

# Wakefield: Random Data Set (Part II)

29 April, 2015

## Contents

<b>1</b>	<b>Brief Package Description</b>	<b>1</b>
<b>2</b>	<b>Improvements</b>	<b>2</b>
2.1	Repeated Measures Series . . . . .	2
2.2	Dummy Coding Expansion of Factors . . . . .	3
2.3	Factor to Numeric Conversion . . . . .	4
2.4	Viewing Whole Data Set . . . . .	5
2.5	Visualizing Column Types and NAs . . . . .	6
<b>3</b>	<b>Table of Variable Functions</b>	<b>7</b>
<b>4</b>	<b>Possible Uses</b>	<b>9</b>
4.1	Testing Methods . . . . .	9
4.2	Unique Student Data for Course Assignments . . . . .	9
4.3	Blogging and Online Help Communities . . . . .	10
<b>5</b>	<b>Getting Involved</b>	<b>11</b>

This post is part II of a series detailing the GitHub package, [wakefield](#), for generating random data sets. The [First Post \(part I\)](#) was a test run to gauge user interest. I received positive feedback and some ideas for improvements, which I'll share below.

The post is broken into the following sections:

You can view just the R code [HERE](#) or PDF version [HERE](#)

## 1 Brief Package Description

First we'll use the [pacman](#) package to grab the **wakefield** package from GitHub and then load it as well as the handy **dplyr** package.

```
if (!require("pacman")) install.packages("pacman"); library(pacman)
p_install_gh("trinker/wakefield")
p_load(dplyr, wakefield)
```

The main function in **wakefield** is `r_data_frame`. It takes `n` (the number of rows) and any number of *variable functions* that generate random columns. The result is a data frame with named, randomly generated columns. Below is an example, for details see [Part I](#) or the [README](#)

```
set.seed(10)

r_data_frame(n = 30,
  id,
  race,
  age(x = 8:14),
  Gender = sex,
  Time = hour,
  iq,
  grade,
  height(mean=50, sd = 10),
  died,
  Scoring = rnorm,
  Smoker = valid
)
```

```
## Source: local data frame [30 x 11]
##
##   ID      Race Age Gender      Time  IQ Grade Height  Died    Scoring
## 1  01      White  11   Male 01:00:00 110  90.7    52 FALSE -1.8227126
## 2  02      White   8   Male 01:00:00 111  91.8    36  TRUE  0.3525440
## 3  03      White   9   Male 01:30:00  87  81.3    39 FALSE -1.3484514
## 4  04 Hispanic  14   Male 01:30:00 111  83.2    46  TRUE  0.7076883
## 5  05      White  10 Female 03:30:00  95  80.1    51  TRUE -0.4108909
## 6  06      White  13 Female 04:00:00  97  93.9    61  TRUE -0.4460452
## 7  07      White  13 Female 05:00:00 109  89.5    44  TRUE -1.0411563
## 8  08      White  14   Male 06:00:00 101  92.3    63  TRUE -0.3292247
## 9  09      White  12   Male 06:30:00 110  90.1    52  TRUE -0.2828216
## 10 10      White  11   Male 09:30:00 107  88.4    47 FALSE  0.4324291
## .. ..      ... ..      ...      ... ..      ... ..      ...
## Variables not shown: Smoker (lgl)
```

## 2 Improvements

### 2.1 Repeated Measures Series

Big thanks to [Ananda Mahto](#) for [suggesting](#) better handling of repeated measures series and provided concise code to extend this capability.

The user may now specify the same *variable function* multiple times and it is named appropriately:

```
set.seed(10)

r_data_frame(
  n = 500,
  id,
  age, age, age,
  grade, grade, grade
)
```

```
## Source: local data frame [500 x 7]
```

```
##
##      ID Age_1 Age_2 Age_3 Grade_1 Grade_2 Grade_3
## 1  001   28   33   32   80.2   87.2   85.6
## 2  002   24   35   31   89.7   91.7   86.8
## 3  003   26   33   23   92.7   85.7   88.7
## 4  004   31   24   28   82.2   90.0   86.0
## 5  005   21   21   29   86.5   87.0   88.4
## 6  006   23   28   25   85.6   93.5   86.7
## 7  007   24   22   26   89.3   90.3   87.6
## 8  008   24   21   23   92.4   88.3   89.3
## 9  009   29   23   32   86.4   84.4   88.2
## 10 010   26   34   32   97.6   84.2   90.6
## .. ...   ...   ...   ...   ...   ...
```

But he went further, recommending a short hand for `variable, variable, variable`. The `r_series` function takes a variable function and `j` number of columns. It can also be renamed with the `name` argument:

```
set.seed(10)

r_data_frame(n=100,
  id,
  age,
  sex,
  r_series(gpa, 2),
  r_series(likert, 3, name = "Question")
)
```

```
## Source: local data frame [100 x 8]
##
##      ID Age  Sex GPA_1 GPA_2 Question_1 Question_2
## 1  001  28  Male  3.00  4.00 Strongly Disagree Strongly Agree
## 2  002  24  Male  3.67  3.67 Disagree Neutral
## 3  003  26  Male  3.00  4.00 Disagree Strongly Disagree
## 4  004  31  Male  3.67  3.67 Neutral Strongly Agree
## 5  005  21 Female  3.00  3.00 Agree Strongly Agree
## 6  006  23 Female  3.67  3.67 Agree Agree
## 7  007  24 Female  3.67  4.00 Disagree Strongly Disagree
## 8  008  24  Male  2.67  3.00 Strongly Agree Neutral
## 9  009  29 Female  4.00  3.33 Neutral Strongly Disagree
## 10 010  26  Male  4.00  3.00 Disagree Strongly Disagree
## .. ...   ...   ...   ...   ...
## Variables not shown: Question_3 (fctr)
```

## 2.2 Dummy Coding Expansion of Factors

It is sometimes nice to expand a factor into `j` (number of groups) dummy coded columns. Here we see a factor version and then a dummy coded version of the same data frame:

```
set.seed(10)

r_data_frame(n=100,
  id,
```

```

    age,
    sex,
    political
)

```

```

## Source: local data frame [100 x 4]
##
##      ID Age  Sex   Political
## 1  001  28  Male Constitution
## 2  002  24  Male Constitution
## 3  003  26  Male   Democrat
## 4  004  31  Male   Democrat
## 5  005  21 Female Constitution
## 6  006  23 Female   Democrat
## 7  007  24 Female   Democrat
## 8  008  24  Male Republican
## 9  009  29 Female Constitution
## 10 010  26  Male   Democrat
## .. ... ..

```

The dummy coded version...

```

set.seed(10)

r_data_frame(n=100,
  id,
  age,
  r_dummy(sex, prefix = TRUE),
  r_dummy(political)
)

```

```

## Source: local data frame [100 x 9]
##
##      ID Age Sex_Male Sex_Female Constitution Democrat Green Libertarian
## 1  001  28      1         0          1          0      0          0
## 2  002  24      1         0          1          0      0          0
## 3  003  26      1         0          0          1      0          0
## 4  004  31      1         0          0          1      0          0
## 5  005  21      0         1          1          0      0          0
## 6  006  23      0         1          0          1      0          0
## 7  007  24      0         1          0          1      0          0
## 8  008  24      1         0          0          0      0          0
## 9  009  29      0         1          1          0      0          0
## 10 010  26      1         0          0          1      0          0
## .. ... ..
## Variables not shown: Republican (int)

```

## 2.3 Factor to Numeric Conversion

There are times when you feel like a factor and the when you feel like an integer version. This is particularly useful with Likert-type data and other ordered factors. The `as_integer` function takes a `data.frame` and allows the user to specify the indices (`j`) to convert from factor to numeric. Here I show a factor `data.frame` and then the integer conversion:

```
set.seed(10)

r_data_frame(5,
  id,
  r_series(likert, j = 4, name = "Item")
)
```

```
## Source: local data frame [5 x 5]
##
##   ID      Item_1 Item_2      Item_3      Item_4
## 1  1      Neutral Agree    Disagree    Neutral
## 2  2        Agree Agree    Neutral    Strongly Agree
## 3  3      Neutral Agree Strongly Agree    Agree
## 4  4    Disagree Disagree    Neutral    Agree
## 5  5 Strongly Agree Neutral    Agree Strongly Disagree
```

As integers...

```
set.seed(10)

r_data_frame(5,
  id,
  r_series(likert, j = 4, name = "Item")
) %>%
  as_integer(-1)
```

```
## Source: local data frame [5 x 5]
##
##   ID Item_1 Item_2 Item_3 Item_4
## 1  1      3      4      2      3
## 2  2      4      4      3      5
## 3  3      3      4      5      4
## 4  4      2      2      3      4
## 5  5      5      3      4      1
```

## 2.4 Viewing Whole Data Set

**dplyr** has a nice print method that hides excessive rows and columns. Typically this is great behavior. Sometimes you want to quickly see the whole width of the data set. We can use **View** but this is still wide and shows all columns. The **peek** function shows minimal rows, truncated columns, and prints wide for quick inspection. This is particularly nice for text strings as data. **dplyr** prints wide data sets like this:

```
r_data_frame(100,
  id,
  name,
  sex,
  sentence
)
```

```
## Source: local data frame [100 x 4]
##
```

```
##      ID      Name      Sex
## 1  001    Gerald    Male
## 2  002     Jason    Male
## 3  003 Mitchell    Male
## 4  004       Joe Female
## 5  005    Mickey    Male
## 6  006    Michal    Male
## 7  007    Dannie Female
## 8  008    Jordan    Male
## 9  009      Rudy Female
## 10 010    Sammie Female
## .. ...      ...      ...
## Variables not shown: Sentence (chr)
```

Now use `peek`:

```
r_data_frame(100,
  id,
  name,
  sex,
  sentence
) %>% peek
```

```
## Source: local data frame [100 x 4]
##
##      ID      Name      Sex      Sentence
## 1  001     Jae Female Excuse me.
## 2  002 Darnell Female Over the l
## 3  003  Elisha Female First of a
## 4  004  Vernon Female Gentlemen,
## 5  005   Scott   Male That's wha
## 6  006   Kasey Female We don't h
## 7  007 Michael   Male You don't
## 8  008   Cecil Female I'll get o
## 9  009    Cruz Female They must
## 10 010  Travis Female Good night
## .. ...      ...      ...      ...
```

## 2.5 Visualizing Column Types and NAs

When we build a large random data set it is nice to get a sense of the column types and the missing values. The `table_heat` (also plot for `tbl_df` class) does this. Here I'll generate a data set, add missing values (`r_na`), and then plot:

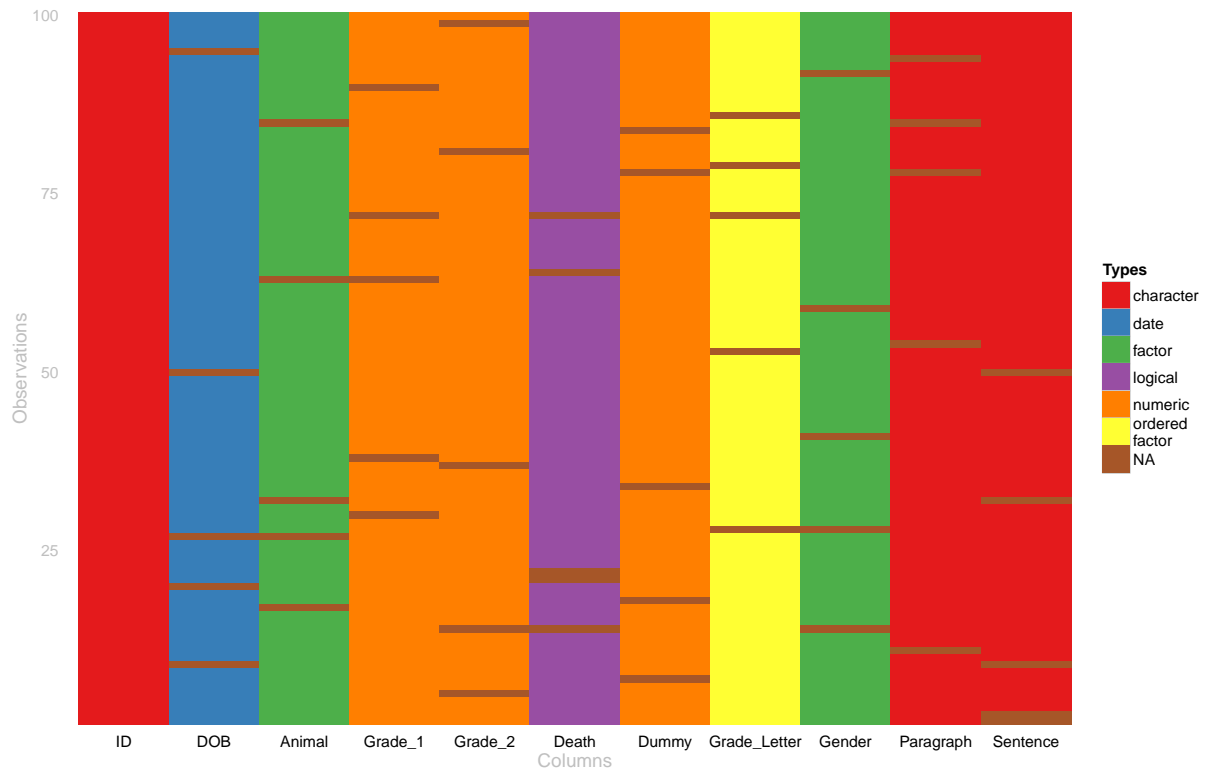
```
set.seed(10)

r_data_frame(n=100,
  id,
  dob,
  animal,
  grade, grade,
  death,
```

```

dummy,
grade_letter,
gender,
paragraph,
sentence
) %>%
  r_na() %>%
  plot(palette = "Set1")

```



### 3 Table of Variable Functions

There are currently 66 **wakefield** based *variable functions* to chose for building columns. Use **variables()** to see them or **variables(TRUE)** to see a list of them broken into variable types. Here's an HTML table version:

% latex table generated in R 3.2.0 by xtable 1.7-4 package % Wed Apr 29 22:08:53 2015

age	dice	grade_letter	level	normal	smokes
animal	died	grade_level	likert	normal_round	speed
answer	dna	group	likert_5	paragraph	speed_kph
area	dob	hair	likert_7	pet	speed_mph
birth	dummy	height	lorem_ipsum	political	state
car	education	height_cm	lower	primary	string
children	employment	height_in	lower_factor	race	upper
coin	eye	income	marital	religion	upper_factor
color	gender	internet_browser	military	sat	valid
date_stamp	gpa	iq	month	sentence	year
death	grade	language	name	sex	zip_code

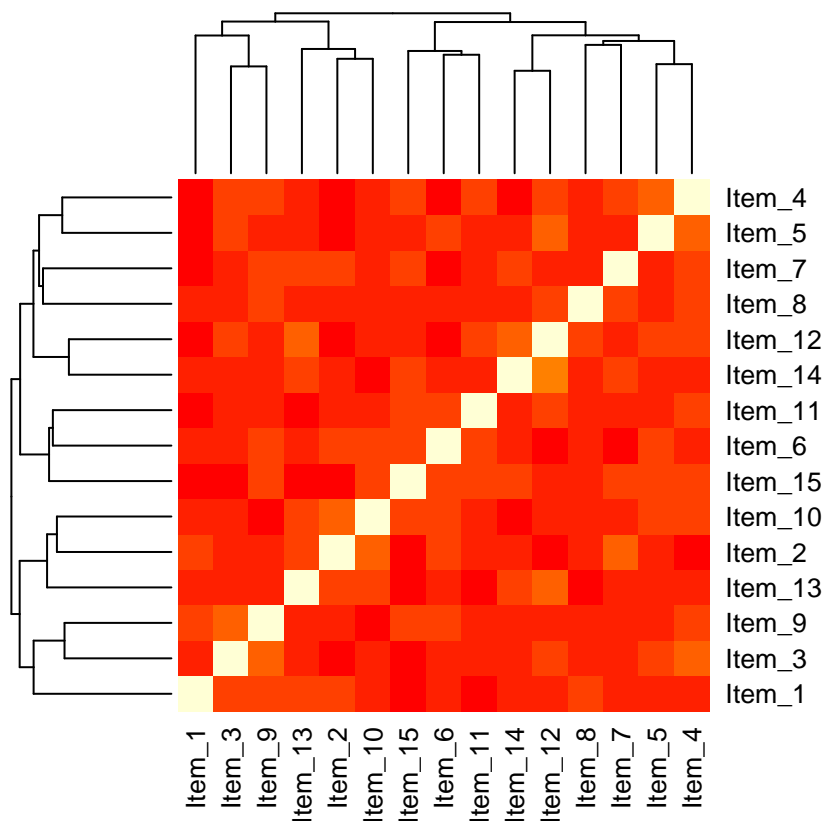


## 4 Possible Uses

### 4.1 Testing Methods

I personally will use this most frequently when I'm testing out a model. For example say you wanted to test psychometric functions, including the `cor` function, on a randomly generated assessment:

```
dat <- r_data_frame(120,  
  id,  
  sex,  
  age,  
  r_series(likert, 15, name = "Item")  
) %>%  
  as_integer(-c(1:3))  
  
dat %>%  
  select(contains("Item")) %>%  
  cor %>%  
  heatmap
```



### 4.2 Unique Student Data for Course Assignments

Sometimes it's nice if students each have their own data set to work with but one in which you control the parameters. Simply supply the students with a unique integer id and they can use this inside of `set.seed`

with a **wakefield** `r_data_frame` you've constructed for them in advance. Viola 25 instant data sets that are structurally the same but randomly different.

```
student_id <- ## INSERT YOUR ID HERE

set.seed(student_id)

dat <- function(1000,
  id,
  gender,
  religion,
  internet_browser,
  language,
  iq,
  sat,
  smokes
)
```

### 4.3 Blogging and Online Help Communities

**wakefield** can make data sharing on blog posts and online help communities (e.g., [TalkStats](#), [StackOverflow](#)) fast, accessible, and with little space or cognitive effort. Use `variables(TRUE)` to see *variable functions* by class and select the ones you want:

```
variables(TRUE)

## $character
## [1] "lorem_ipsum" "lower"      "name"      "paragraph" "sentence"
## [6] "string"      "upper"      "zip_code"
##
## $date
## [1] "birth"      "date_stamp" "dob"
##
## $factor
## [1] "animal"      "answer"      "area"
## [4] "car"         "coin"        "color"
## [7] "dna"         "education"   "employment"
## [10] "eye"         "gender"      "grade_level"
## [13] "group"       "hair"        "internet_browser"
## [16] "language"    "lower_factor" "marital"
## [19] "military"    "month"       "pet"
## [22] "political"   "primary"     "race"
## [25] "religion"    "sex"         "state"
## [28] "upper_factor"
##
## $integer
## [1] "age"         "children" "dice"      "level"     "year"
##
## $logical
## [1] "death"      "died"      "smokes"    "valid"
##
## $numeric
## [1] "dummy"      "gpa"       "grade"     "height"
```

```
## [5] "height_cm"      "height_in"      "income"          "iq"
## [9] "normal"         "normal_round"   "sat"             "speed"
## [13] "speed_kph"      "speed_mph"
##
## $`ordered factor`
## [1] "grade_letter" "likert"          "likert_5"        "likert_7"
```

Then throw the inside of `r_data_fame` to make a quick data set to share.

```
r_data_frame(8,
  name,
  sex,
  r_series(iq, 3)
) %>%
  peek %>%
  dput
```

## 5 Getting Involved

If you're interested in getting involved with use or contributing you can:

1. Install and use [wakefield](#)
2. Provide feedback via comments below
3. Provide feedback (bugs, improvements, and feature requests) via [wakefield's Issues Page](#)
4. Fork from [GitHub](#) and give a Pull Request

Thanks for reading, your feedback is welcomed.

---

*\*Get the R code for this post [HERE](#)*

*Get a PDF version this post [HERE](#)*