Package: panelaggregation (via r-universe)

September 15, 2024

Title Aggregate Longitudinal Survey Data

Description Aggregate Business Tendency Survey Data (and other qualitative surveys) to time series at various aggregation levels. Run aggregation of survey data in a speedy, re-traceable and a easily deployable way. Aggregation is substantially accelerated by use of data.table. This package intends to provide an interface that is less general and abstract than data.table but rather geared towards survey researchers.

Version 0.1.1

Maintainer Matthias Bannert <bannert@kof.ethz.ch>

Depends R (>= 3.0.0), data.table (>= 1.9.4)

License GPL-2

LazyData true

RoxygenNote 5.0.1

NeedsCompilation no

Author Matthias Bannert [aut, cre], Gabriel Bucur [aut]

Date/Publication 2017-01-07 02:40:46

Repository https://will-the-wiz.r-universe.dev

RemoteUrl https://github.com/cran/panelaggregation

RemoteRef HEAD

RemoteSha 594f3e74f06e4a1b65bf930568ead7a8cabb3122

Contents

`btsdemo` *Randomly Generated Panel Dataset*

Description

This data was created by simulation to mimmick a firm level dataset stemming from business tendency surveys. The data was simulated because of privacy concerns with micro level firm data. For convenience the dataset contains two different date notations. Also 5 qualitative 3-item questions are included. Business tendency survey data is often weighted with company size represented by the number of employees. Thus the weight column is quantitative and its distribution is somewhat (!) reasonable with respect to the distribution of employees in a typical firm sample.

Format

A data frame with 27000 rows and 13 variables

Details

- uid unique company identifier
- year numeric year column
- weight quantitative weight
- question_1
- question_2
- question_3
- question_4
- question_5
- group group to mimmick different sectors / branches of trade
- altGroup another alternative grouping columns
- sClass a column denoting discrete size classes small (S), medium (M) and large (L)
- date_qtrly quarterly dates stored in a single column.

Author(s)

Matthias Bannert

Source

Randomly generated in R using the sample generator from https://github.com/mbannert/gateveys/blob/master/R/gateveys.R

`computeBalance` *Compute Balances from a Item Shares*

Description

This function computes balances (i.e. positive - negative items), from item shares stored in a wide format data.table.

Usage

```
computeBalance(data_table, multipliers = list(item_pos = 1, item_eq = 0,
  item_neg = -1))
```

Arguments

`data_table`	a data.table in wide format containing item
`multipliers`	list containing multipliers of items, assigned by item and column names

Author(s)

Matthias Bannert, Gabriel Bucu

`computeShares` *Compute Weighted Shares By Group*

Description

This function computes weighted shares from a data.table. computeShares is performance optimized and designed to work well in bulk operations. The function returns a data.table.

Usage

```
computeShares(data_table, variable, weight, by, wide = T)
```

Arguments

`data_table`	a data.table
`variable`	character name of the variable to focus on. The variable must be in the data.table
`weight`	character name of the data.table column that contains a weight.
`by`	character vector of the columns to group by
`wide`	logical if true the result is returned in wide format dcast.

Author(s)

Matthias Bannert, Gabriel Bucur, Oliver Mueller

Examples

```
# TODO: add new weight columns to BTS demo
# load library and dataset
library(panelaggregation)
data(btsdemo)
head(btsdemo)
# adapt the levels to positive, equal and negative
# in order to suit the naming defaults. other levels work too,
# but you'd need to specify multipliers in computeBalance then
levels(btsdemo$question_1) <- c("pos","eq","neg")

# compute the weighted shares and display store in wide format
# to get a basis for further steps
level1 <- computeShares(btsdemo,"question_1","weight",
                        by = c("date_qtrly","group", "altGroup", "sClass"))

# compute balance, don't have to do much here, because
# (pos, eq, neg) is the default for the possible answers
level1_wbalance <- computeBalance(level1)

# Select a particular grouping combination and a timeseries that
# should be extracted from the level 1 aggregation.
ts1 <- extractTimeSeries(level1_wbalance,
                         "date_qtrly",
                         list(group = "C", altGroup = "a", sClass = "S"),
                         freq = 4,
                         item = "balance",
                         variable = "question_1")
ts1
# Plot a standard R ts using the plot method for ts
plot(ts1, main = attributes(ts1)$ts_key)

# Add weight column to the aggregated results
# In order to join the tables, we need to know what weight to assign to each row.
# This is done by having via a common key, for example c('group', 'altGroup').
# In this example we would assign a different weight for each
#   c('group', 'altGroup') combination (e.g. c('A', 'a')).
btsweight1 <- btsdemo[, list(weight = sum(weight)), by = 'group']
btsagg1 <- joinDataTables(level1_wbalance, btsweight1, 'group')

# Compute second level aggregation, this time on fewer columns and using a different set of weights.
level2_balance <- computeWeightedMeans(btsagg1, c('item_pos', 'item_eq', 'item_neg', 'balance'),
                                       'weight', c("date_qtrly","group", "sClass"))

# Select a particular grouping combination and a timeseries that
# should be extracted from the level 2 aggregation.
ts2 <- extractTimeSeries(level2_balance,
                         "date_qtrly",
                         list(group = "C", sClass = "S"),
                         freq = 4,
                         item = "balance",
                         variable = "question_1")
```

```
ts2
# Plot a standard R ts using the plot method for ts
plot(ts2, main = attributes(ts2)$ts_key)

# Add weight column to the aggregated results
# In order to join the tables, we need to know what weight to assign to each row.
# This is done by having via a common key, for example c('group', 'altGroup').
# In this example we would assign a different weight for each
#   c('group', 'altGroup') combination (e.g. c('A', 'a')).
btsweight2 <- btsdemo[, list(weight = sum(weight)), by = 'sClass']
btsagg2 <- joinDataTables(level2_balance, btsweight2, 'sClass')

# Compute third level of aggregation, on the whole sector, using yet another set of weights.
level3_balance <- computeWeightedMeans(btsagg2, 'balance', 'weight', c("date_qtrly", "sClass"))

# Select a particular grouping combination and a timeseries that
# should be extracted from the level 2 aggregation.
ts3 <- extractTimeSeries(level3_balance,
                         "date_qtrly",
                         list(sClass = "S"),
                         freq = 4,
                         item = "balance",
                         variable = "question_1")
ts3
# Plot a standard R ts using the plot method for ts
plot(ts3, main = attributes(ts3)$ts_key)
```

`computeWeightedMeans` *Compute Weighted Mean by Group*

Description

This function computes the weighted mean of variable groups from a data.table. computeWeighted-Mean is performance optimized and designed to work well in bulk operations. The function returns a data.table.

Usage

```
computeWeightedMeans(data_table, variables, weight, by)
```

Arguments

`data_table`	a data.table
`variables`	character name of the variable(s) to focus on. The variables must be in the data.table
`weight`	character name of the data.table column that contains a weight.
`by`	character vector of the columns to group by

Author(s)

Matthias Bannert, Gabriel Bucur

Examples

```
# TODO: add new weight columns to BTS demo
# load library and dataset
library(panelaggregation)
data(btsdemo)
head(btsdemo)
# adapt the levels to positive, equal and negative
# in order to suit the naming defaults. other levels work too,
# but you'd need to specify multipliers in computeBalance then
levels(btsdemo$question_1) <- c("pos","eq","neg")

# compute the weighted shares and display store in wide format
# to get a basis for further steps
level1 <- computeShares(btsdemo,"question_1","weight",
                        by = c("date_qtrly","group", "altGroup", "sClass"))

# compute balance, don't have to do much here, because
# (pos, eq, neg) is the default for the possible answers
level1_wbalance <- computeBalance(level1)

# Select a particular grouping combination and a timeseries that
# should be extracted from the level 1 aggregation.
ts1 <- extractTimeSeries(level1_wbalance,
                         "date_qtrly",
                         list(group = "C", altGroup = "a", sClass = "S"),
                         freq = 4,
                         item = "balance",
                         variable = "question_1")
ts1
# Plot a standard R ts using the plot method for ts
plot(ts1, main = attributes(ts1)$ts_key)

# Add weight column to the aggregated results
# In order to join the tables, we need to know what weight to assign to each row.
# This is done by having via a common key, for example c('group', 'altGroup').
# In this example we would assign a different weight for each
#   c('group', 'altGroup') combination (e.g. c('A', 'a')).
btsweight1 <- btsdemo[, list(weight = sum(weight)), by = 'group']
btsagg1 <- joinDataTables(level1_wbalance, btsweight1, 'group')

# Compute second level aggregation, this time on fewer columns and using a different set of weights.
level2_balance <- computeWeightedMeans(btsagg1, c('item_pos', 'item_eq', 'item_neg', 'balance'),
                                       'weight', c("date_qtrly","group", "sClass"))

# Select a particular grouping combination and a timeseries that
# should be extracted from the level 2 aggregation.
ts2 <- extractTimeSeries(level2_balance,
                         "date_qtrly",
```

```
                          list(group = "C", sClass = "S"),
                          freq = 4,
                          item = "balance",
                          variable = "question_1")
ts2
# Plot a standard R ts using the plot method for ts
plot(ts2, main = attributes(ts2)$ts_key)

# Add weight column to the aggregated results
# In order to join the tables, we need to know what weight to assign to each row.
# This is done by having via a common key, for example c('group', 'altGroup').
# In this example we would assign a different weight for each
#   c('group', 'altGroup') combination (e.g. c('A', 'a')).
btsweight2 <- btsdemo[, list(weight = sum(weight)), by = 'sClass']
btsagg2 <- joinDataTables(level2_balance, btsweight2, 'sClass')

# Compute third level of aggregation, on the whole sector, using yet another set of weights.
level3_balance <- computeWeightedMeans(btsagg2, 'balance', 'weight', c("date_qtrly", "sClass"))

# Select a particular grouping combination and a timeseries that
# should be extracted from the level 2 aggregation.
ts3 <- extractTimeSeries(level3_balance,
                          "date_qtrly",
                          list(sClass = "S"),
                          freq = 4,
                          item = "balance",
                          variable = "question_1")
ts3
# Plot a standard R ts using the plot method for ts
plot(ts3, main = attributes(ts3)$ts_key)
```

`doUniqueCJ` *Performs a Cross Join of Unique combinations*

Description

This function makes use of `CJ` function of the data.table package to perform a cross join. The function makes sure that the combinations are unique and removes NAs before joining. doUniqueCJ is rather not used as a standalone function but inside `computeShares`.

Usage

```
doUniqueCJ(dt, cols)
```

Arguments

`dt`	data.table
`cols`	character vector that denotes names of relevant columns

Author(s)

Matthias Bannert, Gabriel Bucur

---

extractTimeSeries *Extract a Time Series from a Data.table*

Description

This function extracts time series from data.table columns and returns object of class ts.

Usage

```
extractTimeSeries(data_table, time_column, group_list, freq, item, variable,
  prefix = "CH.KOF.IND")
```

Arguments

data_table	a data.table
time_column	character name of the column which contains the time index
group_list	list or NULL
freq	integer value either 4 denoting quarterly frequency or 12 denoting quarterly frequency
item	character name of the column which contains the item that is extracted from the data.table
variable	character name of the variable selected
prefix	character prefix attached to the dynamically generated key string to identify the time series. Recommend key format: ISOcountry.provider.source.aggregationLevel.selectedGroup.variab

Author(s)

Matthias Bannert, Gabriel Bucur

---

joinDataTables *Joins two data.tables based on keys*

Description

This function joins two data.table objects, given a common key, which can have different names in the two tables. In the latter case, the sequence of the names is crucial. Make sure that the key columns match exactly.

Usage

```
joinDataTables(dt_1, dt_2, key_1, key_2 = key_1)
```

Arguments

dt_1	first data.table
dt_2	second data.table
key_1	character vector of key columns for first data.table
key_2	character vector of key columns for second data.table

Value

joined data.table

Author(s)

Matthias Bannert, Gabriel Bucur

Index