables: utilities <dbl>, privatecap <dbl>, gsp <int>,
#   labor <dbl>, unemp <dbl>
```

Then the two indexes are created with the default names `id1` and `id2`. More relevant names can be indicated using the `idnames` argument and the values of the second index can be indicated, using the `levels` argument.

```
munnell %>% dfidx(48, idnames = c("state", "year"), levels = 1970:1986)
```

```
# A tibble: 816 x 11
# Index:    48 (state) x 17 (year)
# Balanced: yes
  idx    region             president publiccap highway water utilities
  <idx>  <chr>              <chr>         <dbl>   <dbl> <dbl>     <dbl>
1 1:1970 East-South Centr~ Nixon        15033.   7326. 1656.     6051.
2 1:1971 East-South Centr~ Nixon        15502.   7526. 1721.     6255.
3 1:1972 East-South Centr~ Nixon        15972.   7765. 1765.     6442.
# i 813 more rows
# i 4 more variables: privatecap <dbl>, gsp <int>, labor <dbl>,
#   unemp <dbl>
```

The `idx` argument can also be a character of length one or two. In the first case, only the first index is indicated:

```
munnell %>% dfidx("state", idnames = c(NA, "date"), levels = 1970:1986)
```

```
# A tibble: 816 x 12
# Index:    48 (state) x 17 (date)
# Balanced: yes
  idx          year region president publiccap highway water utilities
  <idx>       <int> <chr>  <chr>         <dbl>   <dbl> <dbl>     <dbl>
1 Alaba~:1970  1970 East-~ Nixon        15033.   7326. 1656.     6051.
2 Alaba~:1971  1971 East-~ Nixon        15502.   7526. 1721.     6255.
3 Alaba~:1972  1972 East-~ Nixon        15972.   7765. 1765.     6442.
# i 813 more rows
# i 4 more variables: privatecap <dbl>, gsp <int>, labor <dbl>,
#   unemp <dbl>
```

Note that we've only provided a name for the second index, the `NA` in the first position of the `idnames` argument meaning that we want to keep the original name for the first index. Finally, if the `idx` argument is a character of length 2, it should contain the name of the two indexes.

```
munnell %>% dfidx(c("state", "year"))
```

```
# A tibble: 816 x 11
# Index:    48 (state) x 17 (year)
# Balanced: yes
  idx          region      president publiccap highway water utilities
  <idx>        <chr>       <chr>         <dbl>   <dbl> <dbl>     <dbl>
1 Alabama:1970 East-South~ Nixon        15033.   7326. 1656.     6051.
2 Alabama:1971 East-South~ Nixon        15502.   7526. 1721.     6255.
3 Alabama:1972 East-South~ Nixon        15972.   7765. 1765.     6442.
# i 813 more rows
# i 4 more variables: privatecap <dbl>, gsp <int>, labor <dbl>,
#   unemp <dbl>
```

2 More advanced use of `dfidx`

2.1 Nesting structure

One or both of the indexes may be nested in another series. In this case, the `idx` argument is still a character of length two, but the nesting series is indicated as the name of the corresponding index:

```
mn <- munnell %>% dfidx(c(region = "state", "year"))
mn <- munnell %>% dfidx(c(region = "state", president = "year"))
mn
```

```
# A tibble: 816 x 9
# Index:    48 (state) x 17 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
  idx         publiccap highway water utilities privatecap    gsp labor
  <idx>           <dbl>   <dbl> <dbl>     <dbl>      <dbl>  <int> <dbl>
1 Illi~:1977     62201.  26836. 7670.    27696.    146286. 168627 4656.
2 Illi~:1978     63096.  27300. 8005.    27791.    150855. 173767 4789.
3 Illi~:1979     63643.  27247. 8491.    27904.    156752. 173817 4880
# i 813 more rows
# i 1 more variable: unemp <dbl>
```

The `idx` column is now a tibble containing the two indexes and the nesting variables.

```
mn %>% idx
```

```
# A tibble: 816 x 4
  state    region             year  president
  <chr>    <chr>              <fct> <fct>
1 Illinois East-North Central 1977  Carter
```

```
2 Illinois East-North Central 1978  Carter
3 Illinois East-North Central 1979  Carter
# i 813 more rows
```

2.2 Data frames in wide format

`dfidx` can deal with data frames in wide format, *i.e* for which each series for a given value of the second index is a column of the data frame. This is the case of the `munnell_wide` tibble that contains two series of the original data set (`gsp` and `unemp`).

```
munnell_wide
```

```
# A tibble: 48 x 36
  state   region gsp_1970 gsp_1971 gsp_1972 gsp_1973 gsp_1974 gsp_1975
  <chr>   <chr>     <int>    <int>    <int>    <int>    <int>    <int>
1 Alabama East-~    28418    29375    31303    33430    33749    33604
2 Arizona Monta~    19288    21040    23289    25244    25698    24915
3 Arkans~ West-~    15392    16177    17702    18825    19287    19024
# i 45 more rows
# i 28 more variables: gsp_1976 <int>, gsp_1977 <int>,
#   gsp_1978 <int>, gsp_1979 <int>, gsp_1980 <int>, gsp_1981 <int>,
#   gsp_1982 <int>, gsp_1983 <int>, gsp_1984 <int>, gsp_1985 <int>,
#   gsp_1986 <int>, unemp_1970 <dbl>, unemp_1971 <dbl>,
#   unemp_1972 <dbl>, unemp_1973 <dbl>, unemp_1974 <dbl>,
#   unemp_1975 <dbl>, unemp_1976 <dbl>, unemp_1977 <dbl>, ...
```

Each line is now an American state and, apart the indexes, there are now 34 series with names obtained by the concatenation of the name of the series and the year (for example `gsp_1988`). In this case a supplementary argument called `varying` should be provided. It is a vector of integers indicating the position of the columns that should be merged in the resulting long formatted data frame. The `stats::reshape` function is then used and the `sep` argument can be also provided to indicate the separating character in the names of the series (the default value being `"."`).

```
munnell_wide %>% dfidx(varying = 3:36, sep = "_")
```

```
# A tibble: 816 x 5
# Index:    48 (id1) x 17 (id2)
# Balanced: yes
  idx    state   region               gsp unemp
  <idx>  <chr>   <chr>              <int> <dbl>
1 1:1970 Alabama East-South Central 28418   4.7
2 1:1971 Alabama East-South Central 29375   5.2
3 1:1972 Alabama East-South Central 31303   4.7
# i 813 more rows
```

Better results can be obtained using the `idx` and `idnames` previously described:

```
munnell_wide %>% dfidx(idx = c(region = "state"), varying = 3:36,
                       sep = "_", idnames = c(NA, "year"))
```

```
# A tibble: 816 x 3
# Index:    48 (state) x 17 (year)
# Balanced: yes
# Nesting:  state (region)
  idx              gsp unemp
  <idx>          <int> <dbl>
1 Illinois:1970 145792   4.1
2 Illinois:1971 148503   5.1
3 Illinois:1972 154413   5.1
# i 813 more rows
```

3 Getting the indexes or their names

The name (and the position) of the `idx` column can be obtained as a named integer (the integer being the position of the column and the name its name) using the `idx_name` function:

```
idx_name(mn)
## idx
##   1
```

To get the name of one of the indexes, the second argument, `n`, is set either to 1 or 2 to get the first or the second index, ignoring the nesting variables:

```
idx_name(mn, 2)
## [1] "year"
idx_name(idx(mn), 2)
## [1] "year"
```

Not that `idx_name` can be in this case applied to a `dfidx` or to a `idx` object. To get a nesting variable, the third argument, called `m`, is set to 2:

```r
idx_name(mn, 1, 1)
## [1] "state"
idx_name(mn, 1, 2)
## [1] "region"
```

To extract one or all the indexes, the `idx` function is used. This function has already been encountered when one wants to extract the `idx` column of a `dfidx` object. The same `n` and `m` arguments as for the `idx_name` function can be used in order to extract a specific series. For example, to extract the region index, which nests the state index:

```r
id_index1 <- idx(mn, n = 1, m = 2)
id_index2 <- idx(idx(mn), n = 1, m = 2)
head(id_index1)
## [1] "East-North Central" "East-North Central" "East-North Central"
## [4] "East-North Central" "East-North Central" "East-North Central"
identical(id_index1, id_index2)
## [1] TRUE
```

4 Data frames subsetting

Subsets of data frames are obtained using the `[` and the `[[` operators. The former returns most of the time a data frame as the second one always returns a series.

4.1 Commands that return a data frame

Consider first the use of `[`. If one argument is provided, it indicates the columns that should be selected. The result is always a data frame, even if a single column is selected. If two arguments are provided, the first one indicates the subset of lines and the second one the subset of columns that should be returned. If only one column is selected, the result depends on the value of the `drop` argument. If `TRUE`, a series is returned and if `FALSE`, a one series data frame is returned. An important difference between tibbles and ordinary data frames is that the default value of `drop` is `FALSE` for the former and `TRUE` for the later. Therefore, with tibbles, the use of `[` will always by default return a data frame.

A specific `dfidx` method is provided for one reason: the column that contains the indexes should be "sticky" (we borrow this idea from the `sf` package[2]), which means that it should be always returned while using the extractor operator, even if it is not explicitly selected.

[2]Pebesma and Bivand (2023) and Pebesma (2018).

```r
mn[mn$unemp > 10, ]
```

```
# A tibble: 46 x 9
# Index:    19 (state) x 8 (year)
# Balanced: no
# Nesting:  state (region), year (president)
  idx        publiccap highway  water utilities privatecap    gsp labor
  <idx>          <dbl>   <dbl>  <dbl>     <dbl>      <dbl>  <int> <dbl>
1 Illi~1982     65064.  27568. 10218     27278.    154806. 159778 4593.
2 Illi~1983     64752.  27483  10436.    26833.    157096. 160856 4531.
3 Indi~1982     25109.  10619.  3297.    11193.     82361.  64042 2028
# i 43 more rows
# i 1 more variable: unemp <dbl>
```

```r
mn[mn$unemp > 10, c("highway", "utilities")]
```

```
# A tibble: 46 x 3
# Index:    19 (state) x 8 (year)
# Balanced: no
# Nesting:  state (region), year (president)
  highway utilities idx
    <dbl>     <dbl> <idx>
1  27568.    27278. Illinois:1982
2  27483     26833. Illinois:1983
3  10619.    11193. Indiana:1982
# i 43 more rows
```

```r
mn[mn$unemp > 10, "highway"]
```

```
# A tibble: 46 x 2
# Index:    19 (state) x 8 (year)
# Balanced: no
# Nesting:  state (region), year (president)
  highway idx
    <dbl> <idx>
1  27568. Illinois:1982
2  27483  Illinois:1983
3  10619. Indiana:1982
# i 43 more rows
```

All the previous commands extract the observations where the unemployment rate is greater than 10% and, in the first case all the series, in the second case two of them and in the third case only one series.

4.2 Commmands that return a series

A series can be extracted using any of the following commands:

```r
mn1 <- mn[, "highway", drop = TRUE]
mn2 <- mn[["highway"]]
mn3 <- mn$highway
c(identical(mn1, mn2), identical(mn1, mn3))
## [1] TRUE TRUE
```

The result is a `xseries` which inherits the `idx` column from the data frame it has been extracted from as an attribute :

```r
mn1 %>% print(n = 3)
```

```
# Index: 48 (state) x 17 (year)
[1] 26835.52 27300.22 27247.22
```

```r
class(mn1)
```

```
[1] "xseries" "numeric"
```

```r
idx(mn1) %>% print(n = 3)
```

```
# A tibble: 816 x 4
  state    region             year  president
  <chr>    <chr>              <fct> <fct>
1 Illinois East-North Central 1977  Carter
2 Illinois East-North Central 1978  Carter
3 Illinois East-North Central 1979  Carter
# i 813 more rows
```

Note that, except when `dfidx` hasn't been used with `drop.index = FALSE`, a series which defines the indexes is dropped from the data frame (but is one of the column of the `idx` column of the data frame). It can be therefore retrieved using:

```
mn$idx$president %>% head
```

```
[1] Carter Carter Carter Carter Ford   Ford
Levels: Carter Ford Nixon Reagan
```

or

```
idx(mn)$president %>% head
```

```
[1] Carter Carter Carter Carter Ford   Ford
Levels: Carter Ford Nixon Reagan
```

or more simply by applying the `$` operator as if the series were a stand-alone series in the data frame :

```
mn$president %>% print(n = 3)
```

```
# Index: 48 (state) x 17 (year)
[1] Carter Carter Carter
Levels: Carter Ford Nixon Reagan
```

In this last case, the resulting series is a `xseries`, *ie* it inherits the index data frame as an attribute.

4.3 User defined class for extracted series

While creating the `dfidx`, a `pkg` argument can be indicated, so that the resulting `dfidx` object and its series are respectively of class `c("dfidx_pkg", "dfidx")` and `c("xseries_pkg", "xseries")` which enables the definition of special methods for `dfidx` and `xseries` objects. For example, consider the hypothetical **pnl** package for panel data:

```
mn <- dfidx(munnell, idx = c(region = "state", president = "year"),
                                pkg = "pnl")
mn1 <- mn$gsp
class(mn)
## [1] "dfidx_pnl"  "dfidx"      "tbl_df"     "tbl"        "data.frame"
class(mn1)
## [1] "xseries_pnl" "xseries"     "integer"
```

For example, we want to define a `lag` method for `xseries_pnl` objects. While lagging there should be a `NA` not only on the first position of the resulting vector like for time-series, but each time we encounter a new individual. A minimal `lag` method could therefore be written as:

```
lag.xseries_pnl <- function(x, ...){
    .idx <- idx(x)
    class <- class(x)
    x <- unclass(x)
    id <- .idx[[1]]
    lgt <- length(id)
    lagid <- c("", id[- lgt])
    sameid <- lagid ==  id
    x <- c(NA, x[- lgt])
    x[! sameid] <- NA
    structure(x, class = class, idx = .idx)
}
lmn1 <- stats::lag(mn1)
lmn1 %>% print(n = 3)
```

```
# Index: 48 (state) x 17 (year)
[1]     NA 168627 173767
```

```
class(lmn1)
```

```
[1] "xseries_pnl" "xseries"     "integer"
```

```
rbind(mn1, lmn1)[, 1:20]
```

```
       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
mn1  168627 173767 173817 165722 157366 163112 145792 148503 154413
lmn1     NA 168627 173767 173817 165722 157366 163112 145792 148503
      [,10]  [,11]  [,12]  [,13]  [,14]  [,15]  [,16]  [,17] [,18]
mn1  163125 161725 166029 159778 160856 173602 178493 183849 68832
lmn1 154413 163125 161725 166029 159778 160856 173602 178493    NA
     [,19] [,20]
mn1  71717 72047
lmn1 68832 71717
```

Note the use of `stats::lag` instead of `lag` which ensures that the `stats::lag` function is used, even if the **dplyr** (or **tidyverse**) package is attached.

5 tidyverse

5.1 dplyr

dfidx supports some of the verbs of **dplyr**, namely, for the current version:

- `select` to select columns,
- `filter` to select some rows using logical conditions,
- `arrange` to sort the lines according to one or several variables,
- `mutate` and `transmute` for creating new series,
- `slice` to select some rows using their position.

dplyr's verbs don't work with `dfidx` objects for two main reasons:

- the first one is that with most of the verbs (`select` is an exception), the returned object is a `data.frame` (or a `tibble`) and not a `dfidx`,
- the second one is that the index column should be "sticky", which means that it should be always returned, even while selecting a subset of columns which doesn't include the index column or while using `transmute`.

Therefore, specific methods are provided for **dplyr**'s verb. The general strategy consists on:

1. first save the original attributes of the argument (a `dfidx` object),
2. coerce to a `data.frame` or a tibble using the `as.data.frame` method,
3. use `dplyr`'s verb,
4. add the column containing the index if necessary (*i.e.* while using `transmute` or while `select`ing a subset of columns which don't contain the index column),
5. change some of the attributes if necessary,
6. attach the attributes to the `data.frame` and returns the result.

The following code illustrates the use of **dplyr**'s verbs applied to `dfidx` objects.

```r
select(mn, highway, utilities)
```

```
# A tibble: 816 x 3
# Index:    48 (state) x 17 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
  highway utilities idx
    <dbl>     <dbl> <idx>
1  26836.    27696. Illinois:1977
2  27300.    27791. Illinois:1978
3  27247.    27904. Illinois:1979
# i 813 more rows
```

```r
arrange(mn, desc(unemp))
```

```
# A tibble: 816 x 9
# Index:    48 (state) x 17 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
  idx        publiccap highway  water utilities privatecap    gsp labor
  <idx>          <dbl>   <dbl>  <dbl>     <dbl>      <dbl>  <int> <dbl>
1 West~1983     11079.   7551.   756.     2772.     35933.  20822  582.
2 Mich~1982     51956.  19881. 10759.    21316.    115911. 108627 3193.
3 West~1984     11073.   7562.   809.     2702.     36068.  21615  597.
# i 813 more rows
# i 1 more variable: unemp <dbl>
```

```r
mutate(mn, lgsp = log(gsp), lgsp2 = lgsp ^ 2)
```

```
# A tibble: 816 x 11
# Index:    48 (state) x 17 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
  idx         publiccap highway water utilities privatecap    gsp labor
  <idx>           <dbl>   <dbl> <dbl>     <dbl>      <dbl>  <int> <dbl>
1 Illi~:1977     62201.  26836. 7670.    27696.    146286. 168627 4656.
2 Illi~:1978     63096.  27300. 8005.    27791.    150855. 173767 4789.
3 Illi~:1979     63643.  27247. 8491.    27904.    156752. 173817 4880
# i 813 more rows
# i 3 more variables: unemp <dbl>, lgsp <dbl>, lgsp2 <dbl>
```

```r
transmute(mn, lgsp = log(gsp), lgsp2 = lgsp ^ 2)
```

```
# A tibble: 816 x 3
# Index:    48 (state) x 17 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
   lgsp lgsp2 idx
  <dbl> <dbl> <idx>
1  12.0  145. Illinois:1977
2  12.1  146. Illinois:1978
3  12.1  146. Illinois:1979
# i 813 more rows
```

```
filter(mn, unemp > 10, gsp > 150000)
```

```
# A tibble: 2 x 9
# Index:    1 (state) x 2 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
  idx       publiccap highway  water utilities privatecap    gsp labor
  <idx>         <dbl>   <dbl>  <dbl>     <dbl>      <dbl>  <int> <dbl>
1 Illi~1982    65064.  27568. 10218     27278.    154806. 159778 4593.
2 Illi~1983    64752.  27483  10436.    26833.    157096. 160856 4531.
# i 1 more variable: unemp <dbl>
```

```
slice(mn, 1:3)
```

```
# A tibble: 3 x 9
# Index:    1 (state) x 3 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
  idx        publiccap highway water utilities privatecap    gsp labor
  <idx>          <dbl>   <dbl> <dbl>     <dbl>      <dbl>  <int> <dbl>
1 Illi~:1977    62201.  26836. 7670.    27696.    146286. 168627 4656.
2 Illi~:1978    63096.  27300. 8005.    27791.    150855. 173767 4789.
3 Illi~:1979    63643.  27247. 8491.    27904.    156752. 173817 4880
# i 1 more variable: unemp <dbl>
```

To extract a series, the `pull` function can be used:

```
mn %>% pull(utilities)
```

```
# Index: 48 (state) x 17 (year)
 [1] 27695.71 27791.28 27904.24 27718.08 26728.17 27256.29 22252.68
 [8] 23384.86 24261.85 25032.14
```

6 Model building

The two main steps in **R** in order to estimate a model are to use the `model.frame` function to construct a data frame, using a formula and a data frame and then to extract from it the matrix of covariates using the `model.matrix` function.

6.1 Model frame

The default method of `model.frame` has as first two arguments `formula` and `data`. It returns a data frame with a `terms` attribute. Some other methods exist in the **stats** package, for example for `lm` and `glm` object with a first and main argument called `formula`. This is quite unusual and misleading as for most of the generic functions in **R**, the first argument is called either `x` or `object`.

Another noticeable method for `model.frame` is provided by the **Formula** package and, in this case, the first argument is a `Formula` object, which is an extended formula which can contain several parts on the left and/or on the right hand side of the formula.

We provide a `model.frame` method for `dfidx` objects, mainly because the `idx` column should be returned in the resulting data frame. This leads to an unusual order of the arguments, the data frame first and then the formula. The method then first extract (and subset if necessary the `idx` column), call the `formula`/`Formula` method and then add to the resulting data frame the `idx` column. The resulting data frame is a `dfidx` object.

```
mf_mn <- mn %>% model.frame(gsp ~ utilities + highway | unemp | labor,
                            subset = unemp > 10)
mf_mn
```

```
# A tibble: 46 x 6
# Index:    19 (state) x 8 (year)
# Balanced: no
# Nesting:  state (region), year (president)
     gsp utilities highway unemp labor idx
   <int>     <dbl>   <dbl> <dbl> <dbl> <idx>
1 159778    27278.  27568.    11 4593. Illinois:1982
2 160856    26833.  27483     11 4531. Illinois:1983
3  64042    11193.  10619.    12 2028  Indiana:1982
# i 43 more rows
```

```
formula(mf_mn)
```

```
gsp ~ utilities + highway + unemp + labor + (state + region +
    year + president)
<environment: 0x5ccc712e9720>
```

6.2 Model matrix

`model.matrix` is a generic function and for the default method, the first two arguments are a `terms` object and a data frame. In `lm`, the `terms` attribute is extracted from the `model.frame` internally constructed using the `model.frame` function. This means that, at least in this context, `model.matrix` doesn't need a `formula`/`term` argument and a `data.frame`, but only a data frame returned by the model frame method, i.e. a data frame with a `terms` attribute.

We use this idea for the `model.matrix` method for `dfidx` object; the only required argument is a `dfidx` returned by the `model.frame` function. The formula is then extracted from the `dfidx` and the `Formula` or default method is then called. The result is a matrix of class `dfidx_matrix`, with a printing method that allows the use of the `n` argument:

```
mf_mn %>% model.matrix(rhs = 1)
```

```
# [46 x 3]
   (Intercept) utilities  highway
1            1  27277.69 27568.50
2            1  26832.94 27483.00
3            1  11192.68 10618.71
4            1  11141.74 10558.11
5            1  21281.74 19996.38
6            1  20311.41 19397.17
7            1  21352.04 20024.11
8            1  21316.04 19881.31
9            1  21012.58 19714.51
10           1  20634.78 19505.44
```

```
mf_mn %>% model.matrix(rhs = 2:3) %>% print(n = 5)
```

```
# [46 x 3]
  (Intercept) unemp  labor
1           1    11 4593.3
2           1    11 4530.6
3           1    12 2028.0
4           1    11 2029.5
5           1    12 3442.8
```

```
mn <- dfidx(munnell, idx = c(region = "state", president = "year"),
            name = "index", position = 4)
```

```
mn
```

```
# A tibble: 816 x 9
# Index:    48 (state) x 17 (year)
# Balanced: yes
# Nesting:  state (region), year (president)
  publiccap highway water index      utilities privatecap    gsp labor
      <dbl>   <dbl> <dbl> <idx>          <dbl>      <dbl>  <int> <dbl>
1    62201.  26836. 7670. Illi~:1977    27696.    146286. 168627 4656.
2    63096.  27300. 8005. Illi~:1978    27791.    150855. 173767 4789.
3    63643.  27247. 8491. Illi~:1979    27904.    156752. 173817 4880 
# i 813 more rows
# i 1 more variable: unemp <dbl>
```

References

Bache, Stefan Milton, and Hadley Wickham. 2022. *Magrittr: A Forward-Pipe Operator for r.* https://CRAN.R-project.org/package=magrittr.

Baltagi, B. H. 2013. *Econometric Analysis of Panel Data.* 5th ed. John Wiley; Sons ltd.

Müller, Kirill, and Hadley Wickham. 2023. *Tibble: Simple Data Frames.* https://CRAN.R-project.org/package=tibble.

Munnell, A. 1990. "Why Has Productivity Growth Declined? Productivity and Public Investment." *New England Economic Review*, 3–22.

Pebesma, Edzer. 2018. "Simple Features for R: Standardized Support for Spatial Vector Data." *The R Journal* 10 (1): 439–46. https://doi.org/10.32614/RJ-2018-009.

Pebesma, Edzer, and Roger Bivand. 2023. *Spatial Data Science: With applications in R.* Chapman and Hall/CRC. https://doi.org/10.1201/9780429459016.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. *Dplyr: A Grammar of Data Manipulation.* https://CRAN.R-project.org/package=dplyr.