Wide/Long Data Frame Reshaping

Example with a repeated measures data set

Bruce Dudek

2020-05-04

Contents

1 Introduction, the R environment, and the Data Example

This document is can be a quick template for using various R functions that permit reshaping/restructuring data frames from wide to long formats, and vice versa. It can be generally useful, but is intended for use by students in the APSY511 Statistics II class at the University at Albany. The most common need for switching between these formats is with regard to repeated measures types of designs. Traditionally, most software (such as SPSS methods we have used) expect the data set to be structured in a wide format. In R, the long format is more commonly needed.

1.1 First, load the packages that will be required

```
library(gt)
library(knitr)
library(foreign)
library(ggplot2)
library(ggthemes)
library(tidyr)
```

```r
library(dplyr)
library(nlme)
library(psych)
library(data.table)
```

1.2 Obtain the Data

The data set used here is a textbook example, taken from the Keppel textbook ((Keppel and Wickens, 2004, exercise #1, Ch. 16, pp 366-367). The study oulined in the exercise presumed to evaluate an appetitive behavior among rats, tongue protrusion (called DV in the data set). Tongue protrusions are also used in social situations, presumably as part of a chemical senses system for evaluating airborne molecules that may carry social significance. Accordingly, a study was designed where rats were exposed to bedding types. Two control types of conditions were employed, clean bedding and bedding from the rat's own home cage. Three other conditions were bedding from other species, iguana, whiptail lizard, and kangaroo rat. The IV (which will eventually be called "type" in the long data frame) thus had five levels, with each of ten rats being measured under each level, making the study a simple one-factor repeated measures design.

The data set is found in a .csv file that is in wide format. Notice that the five "levels" of the repeated factor are in columns 2-6.

```
# First, read in the wide data file and look at it.
wide.df <- read.csv("1facrpt_wide1.csv")
kable(wide.df)
```

snum	clean	homecage	iguana	whiptail	krat
1	24	15	41	30	50
2	6	6	0	6	13
3	4	0	5	4	9
4	11	9	10	14	18
5	0	0	0	0	0
6	8	15	10	15	38
7	8	5	2	6	15
8	0	0	0	11	54
9	0	3	1	1	11
10	7	7	4	7	23

2 Convert from Wide to Long Format

There are multiple ways of accomplishing the wide to long format conversion.

- There is a `reshape` function in base R that can do the conversion.
- The **data.table** package also has capabilities for wide/long conversion.
- A **reshape** package also exists, and has been replaced by the **reshape2** package.
- In turn, the **reshape2** package has been superseded by a tidyverse set of functions in the **tidyr** package. The original primary functions for wide/long conversion with **tidyr**, the `gather` and `spread` functions, have been upgraded to the `pivot_longer` and `pivot_wide` functions.

This document will show code for several of these possibilities, but emphasize the newest **tidyr** functions.

2.1 Use `pivot_longer` from the tidyr package

The arguments needed to do the conversion are the wide dataframe name, the columns in which the variables to be converted are found, and the names of the new variables in the long format data frame. "names_to" creates the variable named "type" (for stimulus type) and the values for that variable will be the original column names. The "values_to" argument creates the name of the variable that the original values of the wide-format variables comprise. I just called it "dv" here for the dependent variable, tongue protrusions.

The logic of this function is that we tell it the columns in which the data reside that need to be reshaped (columns 2 to 6 in our wide.df data frame). Any other variables are assumed to be "key" or "id" variables which will be repeated for each of the newly created cases in the long format.

```
long1.df <- pivot_longer(data=wide.df,
                         cols=2:6,
                         names_to="type",
                         values_to="dv")
str(long1.df)
```

```
## tibble [50 x 3] (S3: tbl_df/tbl/data.frame)
##  $ snum: int [1:50] 1 1 1 1 1 2 2 2 2 2 ...
##  $ type: chr [1:50] "clean" "homecage" "iguana" "whiptail" ...
##  $ dv  : int [1:50] 24 15 41 30 50 6 6 0 6 13 ...
```

```
headTail(long1.df)
```

```
##   snum     type  dv
## 1    1    clean  24
## 2    1 homecage  15
## 3    1   iguana  41
## 4    1 whiptail  30
## 5  ...     <NA> ...
## 6   10 homecage   7
## 7   10   iguana   4
## 8   10 whiptail   7
## 9   10     krat  23
```

Note that the `pivot_longer` function produces a type of tidyverse data frame called a tibble. Tibbles have some advantages over standard data frames, but since we have not yet covered that, we can easily convert the tibble to a standard data frame.

```
long1.df <- as.data.frame(long1.df)
str(long1.df)
```

```
## 'data.frame':    50 obs. of  3 variables:
##  $ snum: int  1 1 1 1 1 2 2 2 2 2 ...
##  $ type: chr  "clean" "homecage" "iguana" "whiptail" ...
##  $ dv  : int  24 15 41 30 50 6 6 0 6 13 ...
```

Also note that the IV, called "type" is currently a character variable and "snum" is numeric. Use of these variables in in the repeated measures analyses will require them to be factors.

```
long1.df$snum <- as.factor(long1.df$snum)
long1.df$type <- as.factor(long1.df$type)
str(long1.df)
```

```
## 'data.frame':    50 obs. of  3 variables:
##  $ snum: Factor w/ 10 levels "1","2","3","4",..: 1 1 1 1 1 2 2 2 2 2 ...
##  $ type: Factor w/ 5 levels "clean","homecage",..: 1 2 3 5 4 1 2 3 5 4 ...
##  $ dv  : int  24 15 41 30 50 6 6 0 6 13 ...
```

It is also helpful to be able to control the order of the levels of the IV. For a "factor" variable, such as our "type" variable, R will default to ordering the categories/levels alphabetically. We can change that, and I do so here to match the order that they were in the wide-formate .csv file. This will keep graphics and contrasts comparable to what we produced using other software.

```
long1.df$type <-  ordered(long1.df$type,
                     levels=c("clean","homecage","iguana",
```

```
                                "whiptail", "krat"))
```

2.2 Using other facilities in R for wide to long conversion

The `reshape` function in base R handles the conversion with this approach, but I find it to be somewhat non-intuitive. It takes a while to get used to.

```
long1.dfreshape <- reshape(wide.df, direction="long",
                           varying= list(names(wide.df)[2:6]),
                           v.names="dv",
                           idvar="snum",
                           timevar="type",
                           times=names(wide.df[2:6]))
str(long1.dfreshape)
rownames(long1.dfreshape) <- NULL
```

The **data.table** package has a function called `melt` that does the wide/long conversion with fairly simple syntax. Since `melt` is also a function in **reshape2**, the "data.table::melt" syntax ensures that the correct one is used here.

```
long1.dfdt<- data.table::melt(setDT(wide.df),
                              id.vars="snum",
                              measure.vars= 2:6,
                              variable.name="type")
str(long1.dfdt)
```

The **tidyr** package has another pair of functions, `gather` and `spread` to do the wide/long conversions. These functions have been replaced by `pivot_long` and `pivot_wide` and will no longer be developed, but they can be used. Here, I use `gather` to do the same conversion from wide to long. Note that the way to ensure that the "snum" variable is the id that is repeated for each of the five new cases created for the five IV levels is to use the minus sign in front of it.

```
long1.dfgather <- gather(data=wide.df,
                         key="type",
                         value="dv",
                         -"snum",
                         factor_key=T)
str(long1.dfgather)
```

The **reshape2** package has simple syntax for the wide/long conversion and I have preferred it to the newer `pivot_long` function. It should continue to be on CRAN for a while yet, although the R world seems to be moving to the **tidyr** functions. Note that the function name used here, `melt`, is the same as the one from the **data.table** package illustrated above. Thus the "reshape2::melt" syntax makes sure to use the proper `melt` function.

```
long1.dfreshape2 <- reshape2::melt(wide.df,
                                   id = "snum",
                                   variable.name = "type",
                                   value.name = "dv")
str(long1.dfreshape2)

# sort the data frame to put all data for each subject together
long1.dfreshape2 <- long1.df[order(long1.dfreshape2$snum),]
long1.dfreshape2
```

Each of these methods has quirks in the syntax that may not be intuitive and require some learn-

ing/understanding before applying them to larger data sets than this simple one illustrated here. I have found `reshape2::melt` to be the most straight forward, but `pivot_longer` is becoming the defacto standard function.

3 Conversion from long to wide format data frames

Each of the approaches outlined above for the wide to long conversion has an accompanying function for the opposite, long to wide. I will show code for two of them, `pivot_wider`, and `dcast`.

We will use the long1.df data frame from above, the one originally produced by `pivot_longer` and then converted to a data frame with snum and type defined as factors.

```
wide2.dfpivot <- pivot_wider(data=long1.df,
                             id_cols=snum,
                             names_from=type,
                             values_from=dv)
str(wide2.dfpivot)
```

```
## tibble [10 x 6] (S3: tbl_df/tbl/data.frame)
##  $ snum    : Factor w/ 10 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10
##  $ clean   : int [1:10] 24 6 4 11 0 8 8 0 0 7
##  $ homecage: int [1:10] 15 6 0 9 0 15 5 0 3 7
##  $ iguana  : int [1:10] 41 0 5 10 0 10 2 0 1 4
##  $ whiptail: int [1:10] 30 6 4 14 0 15 6 11 1 7
##  $ krat    : int [1:10] 50 13 9 18 0 38 15 54 11 23
```

```
headTail(wide2.dfpivot)
```

```
##   snum clean homecage iguana whiptail krat
## 1    1    24       15     41       30   50
## 2    2     6        6      0        6   13
## 3    3     4        0      5        4    9
## 4    4    11        9     10       14   18
## 5 <NA>   ...      ...    ...      ...  ...
## 6    7     8        5      2        6   15
## 7    8     0        0      0       11   54
## 8    9     0        3      1        1   11
## 9   10     7        7      4        7   23
```

As was the case with `pivot_longer`, `pivot_wider` produces a tibble. If you want the object to be a traditional data frame, convert with the `as.data.frame` function and note that snum remains as a factor

```
wide2.dfpivot <- as.data.frame(wide2.dfpivot)
str(wide2.dfpivot)
```

```
## 'data.frame':    10 obs. of  6 variables:
##  $ snum    : Factor w/ 10 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10
##  $ clean   : int  24 6 4 11 0 8 8 0 0 7
##  $ homecage: int  15 6 0 9 0 15 5 0 3 7
##  $ iguana  : int  41 0 5 10 0 10 2 0 1 4
##  $ whiptail: int  30 6 4 14 0 15 6 11 1 7
##  $ krat    : int  50 13 9 18 0 38 15 54 11 23
```

We can accomplish the same thing with the `dcast` function from the **reshape2** package. Since a `dcast` function is also found in the **data.table** package, we use the `reshape2::dcast` syntax here to ensure use of the one intended.

```
wide2.dfdcast <- reshape2::dcast(long1.df,
                        snum~type,
                        mean,
                        value='dv')
```

```
## Using dv as value column: use value.var to override.
```

```
str(wide2.dfdcast)
```

```
## 'data.frame':    10 obs. of  6 variables:
##  $ snum    : Factor w/ 10 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10
##  $ clean   : num  24 6 4 11 0 8 8 0 0 7
##  $ homecage: num  15 6 0 9 0 15 5 0 3 7
##  $ iguana  : num  41 0 5 10 0 10 2 0 1 4
##  $ whiptail: num  30 6 4 14 0 15 6 11 1 7
##  $ krat    : num  50 13 9 18 0 38 15 54 11 23
```

```
kable(wide2.dfdcast)
```

snum	clean	homecage	iguana	whiptail	krat
1	24	15	41	30	50
2	6	6	0	6	13
3	4	0	5	4	9
4	11	9	10	14	18
5	0	0	0	0	0
6	8	15	10	15	38
7	8	5	2	6	15
8	0	0	0	11	54
9	0	3	1	1	11
10	7	7	4	7	23

Analogous functions exist from **tidyr** as the older `spread` function and from **data.table** as the `dcast` function (as mentioned above), but I don't show example code here for those functions.

4 A more complicated example

In this section, three things are illustrated in a wide to long conversion:

- Use of the "pipes" programming style so prevalent with tidyverse programmers.
- Converting only some of the variables in a wide data frame to the long format.
- Demonstration of how variables with a common prefix can be taken advantage of to specify the variables and to yield values of the new variable derived from the second part of the variable name.

The data set used in this illustration is the famous Anscombe data set that we have used previously. It consists of eight variables measured on eleven cases. The variables are four "x's" and four "y's". It is found in the base R installation.

```
head(anscombe,5)
```

```
##   x1 x2 x3 x4   y1   y2    y3   y4
## 1 10 10 10  8 8.04 9.14  7.46 6.58
## 2  8  8  8  8 6.95 8.14  6.77 5.76
## 3 13 13 13  8 7.58 8.74 12.74 7.71
## 4  9  9  9  8 8.81 8.77  7.11 8.84
## 5 11 11 11  8 8.33 9.26  7.81 8.47
```

Our goal will be to produce a long format data frame (or tibble) using only the "x" variables, leaving the "y's" out. In addition, we will create a case number variable (snum) artificially from the rownames since the data set did not have a subject number variable.

In addition, this is done in one "flow" using the pipes operator (%>%).

The code can be read this way:

1. Create a new data frame named anscombelong.x, beginning with the basic anscombe data set.
2. First, create a new variable, called "snum" in the anscombe wide formate data file, arbitrarily from the row numbers associated with the data set by using the `mutate` function from **dplyr**.
3. Select only this new snum variable and the four x's to continue.
4. Then use `pivot_longer` to do the wide to long conversion. Ask `pivot_longer` to look for variables in the wide data set that begin with "x" using the `starts_with` specifier. Specify that the common prefix is "x" with the `names_prefix` argument. Create the new variable in the long format called "X" (upper case to differentiate) with the `names_to` specifier. Then name the variable that the values of the x's will become (I called it dv).

```
anscombelong.x <- anscombe %>%
  mutate(snum=rownames(anscombe)) %>%
  select("snum","x1", "x2", "x3", "x4") %>%
  pivot_longer(
    cols=starts_with("x"),
    names_prefix = "x",
    names_to = c("X"),
    values_to = "dv",
  )
headTail(anscombelong.x)
```

```
##   snum    X  dv
## 1    1    1  10
## 2    1    2  10
## 3    1    3  10
## 4    1    4   8
## 5 <NA> <NA> ...
## 6   11    1   5
## 7   11    2   5
## 8   11    3   5
## 9   11    4   8
```

5 Resources on Wide/Long Conversion

https://tidyr.tidyverse.org/reference/pivot_longer.html

http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format/

https://uc-r.github.io/tidyr

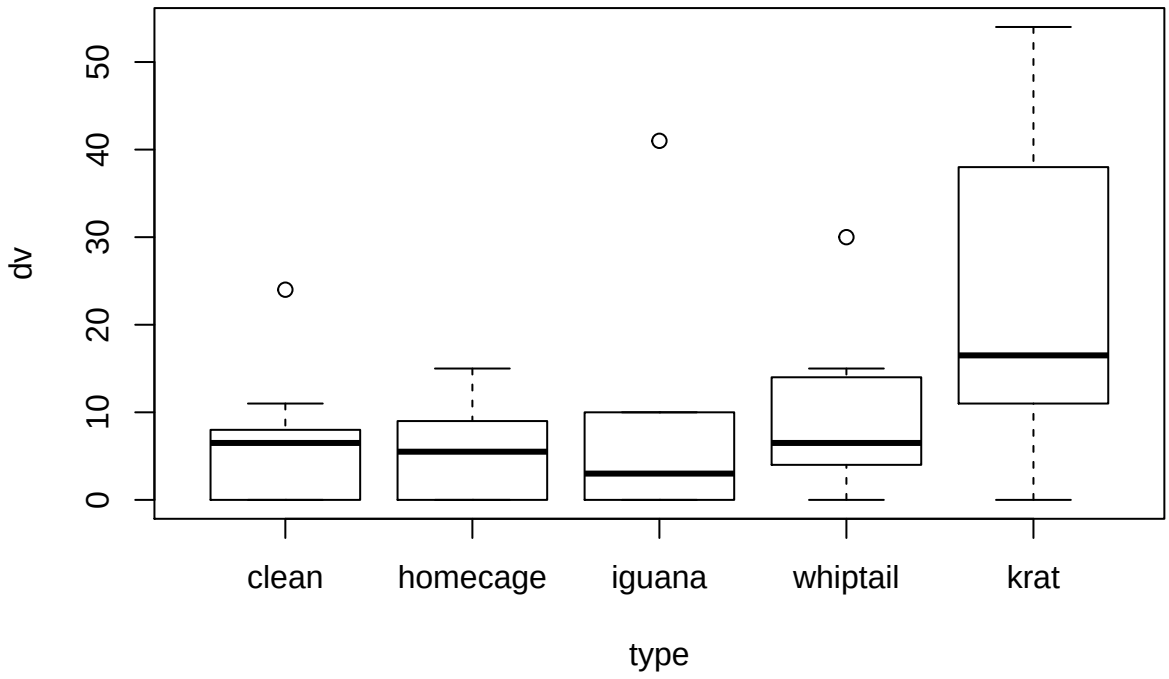
http://www.datasciencemadesimple.com/reshape-in-r-from-wide-to-long-from-long-to-wide/

6 Doing a repeated Measures ANOVA with the long-format data frame

6.1 First, some graphs

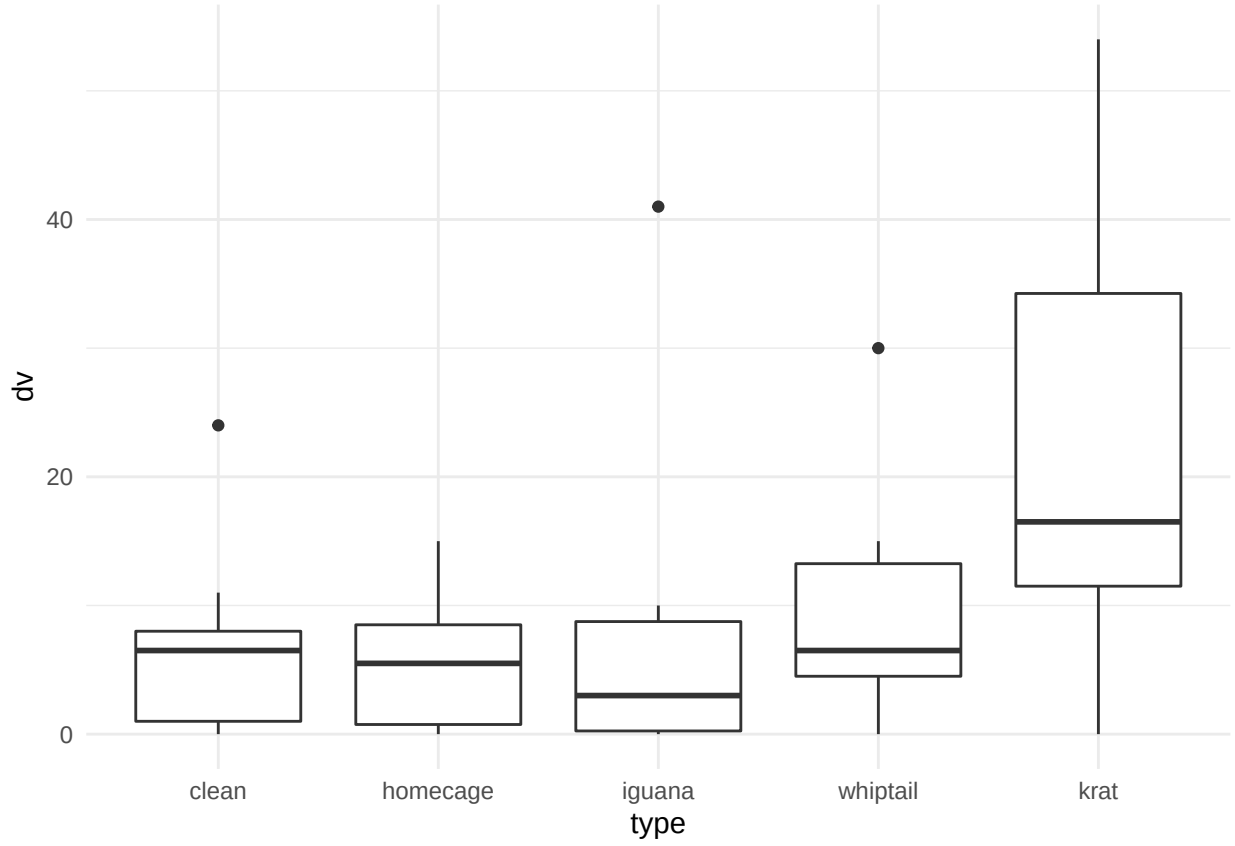
With the long-format data frame, the standard modeling syntax "DV~IV" can be used in base system graphics to plot the data. Here, I demonstrate boxplots.

```
boxplot(dv~type,data=long1.df)
```

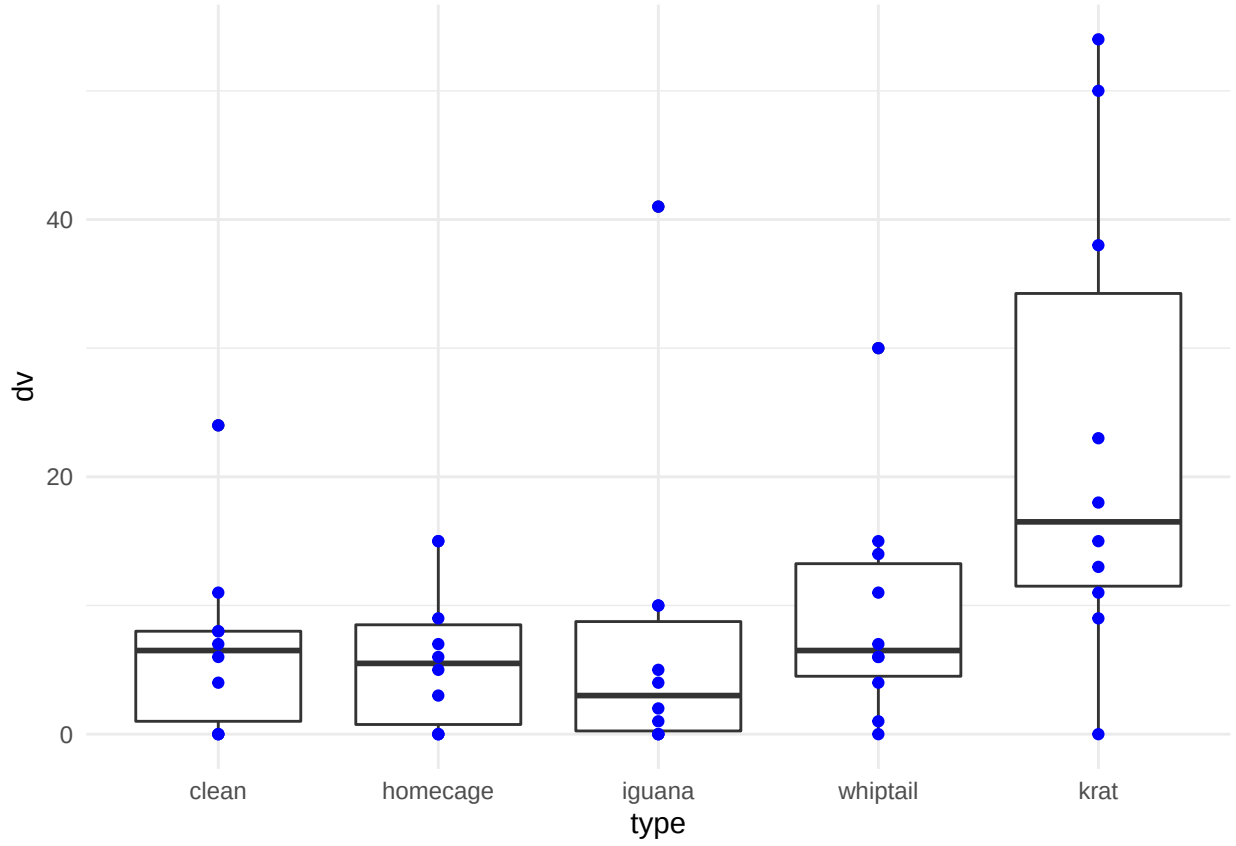


We can also use **ggplot2** to draw boxplots. Here are three different styles.

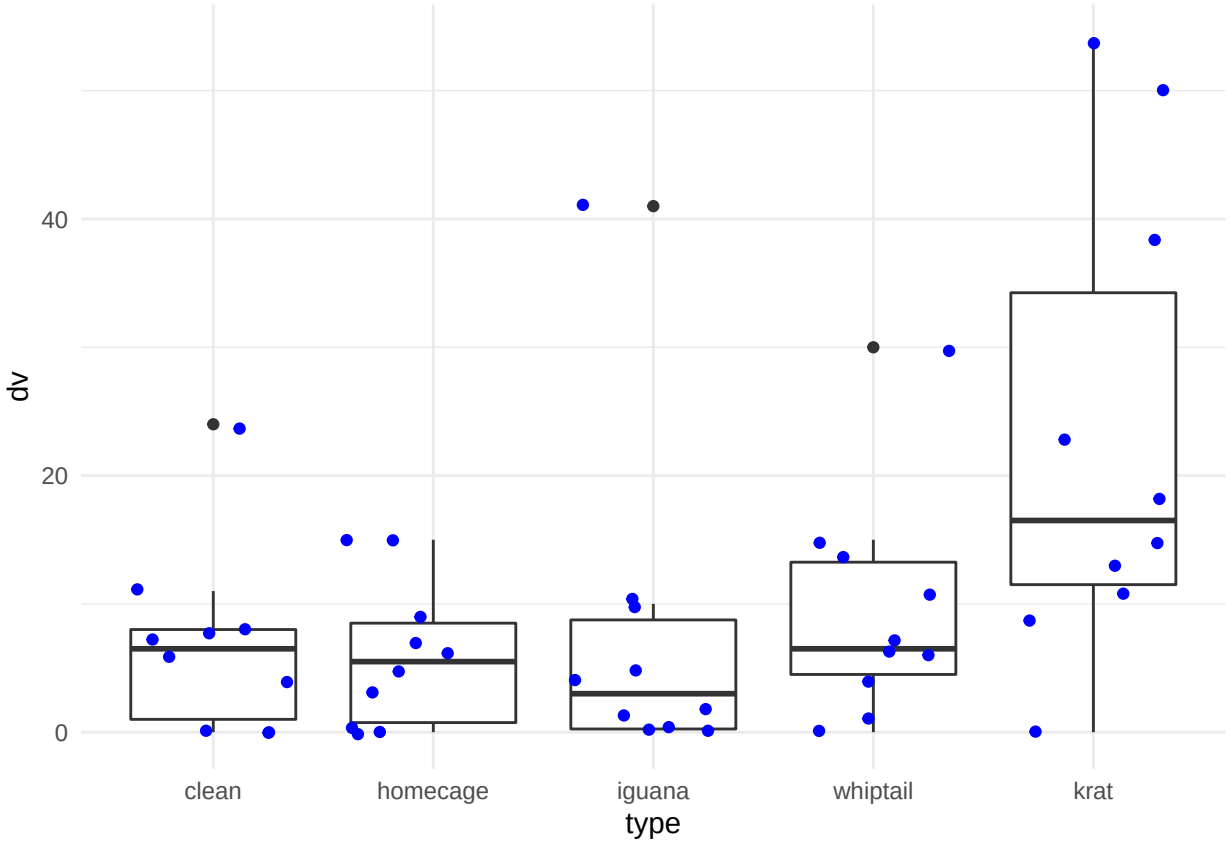
```
# basic plot
ggplot(long1.df, aes(x=type, y=dv)) +
  geom_boxplot() +
  theme_minimal()
```




```
# change color of the data points
ggplot(long1.df, aes(x=type, y=dv)) +
  geom_boxplot() +
  geom_point(color='blue') +
  theme_minimal()
```

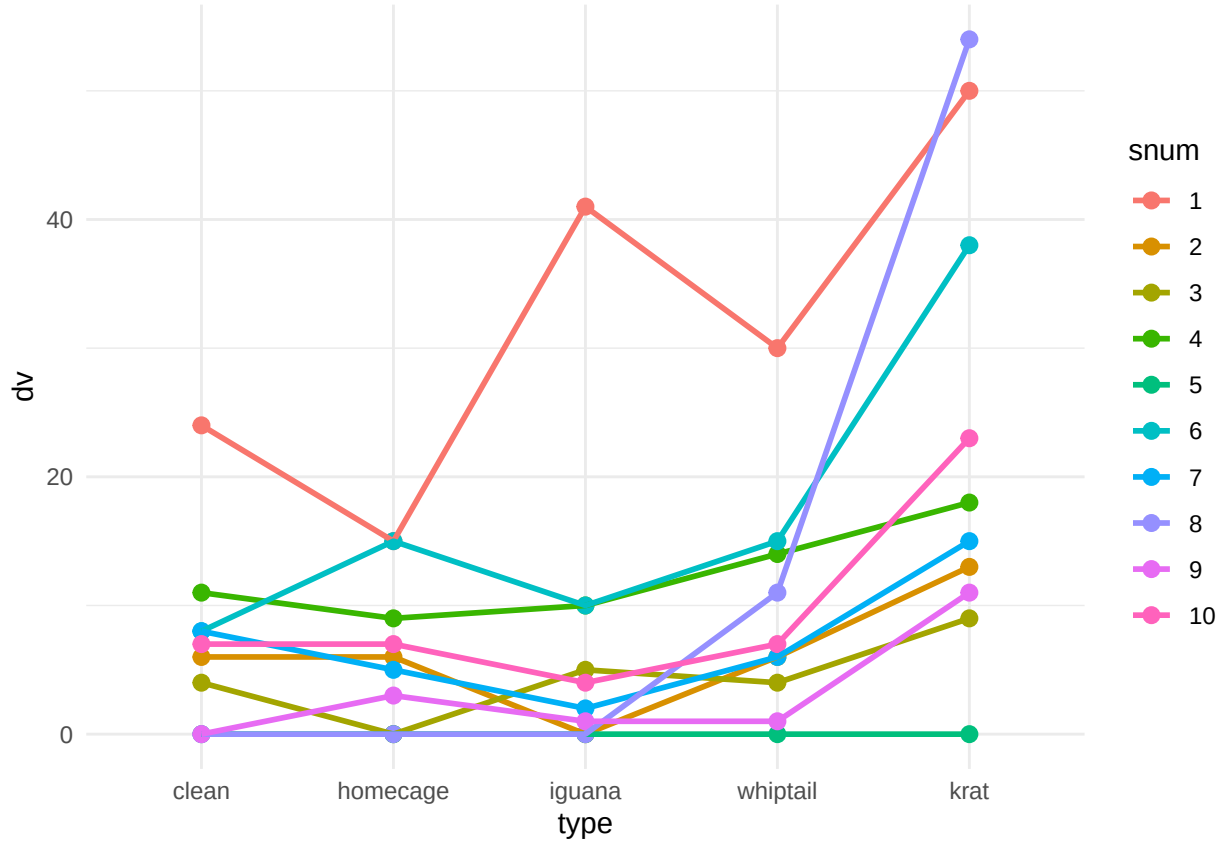



```
# jitter the data points
ggplot(long1.df, aes(x=type, y=dv)) +
  geom_boxplot() +
  geom_jitter(color='blue') +
  theme_minimal()
```



A commonly drawn plot for a design of this type is sometimes called a profile plot. As a line graph, it is probably inappropriate for our categorical IV, but can be useful. It permits comparing the profile of change for each case across the levels of the IV. It would be more appropriate if the repeated measures factor were time. Note that it is a problematic graph in another way - I didn't control the color palette to be color-blind friendly (see the 1-way repeated measures document for an example of choosing a better palette)

```
# Make Profile Plot using ggplot2
library(ggplot2)
ggplot(long1.df, aes(type, dv, colour=snum)) +
    geom_point(size = 2.5) +
    geom_line(aes(group = snum), size = 1) +
  theme_minimal()
```



6.2 Example of 1-factor repeated measures ANOVA using `aov`

The traditional 1-factor repeated measures ANOVA can be viewed as a version of a randomized blocks design where case/subject is the blocking factor. As such, it is just a 2-way layout (treatment by subject). The most rudimentary form of the omnibus analysis can be acheived with the `aov` function. See the accompanying document on 1-factor repeated measures ANOVA for considerably more detailed explanations and additional methods.

Here, the repeated nature aspect of the design is specified with the "Error" argument that indicates the levels of "type" are all found for each snum (subject).

```
fit.1 <- aov(dv~type + Error(snum/type),long1.df)
summary(fit.1)
#report the means of each level
print(model.tables(fit.1,"means"),digits=3)
# report deviation effects for each level
print(model.tables(fit.1,"effects", n=TRUE),digits=3)
```

```
##
## Error: snum
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals  9   3740   415.5
##
## Error: snum:type
##           Df Sum Sq Mean Sq F value   Pr(>F)
## type       4   2042   510.4   8.845 4.42e-05 ***
## Residuals 36   2077    57.7
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Tables of means
## Grand mean
##
## 10.52
##
##  type
## type
##    clean homecage   iguana whiptail     krat
##      6.8      6.0      7.3      9.4     23.1
## Tables of effects
##
##  type
## type
##    clean homecage   iguana whiptail     krat
##    -3.72    -4.52    -3.22    -1.12    12.58
```

6.3 Example of a Linear Mixed Models approach to repeated measures ANOVA

Here is a rudimentary LMM analysis using the `lme` function from the **nlme** package. It also requires the long format data frame. This is just a starting point for LMM modeling with this kind of design. The rudimentary model specified here assumes sphericity and the results for the omnibus F test match what was found with `aov`.

```
# first, default, LMM model assumes sphericity and reproduces  the
# omnibus F test produced as the "average F" in the SPSS MANOVA approach
fit1.lme <- lme(dv ~ type, random = ~1|snum/type, data=long1.df)
anova(fit1.lme)
```

```
##             numDF denDF   F-value p-value
## (Intercept)     1    36 13.317097   8e-04
## type            4    36  8.844723  <.0001
```

```
summary(fit1.lme)
```

```
## Linear mixed-effects model fit by REML
##  Data: long1.df
##        AIC      BIC   logLik
##   357.0839 371.5372 -170.542
##
## Random effects:
##  Formula: ~1 | snum
##         (Intercept)
## StdDev:    8.459511
##
##  Formula: ~1 | type %in% snum
##         (Intercept) Residual
## StdDev:    6.930132 3.110725
##
## Fixed effects: dv ~ type
##                 Value Std.Error DF  t-value p-value
## (Intercept) 10.520000  2.882776 36 3.649260  0.0008
## type.L      11.384200  2.402152 36 4.739167  0.0000
## type.Q       7.964385  2.402152 36 3.315521  0.0021
## type.C       3.004164  2.402152 36 1.250614  0.2191
```

```
## type^4       1.446227  2.402152 36 0.602055  0.5509
##  Correlation:
##        (Intr) type.L type.Q type.C
## type.L 0
## type.Q 0      0
## type.C 0      0      0
## type^4 0      0      0      0
##
## Standardized Within-Group Residuals:
##          Min           Q1          Med           Q3          Max
## -0.756930607 -0.157209242 -0.003407451  0.137647579  1.550657810
##
## Number of Observations: 50
## Number of Groups:
##           snum type %in% snum
##             10             50
```

We can create our own set of orthogonal contrasts for this design and reanalyze. However, the test statistics for the contrasts all have 36 df and are thus not specific to the contrast - they are also potentiall flawed by any non-sphericity.

```
# now create contrasts for the type factor
contrasts.type <- matrix(c(4,-1,-1,-1,-1,
                           0, 3,-1,-1,-1,
                           0, 0,-1,-1, 2,
                           0, 0,-1, 1, 0), ncol = 4)
contrasts(long1.df$type) <- contrasts.type
contrasts(long1.df$type)

fit2.lme <- lme(dv ~ type, random = ~1|snum/type, data=long1.df)
anova(fit2.lme)
summary(fit2.lme)
```

```
##          [,1] [,2] [,3] [,4]
## clean       4    0    0    0
## homecage   -1    3    0    0
## iguana     -1   -1   -1   -1
## whiptail   -1   -1   -1    1
## krat       -1   -1    2    0
##             numDF denDF   F-value p-value
## (Intercept)     1    36 13.317097   8e-04
## type            4    36  8.844723  <.0001
## Linear mixed-effects model fit by REML
##  Data: long1.df
##        AIC      BIC    logLik
##   365.0495 379.5028 -174.5247
##
## Random effects:
##  Formula: ~1 | snum
##         (Intercept)
## StdDev:    8.459511
##
##  Formula: ~1 | type %in% snum
##         (Intercept) Residual
## StdDev:    6.930132 3.110725
```

```
##
## Fixed effects: dv ~ type
##                 Value Std.Error DF   t-value p-value
## (Intercept) 10.520000 2.8827764 36  3.649260  0.0008
## type1       -0.930000 0.5371375 36 -1.731400  0.0919
## type2       -1.816667 0.6934415 36 -2.619784  0.0128
## type3        4.916667 0.9806744 36  5.013557  0.0000
## type4        1.050000 1.6985778 36  0.618164  0.5404
##  Correlation:
##       (Intr) type1 type2 type3
## type1 0
## type2 0      0
## type3 0      0     0
## type4 0      0     0     0
##
## Standardized Within-Group Residuals:
##          Min           Q1          Med           Q3          Max
## -0.756930607 -0.157209242 -0.003407451  0.137647579  1.550657810
##
## Number of Observations: 50
## Number of Groups:
##           snum type %in% snum
##             10             50
```

7 Addendum

Detailed treatment of the repeated measures analyses can be found in the separate document on 1-factor repeated measures analysis.