

2657 Functions

Ananda Mahto

July 27, 2012

Contents

I	Function Descriptions and Examples	1
	concat.split	3
	What it Does	3
	Arguments	3
	Examples	3
	To Do	6
	References	6
	df.sorter	7
	What it Does	7
	Arguments	7
	Examples	7
	To Do	9
	multi.freq.table	10
	What it Does	10
	Arguments	10
	Examples	10
	References	12
	row.extractor	13
	What it Does	13
	Arguments	13
	Examples	13
	References	14
II	The Functions	15
	Where to Get the Functions	17
	concat.split	18
	df.sorter	20
	multi.freq.table	21
	row.extractor	22

III	Snippets and Tips	25
Snippets		27
Load All Scripts and Data Files From Multiple Directories		27
Convert a List of Data Frames Into Individual Data Frames		27
Tips		29
Batch Convert Factor to Character Variables		29

Part I

Function Descriptions and Examples

concat.split

What it Does

The `concat.split` function takes a column with multiple values, splits the values into a list or into separate columns, and returns a new `data.frame`.

Arguments

- `data`: the source `data.frame`.
- `split.col`: the variable that needs to be split; can be specified either by the column number or the variable name.
- `to.list`: logical; should the split column be returned as a single variable list (named “original-variable.list”) or multiple new variables? If `to.list` is `TRUE`, the `mode` argument is ignored and a list of the original values are returned.
- `mode`: can be either `binary` or `value` (where `binary` is default and it recodes values to 1 or NA).
- `sep`: the character separating each value (defaults to “,”).
- `drop.col`: logical (whether to remove the original variable from the output or not; defaults to `TRUE`).

Examples

First load some data from a CSV stored at [github](#). The URL is an HTTPS, so we need to use `getURL` from `RCurl`.

```
require(RCurl)

## Loading required package: RCurl

## Loading required package: bitops

baseURL = c("https://raw.githubusercontent.com/mrdwab/2657-R-Functions/master/")
temp = getURL(paste0(baseURL, "data/concatenated-cells.csv"))
concat.test = read.csv(textConnection(temp))
rm(temp)

# How big is the dataset?
dim(concat.test)

## [1] 48  4

# Just show me the first few rows
head(concat.test)
```

	Name	Likes	Siblings	Hates
## 1	Boyd	1,2,4,5,6	Reynolds , Albert , Ortega	2;4;
## 2	Rufus	1,2,4,5,6	Cohen , Bert , Montgomery	1;2;3;4;
## 3	Dana	1,2,4,5,6	Pierce	2;
## 4	Carole	1,2,4,5,6	Colon , Michelle , Ballard	1;4;
## 5	Ramona	1,2,5,6	Snyder , Joann ,	1;2;3;
## 6	Kelley	1,2,5,6	James , Roxanne ,	1;4;

Notice that the data have been entered in a very silly manner. Let's split it up!

```
# Load the function!
# require(RCurl)
# baseURL = c("https://raw.githubusercontent.com/mrdwab/2657-R-Functions/master/")
source(textConnection(getURL(paste0(baseURL, "scripts/concat.split.R"))))
```

```
# Split up the second column, selecting by column number
head(concat.split(concat.test, 2))
```

```
##      Name      Likes      Siblings      Hates Likes_1 Likes_2 Likes_3
## 1  Boyd 1,2,4,5,6 Reynolds , Albert , Ortega      2;4;      1      1      NA
## 2  Rufus 1,2,4,5,6 Cohen , Bert , Montgomery 1;2;3;4;      1      1      NA
## 3   Dana 1,2,4,5,6      Pierce      2;      1      1      NA
## 4 Carole 1,2,4,5,6 Colon , Michelle , Ballard      1;4;      1      1      NA
## 5 Ramona      1,2,5,6      Snyder , Joann ,      1;2;3;      1      1      NA
## 6 Kelley      1,2,5,6      James , Roxanne ,      1;4;      1      1      NA
## Likes_4 Likes_5 Likes_6
## 1      1      1      1
## 2      1      1      1
## 3      1      1      1
## 4      1      1      1
## 5     NA      1      1
## 6     NA      1      1
```

```
# ... or by name, and drop the offensive first column
head(concat.split(concat.test, "Likes", drop.col=TRUE))
```

```
##      Name      Siblings      Hates Likes_1 Likes_2 Likes_3 Likes_4
## 1  Boyd Reynolds , Albert , Ortega      2;4;      1      1      NA      1
## 2  Rufus Cohen , Bert , Montgomery 1;2;3;4;      1      1      NA      1
## 3   Dana      Pierce      2;      1      1      NA      1
## 4 Carole Colon , Michelle , Ballard      1;4;      1      1      NA      1
## 5 Ramona      Snyder , Joann ,      1;2;3;      1      1      NA      NA
## 6 Kelley      James , Roxanne ,      1;4;      1      1      NA      NA
## Likes_5 Likes_6
## 1      1      1
## 2      1      1
## 3      1      1
## 4      1      1
## 5      1      1
## 6      1      1
```

```
# The "Hates" column uses a different separator:
head(concat.split(concat.test, "Hates", sep=";", drop.col=TRUE))
```

```
##      Name      Likes      Siblings Hates_1 Hates_2 Hates_3 Hates_4
## 1  Boyd 1,2,4,5,6 Reynolds , Albert , Ortega      NA      1      NA      1
## 2  Rufus 1,2,4,5,6 Cohen , Bert , Montgomery      1      1      1      1
## 3   Dana 1,2,4,5,6      Pierce      NA      1      NA      NA
## 4 Carole 1,2,4,5,6 Colon , Michelle , Ballard      1      NA      NA      1
## 5 Ramona      1,2,5,6      Snyder , Joann ,      1      1      1      NA
## 6 Kelley      1,2,5,6      James , Roxanne ,      1      NA      NA      1
```

```
# Retain the original values
head(concat.split(concat.test, 2, mode="value", drop.col=TRUE))
```

```
##      Name                Siblings    Hates Likes_1 Likes_2 Likes_3 Likes_4
## 1  Boyd Reynolds , Albert , Ortega    2;4;      1      2      NA      4
## 2  Rufus  Cohen , Bert , Montgomery 1;2;3;4;    1      2      NA      4
## 3   Dana                Pierce        2;      1      2      NA      4
## 4 Carole Colon , Michelle , Ballard    1;4;      1      2      NA      4
## 5 Ramona                Snyder , Joann , 1;2;3;    1      2      NA      NA
## 6 Kelley                James , Roxanne , 1;4;      1      2      NA      NA
## Likes_5 Likes_6
## 1      5      6
## 2      5      6
## 3      5      6
## 4      5      6
## 5      5      6
## 6      5      6
```

```
# Let's try splitting some strings... Same syntax
head(concat.split(concat.test, 3, drop.col=TRUE))
```

```
##      Name    Likes    Hates Siblings_1 Siblings_2 Siblings_3
## 1  Boyd 1,2,4,5,6    2;4; Reynolds    Albert    Ortega
## 2  Rufus 1,2,4,5,6 1;2;3;4;    Cohen      Bert Montgomery
## 3   Dana 1,2,4,5,6      2;    Pierce      <NA>      <NA>
## 4 Carole 1,2,4,5,6    1;4;    Colon    Michelle    Ballard
## 5 Ramona 1,2,5,6    1;2;3;    Snyder      Joann      <NA>
## 6 Kelley 1,2,5,6    1;4;    James    Roxanne      <NA>
```

```
# Split up the "Likes column" into a list variable; retain original column
head(concat.split(concat.test, 2, to.list=TRUE, drop.col=FALSE))
```

```
##      Name    Likes                Siblings    Hates    Likes_list
## 1  Boyd 1,2,4,5,6 Reynolds , Albert , Ortega    2;4; 1, 2, 4, 5, 6
## 2  Rufus 1,2,4,5,6 Cohen , Bert , Montgomery 1;2;3;4; 1, 2, 4, 5, 6
## 3   Dana 1,2,4,5,6                Pierce        2; 1, 2, 4, 5, 6
## 4 Carole 1,2,4,5,6 Colon , Michelle , Ballard    1;4; 1, 2, 4, 5, 6
## 5 Ramona 1,2,5,6                Snyder , Joann , 1;2;3; 1, 2, 5, 6
## 6 Kelley 1,2,5,6                James , Roxanne , 1;4; 1, 2, 5, 6
```

```
# View the structure of the output for the first 10 rows to verify
# that the new column is a list; note the difference between "Likes"
# and "Likes_list".
str(concat.split(concat.test, 2, to.list=TRUE, drop.col=FALSE)[1:10, c(2, 5)])
```

```
## 'data.frame': 10 obs. of 2 variables:
## $ Likes : Factor w/ 5 levels "1,2,3,4,5","1,2,4,5",...: 3 3 3 3 5 5 3 3 3 4
## $ Likes_list:List of 10
## ..$ : num 1 2 4 5 6
## ..$ : num 1 2 4 5 6
## ..$ : num 1 2 4 5 6
## ..$ : num 1 2 4 5 6
## ..$ : num 1 2 5 6
## ..$ : num 1 2 5 6
## ..$ : num 1 2 4 5 6
## ..$ : num 1 2 4 5 6
## ..$ : num 1 2 4 5 6
## ..$ : num 1 2 5
```


To Do

- Modify the function so that you can split multiple columns in one go?

References

See: <http://stackoverflow.com/q/10100887/1270695>

df.sorter

What it Does

The `df.sorter` function allows you to sort a `data.frame` by columns or rows or both. You can also quickly subset data columns by using the `var.order` argument.

Arguments

- **data**: the source `data.frame`.
- **var.order**: the new order in which you want the variables to appear.
 - Defaults to `names(data)`, which keeps the variables in the original order.
 - Variables can be referred to either by a vector of their index numbers or by a vector of the variable name; partial name matching also works, but requires that the partial match identifies similar columns uniquely (see examples).
 - Basic subsetting can also be done using `var.order` simply by omitting the variables you want to drop.
- **col.sort**: the columns *within* which there is data that need to be sorted.
 - Defaults to `NULL`, which means no sorting takes place.
 - Variables can be referred to either by a vector of their index numbers or by a vector of the variable names; full names must be provided.
- **at.start**: Should the pattern matching be from the start of the variable name? Defaults to `"TRUE"`.

NOTE: If you are sorting both by variables and within the columns, the `col.sort` order should be based on the location of the columns in the *new data.frame*, not the original `data.frame`.

Examples

```
# Load the function!
# require(RCurl)
# baseURL = c("https://raw.githubusercontent.com/mrdwab/2657-R-Functions/master/")
source(textConnection(getURL(paste0(baseURL, "scripts/df.sorter.R"))))

# Make up some data
set.seed(1)
dat = data.frame(id = rep(1:5, each=3), times = rep(1:3, 5),
  measure1 = rnorm(15), score1 = sample(300, 15),
  code1 = replicate(15, paste(sample(LETTERS[1:5], 3),
    sep="", collapse="")),
  measure2 = rnorm(15), score2 = sample(150:300, 15),
  code2 = replicate(15, paste(sample(LETTERS[1:5], 3),
    sep="", collapse="")))

# Preview your data
dat
```

	id	times	measure1	score1	code1	measure2	score2	code2
## 1	1	1	-0.6265	145	DAB	-0.7075	299	CEB
## 2	1	2	0.1836	180	DCB	0.3646	224	ECD
## 3	1	3	-0.8356	148	EBA	0.7685	222	DAE

```
## 4 2 1 1.5953 56 AED -0.1123 175 DBA
## 5 2 2 0.3295 245 CEB 0.8811 260 DAC
## 6 2 3 -0.8205 198 EBD 0.3981 216 DCA
## 7 3 1 0.4874 234 BCA -0.6120 300 CEA
## 8 3 2 0.7383 32 CDA 0.3411 179 CAD
## 9 3 3 0.5758 212 EBC -1.1294 182 BEC
## 10 4 1 -0.3054 120 BED 1.4330 234 CDE
## 11 4 2 1.5118 239 EDB 1.9804 231 CAB
## 12 4 3 0.3898 188 DEB -0.3672 160 DBE
## 13 5 1 -0.6212 226 DBA -1.0441 154 EDB
## 14 5 2 -2.2147 159 DAC 0.5697 238 BDE
## 15 5 3 1.1249 152 AED -0.1351 277 DCE
```

```
# Change the variable order, grouping related columns
# Note that you do not need to specify full variable names,
# just enough that the variables can be uniquely identified
head(df.sorter(dat, var.order = c("id", "ti", "cod", "mea", "sco")))
```

```
## id times code1 code2 measure1 measure2 score1 score2
## 1 1 1 DAB CEB -0.6265 -0.7075 145 299
## 2 1 2 DCB ECD 0.1836 0.3646 180 224
## 3 1 3 EBA DAE -0.8356 0.7685 148 222
## 4 2 1 AED DBA 1.5953 -0.1123 56 175
## 5 2 2 CEB DAC 0.3295 0.8811 245 260
## 6 2 3 EBD DCA -0.8205 0.3981 198 216
```

```
# Same output, but with a more awkward syntax
head(df.sorter(dat, var.order = c(1, 2, 5, 8, 3, 6, 4, 7)))
```

```
## id times code1 code2 measure1 measure2 score1 score2
## 1 1 1 DAB CEB -0.6265 -0.7075 145 299
## 2 1 2 DCB ECD 0.1836 0.3646 180 224
## 3 1 3 EBA DAE -0.8356 0.7685 148 222
## 4 2 1 AED DBA 1.5953 -0.1123 56 175
## 5 2 2 CEB DAC 0.3295 0.8811 245 260
## 6 2 3 EBD DCA -0.8205 0.3981 198 216
```

```
# As above, but sorted by 'times' and then 'id'
head(df.sorter(dat, var.order = c("id", "tim", "cod", "mea", "sco"),
  col.sort = c(2, 1)))
```

```
## id times code1 code2 measure1 measure2 score1 score2
## 1 1 1 DAB CEB -0.6265 -0.7075 145 299
## 4 2 1 AED DBA 1.5953 -0.1123 56 175
## 7 3 1 BCA CEA 0.4874 -0.6120 234 300
## 10 4 1 BED CDE -0.3054 1.4330 120 234
## 13 5 1 DBA EDB -0.6212 -1.0441 226 154
## 2 1 2 DCB ECD 0.1836 0.3646 180 224
```

```
# Drop 'measure1' and 'measure2', sort by 'times', and 'score1'
head(df.sorter(dat, var.order = c("id", "tim", "sco", "cod"),
  col.sort = c(2, 3)))
```

```
## id times score1 score2 code1 code2
## 4 2 1 56 175 AED DBA
## 10 4 1 120 234 BED CDE
```

```
## 1 1 1 145 299 DAB CEB
## 13 5 1 226 154 DBA EDB
## 7 3 1 234 300 BCA CEA
## 8 3 2 32 179 CDA CAD
```

As above, but using names

```
head(df.sorter(dat, var.order = c("id", "tim", "sco", "cod"),
  col.sort = c("times", "score1")))
```

```
## id times score1 score2 code1 code2
## 4 2 1 56 175 AED DBA
## 10 4 1 120 234 BED CDE
## 1 1 1 145 299 DAB CEB
## 13 5 1 226 154 DBA EDB
## 7 3 1 234 300 BCA CEA
## 8 3 2 32 179 CDA CAD
```

Just sort by columns, first by 'times' then by 'id'

```
head(df.sorter(dat, col.sort = c("times", "id")))
```

```
## id times measure1 score1 code1 measure2 score2 code2
## 1 1 1 -0.6265 145 DAB -0.7075 299 CEB
## 4 2 1 1.5953 56 AED -0.1123 175 DBA
## 7 3 1 0.4874 234 BCA -0.6120 300 CEA
## 10 4 1 -0.3054 120 BED 1.4330 234 CDE
## 13 5 1 -0.6212 226 DBA -1.0441 154 EDB
## 2 1 2 0.1836 180 DCB 0.3646 224 ECD
```

```
head(df.sorter(dat, col.sort = c("code1"))) # Sorting by character values
```

```
## id times measure1 score1 code1 measure2 score2 code2
## 4 2 1 1.5953 56 AED -0.1123 175 DBA
## 15 5 3 1.1249 152 AED -0.1351 277 DCE
## 7 3 1 0.4874 234 BCA -0.6120 300 CEA
## 10 4 1 -0.3054 120 BED 1.4330 234 CDE
## 8 3 2 0.7383 32 CDA 0.3411 179 CAD
## 5 2 2 0.3295 245 CEB 0.8811 260 DAC
```

Pattern matching anywhere in the variable name

```
head(df.sorter(dat, var.order= "co", at.start=FALSE))
```

```
## code1 code2 score1 score2
## 1 DAB CEB 145 299
## 2 DCB ECD 180 224
## 3 EBA DAE 148 222
## 4 AED DBA 56 175
## 5 CEB DAC 245 260
## 6 EBD DCA 198 216
```

To Do

- Add an option to sort ascending or descending—at the moment, not supported.

multi.freq.table

What it Does

The `multi.freq.table` function takes a data frame containing Boolean responses to multiple response questions and tabulates the number of responses by the possible combinations of answers. In addition to tabulating the frequency (**Freq**), there are two other columns in the output: *Percent of Responses* (**Pct.of.Resp**) and *Percent of Cases* (**Pct.of.Cases**). *Percent of Responses* is the frequency divided by the total number of answers provided; this column should sum to 100%. *Percent of Cases* is the frequency divided by the total number of valid cases; this column would not likely sum to more than 100% since each respondent (case) can select multiple answers.

Arguments

- **data**: The multiple responses that need to be tabulated.
- **sep**: The desired separator for collapsing the combinations of options; defaults to "" (collapsing with no space between each option name).
- **dropzero**: Should combinations with a frequency of zero be dropped from the final table? Defaults to FALSE.
- **clean**: Should the original tabulated data be retained or dropped from the final table? Defaults to TRUE.
- **basic**: Should a basic table of each item, rather than combinations of items, be created? Defaults to FALSE.

Examples

```
# Load the function!
# require(RCurl)
# baseURL = c("https://raw.githubusercontent.com/mrdwab/2657-R-Functions/master/")
source(textConnection(getURL(paste0(baseURL, "scripts/multi.freq.table.R"))))

# Make up some data
set.seed(1)
dat = data.frame(A = sample(c(0, 1), 20, replace=TRUE),
                 B = sample(c(0, 1), 20, replace=TRUE),
                 C = sample(c(0, 1), 20, replace=TRUE),
                 D = sample(c(0, 1), 20, replace=TRUE),
                 E = sample(c(0, 1), 20, replace=TRUE))

# View your data
dat

##      A B C D E
## 1  0 1 1 1 0
## 2  0 0 1 0 1
## 3  1 1 1 0 0
## 4  1 0 1 0 0
## 5  0 0 1 1 1
## 6  1 0 1 0 0
## 7  1 0 0 0 1
## 8  1 0 0 1 0
## 9  1 1 1 0 0
## 10 0 0 1 1 0
## 11 0 0 0 0 0
```

```
## 12 0 1 1 1 0
## 13 1 0 0 0 1
## 14 0 0 0 0 1
## 15 1 1 0 0 1
## 16 0 1 0 1 1
## 17 1 1 0 1 0
## 18 1 0 1 0 0
## 19 0 1 1 1 1
## 20 1 0 0 1 1
```

```
# Apply the function with all defaults accepted
multi.freq.table(dat)
```

##	Combn	Freq	Pct.of.Resp	Pct.of.Cases
## 1		1	2.083	5
## 2	A	0	0.000	0
## 3	B	0	0.000	0
## 4	AB	0	0.000	0
## 5	C	0	0.000	0
## 6	AC	3	6.250	15
## 7	BC	0	0.000	0
## 8	ABC	2	4.167	10
## 9	D	0	0.000	0
## 10	AD	1	2.083	5
## 11	BD	0	0.000	0
## 12	ABD	1	2.083	5
## 13	CD	1	2.083	5
## 14	ACD	0	0.000	0
## 15	BCD	2	4.167	10
## 16	ABCD	0	0.000	0
## 17	E	1	2.083	5
## 18	AE	2	4.167	10
## 19	BE	0	0.000	0
## 20	ABE	1	2.083	5
## 21	CE	1	2.083	5
## 22	ACE	0	0.000	0
## 23	BCE	0	0.000	0
## 24	ABCE	0	0.000	0
## 25	DE	0	0.000	0
## 26	ADE	1	2.083	5
## 27	BDE	1	2.083	5
## 28	ABDE	0	0.000	0
## 29	CDE	1	2.083	5
## 30	ACDE	0	0.000	0
## 31	BCDE	1	2.083	5
## 32	ABCDE	0	0.000	0

```
# Tabulate only on variables "A", "B", and "D", with a different
# separator, dropping any zero frequency values, and keeping the original tabulations.
# Note that there are no solitary "B" responses.
multi.freq.table(dat[c(1, 2, 4)], sep="-", dropzero=TRUE, clean=FALSE)
```

##	A	B	D	Freq	Combn	Pct.of.Resp	Pct.of.Cases
## 1	0	0	0	3		10.714	15
## 2	1	0	0	5	A	17.857	25
## 4	1	1	0	3	A-B	10.714	15
## 5	0	0	1	2	D	7.143	10

```
## 6 1 0 1    2    A-D      7.143      10
## 7 0 1 1    4    B-D     14.286      20
## 8 1 1 1    1 A-B-D      3.571       5
```

```
# View a basic table.
multi.freq.table(dat, basic=TRUE)
```

```
##   Freq Pct.of.Resp Pct.of.Cases
## A   11      22.92         55
## B    8      16.67         40
## C   11      22.92         55
## D    9      18.75         45
## E    9      18.75         45
```

References

apply shortcut for creating the `Combn` column in the output by [Justin](#)

See: <http://stackoverflow.com/q/11348391/1270695> and <http://stackoverflow.com/q/11622660/1270695>

row.extractor

What it Does

The `row.extractor` function takes a `data.frame` and extracts rows with the `min`, `median`, or `max` values of a given variable, or extracts rows with specific quantiles of a given variable.

Arguments

- `data`: the source `data.frame`.
- `extract.by`: the column which will be used as the reference for extraction; can be specified either by the column number or the variable name.
- `what`: options are `min` (for all rows matching the minimum value), `median` (for the median row or rows), `max` (for all rows matching the maximum value), or `all` (for `min`, `median`, and `max`); alternatively, a numeric vector can be specified with the desired quantiles, for instance `c(0, .25, .5, .75, 1)`

Examples

```
# Load the function!
# require(RCurl)
# baseURL = c("https://raw.githubusercontent.com/mrdwab/2657-R-Functions/master/")
source(textConnection(getURL(paste0(baseURL, "scripts/row.extractor.R"))))

# Make up some data
set.seed(1)
dat = data.frame(V1 = 1:50, V2 = rnorm(50),
                 V3 = round(abs(rnorm(50)), digits=2),
                 V4 = sample(1:30, 50, replace=TRUE))

# Get a summary of the data
summary(dat)
```

##	V1	V2	V3	V4
## Min.	: 1.0	Min. : -2.215	Min. : 0.000	Min. : 2.00
## 1st Qu.:	13.2	1st Qu.: -0.372	1st Qu.: 0.347	1st Qu.: 8.25
## Median :	25.5	Median : 0.129	Median : 0.590	Median : 13.00
## Mean :	25.5	Mean : 0.100	Mean : 0.774	Mean : 14.80
## 3rd Qu.:	37.8	3rd Qu.: 0.728	3rd Qu.: 1.175	3rd Qu.: 20.75
## Max.	: 50.0	Max. : 1.595	Max. : 2.400	Max. : 29.00

```
# Get the rows corresponding to the 'min', 'median', and 'max' of 'V4'
row.extractor(dat, 4)
```

##	V1	V2	V3	V4
## 28 28	-1.4708	0.00	2	
## 47 47	0.3646	1.28	13	
## 29 29	-0.4782	0.07	13	
## 11 11	1.5118	2.40	29	
## 14 14	-2.2147	0.03	29	
## 18 18	0.9438	1.47	29	
## 19 19	0.8212	0.15	29	
## 50 50	0.8811	0.47	29	

```
# Get the 'min' rows only, referenced by the variable name
row.extractor(dat, "V4", "min")
```



```
##      V1      V2 V3 V4
## 28 28 -1.471  0  2

# Get the 'median' rows only. Notice that there are two rows
#   since we have an even number of cases and true median
#   is the mean of the two central sorted values
row.extractor(dat, "V4", "median")

##      V1      V2  V3 V4
## 47 47  0.3646 1.28 13
## 29 29 -0.4782 0.07 13

# Get the rows corresponding to the deciles of 'V3'
row.extractor(dat, "V3", seq(0.1, 1, 0.1))

##      V1      V2  V3 V4
## 10 10 -0.30539 0.14 22
## 26 26 -0.05613 0.29 16
## 39 39  1.10003 0.37 13
## 41 41 -0.16452 0.54 10
## 30 30  0.41794 0.59 26
## 44 44  0.55666 0.70  5
## 37 37 -0.39429 1.06 21
## 49 49 -0.11235 1.22 14
## 34 34 -0.05381 1.52 19
## 11 11  1.51178 2.40 29
```

References

which.quantile function by [cbeleites](#)
 See: <http://stackoverflow.com/q/10256503/1270695>

Part II

The Functions

Where to Get the Functions

The most current source code for the functions described in this document follow.

To load the functions, you can directly source them from the 2657 R Functions page at github: <https://github.com/mrdwab/2657-R-Functions>

You should be able to load the functions using the following (replace ----- with the function name):

```
require(RCurl)
baseURL = c("https://raw.githubusercontent.com/mrdwab/2657-R-Functions/master/")
source(textConnection(getURL(paste0(baseURL, "scripts/-----R"))))
```

concat.split

```
concat.split = function(data, split.col, to.list=FALSE, mode=NULL,
                        sep=",", drop.col=FALSE) {
  # Takes a column with multiple values, splits the values into
  # separate columns, and returns a new data.frame.
  # 'data' is the source data.frame; 'split.col' is the variable that
  # needs to be split; 'to.list' is whether the split output should
  # be added as a single variable list (defaults to "FALSE");
  # mode' can be either 'binary' or 'value' (where 'binary' is
  # default and it recodes values to 1 or NA); 'sep' is the
  # character separating each value (defaults to ',');
  # and 'drop.col' is logical (whether to remove the original
  # variable from the output or not.
  #
  # === EXAMPLES ===
  #
  #       dat = data.frame(V1 = c("1, 2, 4", "3, 4, 5",
  #                               "1, 2, 5", "4", "1, 2, 3, 5"),
  #                         V2 = c("1;2;3;4", "1", "2;5",
  #                               "3;2", "2;3;4"))
  #       dat2 = data.frame(V1 = c("Fred, John, Sue", "Jerry, Jill",
  #                                "Sally, Ryan", "Susan, Amos, Ben"))
  #
  #       concat.split(dat, 1)
  #       concat.split(dat, 2, sep=";")
  #       concat.split(dat, "V2", sep=";", mode="value")
  #       concat.split(dat, "V1", mode="binary")
  #       concat.split(dat2, 1)
  #       concat.split(dat2, "V1", drop.col=TRUE)
  #
  # See: http://stackoverflow.com/q/10100887/1270695

  if (is.numeric(split.col)) split.col = split.col
  else split.col = which(colnames(data) %in% split.col)

  a = as.character(data[, split.col])
  b = strsplit(a, sep)

  if (isTRUE(to.list)) {
    varname = paste(names(data[split.col]), "_list", sep="")
    if (suppressWarnings(is.na(try(max(as.numeric(unlist(b))))))) {
      data[varname] = list(lapply(lapply(b, as.character),
                                     function(x) gsub("^\\s+|\\s+$",
                                                         "", x)))
    } else if (!is.na(try(max(as.numeric(unlist(b)))))) {
      data[varname] = list(lapply(b, as.numeric))
    }
  }
  if (isTRUE(drop.col)) data[-split.col]
  else data
} else if (!isTRUE(to.list)) {
  if (suppressWarnings(is.na(try(max(as.numeric(unlist(b))))))) {
    what = "string"
    ncol = max(unlist(lapply(b, function(i) length(i))))
  } else if (!is.na(try(max(as.numeric(unlist(b)))))) {
    what = "numeric"
    ncol = max(as.numeric(unlist(b)))
  }
}
```

```

m = matrix(nrow = nrow(data), ncol = ncol)
v = vector("list", nrow(data))

if (identical(what, "string")) {
  temp = as.data.frame(t(sapply(b, '[', 1:ncol)))
  names(temp) = paste(names(data[split.col]), "_", 1:ncol, sep="")
  temp = apply(temp, 2, function(x) gsub("^\\s+|\\s+$", "", x))
  temp1 = cbind(data, temp)
} else if (identical(what, "numeric")) {
  for (i in 1:nrow(data)) {
    v[[i]] = as.numeric(strsplit(a, sep)[[i]])
  }

  temp = v

  for (i in 1:nrow(data)) {
    m[i, temp[[i]]] = temp[[i]]
  }

  m = data.frame(m)
  names(m) = paste(names(data[split.col]), "_", 1:ncol, sep="")

  if (is.null(mode) || identical(mode, "binary")) {
    temp1 = cbind(data, replace(m, m != "NA", 1))
  } else if (identical(mode, "value")) {
    temp1 = cbind(data, m)
  }
}

if (isTRUE(drop.col)) temp1[-split.col]
else temp1
}

```

df.sorter

```
df.sorter = function(data, var.order=names(data), col.sort=NULL, at.start=TRUE ) {
  # Sorts a data.frame by columns or rows or both.
  # Can also subset the data columns by using 'var.order'.
  # Can refer to variables either by names or number.
  # If referring to variable by number, and sorting both the order
  #   of variables and the sorting within variables, refer to the
  #   variable numbers of the final data.frame.
  #
  # === EXAMPLES ===
  #
  #   library(foreign)
  #   temp = "http://www.ats.ucla.edu/stat/stata/modules/kidshtwt.dta"
  #   kidshtwt = read.dta(temp); rm(temp)
  #   df.sorter(kidshtwt, var.order = c("fam", "bir", "wt", "ht"))
  #   df.sorter(kidshtwt, var.order = c("fam", "bir", "wt", "ht"),
  #             col.sort = c("birth", "famid")) # USE FULL NAMES HERE
  #   df.sorter(kidshtwt, var.order = c(1:4),    # DROP THE WT COLUMNS
  #             col.sort = 3)                   # SORT BY HT1

  if (is.numeric(var.order))
    var.order = colnames(data)[var.order]
  else var.order = var.order

  a = names(data)
  b = length(var.order)
  subs = vector("list", b)

  if (isTRUE(at.start)) {
    for (i in 1:b) {
      subs[[i]] = sort(grep(paste("^", var.order[i],
                                sep="", collapse=""),
                            a, value=TRUE))
    }
  } else if (!isTRUE(at.start)) {
    for (i in 1:b) {
      subs[[i]] = sort(grep(var.order[i], a, value=TRUE))
    }
  }

  x = unlist(subs)
  y = data[ , x ]

  if (is.null(col.sort)) {
    y
  } else if (is.numeric(col.sort)) {
    col.sort = colnames(y)[col.sort]
    y[do.call(order, y[col.sort]), ]
  } else if (!is.numeric(col.sort)) {
    col.sort = col.sort
    y[do.call(order, y[col.sort]), ]
  }
}
```

multi.freq.table

```

multi.freq.table = function(data, sep="", dropzero=FALSE,
                             clean=TRUE, basic=FALSE) {
  # Takes boolean multiple-response data and tabulates it according
  #   to the possible combinations of each variable.
  #
  # === EXAMPLES ===
  #   set.seed(1)
  #   dat = data.frame(A = sample(c(0, 1), 20, replace=TRUE),
  #                     B = sample(c(0, 1), 20, replace=TRUE),
  #                     C = sample(c(0, 1), 20, replace=TRUE),
  #                     D = sample(c(0, 1), 20, replace=TRUE),
  #                     E = sample(c(0, 1), 20, replace=TRUE))
  #   multi.freq.table(dat)
  #   multi.freq.table(dat[1:3], sep="-", dropzero=TRUE)
  #
  # See: http://stackoverflow.com/q/11348391/1270695
  #      http://stackoverflow.com/q/11622660/1270695

  if(isTRUE(basic)) {
    counts = data.frame(Freq = colSums(data),
                        Pct.of.Resp = (colSums(data)/sum(data))*100,
                        Pct.of.Cases = (colSums(data)/nrow(data))*100)
  } else if (!isTRUE(basic)) {
    counts = data.frame(table(data))
    N = ncol(counts)
    counts$Combn = apply(counts[-N] == 1, 1,
                        function(x) paste(names(counts[-N])[x],
                                           collapse=sep))

    counts$Pct.of.Resp = (counts$Freq/sum(data))*100
    counts$Pct.of.Cases = (counts$Freq/nrow(data))*100
    if (isTRUE(dropzero)) {
      counts = counts[counts$Freq != 0, ]
    } else if (!isTRUE(dropzero)) {
      counts = counts
    }
    if (isTRUE(clean)) {
      counts = data.frame(Combn = counts$Combn, Freq = counts$Freq,
                          Pct.of.Resp = counts$Pct.of.Resp,
                          Pct.of.Cases = counts$Pct.of.Cases)
    }
  }
  counts
}

```


row.extractor

```

row.extractor = function(data, extract.by, what="all") {
  # Extracts rows with min, median, and max values, or by quantiles.
  # Values for "what" can be "min", "median", "max", "all", or a
  #   vector specifying the desired quantiles.
  # Values for "extract.by" can be the variable name or number.
  #
  # === EXAMPLES ===
  #
  #   set.seed(1)
  #   dat = data.frame(V1 = 1:10, V2 = rnorm(10), V3 = rnorm(10),
  #                     V4 = sample(1:20, 10, replace=T))
  #   dat2 = dat[-10,]
  #   row.extractor(dat, 4, "all")
  #   row.extractor(dat1, 4, "min")
  #   row.extractor(dat, "V4", "median")
  #   row.extractor(dat, 4, c(0, .5, 1))
  #   row.extractor(dat, "V4", c(0, .25, .5, .75, 1))
  #
  # "which.quantile" function by cbeleites:
  # http://stackoverflow.com/users/755257/cbeleites
  # See: http://stackoverflow.com/q/10256503/1270695

  if (is.numeric(extract.by)) {
    extract.by = extract.by
  } else if (is.numeric(extract.by) != 0) {
    extract.by = which(colnames(data) %in% "extract.by")
  }

  if (is.character(what)) {
    which.median = function(data, extract.by) {
      a = data[, extract.by]
      if (length(a) %% 2 != 0) {
        which(a == median(a))
      } else if (length(a) %% 2 == 0) {
        b = sort(a)[c(length(a)/2, length(a)/2+1)]
        c(max(which(a == b[1])), min(which(a == b[2])))
      }
    }
  }

  X1 = data[which(data[extract.by] == min(data[extract.by])), ] # min
  X2 = data[which(data[extract.by] == max(data[extract.by])), ] # max
  X3 = data[which.median(data, extract.by), ] # median

  if (identical(what, "min")) {
    X1
  } else if (identical(what, "max")) {
    X2
  } else if (identical(what, "median")) {
    X3
  } else if (identical(what, "all")) {
    rbind(X1, X3, X2)
  }
} else if (is.numeric(what)) {
  which.quantile <- function (data, extract.by, what, na.rm = FALSE) {

    x = data[, extract.by]

```

```

    if (! na.rm & any (is.na (x)))
      return (rep (NA_integer_, length (what)))

    o <- order (x)
    n <- sum (! is.na (x))
    o <- o [seq_len (n)]

    nppm <- n * what - 0.5
    j <- floor(nppm)
    h <- ifelse((nppm == j) & ((j%2L) == 0L), 0, 1)
    j <- j + h

    j [j == 0] <- 1
    o[j]
  }
  data[which.quantile(data, extract.by, what), ] # quantile
}

```


Part III

Snippets and Tips

Snippets

Load All Scripts and Data Files From Multiple Directories

```
load.scripts.and.data = function(path,
                                pattern=list(scripts = "*.R$",
                                              data = "*.rda$|*.Rdata$"),
                                ignore.case=TRUE) {
  # Reads all the data files and scripts from specified directories.
  #   In general, should only need to specify the directories.
  #   Specify directories without trailing slashes.
  #
  # === EXAMPLE ===
  #
  #   load.scripts.and.data(c("~/Dropbox/Public",
  #                           "~/Dropbox/Public/R Functions"))

  file.sources = list.files(path, pattern=pattern$scripts,
                             full.names=TRUE, ignore.case=ignore.case)
  data.sources = list.files(path, pattern=pattern$data,
                             full.names=TRUE, ignore.case=ignore.case)
  sapply(data.sources, load, .GlobalEnv)
  sapply(file.sources, source, .GlobalEnv)
}
```

Convert a List of Data Frames Into Individual Data Frames

```
unlist.dfs = function(data) {
  # Specify the quoted name of the source list.
  q = get(data)
  prefix = paste0(data, "_", 1:length(q))
  for (i in 1:length(q)) assign(prefix[i], q[[i]], envir=.GlobalEnv)
}
```

Example

Note that the list name must be quoted.

```
# Sample data
temp = list(A = data.frame(A = 1:2, B = 3:4),
            B = data.frame(C = 5:6, D = 7:8))
# View it
temp

## $A
##   A B
## 1 1 3
## 2 2 4
##
## $B
##   C D
## 1 5 7
## 2 6 8
##
```

```
# Remove any files with similar names to output
rm(list=ls(pattern="temp_"))
# The following should not work
temp_1
```

```
## Error: object 'temp_1' not found
```

```
# Split it up!
unlist.dfs("temp")
# List files with the desired pattern
ls(pattern="temp_")
```

```
## [1] "temp_1" "temp_2"
```

```
# View the new files
temp_1
```

```
##      A B
## 1 1 3
## 2 2 4
```

```
temp_2
```

```
##      C D
## 1 5 7
## 2 6 8
```

Tips

Batch Convert Factor to Character Variables

In the example data below, `author` and `title` are automatically converted to factor (unless you add the argument `stringsAsFactor = FALSE` when you are creating the data). What if you forgot?

Use `sapply` to identify factors and convert to string.

```
dat = data.frame(title = c("title1", "title2", "title3"),
                 author = c("author1", "author2", "author3"),
                 customerID = c(1, 2, 1))

str(dat)

## 'data.frame':   3 obs. of  3 variables:
## $ title      : Factor w/ 3 levels "title1","title2",...: 1 2 3
## $ author     : Factor w/ 3 levels "author1","author2",...: 1 2 3
## $ customerID: num  1 2 1

dat[sapply(dat, is.factor)] = lapply(dat[sapply(dat, is.factor)], as.character)
str(dat)

## 'data.frame':   3 obs. of  3 variables:
## $ title      : chr  "title1" "title2" "title3"
## $ author     : chr  "author1" "author2" "author3"
## $ customerID: num  1 2 1
```