This blog post is an excerpt of my ebook Modern R with the tidyverse that you can read for free  
[here](https://b-rodrigues.github.io/modern_R/functional-programming.html#modeling-with-functional-programming).  
This is taken from Chapter 8, in which I discuss advanced functional programming methods for  
modeling.

As written just above (note: as written above *in the book*), map() simply applies a function  
to a list of inputs, and in the previous  
section we mapped ggplot() to generate many plots at once. This approach can also be used to  
map any modeling functions, for instance lm() to a list of datasets.

For instance, suppose that you wish to perform a Monte Carlo simulation. Suppose that you are  
dealing with a binary choice problem; usually, you would use a logistic regression for this.

However, in certain disciplines, especially in the social sciences, the so-called Linear Probability  
Model is often used as well. The LPM is a simple linear regression, but unlike the standard setting  
of a linear regression, the dependent variable, or target, is a binary variable, and not a continuous  
variable. Before you yell “Wait, that’s illegal”, you should know that in practice LPMs do a good  
job of estimating marginal effects, which is what social scientists and econometricians are often  
interested in. Marginal effects are another way of interpreting models, giving how the outcome  
(or the target) changes given a change in a independent variable (or a feature). For instance,  
a marginal effect of 0.10 for age would mean that probability of success would increase by 10% for  
each added year of age.

There has been a lot of discussion on logistic regression vs LPMs, and there are pros and cons  
of using LPMs. Micro-econometricians are still fond of LPMs, even though the pros of LPMs are  
not really convincing. However, quoting Angrist and Pischke:

“While a nonlinear model may fit the CEF (population conditional expectation function) for LDVs  
(limited dependent variables) more closely than a linear model, when it comes to marginal effects,  
this probably matters little” (source: *Mostly Harmless Econometrics*)

so LPMs are still used for estimating marginal effects.

Let us check this assessment with one example. First, we simulate some data, then  
run a logistic regression and compute the marginal effects, and then compare with a LPM:

set.seed(1234)

x1 <- rnorm(100)

x2 <- rnorm(100)

z <- .5 + 2\*x1 + 4\*x2

p <- 1/(1 + exp(-z))

y <- rbinom(100, 1, p)

df <- tibble(y = y, x1 = x1, x2 = x2)

This data generating process generates data from a binary choice model. Fitting the model using a  
logistic regression allows us to recover the structural parameters:

logistic\_regression <- glm(y ~ ., data = df, family = binomial(link = "logit"))

Let’s see a summary of the model fit:

summary(logistic\_regression)

##

## Call:

## glm(formula = y ~ ., family = binomial(link = "logit"), data = df)

##

## Deviance Residuals:

## Min 1Q Median 3Q Max

## -2.91941 -0.44872 0.00038 0.42843 2.55426

##

## Coefficients:

## Estimate Std. Error z value Pr(>|z|)

## (Intercept) 0.0960 0.3293 0.292 0.770630

## x1 1.6625 0.4628 3.592 0.000328 \*\*\*

## x2 3.6582 0.8059 4.539 5.64e-06 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## (Dispersion parameter for binomial family taken to be 1)

##

## Null deviance: 138.629 on 99 degrees of freedom

## Residual deviance: 60.576 on 97 degrees of freedom

## AIC: 66.576

##

## Number of Fisher Scoring iterations: 7

We do recover the parameters that generated the data, but what about the marginal effects? We can  
get the marginal effects easily using the {margins} package:

library(margins)

margins(logistic\_regression)

## Average marginal effects

## glm(formula = y ~ ., family = binomial(link = "logit"), data = df)

## x1 x2

## 0.1598 0.3516

Or, even better, we can compute the *true* marginal effects, since we know the data  
generating process:

meffects <- function(dataset, coefs){

X <- dataset %>%

select(-y) %>%

as.matrix()

dydx\_x1 <- mean(dlogis(X%\*%c(coefs[2], coefs[3]))\*coefs[2])

dydx\_x2 <- mean(dlogis(X%\*%c(coefs[2], coefs[3]))\*coefs[3])

tribble(~term, ~true\_effect,

"x1", dydx\_x1,

"x2", dydx\_x2)

}

(true\_meffects <- meffects(df, c(0.5, 2, 4)))

## # A tibble: 2 x 2

## term true\_effect

##

## 1 x1 0.175

## 2 x2 0.350

Ok, so now what about using this infamous Linear Probability Model to estimate the marginal effects?

lpm <- lm(y ~ ., data = df)

summary(lpm)

##

## Call:

## lm(formula = y ~ ., data = df)

##

## Residuals:

## Min 1Q Median 3Q Max

## -0.83953 -0.31588 -0.02885 0.28774 0.77407

##

## Coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 0.51340 0.03587 14.314 < 2e-16 \*\*\*

## x1 0.16771 0.03545 4.732 7.58e-06 \*\*\*

## x2 0.31250 0.03449 9.060 1.43e-14 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Residual standard error: 0.3541 on 97 degrees of freedom

## Multiple R-squared: 0.5135, Adjusted R-squared: 0.5034

## F-statistic: 51.18 on 2 and 97 DF, p-value: 6.693e-16

It’s not too bad, but maybe it could have been better in other circumstances. Perhaps if we had more  
observations, or perhaps for a different set of structural parameters the results of the LPM  
would have been closer. The LPM estimates the marginal effect of x1 to be  
0.1677134 vs 0.1597956  
for the logistic regression and for x2, the LPM estimation is 0.3124966  
vs 0.351607. The *true* marginal effects are  
0.1750963 and 0.3501926 for x1 and x2 respectively.

Just as to assess the accuracy of a model data scientists perform cross-validation, a Monte Carlo  
study can be performed to asses how close the estimation of the marginal effects using a LPM is  
to the marginal effects derived from a logistic regression. It will allow us to test with datasets  
of different sizes, and generated using different structural parameters.

First, let’s write a function that generates data. The function below generates 10 datasets of size  
100 (the code is inspired by this [StackExchange answer](https://stats.stackexchange.com/a/46525)):

generate\_datasets <- function(coefs = c(.5, 2, 4), sample\_size = 100, repeats = 10){

generate\_one\_dataset <- function(coefs, sample\_size){

x1 <- rnorm(sample\_size)

x2 <- rnorm(sample\_size)

z <- coefs[1] + coefs[2]\*x1 + coefs[3]\*x2

p <- 1/(1 + exp(-z))

y <- rbinom(sample\_size, 1, p)

df <- tibble(y = y, x1 = x1, x2 = x2)

}

simulations <- rerun(.n = repeats, generate\_one\_dataset(coefs, sample\_size))

tibble("coefs" = list(coefs), "sample\_size" = sample\_size, "repeats" = repeats, "simulations" = list(simulations))

}

Let’s first generate one dataset:

one\_dataset <- generate\_datasets(repeats = 1)

Let’s take a look at one\_dataset:

one\_dataset

## # A tibble: 1 x 4

## coefs sample\_size repeats simulations

##

## 1 100 1

As you can see, the tibble with the simulated data is inside a list-column called simulations.  
Let’s take a closer look:

str(one\_dataset$simulations)

## List of 1

## $ :List of 1

## ..$ :Classes 'tbl\_df', 'tbl' and 'data.frame': 100 obs. of 3 variables:

## .. ..$ y : int [1:100] 0 1 1 1 0 1 1 0 0 1 ...

## .. ..$ x1: num [1:100] 0.437 1.06 0.452 0.663 -1.136 ...

## .. ..$ x2: num [1:100] -2.316 0.562 -0.784 -0.226 -1.587 ...

The structure is quite complex, and it’s important to understand this, because it will have an  
impact on the next lines of code; it is a list, containing a list, containing a dataset! No worries  
though, we can still map over the datasets directly, by using modify\_depth() instead of map().

Now, let’s fit a LPM and compare the estimation of the marginal effects with the *true* marginal  
effects. In order to have some confidence in our results,  
we will not simply run a linear regression on that single dataset, but will instead simulate hundreds,  
then thousands and ten of thousands of data sets, get the marginal effects and compare  
them to the true ones (but here I won’t simulate more than 500 datasets).

Let’s first generate 10 datasets:

many\_datasets <- generate\_datasets()

Now comes the tricky part. I have this object, many\_datasets looking like this:

many\_datasets

## # A tibble: 1 x 4

## coefs sample\_size repeats simulations

##

## 1 100 10

I would like to fit LPMs to the 10 datasets. For this, I will need to use all the power of functional  
programming and the {tidyverse}. I will be adding columns to this data frame using mutate()  
and mapping over the simulations list-column using modify\_depth(). The list of data frames is  
at the second level (remember, it’s a list containing a list containing data frames).

I’ll start by fitting the LPMs, then using broom::tidy() I will get a nice data frame of the  
estimated parameters. I will then only select what I need, and then bind the rows of all the  
data frames. I will do the same for the *true* marginal effects.

I highly suggest that you run the following lines, one after another. It is complicated to understand  
what’s going on if you are not used to such workflows. However, I hope to convince you that once  
it will click, it’ll be much more intuitive than doing all this inside a loop. Here’s the code:

results <- many\_datasets %>%

mutate(lpm = modify\_depth(simulations, 2, ~lm(y ~ ., data = .x))) %>%

mutate(lpm = modify\_depth(lpm, 2, broom::tidy)) %>%

mutate(lpm = modify\_depth(lpm, 2, ~select(., term, estimate))) %>%

mutate(lpm = modify\_depth(lpm, 2, ~filter(., term != "(Intercept)"))) %>%

mutate(lpm = map(lpm, bind\_rows)) %>%

mutate(true\_effect = modify\_depth(simulations, 2, ~meffects(., coefs = coefs[[1]]))) %>%

mutate(true\_effect = map(true\_effect, bind\_rows))

This is how results looks like:

results

## # A tibble: 1 x 6

## coefs sample\_size repeats simulations lpm true\_effect

##

## 1 100 10

Let’s take a closer look to the lpm and true\_effect columns:

results$lpm

## [[1]]

## # A tibble: 20 x 2

## term estimate

##

## 1 x1 0.228

## 2 x2 0.353

## 3 x1 0.180

## 4 x2 0.361

## 5 x1 0.165

## 6 x2 0.374

## 7 x1 0.182

## 8 x2 0.358

## 9 x1 0.125

## 10 x2 0.345

## 11 x1 0.171

## 12 x2 0.331

## 13 x1 0.122

## 14 x2 0.309

## 15 x1 0.129

## 16 x2 0.332

## 17 x1 0.102

## 18 x2 0.374

## 19 x1 0.176

## 20 x2 0.410

results$true\_effect

## [[1]]

## # A tibble: 20 x 2

## term true\_effect

##

## 1 x1 0.183

## 2 x2 0.366

## 3 x1 0.166

## 4 x2 0.331

## 5 x1 0.174

## 6 x2 0.348

## 7 x1 0.169

## 8 x2 0.339

## 9 x1 0.167

## 10 x2 0.335

## 11 x1 0.173

## 12 x2 0.345

## 13 x1 0.157

## 14 x2 0.314

## 15 x1 0.170

## 16 x2 0.340

## 17 x1 0.182

## 18 x2 0.365

## 19 x1 0.161

## 20 x2 0.321

Let’s bind the columns, and compute the difference between the *true* and estimated marginal  
effects:

simulation\_results <- results %>%

mutate(difference = map2(.x = lpm, .y = true\_effect, bind\_cols)) %>%

mutate(difference = map(difference, ~mutate(., difference = true\_effect - estimate))) %>%

mutate(difference = map(difference, ~select(., term, difference))) %>%

pull(difference) %>%

.[[1]]

Let’s take a look at the simulation results:

simulation\_results %>%

group\_by(term) %>%

summarise(mean = mean(difference),

sd = sd(difference))

## # A tibble: 2 x 3

## term mean sd

##

## 1 x1 0.0122 0.0370

## 2 x2 -0.0141 0.0306

Already with only 10 simulated datasets, the difference in means is not significant. Let’s rerun  
the analysis, but for difference sizes. In order to make things easier, we can put all the code  
into a nifty function:

monte\_carlo <- function(coefs, sample\_size, repeats){

many\_datasets <- generate\_datasets(coefs, sample\_size, repeats)

results <- many\_datasets %>%

mutate(lpm = modify\_depth(simulations, 2, ~lm(y ~ ., data = .x))) %>%

mutate(lpm = modify\_depth(lpm, 2, broom::tidy)) %>%

mutate(lpm = modify\_depth(lpm, 2, ~select(., term, estimate))) %>%

mutate(lpm = modify\_depth(lpm, 2, ~filter(., term != "(Intercept)"))) %>%

mutate(lpm = map(lpm, bind\_rows)) %>%

mutate(true\_effect = modify\_depth(simulations, 2, ~meffects(., coefs = coefs[[1]]))) %>%

mutate(true\_effect = map(true\_effect, bind\_rows))

simulation\_results <- results %>%

mutate(difference = map2(.x = lpm, .y = true\_effect, bind\_cols)) %>%

mutate(difference = map(difference, ~mutate(., difference = true\_effect - estimate))) %>%

mutate(difference = map(difference, ~select(., term, difference))) %>%

pull(difference) %>%

.[[1]]

simulation\_results %>%

group\_by(term) %>%

summarise(mean = mean(difference),

sd = sd(difference))

}

And now, let’s run the simulation for different parameters and sizes:

monte\_carlo(c(.5, 2, 4), 100, 10)

## # A tibble: 2 x 3

## term mean sd

##

## 1 x1 -0.00826 0.0291

## 2 x2 -0.00732 0.0412

monte\_carlo(c(.5, 2, 4), 100, 100)

## # A tibble: 2 x 3

## term mean sd

##

## 1 x1 0.00360 0.0392

## 2 x2 0.00517 0.0446

monte\_carlo(c(.5, 2, 4), 100, 500)

## # A tibble: 2 x 3

## term mean sd

##

## 1 x1 -0.00152 0.0371

## 2 x2 -0.000701 0.0423

monte\_carlo(c(pi, 6, 9), 100, 10)

## # A tibble: 2 x 3

## term mean sd

##

## 1 x1 -0.00829 0.0546

## 2 x2 0.00178 0.0370

monte\_carlo(c(pi, 6, 9), 100, 100)

## # A tibble: 2 x 3

## term mean sd

##

## 1 x1 0.0107 0.0608

## 2 x2 0.00831 0.0804

monte\_carlo(c(pi, 6, 9), 100, 500)

## # A tibble: 2 x 3

## term mean sd

##

## 1 x1 0.00879 0.0522

## 2 x2 0.0113 0.0668

We see that, at least for this set of parameters, the LPM does a good job of estimating marginal  
effects.

Now, this study might in itself not be very interesting to you, but I believe the general approach  
is quite useful and flexible enough to be adapted to all kinds of use-cases.