In this post I am specifically focusing on having many response variables with the same explanatory variables, using purrr::map() and friends for the looping.

**R packages**

I’ll be using **purrr** for looping and will make residual plots with **ggplot2** and **patchwork**. I’ll use **broom** to extract tidy results from models.

library(purrr) # v. 0.3.2

library(ggplot2) # v. 3.2.0

library(patchwork) # v. 0.0.1, github only

library(broom) # v. 0.5.2

**The dataset**

I made a dataset with three response variables, resp, slp, and grad, along with a 2-level explanatory variable group.

dat = structure(list(group = structure(c(1L, 1L, 1L, 1L, 1L, 1L, 1L,

1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,

2L, 2L, 2L, 2L, 2L, 2L, 2L), .Label = c("a", "b"), class = "factor"),

resp = c(10.48, 9.87, 11.1, 8.56, 11.15, 9.53, 8.99, 10.06,

11.02, 10.57, 11.85, 10.11, 9.25, 11.66, 10.72, 8.34, 10.58,

10.47, 9.46, 11.13, 8.35, 9.69, 9.82, 11.47, 9.13, 11.53,

11.05, 11.03, 10.84, 10.22), slp = c(38.27, 46.33, 44.29,

35.57, 34.78, 47.81, 50.45, 46.31, 47.82, 42.07, 31.75, 65.65,

47.42, 41.51, 38.69, 47.84, 46.22, 50.66, 50.69, 44.09, 47.3,

52.53, 53.63, 53.38, 27.34, 51.83, 56.63, 32.99, 77.5, 38.24

), grad = c(0.3, 0.66, 0.57, 0.23, 0.31, 0.48, 0.5, 0.49,

2.41, 0.6, 0.27, 0.89, 2.43, 1.02, 2.17, 1.38, 0.17, 0.47,

1.1, 3.28, 6.14, 3.8, 4.35, 0.85, 1.13, 1.11, 2.93, 1.13,

4.52, 0.13)), class = "data.frame", row.names = c(NA, -30L) )

head(dat)

# group resp slp grad

# 1 a 10.48 38.27 0.30

# 2 a 9.87 46.33 0.66

# 3 a 11.10 44.29 0.57

# 4 a 8.56 35.57 0.23

# 5 a 11.15 34.78 0.31

# 6 a 9.53 47.81 0.48

**A function for model fitting**

The analysis in the example I’m using today amounts to a two-sample t-test. I will fit this as a linear model with lm().

Since the response variable needs to vary among models but the dataset and explanatory variable do not, my function will have a single argument for setting the response variable. Building the model formula in my function ttest\_fun() relies on paste() and as.formula().

ttest\_fun = function(response) {

form = paste(response, "~ group")

lm(as.formula(form), data = dat)

}

This function takes the response variable as a string and returns a model object.

ttest\_fun(response = "resp")

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# 10.3280 -0.1207

**Looping through the response variables**

I’ll make a vector of the response variable names as strings so I can loop through them and fit a model for each one. I pull my response variable names out of the dataset with names(). This step may take more work for you if you have many response variables that aren’t neatly listed all in a row like mine are. 

vars = names(dat)[2:4]

vars

# [1] "resp" "slp" "grad"

I want to keep track of which variable goes with which model. This can be accomplished by naming the vector I’m going to loop through. I name the vector of strings with itself using purrr::set\_names().

vars = set\_names(vars)

vars

# resp slp grad

# "resp" "slp" "grad"

Now I’m ready to loop through the variables and fit a model for each one with purrr::map(). Since my function takes a single argument, the response variable, I can list the function by name within map() without using a formula (~) or an anonymous function.

models = vars %>%

map(ttest\_fun)

The output is a list containing three models, one for each response variable. Notice that the output list is a *named* list, where the names of each list element is the response variable used in that model. This is the reason I took the time to name the response variable vector.

models

# $resp

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# 10.3280 -0.1207

#

#

# $slp

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# 43.91 4.81

#

#

# $grad

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# 0.8887 1.2773

Note I could have done the set\_names() step within the pipe chain rather than as a separate step.

vars %>%

set\_names() %>%

map(ttest\_fun)

**Create residual plots for each model**

I’m working with a simple model fitting function, where the output only contains the fitted model. To extract other output I can loop through the list of models in a separate step. An alternative is to create all the output within the modeling function and then pull whatever you want out of the list of results.

In this case, my next step is to loop through the models and make residual plots. I want to look at a residuals vs fitted values plot as well as a plot to look at residual normality (like a boxplot, a histogram, or a quantile-quantile normal plot). In more complicated models I might also make plots of residuals vs explanatory variables.

I’ll make a function to build the two residuals plots. My function takes a model and the model name as arguments. I extract residuals and fitted values via broom::augment() and make the two plots with **ggplot2** functions. I combine the plots via **patchwork**. I add a title to the combined plot with the name of the response variable from each model to help me keep track of things.

resid\_plots = function(model, modelname) {

output = augment(model)

res.v.fit = ggplot(output, aes(x = .fitted, y = .resid) ) +

geom\_point() +

theme\_bw(base\_size = 16)

res.box = ggplot(output, aes(x = "", y = .resid) ) +

geom\_boxplot() +

theme\_bw(base\_size = 16) +

labs(x = NULL)

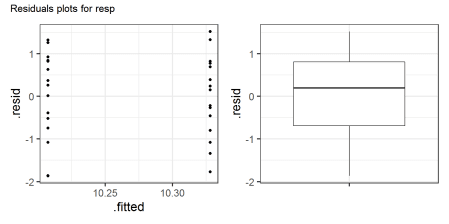
res.v.fit + res.box +

plot\_annotation(title = paste("Residuals plots for", modelname) )

}

The output of this function is a combined plot of the residuals. Here is an example for one model (printed at 8″ wide by 4″ tall).

resid\_plots(model = models[[1]], modelname = names(models)[1])



I can use purrr::imap() to loop through all models and the model names simultaneously to make the plots with the title for each variable.

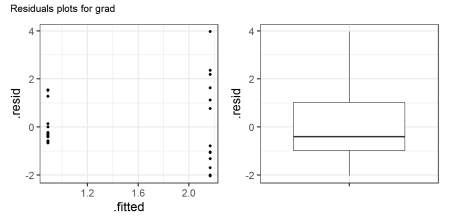
residplots = imap(models, resid\_plots)

**Examining the plots**

In a situation where I have many response variables, I like to save my plots out into a PDF so I can easily page through them outside of R.

Since I have only a few plots I can print them in R. The last plot, shown below, looks potentially problematic. I see the variance increasing with the mean and right skew in the residuals.

residplots[[3]]



**Re-fitting a model**

If you find a problematic model fit you’ll need to spend some time working with that variable to find a more appropriate model.

Once you have a model you’re happy with, you can manually add the new model to the list (if needed). In my example, let’s say the grad model needed a log transformation.

gradmod = ttest\_fun("log(grad)")

If I’m happy with the fit of the new model I add it to the list with the other models to automate extracting results.

models$log\_grad = gradmod

I remove the original model by setting it to NULL. I don’t want any results from that model and if I leave it in I know I’ll ultimately get confused about which model is the final model. 

models$grad = NULL

Now the output list has three models, with the new log\_grad model and the old grad model removed.

models

# $resp

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# 10.3280 -0.1207

#

#

# $slp

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# 43.91 4.81

#

#

# $log\_grad

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# -0.4225 0.7177

I could have removed models from the list via subsetting by name. Here’s an example, showing what the list looks like if I remove the slp model.

models[!names(models) %in% "slp"]

# $resp

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# 10.3280 -0.1207

#

#

# $log\_grad

#

# Call:

# lm(formula = as.formula(form), data = dat)

#

# Coefficients:

# (Intercept) groupb

# -0.4225 0.7177

**Getting model results**

Once you are happy with model fit of all models it’s time to extract any output of interest. For a t-test we would commonly want the estimated difference between the two groups, which is in the summary() output. I’ll pull this information from the model as a data.frame with broom::tidy(). This returns the estimated coefficients, statistical tests, and (optionally) confidence intervals for coefficients

I switch to map\_dfr() for looping to get the output combined into a single data.frame. I use the .id argument to add the response variable name to the output dataset.

Since some of the response variables are log transformed, it would make sense to back-transform coefficients in this step. I don’t show this here, but would likely approach this using an if() statement based on log transformed variables containing "log" in their names.

res\_anova = map\_dfr(models, tidy, conf.int = TRUE, .id = "variable")

res\_anova

# # A tibble: 6 x 8

# variable term estimate std.error statistic p.value conf.low conf.high

#

# 1 resp (Inter~ 10.3 0.260 39.7 3.60e-26 9.80 10.9

# 2 resp groupb -0.121 0.368 -0.328 7.45e- 1 -0.874 0.632

# 3 slp (Inter~ 43.9 2.56 17.2 2.18e-16 38.7 49.2

# 4 slp groupb 4.81 3.62 1.33 1.95e- 1 -2.61 12.2

# 5 log\_grad (Inter~ -0.423 0.255 -1.66 1.09e- 1 -0.945 0.0997

# 6 log\_grad groupb 0.718 0.361 1.99 5.64e- 2 -0.0208 1.46

The primary interest in this output would be in the groupb row for each variable. Since the output is a data frame (thanks broom::tidy()!) you can use standard data manipulation tools to pull out only rows and columns of interest.

Other output, like, e.g., estimated marginal means for more complicated models, can be extracted and saved in a similar way.

**Alternative approach to fitting many models**

When I am working with many response variables with widely varying ranges, it feels most natural to me to keep the different variables in different columns and loop through them as I have shown above. However, a reasonable alternative is to *reshape* your dataset so all the values of all variables are in a single column. A second, categorical column will contain the variable names so we know which variable each row is associated with. Such reshaping is an example of making a *wide* dataset into a *long* dataset.

**Just the code, please**

Here’s the code without all the discussion. Copy and paste the code below or you can download an R script of uncommented code [from here](https://aosmith.rbind.io/script/2019-07-22-automate-model-fitting-with-loops.R).

library(purrr) # v. 0.3.2

library(ggplot2) # v. 3.2.0

library(patchwork) # v. 0.0.1, github only

library(broom) # v. 0.5.2

dat = structure(list(group = structure(c(1L, 1L, 1L, 1L, 1L, 1L, 1L,

1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,

2L, 2L, 2L, 2L, 2L, 2L, 2L), .Label = c("a", "b"), class = "factor"),

resp = c(10.48, 9.87, 11.1, 8.56, 11.15, 9.53, 8.99, 10.06,

11.02, 10.57, 11.85, 10.11, 9.25, 11.66, 10.72, 8.34, 10.58,

10.47, 9.46, 11.13, 8.35, 9.69, 9.82, 11.47, 9.13, 11.53,

11.05, 11.03, 10.84, 10.22), slp = c(38.27, 46.33, 44.29,

35.57, 34.78, 47.81, 50.45, 46.31, 47.82, 42.07, 31.75, 65.65,

47.42, 41.51, 38.69, 47.84, 46.22, 50.66, 50.69, 44.09, 47.3,

52.53, 53.63, 53.38, 27.34, 51.83, 56.63, 32.99, 77.5, 38.24

), grad = c(0.3, 0.66, 0.57, 0.23, 0.31, 0.48, 0.5, 0.49,

2.41, 0.6, 0.27, 0.89, 2.43, 1.02, 2.17, 1.38, 0.17, 0.47,

1.1, 3.28, 6.14, 3.8, 4.35, 0.85, 1.13, 1.11, 2.93, 1.13,

4.52, 0.13)), class = "data.frame", row.names = c(NA, -30L) )

head(dat)

ttest\_fun = function(response) {

form = paste(response, "~ group")

lm(as.formula(form), data = dat)

}

ttest\_fun(response = "resp")

vars = names(dat)[2:4]

vars

vars = set\_names(vars)

vars

models = vars %>%

map(ttest\_fun)

models

vars %>%

set\_names() %>%

map(ttest\_fun)

resid\_plots = function(model, modelname) {

output = augment(model)

res.v.fit = ggplot(output, aes(x = .fitted, y = .resid) ) +

geom\_point() +

theme\_bw(base\_size = 16)

res.box = ggplot(output, aes(x = "", y = .resid) ) +

geom\_boxplot() +

theme\_bw(base\_size = 16) +

labs(x = NULL)

res.v.fit + res.box +

plot\_annotation(title = paste("Residuals plots for", modelname) )

}

resid\_plots(model = models[[1]], modelname = names(models)[1])

residplots = imap(models, resid\_plots)

residplots[[3]]

gradmod = ttest\_fun("log(grad)")

models$log\_grad = gradmod

models$grad = NULL

models

models[!names(models) %in% "slp"]

res\_anova = map\_dfr(models, tidy, conf.int = TRUE, .id = "variable")

res\_anova