Package 'midfieldr'

May 17, 2024

Title Tools and Methods for Working with MIDFIELD Data in 'R'

Version 1.0.2

Description Provides tools and demonstrates methods for working with individual undergraduate student-level records (registrar's data) in 'R'. Tools include filters for program codes, data sufficiency, and timely completion. Methods include gathering blocs of records, computing quantitative metrics such as graduation rate, and creating charts to visualize comparisons. 'midfieldr' interacts with practice data provided in 'midfielddata', an R data package available at <`https://midfieldr.github.io/midfielddata/`>. 'midfieldr' also interacts with the full MIDFIELD database for users who have access. This work is supported by the US National Science Foundation through grant numbers 1545667 and 2142087.

Maintainer Richard Layton <`graphdoctor@gmail.com`>

Additional_repositories `http://midfieldr.github.io/drat`

BugReports `https://github.com/MIDFIELDR/midfieldr/issues`

URL `https://midfieldr.github.io/midfieldr/`

RoxygenNote 7.3.1

LazyDataCompression bzip2

LazyData TRUE

VignetteBuilder knitr, rmarkdown

Depends R (>= 3.5.0)

Encoding UTF-8

Language en-US

Imports checkmate, data.table (>= 1.9.8), stats, wrapr

Suggests ggplot2, here, kableExtra, knitr, mice, midfielddata (>= 0.2.0), rmarkdown, tinytest

License MIT + file LICENSE

NeedsCompilation no

Author Richard Layton [cre, aut, cph],
Russell Long [aut, cph, dtm],
Matthew Ohland [aut, cph],
Marisa Orr [aut, cph],
Susan Lord [aut, cph]

Repository CRAN

Date/Publication 2024-05-16 22:50:02 UTC

R topics documented:

`add_completion_status` *Determine completion status for every student*

Description

Add columns to a data frame of student-level records that indicate whether a student completed a degree, and if so, whether their completion was timely.

Usage

```
add_completion_status(dframe, midfield_degree = degree)
```

Arguments

`dframe`	Working data frame of student-level records to which completion-status columns are to be added. Required variables are `mcid` and `timely_term`. See also `add_timely_term()`.
`midfield_degree`	MIDFIELD `degree` data table or equivalent with required variables `mcid` and `term_degree`.

Details

By "completion" we mean an undergraduate earning their first baccalaureate degree (or degrees, for students earning more than one degree in the same term). Additional degrees, if any, earned later than the term of the first degree are ignored.

In many studies, students must complete a degree in a specified time span, for example 4-, 6-, or 8-years after admission. If they do, their completion is timely; if not, their completion is late and they are grouped with the non-completers when computing a metric such as graduation rate.

Completion status is "timely" for students completing a degree no later than their timely completion terms. See also `add_timely_term()`.

Value

A data frame in `data.table` format with the following properties: rows are preserved; columns are preserved with the exception that columns added by the function overwrite existing columns of the same name (if any); grouping structures are not preserved. The added columns are:

`term_degree` Character. Term in which the first degree(s) are completed. Encoded YYYYT. Joined from `midfield_degree` data table.

`completion_status` Character. Label each observation to indicate completion status. Possible values are: "timely", indicating completion no later than the timely completion term; "late", indicating completion after the timely completion term; and "NA" indicating non-completion.

See Also

Other add_*: `add_data_sufficiency()`, `add_timely_term()`

Examples

```
# Start with an excerpt from the student data set
dframe <- toy_student[1:10, .(mcid)]

# Timely term column is required to add completion status column
dframe <- add_timely_term(dframe, toy_term)

# Add completion status column
add_completion_status(dframe, toy_degree)

# Existing completion_status column, if any, is overwritten
dframe[, completion_status := NA_character_]
add_completion_status(dframe, toy_degree)
```

`add_data_sufficiency` *Determine data sufficiency for every student*

Description

Add a column to a data frame of student-level records that labels each row for inclusion or exclusion based on data sufficiency near the upper and lower bounds of an institution's data range.

Usage

```
add_data_sufficiency(dframe, midfield_term = term)
```

Arguments

`dframe`	Working data frame of student-level records to which data-sufficiency columns are to be added. Required variables are `mcid` and `timely_term`. See also `add_timely_term()`.
`midfield_term`	MIDFIELD `term` data table or equivalent with required variables `mcid`, `institution`, and `term`.

Details

The time span of MIDFIELD term data varies by institution, each having their own lower and upper bounds. For some student records, being at or near these bounds creates unavoidable ambiguity when trying to assess degree completion. Such records must be identified and in most cases excluded to prevent false summary counts.

Value

A data frame in `data.table` format with the following properties: rows are preserved; columns are preserved with the exception that columns added by the function overwrite existing columns of the same name (if any); grouping structures are not preserved. The added columns are:

`term_i` Character. Initial term of a student's longitudinal record, encoded YYYYT. Not overwritten if present in `dframe`.

`lower_limit` Character. Initial term of an institution's data range, encoded YYYYT

`upper_limit` Character. Final term of an institution's data range, encoded YYYYT

`data_sufficiency` Character. Label each observation for inclusion or exclusion based on data sufficiency. Possible values are: `include`, indicating that available data are sufficient for estimating timely completion; `exclude-upper`, indicating that data are insufficient at the upper limit of a data range; and `exclude-lower`, indicating that data are insufficient at the lower limit.

See Also

Other add_*: `add_completion_status()`, `add_timely_term()`

Examples

```
# Start with an excerpt from the student data set
dframe <- toy_student[1:10, .(mcid)]

# Timely term column is required to add data sufficiency column
dframe <- add_timely_term(dframe, midfield_term = toy_term)

# Add data sufficiency column
add_data_sufficiency(dframe, midfield_term = toy_term)

# Existing data_sufficiency column, if any, is overwritten
dframe[, data_sufficiency := NA_character_]
add_data_sufficiency(dframe, midfield_term = toy_term)
```

---

add_timely_term *Calculate a timely completion term for every student*

Description

Add a column to a data frame of student-level records that indicates the latest term by which degree completion would be considered timely for every student.

Usage

```
add_timely_term(
  dframe,
  midfield_term = term,
  ...,
  span = NULL,
  sched_span = NULL
)
```

Arguments

`dframe`	Working data frame of student-level records to which timely-term columns are to be added. Required variable is `mcid`.
`midfield_term`	MIDFIELD `term` data table or equivalent with required variables `mcid`, `term`, and `level`.
`...`	Not used for passing values; forces subsequent arguments to be referable only by name.
`span`	Optional integer scalar, number of years to define timely completion. Commonly used values are are 100, 150, and 200 percent of `sched_span`. Default 6 years.
`sched_span`	Optional integer scalar, the number of years an institution officially schedules for completing a program. Default 4 years.

Details

By "completion" we mean an undergraduate earning their first baccalaureate degree (or degrees, for students earning more than one degree in the same term).

In many studies, students must complete their programs in a specified time span, for example 4-, 6-, or 8-years after admission. If they do, their completion is timely; if not, their completion is late and they are grouped with the non-completers when computing a metric such as graduation rate.

Our heuristic assigns `span` number of years (default is 6 years) to every student. For students admitted at second-year level or higher, the span is reduced by one year for each full year the student is assumed to have completed. For example, a student admitted at the second-year level is assumed to have completed one year of a program, so their span is reduced by one year.

The adjusted span is added to the initial term to create the timely completion term in the `timely_term` column.

Value

A data frame in `data.table` format with the following properties: rows are preserved; columns are preserved with the exception that columns added by the function overwrite existing columns of the same name (if any); grouping structures are not preserved. The added columns are:

`term_i` Character. Initial term of a student's longitudinal record, encoded YYYYT

`level_i` Character. Student level (01 Freshman, 02 Sophomore, etc.) in their initial term

`adj_span` Numeric. Integer span of years for timely completion adjusted for a student's initial level.

`timely_term` Character. Latest term by which program completion would be considered timely for every student. Encoded YYYYT.

See Also

Other add_*: `add_completion_status`(), `add_data_sufficiency`()

Examples

```
# Start with an excerpt from the student data set
dframe <- toy_student[1:10, .(mcid)]

# Add timely completion term column
add_timely_term(dframe, midfield_term = toy_term)

# Define timely completion as 200% of scheduled span (8 years)
add_timely_term(dframe, midfield_term = toy_term, span = 8)

# Existing timely_term column, if any, is overwritten
dframe[, timely_term := NA_character_]
add_timely_term(dframe, midfield_term = toy_term)
```

`baseline_mcid` *Baseline ID bloc to start a typical analysis*

Description

Data frame of IDs after processing the practice data for data sufficiency and degree seeking. Provides a convenient bloc to start many of the analysis illustrated in the package articles.

Usage

```
baseline_mcid
```

Format

`data.table` with 76875 rows and 1 column:

`mcid` Character, de-identified student ID.

See Also

Other case-study-data: `study_observations`, `study_programs`, `study_results`

`cip` *Table of academic programs*

Description

A data table based on the US National Center for Education Statistics (NCES), Integrated Postsecondary Education Data System (IPEDS), 2010 CIP, `http://nces.ed.gov/ipeds/cipcode/`. The data are codes and names for 1582 instructional programs organized on three levels: a 2-digit series, a 4-digit series, and a 6-digit series.

Usage

```
cip
```

Format

`data.table` with 1582 rows and 6 columns keyed by the 6-digit CIP code:

`cip6` Character 6-digit code representing "specific instructional programs" (US National Center for Education Statistics).

`cip6name` Character program name at the 6-digit level

`cip4` Character 4-digit code (the first 4 digits of `cip6`) representing "intermediate groupings of programs that have comparable content and objectives."

`cip4name` Character program name at the 4-digit level.

`cip2` Character 2-digit code (the first 2 digits of `cip6`) representing "the most general groupings of related programs."

`cip2name` Character program name at the 2-digit level.

Details

The midfielddata taxonomy includes one non-IPEDS code (999999) for Undecided or Unspecified, instances in which institutions reported no program information or that students were not enrolled in a program.

The MIDFIELD research database include CIPs for undergraduate pre-majors such as pre-med (511102), pre-law (220001), and pre-vet (511104).

See Also

Other cip-data: `fye_proxy`

`filter_cip` *Subset rows that include matches to search strings*

Description

Subset a CIP data frame, retaining rows that match or partially match a vector of character strings. Columns are not subset unless selected in an optional argument.

Usage

```
filter_cip(keep_text = NULL, ..., drop_text = NULL, cip = NULL, select = NULL)
```

Arguments

`keep_text`	Character vector of search text for retaining rows, not case-sensitive. Can be empty if `drop_text` is used.
`...`	Not used for passing values; forces subsequent arguments to be referable only by name.
`drop_text`	Optional character vector of search text for dropping rows, default NULL.
`cip`	Data frame to be searched. Default `cip`.
`select`	Optional character vector of column names to return, default all columns.

Details

Search terms can include regular expressions. Uses `grepl()`, therefore non-character columns (if any) that can be coerced to character are also searched for matches. Columns are subset by the values in `select` after the search concludes.

If none of the optional arguments are specified, the function returns the original data frame.

Value

A data frame in `data.table` format, a subset of `cip`, with the following properties: exclude rows that match elements of `drop_text`; of the remaining rows, include those that match elements of `keep_text`; if `select` is empty, all columns are preserved, otherwise only columns included in `select` are retained; grouping structures are not preserved.

Examples

```
# Subset using keywords
filter_cip(keep_text = "engineering")


    # Multiple passes to narrow the results
    first_pass <- filter_cip("civil")
    second_pass <- filter_cip("engineering", cip = first_pass)
    filter_cip(drop_text = "technology", cip = second_pass)

    # drop_text argument, when used, must be named
    filter_cip("civil engineering", drop_text = "technology")

    # Subset using numerical codes
    filter_cip(keep_text = c("050125", "160501"))

    # Subset using regular expressions
    filter_cip(keep_text = "^54")
    filter_cip(keep_text = c("^1407", "^1408"))

    # Select columns
    filter_cip(keep_text = "^54", select = c("cip6", "cip4name"))
```

`fye_proxy` *Starting program proxies for FYE students*

Description

Proxies are the degree-granting engineering programs we estimate that First-Year Engineering (FYE) students would have declared had they not been required to enroll in FYE. Keyed by student ID. Proxies are provided for all students in the midfielddata practice data who enroll in FYE in their first term.

Usage

```
fye_proxy
```

Format

`data.table` with 4623 rows and 2 columns keyed by student ID:

`mcid` Character, de-identified student ID.

`proxy` Character, 6-digit CIP code of the estimated proxy program.

Details

The proxy variable contains 6-digit CIP codes of degree-granting engineering programs, e.g., Electrical Engineering, Mechanical Engineering, etc., that are substituted for the FYE CIP code when an analysis requires degree-granting starting programs. The most common application is a graduation rate calculation.

The estimation is based on students' first post-FYE programs and a multiple imputation suitable for categorical variables using the mice package. The predictor variables are institution, race, and sex. The estimated variable is the 6-digit CIP code of a degree-granting engineering program at their institution.

`fye_proxy` holds only for the practice data in midfielddata—these values cannot be commingled with the MIDFIELD research database.

See Also

Other cip-data: `cip`

`order_multiway` *Order categorical variables of multiway data*

Description

Transform a data frame such that two independent categorical variables are factors with levels ordered for display in a multiway dot plot. Multiway data comprise a single quantitative value (or response) for every combination of levels of two categorical variables. The ordering of the rows and panels is crucial to the perception of effects (Cleveland, 1993).

Usage

```
order_multiway(
  dframe,
  quantity,
  categories,
  ...,
  method = NULL,
  ratio_of = NULL
)
```

Arguments

`dframe`	Data frame containing a single quantitative value (or response) for every combination of levels of two categorical variables. Categories may be class character or factor. Two additional numeric columns are required when using the "percent" ordering method.
`quantity`	Character, name (in quotes) of the single multiway quantitative variable
`categories`	Character, vector of names (in quotes) of the two multiway categorical variables
`...`	Not used for passing values; forces subsequent arguments to be referable only by name.
`method`	Character, “median” (default) or “percent”, method of ordering the levels of the categories. The median method computes the medians of the quantitative column grouped by category. The percent method computes percentages based on the same ratio underlying the quantitative percentage variable except grouped by category.
`ratio_of`	Character vector with the names (in quotes) of the numerator and denominator columns that produced the quantitative variable, required when `method` is "percent". Names can be in any order; the algorithm assumes that the parameter with the larger column sum is the denominator of the ratio.

Details

In our context, "multiway" refers to the data structure and graph design defined by Cleveland (1993), not to the methods of analysis described by Kroonenberg (2008).

Multiway data comprise three variables: a categorical variable of *m* levels; a second independent categorical variable of *n* levels; and a quantitative variable (or *response*) of length *mn* that cross-classifies the categories, that is, there is a value of the response for each combination of levels of the two categorical variables.

In a multiway dot plot, one category is encoded by the panels, the second category is encoded by the rows of each panel, and the quantitative variable is encoded along identical horizontal scales.

Value

A data frame in `data.table` format with the following properties: rows are preserved; columns specified by `categories` are converted to factors and ordered; the column specified by `quantity` is converted to type double; other columns are preserved with the exception that columns added by the function overwrite existing columns of the same name (if any); grouping structures are not preserved. The added columns are:

`CATEGORY_median` **columns (when ordering method is "median")** Numeric. Two columns of medians of the quantitative variable grouped by the categorical variables. The `CATEGORY` placeholder in the column name is replaced by a category name from the `categories` argument. For example, suppose `categories = c("program", "people")` and `method = "median"`. The two new column names would be `program_median` and `people_median`.

`CATEGORY_QUANTITY` **columns (when ordering method is "percent")** Numeric. Two columns of percentages based on the same ratio that produces the quantitative variable except grouped by the categorical variables. The `CATEGORY` placeholder in the column name is replaced by a category name from the categories argument; the QUANTITY placeholder is replaced by the quantitative variable name in the quantity argument. For example, suppose categories = c("program", "people"), and quantity = "grad_rate", and method = "percent". The two new column names would be program_grad_rate and people_grad_rate.

References

Cleveland WS (1993). *Visualizing Data*. Hobart Press, Summit, NJ.

Kroonenberg PM (2008). *Applied Multiway Data Analysis*. Wiley, Hoboken, NJ.

Examples

```
# Subset of built-in data set
dframe <- study_results[program == "EE" | program == "ME"]
dframe[, people := paste(race, sex)]
dframe[, c("race", "sex") := NULL]
data.table::setcolorder(dframe, c("program", "people"))

# Class before ordering
class(dframe$program)
class(dframe$people)

# Class and levels after ordering
mw1 <- order_multiway(dframe,
                      quantity = "stickiness",
                      categories = c("program", "people"))
class(mw1$program)
levels(mw1$program)
class(mw1$people)
levels(mw1$people)

# Display category medians
mw1

# Existing factors (if any) are re-ordered
mw2 <- dframe
mw2$program <- factor(mw2$program, levels = c("ME", "EE"))

# Levels before conditioning
levels(mw2$program)

# Levels after conditioning
mw2 <- order_multiway(dframe,
                      quantity = "stickiness",
                      categories = c("program", "people"))
levels(mw2$program)

# Ordering using percent method
order_multiway(dframe,
               quantity = "stickiness",
               categories = c("program", "people"),
               method = "percent",
                ratio_of = c("graduates", "ever_enrolled"))
```

prep_fye_mice *Prepare FYE data for multiple imputation*

Description

Constructs a data frame of student-level records of First-Year Engineering (FYE) programs and conditions the data for later use as an input to the mice R package for multiple imputation. Sets up three variables as predictors (institution, race/ethnicity, and sex) and one variable to be estimated (program CIP code).

Usage

```
prep_fye_mice(
  midfield_student = student,
  midfield_term = term,
  ...,
  fye_codes = NULL
)
```

Arguments

`midfield_student`	MIDFIELD `student` data table or equivalent with required variables `mcid`, `race`, and `sex`.
`midfield_term`	MIDFIELD `term` data table or equivalent with required variables `mcid`, `institution`, `term`, and `cip6`.
`...`	Not used for passing values; forces subsequent arguments to be referable only by name.
`fye_codes`	Optional character vector of 6-digit CIP codes to identify FYE programs, default "140102". Codes must be 6-digit strings of numbers; regular expressions are prohibited. Non-engineering codes—those that do not start with 14—produce an error.

Details

At some US institutions, engineering students are required to complete a First-Year Engineering (FYE) program as a prerequisite for declaring an engineering major. Administratively, degree-granting engineering programs such as Electrical Engineering or Mechanical Engineering treat their incoming post-FYE students as their "starting" cohorts. However, when computing a metric that requires a count of starters—graduation rate, for example—FYE records must be treated with special care to avoid a miscount.

To illustrate the potential for miscounting starters, suppose we wish to calculate a Mechanical Engineering (ME) graduation rate. Students starting in ME constitute the starting pool and the fraction of that pool graduating in ME is the graduation rate. At FYE institutions, an ME program would typically define their starting pool as the post-FYE cohort entering their program. This may be the best information available, but it invariably undercounts starters by failing to account for FYE students who do not transition (post-FYE) to degree-granting engineering programs—students who may have left the institution or switched to non-engineering majors. In either case, in the absence of the FYE requirement, some of these students would have been ME starters. By neglecting these students, the count of ME starters is artificially low resulting in an ME graduation rate that is artificially high. The same is true for every degree-granting engineering discipline in an FYE institution.

Therefore, to avoid miscounting starters at FYE institutions, we have to estimate an "FYE proxy", that is, the 6-digit CIP codes of the degree-granting engineering programs that FYE students would have declared had they not been required to enroll in FYE. The purpose of ‘prep_fye_mice()“ is to prepare the data for making that estimation.

After running `prep_fye_mice()` but before running `mice()`, one can edit variables or add variables to create a custom set of predictors. The mice package expects all predictors and the proxy variables to be factors. Do not delete the institution variable because it ensures that a student’s imputed program is available at their institution.

In addition, ensure that the only missing values are in the proxy column. Other variables are expected to be complete (no NA values). A value of "unknown" in a predictor column, e.g., race/ethnicity or sex, is an acceptable value, not missing data. Observations with missing or unknown values in the ID or institution columns (if any) should be removed.

Value

A data frame in `data.table` format conditioned for later use as an input to the mice R package for multiple imputation. The data frame comprises one row for every FYE student, first-term and migrator. Grouping structures are not preserved. The columns returned are:

`mcid` Character, anonymized student identifier. Returned as-is.

`race` Factor, race/ethnicity as self-reported by the student. An imputation predictor variable.

`sex` Factor, sex as self-reported by the student. An imputation predictor variable.

`institution` Factor, anonymized institution name. An imputation predictor variable.

`proxy` Factor, 6-digit CIP code of a student’s known, post-FYE engineering program or NA representing missing values to be imputed.

Method

The function extracts all terms for all FYE students, including those who migrate to enter Engineering after their first term, and identifies the first post-FYE program in which they enroll, if any. This treatment yields two possible outcomes for values returned in the `proxy` column:

1. The student completes FYE and enrolls in an engineering major. For this outcome, we know that at the student’s first opportunity, they enrolled in an engineering program of their choosing. The CIP code of that program is returned as the student’s FYE proxy.
2. The student does not enroll post-FYE in an engineering major. Such students have no further records in the database or switched from Engineering to another program. For this outcome, the data provide no information regarding what engineering program the student would have declared originally had the institution not required them to enroll in FYE. For these students a proxy value of NA is returned. These are the data treated as missing values to be imputed by `mice()`.

In cases where students enter FYE, change programs, and re-enter FYE, only the first group of FYE terms is considered. Any programs before FYE are ignored.

The resulting data frame is ready for use as input for the mice package, with all variables except `mcid` returned as factors.

Examples

```
# Using toy data
prep_fye_mice(toy_student, toy_term)

# Other columns, if any, are dropped
colnames(toy_student)
colnames(prep_fye_mice(toy_student, toy_term))

# Optional argument permits multiple CIP codes for FYE
prep_fye_mice(midfield_student = toy_student,
              midfield_term =toy_term,
              fye_codes = c("140101", "140102"))
```

`same_content` *Test for equal content between two data tables*

Description

Test of data equality between data.table objects. Convenience function used in 'midfieldr' articles.

Usage

```
same_content(x, y)
```

Arguments

`x`	Data frame to be compared. Same as `target` argument in `all.equal()`
`y`	Data frame to be compared. Same as `current` argument in `all.equal()`

Details

Wrapper around `all.equal()` for class data.table that ignores row order, column order, and data.table keys. Both inputs must be date frames. Equivalent to:

`all.equal(target, current, ignore.row.order = TRUE, ignore.col.order = TRUE)`

Value

Either TRUE or a description of the differences between `x` and `y`.

Examples

```
# Same information and ignore row order, column order
x <- toy_student[order(mcid), .(mcid, institution)]
y <- toy_student[order(institution), .(institution, mcid)]
same_content(x, y)

# Different number of rows
x <- toy_student[1:10]
y <- toy_student[1:11]
same_content(x, y)

# Different column names
x <- toy_student[, .(mcid)]
y <- toy_student[, .(institution)]
same_content(x, y)

# Different number of columns and column names
x <- toy_student[, .(mcid)]
y <- toy_student[, .(mcid, institution)]
same_content(x, y)

# Different number of rows, number of columns, and column names
x <- toy_student
y <- toy_term
same_content(x, y)

# Different row content
x <- toy_student[1:10]
y <- toy_student[2:11]
same_content(x, y)
```

`select_required` *Select required midfieldr variables*

Description

Subset a data frame, selecting columns by matching or partially matching a vector of character strings. A convenience function to reduce the dimensions of a MIDFIELD data table at the start of a session by selecting only those columns typically required by other midfieldr functions. Particularly useful in interactive sessions when viewing the data tables at various stages of an analysis.

Usage

```
select_required(midfield_x, ..., select_add = NULL)
```

Arguments

`midfield_x`	Data frame from which columns are selected, typically `student`, `term`, `degree` or their subsets.

`...`	Not used for passing values; forces subsequent arguments to be referable only by name.
`select_add`	Character vector of column names to be added (optionally) to the default `c("mcid", "institution", "race", "sex", "^term", "cip6", "level")`.

Details

Several midfieldr functions are designed to operate on one or more of the MIDFIELD data tables, usually `student`, `term`, or `degree`. This family of functions requires only a small subset of available variables: the required columns built in to the function are `mcid`, `institution`, `race`, `sex`, `^term`, `cip6`, and `level`. Additional column names or partial names can be included by using the `select_add` argument.

The column names of `midfield_x` are searched for matches or partial matches using `grep()`, thus search terms can include regular expressions. Variables with names that match or partially match the search terms are returned; all other columns are dropped. Rows are unaffected. Search terms not present are silently ignored.

One could use this function to select columns from a non-MIDFIELD data frame, but with no benefit to the user—conventional column selection syntax is better suited to that task. Here, we specialize the column selection to serve midfieldr functions.

Value

A data frame in `data.table` format with the following properties: rows are preserved; columns with names that match or partially match values in `select_add` are retained; grouping structures are not preserved.

Examples

```
# Default character vector for selecting columns
default_cols<- c("mcid", "institution", "race", "sex", "^term", "cip6", "level")

# Create one string separated by OR
search_pattern <- paste(default_cols, collapse = "|")

# Find names of columns matching or partially matching
x <- select_required(toy_student)
names(x)
grepl(search_pattern, names(x))

x <- select_required(toy_term)
names(x)
grepl(search_pattern, names(x))

x <- select_required(toy_degree)
names(x)
grepl(search_pattern, names(x))

x <- select_required(toy_course)
names(x)
grepl(search_pattern, names(x))
```

```
# Adding search terms
x <- select_required(toy_course, select_add = c("abbrev", "number", "grade"))
names(x)
```

`study_observations` *Case-study observations*

Description

Data table of post-processed observations of students ever enrolled in, and students graduating from, the four programs of the case study. Keyed by student ID. Provided for the convenience of vignette users.

Usage

```
study_observations
```

Format

`data.table` with 8917 rows and 5 columns. The variables are:

`mcid` Character, de-identified student ID.

`race` Character, race/ethnicity as self-reported by the student, e.g., Asian, Black, Latine, etc.

`sex` Character, sex as self-reported by the student, possible values are Female, Male, and Unknown.

`program` Character, academic program label.

`bloc` Character, indicating the grouping (`ever_enrolled` or `graduates`) to which an observation belongs.

Details

Starting with the case-study starting pool of students ever enrolled in the four programs of the study (Civil, Electrical, Industrial/Systems, and Mechanical Engineering), we filtered the data for data sufficiency, degree seeking, program, and timely completion.

A data frame of "ever enrolled" and a data frame of "timely graduates" were bound using shared column names and are distinguished in the `bloc` variable. This data structure facilitates grouping and summarizing by race, sex, program, and group.

See Also

Other case-study-data: `baseline_mcid`, `study_programs`, `study_results`

`study_programs` *Case-study program labels and codes*

Description

Data table of program CIP codes and labels of the four programs of the case study. Keyed by 6-digit CIPs. Provided for the convenience of vignette users.

Usage

```
study_programs
```

Format

`data.table` with 15 rows and 2 columns. The variables are:

`cip6` Character, 6-digit CIP code of program in which a student is enrolled in a term.

`program` Character, abbreviated labels for four engineering programs. Values are "CE" (Civil Engineering), "EE" (Electrical Engineering), "ISE" (Industrial/Systems Engineering), and "ME" (Mechanical Engineering).

Details

Starting with the midfieldr `cip` data set, we extracted the CIPs of the four programs of the case study and assigned them a custom label to be used for grouping and summarizing.

See Also

Other case-study-data: `baseline_mcid`, `study_observations`, `study_results`

`study_results` *Case-study results*

Description

Data table of longitudinal stickiness for the four programs of the case study (Civil, Electrical, Industrial/Systems, and Mechanical Engineering) grouped by program, race/ethnicity, and sex. Provided for the convenience of vignette users.

Usage

```
study_results
```

Format

`data.table` with 50 rows and 6 columns:

`program` Character, academic program label.

`sex` Character, sex as self-reported by the student, possible values are Female, Male, and Unknown.

`race` Character, race/ethnicity as self-reported by the student, e.g., Asian, Black, Latine, etc.

`ever_enrolled` Numerical, number of students ever enrolled in a program.

`graduates` Numerical, number of students completing a program.

`stickiness` Numerical, program stickiness, the ratio of `graduates` to `ever_enrolled`, in percent.

Details

Longitudinal stickiness is the ratio of the number of students graduating from a program to the number of students ever enrolled in the program over the time span of available data. Results are based on data that have been filtered for data sufficiency, degree seeking, and timely completion.

See Also

Other case-study-data: `baseline_mcid`, `study_observations`, `study_programs`

`toy_course` *Course data for examples*

Description

Selected variables modeled on those in the `course` practice data for use in package examples and articles. Sampled from an early version of the practice data, the toy data are not a current practice data sample.

Usage

```
toy_course
```

Format

`data.table` with 4616 rows and 6 columns keyed by student ID:

`mcid` Character, de-identified student ID.

`institution` Character, de-identified institution name, e.g., Institution A, Institution B, etc.

`term` Character, academic year and term, format YYYYT.

`abbrev` Character, course alphabetical identifier, e.g. ENGR, MATH, ENGL.

`number` Character, course numeric identifier, e.g. 101, 3429.

`grade` Character, course grade, e.g., A+, A, A-, B+, I, NG, etc.

See Also

Other toy-data: `toy_degree`, `toy_student`, `toy_term`

toy_degree *Degree data for examples*

Description

Selected variables modeled on those in the `degree` practice data for use in package examples and articles. Sampled from an early version of the practice data, the toy data are not a current practice data sample.

Usage

```
toy_degree
```

Format

`data.table` with 65 rows and 4 columns keyed by student ID. The variables are:

`mcid` Character, de-identified student ID.

`institution` Character, de-identified institution name, e.g., Institution A, Institution B, etc.

`term_degree` Character, academic year and term in which a student completes their program, format YYYYT.

`cip6` Character, 6-digit CIP code of program in which a student is enrolled in a term.

See Also

Other toy-data: `toy_course`, `toy_student`, `toy_term`

toy_student *Student data for examples*

Description

Selected variables modeled on those in the `student` practice data for use in package examples and articles. Sampled from an early version of the practice data, the toy data are not a current practice data sample.

Usage

```
toy_student
```

Format

`data.table` with 99 rows and 4 columns keyed by student ID:

`mcid` Character, de-identified student ID.

`institution` Character, de-identified institution name, e.g., Institution A, Institution B, etc.

`race` Character, race/ethnicity as self-reported by the student, e.g., Asian, Black, Latine, etc.

`sex` Character, sex as self-reported by the student, possible values are Female, Male, and Unknown.

See Also

Other toy-data: `toy_course`, `toy_degree`, `toy_term`

`toy_term` *Term data for examples*

Description

Selected variables modeled on those in the `term` practice data for use in package examples and articles. Sampled from an early version of the practice data, the toy data are not a current practice data sample.

Usage

```
toy_term
```

Format

`data.table` with 150 rows and 5 columns keyed by student ID. The variables are:

`mcid` Character, de-identified student ID.

`institution` Character, de-identified institution name, e.g., Institution A, Institution B, etc.

`term` Character, academic year and term, format YYYYT.

`cip6` Character, 6-digit CIP code of program in which a student is enrolled in a term.

`level` Character, 01 Freshman, 02 Sophomore, etc. The equivalent values in the current practice data are 01 First-Year, 02-Second Year, etc.

See Also

Other toy-data: `toy_course`, `toy_degree`, `toy_student`

Index