Panel Data and Optimization with R

Fan Wang

2020-04-12

Contents

Preface

This is a work-in-progress website consisting of R panel data and optimization examples for Statistics/Econometrics/Economic Analysis. Materials gathered from various projects in which R code is used. Files are from **Fan**'s R4Econ repository. This is not a R package, but a list of examples in PDF/HTML/Rmd formats. REconTools is a package that can be installed with tools used in projects involving R.

Bullet points show which base R, tidyverse or other functions/commands are used to achieve various objectives. An effort is made to use only base R (R Core Team, 2019) and tidyverse (Wickham, 2019) packages whenever possible to reduce dependencies. The goal of this repository is to make it easier to find/re-use codes produced for various projects. Some functions also rely on or correspond to functions from REconTools (Wang, 2020).

From Fan's other repositories: For dynamic borrowing and savings problems, see Dynamic Asset Repository; For code examples, see also Matlab Example Code and Stata Example Code; For intro econ with Matlab, see Intro Mathematics for Economists, and for intro stat with R, see Intro Statistics for Undergraduates. See here for all of Fan's public repositories.

The site is built using Bookdown (Xie, 2020).

Please contact FanWangEcon for issues or problems.

Chapter 1

Array, Matrix, Dataframe

1.1 List

1.1.1 Multiple Dimensional List

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

- r list tutorial
- r vector vs list
- r initialize empty multiple element list
- r name rows and columns of 2 dimensional list
- r row and colum names of list
- list dimnames

1.1.1.1 One Dimensional Named List

1. define list
2. slice list

```
# Define Lists
ls_num <- list(1,2,3)
ls_str <- list('1','2','3')
ls_num_str <- list(1,2,'3')

# Named Lists
ar_st_names <- c('e1','e2','e3')
ls_num_str_named <- ls_num_str
names(ls_num_str_named) <- ar_st_names

# Add Element to Named List
ls_num_str_named$e4 <- 'this is added'

# display
print(paste0('ls_num:', str(ls_num)))
```

```
## List of 3
##  $ : num 1
##  $ : num 2
##  $ : num 3
## [1] "ls_num:"
```

```
print(paste0('ls_num[2:3]:', str(ls_num[2:3])))
```

```
## List of 2
```

```
##  $ : num 2
##  $ : num 3
## [1] "ls_num[2:3]:"
```

```
print(paste0('ls_str:', str(ls_str)))
```

```
## List of 3
##  $ : chr "1"
##  $ : chr "2"
##  $ : chr "3"
## [1] "ls_str:"
```

```
print(paste0('ls_str[2:3]:', str(ls_str[2:3])))
```

```
## List of 2
##  $ : chr "2"
##  $ : chr "3"
## [1] "ls_str[2:3]:"
```

```
print(paste0('ls_num_str:', str(ls_num_str)))
```

```
## List of 3
##  $ : num 1
##  $ : num 2
##  $ : chr "3"
## [1] "ls_num_str:"
```

```
print(paste0('ls_num_str[2:4]:', str(ls_num_str[2:4])))
```

```
## List of 3
##  $ : num 2
##  $ : chr "3"
##  $ : NULL
## [1] "ls_num_str[2:4]:"
```

```
print(paste0('ls_num_str_named:', str(ls_num_str_named)))
```

```
## List of 4
##  $ e1: num 1
##  $ e2: num 2
##  $ e3: chr "3"
##  $ e4: chr "this is added"
## [1] "ls_num_str_named:"
```

```
print(paste0('ls_num_str_named[c(\'e2\',\'e3\',\'e4\')]', str(ls_num_str_named[c('e2','e3','e4')])))
```

```
## List of 3
##  $ e2: num 2
##  $ e3: chr "3"
##  $ e4: chr "this is added"
## [1] "ls_num_str_named[c('e2','e3','e4')]"
```

1.1.1.2 Two Dimensional Unnamed List

Generate a multiple dimensional list:

1. Initiate with an N element empty list
2. Reshape list to M by Q
3. Fill list elements
4. Get list element by row and column number

List allows for different data types to be stored together.

Note that element specific names in named list are not preserved when the list is reshaped to be two dimensional. Two dimensional list, however, could have row and column names.

```
# Dimensions
it_M <- 2
it_Q <- 3
it_N <- it_M*it_Q

# Initiate an Empty MxQ=N element list
ls_2d_flat <- vector(mode = "list", length = it_N)
ls_2d <- ls_2d_flat

# Named flat
ls_2d_flat_named <- ls_2d_flat
names(ls_2d_flat_named) <- paste0('e',seq(1,it_N))
ls_2d_named <- ls_2d_flat_named

# Reshape
dim(ls_2d) <- c(it_M, it_Q)
# named 2d list can not carry 1d name after reshape
dim(ls_2d_named) <- c(it_M, it_Q)

# display
print('ls_2d_flat')
```

```
## [1] "ls_2d_flat"
```

```
print(ls_2d_flat)
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
```

```
print('ls_2d_flat_named')
```

```
## [1] "ls_2d_flat_named"
```

```
print(ls_2d_flat_named)
```

```
## $e1
## NULL
##
## $e2
## NULL
##
## $e3
## NULL
```

```
##
## $e4
## NULL
##
## $e5
## NULL
##
## $e6
## NULL
```

```
print('ls_2d')
```

```
## [1] "ls_2d"
```

```
print(ls_2d)
```

```
##      [,1] [,2] [,3]
## [1,] NULL NULL NULL
## [2,] NULL NULL NULL
```

```
print('ls_2d_named')
```

```
## [1] "ls_2d_named"
```

```
print(ls_2d_named)
```

```
##      [,1] [,2] [,3]
## [1,] NULL NULL NULL
## [2,] NULL NULL NULL
```

```
# Select Values, double bracket to select from 2dim list
print('ls_2d[[1,2]]')
```

```
## [1] "ls_2d[[1,2]]"
```

```
print(ls_2d[[1,2]])
```

```
## NULL
```

1.1.1.3 Define Two Dimensional Named LIst

For naming two dimensional lists, *rowname* and *colname* does not work. Rather, we need to use *dimnames*. Note that in addition to dimnames, we can continue to have element specific names. Both can co-exist. But note that the element specific names are not preserved after dimension transform, so need to be redefined afterwards.

How to select an element of a two dimensional list:

1. row and column names: *dimnames*, *ls_2d_flat_named[['row2','col2']]*
2. named elements: *names*, *ls_2d_flat_named[['e5']]*
3. select by index: *index*, *ls_2d_flat_named[[5]]*

Neither *dimnames* nor *names* are required, but both can be used to select elements.

```
# Dimensions
it_M <- 3
it_Q <- 4
it_N <- it_M*it_Q

# Initiate an Empty MxQ=N element list
ls_2d_flat_named <- vector(mode = "list", length = it_N)
dim(ls_2d_flat_named) <- c(it_M, it_Q)

# Fill with values
for (it_Q_ctr in seq(1,it_Q)) {
```

```
  for (it_M_ctr in seq(1,it_M)) {
    # linear index
    ls_2d_flat_named[[it_M_ctr, it_Q_ctr]] <- (it_Q_ctr-1)*it_M+it_M_ctr
  }
}

# Replace row names, note rownames does not work
dimnames(ls_2d_flat_named)[[1]] <- paste0('row',seq(1,it_M))
dimnames(ls_2d_flat_named)[[2]] <- paste0('col',seq(1,it_Q))

# Element Specific Names
names(ls_2d_flat_named) <- paste0('e',seq(1,it_N))

# These are not element names, can still name each element
# display
print('ls_2d_flat_named')
```

```
## [1] "ls_2d_flat_named"
```

```
print(ls_2d_flat_named)
```

```
##      col1 col2 col3 col4
## row1 1    4    7    10
## row2 2    5    8    11
## row3 3    6    9    12
## attr(,"names")
##  [1] "e1"  "e2"  "e3"  "e4"  "e5"  "e6"  "e7"  "e8"  "e9"  "e10" "e11" "e12"
```

```
print('str(ls_2d_flat_named)')
```

```
## [1] "str(ls_2d_flat_named)"
```

```
print(str(ls_2d_flat_named))
```

```
## List of 12
##  $ e1 : num 1
##  $ e2 : num 2
##  $ e3 : num 3
##  $ e4 : num 4
##  $ e5 : num 5
##  $ e6 : num 6
##  $ e7 : num 7
##  $ e8 : num 8
##  $ e9 : num 9
##  $ e10: num 10
##  $ e11: num 11
##  $ e12: num 12
##  - attr(*, "dim")= int [1:2] 3 4
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:3] "row1" "row2" "row3"
##   ..$ : chr [1:4] "col1" "col2" "col3" "col4"
## NULL
```

```
# Select elements with with dimnames
print('ls_2d_flat_named[[\'row2\',\'col2\']]')
```

```
## [1] "ls_2d_flat_named[['row2','col2']]"
```

```
print(ls_2d_flat_named[['row2','col2']])
```

```
## [1] 5
```

```
# Select elements with element names
print('ls_2d_flat_named[[\'e5\']]')
```

```
## [1] "ls_2d_flat_named[['e5']]"
```

```
print(ls_2d_flat_named[['e5']])
```

```
## [1] 5
```

```
# Select elements with index
print('ls_2d_flat_named[[5]]')
```

```
## [1] "ls_2d_flat_named[[5]]"
```

```
print(ls_2d_flat_named[[5]])
```

```
## [1] 5
```

1.2 Array

1.2.1 Array Basics

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.2.1.1 Multidimesional Arrays

```
# Multidimensional Array
# 1 is r1c1t1, 1.5 in r2c1t1, 0 in r1c2t1, etc.
# Three dimensions, row first, column second, and tensor third
x <- array(c(1, 1.5, 0, 2, 0, 4, 0, 3), dim=c(2, 2, 2))
dim(x)
```

1.2.1.1.1 Generate 2 Dimensional Array

```
## [1] 2 2 2
```

```
print(x)
```

```
## , , 1
##
##      [,1] [,2]
## [1,]  1.0    0
## [2,]  1.5    2
##
## , , 2
##
##      [,1] [,2]
## [1,]    0    0
## [2,]    4    3
```

1.2.1.2 Array Slicing

```
# Remove last element of array
vars.group.bydf <- c('23','dfa', 'wer')
vars.group.bydf[-length(vars.group.bydf)]
```

1.2.1.2.1 Remove Elements of Array

```
## [1] "23"  "dfa"
```

1.2.1.3 NA in Array

```
# Convert Inf and -Inf to NA
x <- c(1, -1, Inf, 10, -Inf)
na_if(na_if(x, -Inf), Inf)
```

1.2.1.3.1 Check if NA is in Array

```
## [1]  1 -1 NA 10 NA
```

1.2.2 Generate Arrays

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.2.2.1 Generate Special Arrays

1.2.2.1.1 Log Space Arrays Often need to generate arrays on log rather than linear scale, below is log 10 scaled grid.

```
# Parameters
it.lower.bd.inc.cnt <- 3
fl.log.lower <- -10
fl.log.higher <- -9
fl.min.rescale <- 0.01
it.log.count <- 4
# Generate
ar.fl.log.rescaled <- exp(log(10)*seq(log10(fl.min.rescale),
                                      log10(fl.min.rescale +
                                              (fl.log.higher-fl.log.lower)),
                                      length.out=it.log.count))
ar.fl.log <- ar.fl.log.rescaled + fl.log.lower - fl.min.rescale
# Print
ar.fl.log
```

```
## [1] -10.000000  -9.963430  -9.793123  -9.000000
```

1.2.3 String Arrays

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.2.3.1 String Replace

```
# String replacement
gsub(x = paste0(unique(df.slds.stats.perc$it.inner.counter), ':',
                unique(df.slds.stats.perc$z_n_a_n), collapse = ';'),
     pattern = "\n",
     replacement = "")
gsub(x = var,  pattern = "\n", replacement = "")
gsub(x = var.input,  pattern = "\\.", replacement = "_")
```

1.2.3.1.1 String Contains

- r if string contains

```
st_example_a <- 'C:/Users/fan/R4Econ/amto/tibble/fs_tib_basics.Rmd'
st_example_b <- 'C:/Users/fan/R4Econ/amto/tibble/_main.html'
grepl('_main', st_example_a)
```

```
## [1] FALSE
```

```
grepl('_main', st_example_b)
```

```
## [1] TRUE
```

1.2.3.2 String Concatenate

```
# Simple Collapse
vars.group.by <- c('abc', 'efg')
paste0(vars.group.by, collapse='|')
```

```
## [1] "abc|efg"
```

1.2.3.3 String Add Leading Zero

```
# Add Leading zero for integer values to allow for sorting when
# integers are combined into strings
it_z_n <- 1
it_a_n <- 192
print(sprintf("%02d", it_z_n))
```

```
## [1] "01"
```

```
print(sprintf("%04d", it_a_n))
```

```
## [1] "0192"
```

1.2.3.4 Substring and File Name

From path, get file name without suffix.

- r string split
- r list last element
- r get file name from path
- r get file path no name

```
st_example <- 'C:/Users/fan/R4Econ/amto/tibble/fs_tib_basics.Rmd'
st_file_wth_suffix <- tail(strsplit(st_example, "/")[[1]],n=1)
st_file_wno_suffix <- sub('\\.Rmd$', '', basename(st_example))
st_fullpath_nosufx <- sub('\\.Rmd$', '', st_example)
st_lastpath_noname <- basename(dirname(st_example))
st_fullpath_noname <- dirname(st_example)

print(strsplit(st_example, "/"))
```

```
## [[1]]
## [1] "C:"                "Users"             "fan"               "R4Econ"            "amto"
```

```
print(st_file_wth_suffix)
```

```
## [1] "fs_tib_basics.Rmd"
```

```
print(st_file_wno_suffix)
```

```
## [1] "fs_tib_basics"
```

```
print(st_fullpath_nosufx)
```

```
## [1] "C:/Users/fan/R4Econ/amto/tibble/fs_tib_basics"
```

```
print(st_lastpath_noname)
```

```
## [1] "tibble"
```

```r
print(st_fullpath_noname)
```

```
## [1] "C:/Users/fan/R4Econ/amto/tibble"
```

1.2.4 Mesh Matrix and Vector

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

- r expand.grid meshed array to matrix
- r meshgrid
- r array to matrix
- r reshape array to matrix
- dplyr permuations rows of matrix and element of array
- tidyr expand_grid mesh matrix and vector

In the example below, we have a matrix that is 5 by 2, and a vector that is 1 by 3. We want to generate a tibble dataset that meshes the matrix and the vector, so that all combinations show up.

Note *expand_grid* is a from tidyr 1.0.0.

```r
# it_child_count = N, the number of children
it_N_child_cnt = 5
# P fixed parameters, nN is N dimensional, nP is P dimensional
ar_nN_A = seq(-2, 2, length.out = it_N_child_cnt)
ar_nN_alpha = seq(0.1, 0.9, length.out = it_N_child_cnt)
mt_nP_A_alpha = cbind(ar_nN_A, ar_nN_alpha)

# Choice Grid
it_N_choice_cnt = 3
fl_max = 10
fl_min = 0
ar_nN_alpha = seq(fl_min, fl_max, length.out = it_N_choice_cnt)

# expand grid with dplyr
expand_grid(x = 1:3, y = 1:2)
```

```
## # A tibble: 6 x 2
##       x     y
##   <int> <int>
## 1     1     1
## 2     1     2
## 3     2     1
## 4     2     2
## 5     3     1
## 6     3     2
```

```r
tb_expanded <- as_tibble(mt_nP_A_alpha) %>% expand_grid(choices = ar_nN_alpha)

# display
kable(tb_expanded) %>% kable_styling_fc()
```

1.2.4.1 Define Two Arrays and Mesh Them using expand.grid

Given two arrays, mesh the two arrays together.

```r
# use expand.grid to generate all combinations of two arrays

it_ar_A = 5
it_ar_alpha = 10
```

ar_nN_A	ar_nN_alpha	choices
-2	0.1	0
-2	0.1	5
-2	0.1	10
-1	0.3	0
-1	0.3	5
-1	0.3	10
0	0.5	0
0	0.5	5
0	0.5	10
1	0.7	0
1	0.7	5
1	0.7	10
2	0.9	0
2	0.9	5
2	0.9	10

```r
ar_A = seq(-2, 2, length.out=it_ar_A)
ar_alpha = seq(0.1, 0.9, length.out=it_ar_alpha)

mt_A_alpha = expand.grid(A = ar_A, alpha = ar_alpha)

mt_A_meshed = mt_A_alpha[,1]
dim(mt_A_meshed) = c(it_ar_A, it_ar_alpha)

mt_alpha_meshed = mt_A_alpha[,2]
dim(mt_alpha_meshed) = c(it_ar_A, it_ar_alpha)

# display
kable(mt_A_meshed) %>%
  kable_styling_fc()
```

-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2

```r
kable(mt_alpha_meshed) %>%
  kable_styling_fc_wide()
```

0.1	0.1888889	0.2777778	0.3666667	0.4555556	0.5444444	0.6333333	0.7222222	0.8111111	0.9
0.1	0.1888889	0.2777778	0.3666667	0.4555556	0.5444444	0.6333333	0.7222222	0.8111111	0.9
0.1	0.1888889	0.2777778	0.3666667	0.4555556	0.5444444	0.6333333	0.7222222	0.8111111	0.9
0.1	0.1888889	0.2777778	0.3666667	0.4555556	0.5444444	0.6333333	0.7222222	0.8111111	0.9
0.1	0.1888889	0.2777778	0.3666667	0.4555556	0.5444444	0.6333333	0.7222222	0.8111111	0.9

1.2.4.2 Two Identical Arrays, Mesh to Generate Square using expand.grid

Two Identical Arrays, individual attributes, each column is an individual for a matrix, and each row is also an individual

```r
# use expand.grid to generate all combinations of two arrays

it_ar_A = 5

ar_A = seq(-2, 2, length.out=it_ar_A)
```

```
mt_A_A = expand.grid(Arow = ar_A, Arow = ar_A)
mt_Arow = mt_A_A[,1]
dim(mt_Arow) = c(it_ar_A, it_ar_A)
mt_Acol = mt_A_A[,2]
dim(mt_Acol) = c(it_ar_A, it_ar_A)

# display
kable(mt_Arow) %>%
  kable_styling_fc()
```

-2	-2	-2	-2	-2
-1	-1	-1	-1	-1
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2

```
kable(mt_Acol) %>%
  kable_styling_fc()
```

-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2

1.3 Matrix

1.3.1 Generate Matrixes

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.3.1.1 Create a N by 2 Matrix from 3 arrays

Names of each array become row names automatically.

```
ar_row_one <- c(-1,+1)
ar_row_two <- c(-3,-2)
ar_row_three <- c(0.35,0.75)

mt_n_by_2 <- rbind(ar_row_one, ar_row_two, ar_row_three)
kable(mt_n_by_2) %>%
  kable_styling_fc()
```

ar_row_one	-1.00	1.00
ar_row_two	-3.00	-2.00
ar_row_three	0.35	0.75

1.3.1.2 Generate Random Matrixes

Random draw from the normal distribution, random draw from the uniform distribution, and combine resulting matrixes.

```
# Generate 15 random normal, put in 5 rows, and 3 columns
mt_rnorm <- matrix(rnorm(15,mean=0,sd=1), nrow=5, ncol=3)

# Generate 15 random normal, put in 5 rows, and 3 columns
```

```
mt_runif <- matrix(runif(15,min=0,max=1), nrow=5, ncol=5)

# Combine
mt_rnorm_runif <- cbind(mt_rnorm, mt_runif)

# Display
kable(mt_rnorm_runif) %>%
  kable_styling_fc_wide()
```

-1.1858745	0.7264546	-2.1613182	0.2068418	0.9547658	0.6578097	0.2068418	0.9547658
-2.0055130	0.7136567	0.3952199	0.1146044	0.4543614	0.1698893	0.1146044	0.4543614
0.0075099	-0.6500629	-0.3948340	0.7504459	0.1925193	0.7443364	0.7504459	0.1925193
0.5194904	1.4986962	-0.3097584	0.9334095	0.4198546	0.0552954	0.9334095	0.4198546
-0.7462955	-1.4358281	1.3308266	0.4146961	0.1078679	0.5422845	0.4146961	0.1078679

1.3.2 Linear Algebra

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.3.2.1 Matrix Multiplication

Multiply Together a 3 by 2 matrix and a 2 by 1 vector

```
ar_row_one <- c(-1,+1)
ar_row_two <- c(-3,-2)
ar_row_three <- c(0.35,0.75)
mt_n_by_2 <- rbind(ar_row_one, ar_row_two, ar_row_three)

ar_row_four <- c(3,4)

# Matrix Multiplication
mt_out <- mt_n_by_2 %*% ar_row_four
print(mt_n_by_2)
```

```
##               [,1]  [,2]
## ar_row_one   -1.00  1.00
## ar_row_two   -3.00 -2.00
## ar_row_three  0.35  0.75
```

```
print(ar_row_four)
```

```
## [1] 3 4
```

```
print(mt_out)
```

```
##                [,1]
## ar_row_one     1.00
## ar_row_two   -17.00
## ar_row_three   4.05
```

1.4 Dataframes (Tibble)

1.4.1 Generate Dataframe

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.4.1.1 Generate Tibble given Matrixes and Arrays

Given Arrays and Matrixes, Generate Tibble and Name Variables/Columns

- naming tibble columns
- tibble variable names
- dplyr rename tibble
- dplyr rename tibble all variables
- dplyr rename all columns by index
- dplyr tibble add index column
- see also: SO-51205520

```
# Base Inputs
ar_col <- c(-1,+1)
mt_rnorm_a <- matrix(rnorm(4,mean=0,sd=1), nrow=2, ncol=2)
mt_rnorm_b <- matrix(rnorm(4,mean=0,sd=1), nrow=2, ncol=4)

# Combine Matrix
mt_combine <- cbind(ar_col, mt_rnorm_a, mt_rnorm_b)
colnames(mt_combine) <- c('ar_col',
                          paste0('matcolvar_grpa_', seq(1,dim(mt_rnorm_a)[2])),
                          paste0('matcolvar_grpb_', seq(1,dim(mt_rnorm_b)[2])))

# Variable Names
ar_st_varnames <- c('var_one',
                    paste0('tibcolvar_ga_', c(1,2)),
                    paste0('tibcolvar_gb_', c(1,2,3,4)))

# Combine to tibble, add name col1, col2, etc.
tb_combine <- as_tibble(mt_combine) %>% rename_all(~c(ar_st_varnames))

# Add an index column to the dataframe, ID column
tb_combine <- tb_combine %>% rowid_to_column(var = "ID")

# Change all gb variable names
tb_combine <- tb_combine %>%
                  rename_at(vars(starts_with("tibcolvar_gb_")),
                            funs(str_replace(., "_gb_", "_gbrenamed_")))

# Tibble back to matrix
mt_tb_combine_back <- data.matrix(tb_combine)

# Display
kable(mt_combine) %>% kable_styling_fc_wide()
```

ar_col	matcolvar_grpa_1	matcolvar_grpa_2	matcolvar_grpb_1	matcolvar_grpb_2	matcolvar_grpb_3	matcolvar_grpb_4
-1	-0.6015056	0.0209320	0.1754664	1.0928359	0.1754664	1.0928359
1	-2.4379080	0.7102217	1.0162244	-0.1119114	1.0162244	-0.1119114

```
kable(tb_combine) %>% kable_styling_fc_wide()
```

ID	var_one	tibcolvar_ga_1	tibcolvar_ga_2	tibcolvar_gbrenamed_1	tibcolvar_gbrenamed_2	tibcolvar_gbrenamed_3	tibcolvar_gbrenamed_4
1	-1	-0.6015056	0.0209320	0.1754664	1.0928359	0.1754664	1.0928359
2	1	-2.4379080	0.7102217	1.0162244	-0.1119114	1.0162244	-0.1119114

```
kable(mt_tb_combine_back) %>% kable_styling_fc_wide()
```

ID	var_one	tibcolvar_ga_1	tibcolvar_ga_2	tibcolvar_gbrenamed_1	tibcolvar_gbrenamed_2	tibcolvar_gbrenamed_3	tibcolvar_gbrenamed_4
1	-1	-0.6015056	0.0209320	0.1754664	1.0928359	0.1754664	1.0928359
2	1	-2.4379080	0.7102217	1.0162244	-0.1119114	1.0162244	-0.1119114

1.4.1.2 Rename Tibble with Numeric Column Names

After reshaping, often could end up with variable names that are all numeric, intgers for example, how to rename these variables to add a common prefix for example.

```
# Base Inputs
ar_col <- c(-1,+1)
mt_rnorm_c <- matrix(rnorm(4,mean=0,sd=1), nrow=5, ncol=10)
```

```
## Warning in matrix(rnorm(4, mean = 0, sd = 1), nrow = 5, ncol = 10): data length [4] is not a sub-
```

```
mt_combine <- cbind(ar_col, mt_rnorm_c)
```

```
## Warning in cbind(ar_col, mt_rnorm_c): number of rows of result is not a multiple of vector length
```

```
# Variable Names
ar_it_cols_ctr <- seq(1, dim(mt_rnorm_c)[2])
ar_st_varnames <- c('var_one', ar_it_cols_ctr)

# Combine to tibble, add name col1, col2, etc.
tb_combine <- as_tibble(mt_combine) %>% rename_all(~c(ar_st_varnames))

# Add an index column to the dataframe, ID column
tb_combine_ori <- tb_combine %>% rowid_to_column(var = "ID")

# Change all gb variable names
tb_combine <- tb_combine_ori %>%
                  rename_at(
                    vars(num_range('',ar_it_cols_ctr)),
                    funs(paste0("rho", . , 'var'))
                    )

# Display
kable(tb_combine_ori) %>% kable_styling_fc_wide()
```

ID	var_one	1	2	3	4	5	6	7	8	9	10
1	-1	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419
2	1	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187
3	-1	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174
4	1	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521
5	-1	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419

```
kable(tb_combine) %>% kable_styling_fc_wide()
```

ID	var_one	rho1var	rho2var	rho3var	rho4var	rho5var	rho6var	rho7var	rho8var	rho9var	rho10var
1	-1	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419
2	1	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187
3	-1	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174
4	1	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521
5	-1	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419	-0.2480187	0.7538174	1.0846521	-0.3837419

1.4.1.3 Tibble Row and Column and Summarize

Show what is in the table: 1, column and row names; 2, contents inside table.

```
tb_iris <- as_tibble(iris)
print(rownames(tb_iris))
```

```
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"  "13"  "14"  "15"  "
##  [26] "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"  "37"  "38"  "39"  "40"  "
##  [51] "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"  "65"  "
##  [76] "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"  "89"  "90"  "
## [101] "101" "102" "103" "104" "105" "106" "107" "108" "109" "110" "111" "112" "113" "114" "115" "
## [126] "126" "127" "128" "129" "130" "131" "132" "133" "134" "135" "136" "137" "138" "139" "140" "
```

```
colnames(tb_iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
colnames(tb_iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
summary(tb_iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

1.4.1.4 Tibble Sorting

- dplyr arrange desc reverse
- dplyr sort

```
# Sort in Ascending Order
tb_iris %>% select(Species, Sepal.Length, everything()) %>%
  arrange(Species, Sepal.Length) %>% head(10) %>%
  kable() %>% kable_styling_fc()
```

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	4.3	3.0	1.1	0.1
setosa	4.4	2.9	1.4	0.2
setosa	4.4	3.0	1.3	0.2
setosa	4.4	3.2	1.3	0.2
setosa	4.5	2.3	1.3	0.3
setosa	4.6	3.1	1.5	0.2
setosa	4.6	3.4	1.4	0.3
setosa	4.6	3.6	1.0	0.2
setosa	4.6	3.2	1.4	0.2
setosa	4.7	3.2	1.3	0.2

```
# Sort in Descending Order
tb_iris %>% select(Species, Sepal.Length, everything()) %>%
  arrange(desc(Species), desc(Sepal.Length)) %>% head(10) %>%
  kable() %>% kable_styling_fc()
```

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
virginica	7.9	3.8	6.4	2.0
virginica	7.7	3.8	6.7	2.2
virginica	7.7	2.6	6.9	2.3
virginica	7.7	2.8	6.7	2.0
virginica	7.7	3.0	6.1	2.3
virginica	7.6	3.0	6.6	2.1
virginica	7.4	2.8	6.1	1.9
virginica	7.3	2.9	6.3	1.8
virginica	7.2	3.6	6.1	2.5
virginica	7.2	3.2	6.0	1.8

1.4.1.5 REconTools Summarize over Tible

Use R4Econ's summary tool.

```
df_summ_stats <- ff_summ_percentiles(tb_iris)
kable(t(df_summ_stats)) %>% kable_styling_fc_wide()
```

stats	n	NAobs	ZEROobs	mean	sd	cv	min	p01	p05	p10	p25	p50	p75	p90	p95	p99	max
Petal.Length	150	0	0	3.758000	1.7652982	0.4697441	1.0	1.149	1.300	1.4	1.6	4.35	5.1	5.80	6.100	6.700	6.9
Petal.Width	150	0	0	1.199333	0.7622377	0.6355511	0.1	0.100	0.200	0.2	0.3	1.30	1.8	2.20	2.300	2.500	2.5
Sepal.Length	150	0	0	5.843333	0.8280661	0.1417113	4.3	4.400	4.600	4.8	5.1	5.80	6.4	6.90	7.255	7.700	7.9
Sepal.Width	150	0	0	3.057333	0.4358663	0.1425642	2.0	2.200	2.345	2.5	2.8	3.00	3.3	3.61	3.800	4.151	4.4

1.4.2 Draw Random Rows

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.4.2.1 Draw Random Subset of Sample

- r random discrete

We have a sample of N individuals in some dataframe. Draw without replacement a subset $M < N$ of rows.

```
# parameters, it_M < it_N
it_N <- 10
it_M <- 5

# Draw it_m from indexed list of it_N
set.seed(123)
ar_it_rand_idx <- sample(it_N, it_M, replace=FALSE)

# dataframe
df_full <- as_tibble(matrix(rnorm(4,mean=0,sd=1), nrow=it_N, ncol=4)) %>% rowid_to_column(var = "ID"

# random Subset
df_rand_sub_a <- df_full[ar_it_rand_idx,]

# Random subset also
df_rand_sub_b <- df_full[sample(dim(df_full)[1], it_M, replace=FALSE),]

# Print
# Display
kable(df_full) %>% kable_styling_fc()
```

ID	V1	V2	V3	V4
1	0.1292877	0.4609162	0.1292877	0.4609162
2	1.7150650	-1.2650612	1.7150650	-1.2650612
3	0.4609162	0.1292877	0.4609162	0.1292877
4	-1.2650612	1.7150650	-1.2650612	1.7150650
5	0.1292877	0.4609162	0.1292877	0.4609162
6	1.7150650	-1.2650612	1.7150650	-1.2650612
7	0.4609162	0.1292877	0.4609162	0.1292877
8	-1.2650612	1.7150650	-1.2650612	1.7150650
9	0.1292877	0.4609162	0.1292877	0.4609162
10	1.7150650	-1.2650612	1.7150650	-1.2650612

```
kable(df_rand_sub_a) %>% kable_styling_fc()
```

```
kable(df_rand_sub_b) %>% kable_styling_fc()
```

ID	V1	V2	V3	V4
3	0.4609162	0.1292877	0.4609162	0.1292877
10	1.7150650	-1.2650612	1.7150650	-1.2650612
2	1.7150650	-1.2650612	1.7150650	-1.2650612
8	-1.2650612	1.7150650	-1.2650612	1.7150650
6	1.7150650	-1.2650612	1.7150650	-1.2650612

ID	V1	V2	V3	V4
5	0.1292877	0.4609162	0.1292877	0.4609162
3	0.4609162	0.1292877	0.4609162	0.1292877
9	0.1292877	0.4609162	0.1292877	0.4609162
1	0.1292877	0.4609162	0.1292877	0.4609162
4	-1.2650612	1.7150650	-1.2650612	1.7150650

1.4.2.2 Random Subset of Panel

There are N individuals, each could be observed M times, but then select a subset of rows only, so each person is randomly observed only a subset of times. Specifically, there there are 3 unique students with student ids, and the second variable shows the random dates in which the student showed up in class, out of the 10 classes available.

```
# Define
it_N <- 3
it_M <- 10
svr_id <- 'student_id'

# dataframe
set.seed(123)
df_panel_rand <- as_tibble(matrix(it_M, nrow=it_N, ncol=1)) %>%
  rowid_to_column(var = svr_id) %>%
  uncount(V1) %>%
  group_by(!!sym(svr_id)) %>% mutate(date = row_number()) %>%
  ungroup() %>% mutate(in_class = case_when(rnorm(n(),mean=0,sd=1) < 0 ~ 1, TRUE ~ 0)) %>%
  filter(in_class == 1) %>% select(!!sym(svr_id), date) %>%
  rename(date_in_class = date)

# Print
kable(df_panel_rand) %>% kable_styling_fc()
```

student_id	date_in_class
1	1
1	2
1	8
1	9
1	10
2	5
2	8
2	10
3	1
3	2
3	3
3	4
3	5
3	6
3	9

1.4.3 Variable NA Values

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.4.3.1 Find and Replace

Find and Replace in Dataframe.

```
# For dataframe
df.reg <-df.reg %>% na_if(-Inf) %>% na_if(Inf)
# For a specific variable in dataframe
df.reg.use %>% mutate(!!(var.input) := na_if(!!sym(var.input), 0))

# Setting to NA
df.reg.use <- df.reg.guat %>% filter(!!sym(var.mth) != 0)
df.reg.use.log <- df.reg.use
df.reg.use.log[which(is.nan(df.reg.use$prot.imputed.log)),] = NA
df.reg.use.log[which(df.reg.use$prot.imputed.log==Inf),] = NA
df.reg.use.log[which(df.reg.use$prot.imputed.log==-Inf),] = NA
df.reg.use.log <- df.reg.use.log %>% drop_na(prot.imputed.log)
# df.reg.use.log$prot.imputed.log
```

1.4.4 String Values

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

1.4.4.1 Find and Replace

Find and Replace in Dataframe.

```
# if string value is contained in variable
("bridex.B" %in% (df.reg.out.all$vars_var.y))
# if string value is not contained in variable:
# 1. type is variable name
# 2. Toyota|Mazda are strings to be excluded
filter(mtcars, !grepl('Toyota|Mazda', type))

# filter does not contain string
rs_hgt_prot_log_tidy %>% filter(!str_detect(term, 'prot'))
```

Chapter 2

Summarize Data

2.1 Counting Observation

2.1.1 Uncount

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

In some panel, there are N individuals, each observed for Y_i years. Given a dataset with two variables, the individual index, and the Y_i variable, expand the dataframe so that there is a row for each individual index's each unique year in the survey.

Search:

- r duplicate row by variable

Links:

- see: Create duplicate rows based on a variable

Algorithm:

1. generate testing frame, the individual attribute dataset with invariant information over panel
2. uncount, duplicate rows by years in survey
3. group and generate sorted index
4. add indiviual specific stat year to index

```
# 1. Array of Years in the Survey
ar_years_in_survey <- c(2,3,1,10,2,5)
ar_start_yaer <- c(1,2,3,1,1,1)
ar_end_year <- c(2,4,3,10,2,5)
mt_combine <- cbind(ar_years_in_survey, ar_start_yaer, ar_end_year)

# This is the individual attribute dataset, attributes that are invariant acrosss years
tb_indi_attributes <- as_tibble(mt_combine) %>% rowid_to_column(var = "ID")

# 2. Sort and generate variable equal to sorted index
tb_indi_panel <- tb_indi_attributes %>% uncount(ar_years_in_survey)

# 3. Panel now construct exactly which year in survey, note that all needed is sort index
# Note sorting not needed, all rows identical now
tb_indi_panel <- tb_indi_panel %>%
                    group_by(ID) %>%
                    mutate(yr_in_survey = row_number())

tb_indi_panel <- tb_indi_panel %>%
                    mutate(calendar_year = yr_in_survey + ar_start_yaer - 1)
```

```
# Show results Head 10
tb_indi_panel %>% head(10) %>%
  kable() %>%
  kable_styling_fc()
```

ID	ar_start_yaer	ar_end_year	yr_in_survey	calendar_year
1	1	2	1	1
1	1	2	2	2
2	2	4	1	2
2	2	4	2	3
2	2	4	3	4
3	3	3	1	3
4	1	10	1	1
4	1	10	2	2
4	1	10	3	3
4	1	10	4	4

2.2 Sorting, Indexing, Slicing

2.2.1 Sorting

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

2.2.1.1 Generate Sorted Index within Group with Repeating Values

There is a variable, sort by this variable, then generate index from 1 to N representing sorted values of this index. If there are repeating values, still assign index, different index each value.

- r generate index sort
- dplyr mutate equals index

```
# Sort and generate variable equal to sorted index
df_iris <- iris %>% arrange(Sepal.Length) %>%
              mutate(Sepal.Len.Index = row_number()) %>%
              select(Sepal.Length, Sepal.Len.Index, everything())

# Show results Head 10
df_iris %>% head(10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

Sepal.Length	Sepal.Len.Index	Sepal.Width	Petal.Length	Petal.Width	Species
4.3	1	3.0	1.1	0.1	setosa
4.4	2	2.9	1.4	0.2	setosa
4.4	3	3.0	1.3	0.2	setosa
4.4	4	3.2	1.3	0.2	setosa
4.5	5	2.3	1.3	0.3	setosa
4.6	6	3.1	1.5	0.2	setosa
4.6	7	3.4	1.4	0.3	setosa
4.6	8	3.6	1.0	0.2	setosa
4.6	9	3.2	1.4	0.2	setosa
4.7	10	3.2	1.3	0.2	setosa

2.2.1.2 Populate Value from Lowest Index to All other Rows

We would like to calculate for example the ratio of each individual's highest to the the person with the lowest height in a dataset. We first need to generated sorted index from lowest to highest, and then populate the lowest height to all rows, and then divide.

Search Terms:

- r spread value to all rows from one row
- r other rows equal to the value of one row
- Conditional assignment of one variable to the value of one of two other variables
- dplyr mutate conditional
- dplyr value from one row to all rows
- dplyr mutate equal to value in another cell

Links:

- see: dplyr rank
- see: dplyr case_when

2.2.1.2.1 Short Method: mutate and min We just want the lowest value to be in its own column, so that we can compute various statistics using the lowest value variable and the original variable.

```
# 1. Sort
df_iris_m1 <- iris %>% mutate(Sepal.Len.Lowest.all = min(Sepal.Length)) %>%
                select(Sepal.Length, Sepal.Len.Lowest.all, everything())


# Show results Head 10
df_iris_m1 %>% head(10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

Sepal.Length	Sepal.Len.Lowest.all	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	4.3	3.5	1.4	0.2	setosa
4.9	4.3	3.0	1.4	0.2	setosa
4.7	4.3	3.2	1.3	0.2	setosa
4.6	4.3	3.1	1.5	0.2	setosa
5.0	4.3	3.6	1.4	0.2	setosa
5.4	4.3	3.9	1.7	0.4	setosa
4.6	4.3	3.4	1.4	0.3	setosa
5.0	4.3	3.4	1.5	0.2	setosa
4.4	4.3	2.9	1.4	0.2	setosa
4.9	4.3	3.1	1.5	0.1	setosa

2.2.1.2.2 Long Method: row_number and case_when This is the long method, using row_number, and case_when. The benefit of this method is that it generates several intermediate variables that might be useful. And the key final step is to set a new variable (A=*Sepal.Len.Lowest.all*) equal to another variable's (B=*Sepal.Length*'s) value at the index that satisfies condition based a third variable (C=*Sepal.Len.Index*).

```
# 1. Sort
# 2. generate index
# 3. value at lowest index (case_when)
# 4. spread value from lowest index to other rows
# Note step 4 does not require step 3
df_iris_m2 <- iris %>% arrange(Sepal.Length) %>%
              mutate(Sepal.Len.Index = row_number()) %>%
              mutate(Sepal.Len.Lowest.one =
```

```
                case_when(row_number()==1 ~ Sepal.Length)) %>%
         mutate(Sepal.Len.Lowest.all =
                Sepal.Length[Sepal.Len.Index==1]) %>%
         select(Sepal.Length, Sepal.Len.Index,
               Sepal.Len.Lowest.one, Sepal.Len.Lowest.all)


# Show results Head 10
df_iris_m2 %>% head(10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

Sepal.Length	Sepal.Len.Index	Sepal.Len.Lowest.one	Sepal.Len.Lowest.all
4.3	1	4.3	4.3
4.4	2	NA	4.3
4.4	3	NA	4.3
4.4	4	NA	4.3
4.5	5	NA	4.3
4.6	6	NA	4.3
4.6	7	NA	4.3
4.6	8	NA	4.3
4.6	9	NA	4.3
4.7	10	NA	4.3

2.2.1.3 Generate Sorted Index based on Deviations

Generate Positive and Negative Index based on Ordered Deviation from some Number.

There is a variable that is continuous, substract a number from this variable, and generate index based on deviations. Think of the index as generating intervals indicating where the value lies. 0th index indicates the largest value in sequence that is smaller than or equal to number x, 1st index indicates the smallest value in sequence that is larger than number x.

The solution below is a little bit convoluated and long, there is likely a much quicker way. The process below shows various intermediary outputs that help arrive at deviation index *Sepal.Len.Devi.Index* from initial sorted index *Sepal.Len.Index*.

search:

- dplyr arrange ignore na
- dplyr index deviation from order number sequence
- dplyr index below above
- dplyr index order below above value

```
# 1. Sort and generate variable equal to sorted index
# 2. Plus or minus deviations from some value
# 3. Find the zero, which means, the number closests to zero including zero from the negative side
# 4. Find the index at the highest zero and below deviation point
# 5. Difference of zero index and original sorted index
sc_val_x <- 4.65
df_iris_deviate <- iris %>% arrange(Sepal.Length) %>%
              mutate(Sepal.Len.Index = row_number()) %>%
              mutate(Sepal.Len.Devi = (Sepal.Length - sc_val_x)) %>%
              mutate(Sepal.Len.Devi.Neg =
                       case_when(Sepal.Len.Devi <= 0 ~ (-1)*(Sepal.Len.Devi))) %>%
              arrange((Sepal.Len.Devi.Neg), desc(Sepal.Len.Index)) %>%
```

```
            mutate(Sepal.Len.Index.Zero =
                     case_when(row_number() == 1 ~ Sepal.Len.Index)) %>%
            mutate(Sepal.Len.Devi.Index =
                     Sepal.Len.Index - Sepal.Len.Index.Zero[row_number() == 1]) %>%
            arrange(Sepal.Len.Index) %>%
            select(Sepal.Length, Sepal.Len.Index, Sepal.Len.Devi,
                   Sepal.Len.Devi.Neg, Sepal.Len.Index.Zero, Sepal.Len.Devi.Index)


# Show results Head 10
df_iris_deviate %>% head(20) %>%
  kable() %>%
  kable_styling_fc_wide()
```

Sepal.Length	Sepal.Len.Index	Sepal.Len.Devi	Sepal.Len.Devi.Neg	Sepal.Len.Index.Zero	Sepal.Len.Devi.Index
4.3	1	-0.35	0.35	NA	-8
4.4	2	-0.25	0.25	NA	-7
4.4	3	-0.25	0.25	NA	-6
4.4	4	-0.25	0.25	NA	-5
4.5	5	-0.15	0.15	NA	-4
4.6	6	-0.05	0.05	NA	-3
4.6	7	-0.05	0.05	NA	-2
4.6	8	-0.05	0.05	NA	-1
4.6	9	-0.05	0.05	9	0
4.7	10	0.05	NA	NA	1
4.7	11	0.05	NA	NA	2
4.8	12	0.15	NA	NA	3
4.8	13	0.15	NA	NA	4
4.8	14	0.15	NA	NA	5
4.8	15	0.15	NA	NA	6
4.8	16	0.15	NA	NA	7
4.9	17	0.25	NA	NA	8
4.9	18	0.25	NA	NA	9
4.9	19	0.25	NA	NA	10
4.9	20	0.25	NA	NA	11

2.3 Group Statistics

2.3.1 Groups Statistics

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

2.3.1.1 Aggrgate Groups only Unique Group and Count

There are two variables that are numeric, we want to find all the unique groups of these two variables in a dataset and count how many times each unique group occurs

- r unique occurrence of numeric groups
- How to add count of unique values by group to R data.frame

```
# Numeric value combinations unique Groups
vars.group <- c('hgt0', 'wgt0')

# dataset subsetting
df_use <- df_hgt_wgt %>% select(!!!syms(c(vars.group))) %>%
            mutate(hgt0 = round(hgt0/5)*5, wgt0 = round(wgt0/2000)*2000) %>%
            drop_na()

# Group, count and generate means for each numeric variables
# mutate_at(vars.group, funs(as.factor(.))) %>%
```

```
df.group.count <- df_use %>% group_by(!!!syms(vars.group)) %>%
                    arrange(!!!syms(vars.group)) %>%
                    summarise(n_obs_group=n())

# Show results Head 10
df.group.count %>% kable() %>% kable_styling_fc()
```

hgt0	wgt0	n_obs_group
40	2000	122
45	2000	4586
45	4000	470
50	2000	9691
50	4000	13106
55	2000	126
55	4000	1900
60	6000	18

2.3.1.2 Aggrgate Groups only Unique Group Show up With Means

Several variables that are grouping identifiers. Several variables that are values which mean be unique for each group members. For example, a Panel of income for N households over T years with also household education information that is invariant over time. Want to generate a dataset where the unit of observation are households, rather than household years. Take average of all numeric variables that are household and year specific.

A complicating factor potentially is that the number of observations differ within group, for example, income might be observed for all years for some households but not for other households.

- r dplyr aggregate group average
- Aggregating and analyzing data with dplyr
- column can't be modified because it is a grouping variable
- see also: Aggregating and analyzing data with dplyr

```
# In the df_hgt_wgt from R4Econ, there is a country id, village id,
# and individual id, and various other statistics
vars.group <- c('S.country', 'vil.id', 'indi.id')
vars.values <- c('hgt', 'momEdu')

# dataset subsetting
df_use <- df_hgt_wgt %>% select(!!!syms(c(vars.group, vars.values)))

# Group, count and generate means for each numeric variables
df.group <- df_use %>% group_by(!!!syms(vars.group)) %>%
            arrange(!!!syms(vars.group)) %>%
            summarise_if(is.numeric,
                         funs(mean = mean(., na.rm = TRUE),
                              sd = sd(., na.rm = TRUE),
                              n = sum(is.na(.)==0)))

# Show results Head 10
df.group %>% head(10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

```
# Show results Head 10
df.group %>% tail(10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

S.country	vil.id	indi.id	hgt_mean	momEdu_mean	hgt_sd	momEdu_sd	hgt_n	momEdu_n
Cebu	1	1	61.80000	5.3	9.520504	0	7	18
Cebu	1	2	68.86154	7.1	9.058931	0	13	18
Cebu	1	3	80.45882	9.4	29.894231	0	17	18
Cebu	1	4	88.10000	13.9	35.533166	0	18	18
Cebu	1	5	97.70556	11.3	41.090366	0	18	18
Cebu	1	6	87.49444	7.3	35.586439	0	18	18
Cebu	1	7	90.79412	10.4	38.722385	0	17	18
Cebu	1	8	68.45385	13.5	10.011961	0	13	18
Cebu	1	9	86.21111	10.4	35.126057	0	18	18
Cebu	1	10	87.67222	10.5	36.508127	0	18	18

S.country	vil.id	indi.id	hgt_mean	momEdu_mean	hgt_sd	momEdu_sd	hgt_n	momEdu_n
Guatemala	14	2014	66.97000	NaN	8.967974	NaN	10	0
Guatemala	14	2015	71.71818	NaN	11.399984	NaN	11	0
Guatemala	14	2016	66.33000	NaN	9.490352	NaN	10	0
Guatemala	14	2017	76.40769	NaN	14.827871	NaN	13	0
Guatemala	14	2018	74.55385	NaN	12.707846	NaN	13	0
Guatemala	14	2019	70.47500	NaN	11.797390	NaN	12	0
Guatemala	14	2020	60.28750	NaN	7.060036	NaN	8	0
Guatemala	14	2021	84.96000	NaN	15.446193	NaN	10	0
Guatemala	14	2022	79.38667	NaN	15.824749	NaN	15	0
Guatemala	14	2023	66.50000	NaN	8.613113	NaN	8	0

2.3.2 One Variable Group Summary

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

There is a categorical variable (based on one or the interaction of multiple variables), there is a continuous variable, obtain statistics for the continuous variable conditional on the categorical variable, but also unconditionally.

Store results in a matrix, but also flatten results wide to row with appropriate keys/variable-names for all group statistics.

Pick which statistics to be included in final wide row

2.3.2.1 Build Program

```
# Single Variable Group Statistics (also generate overall statistics)
ff_summ_by_group_summ_one <- function(
  df, vars.group, var.numeric, str.stats.group = 'main',
  str.stats.specify = NULL, boo.overall.stats = TRUE){

    # List of statistics
    # https://rdrr.io/cran/dplyr/man/summarise.html
    strs.center <- c('mean', 'median')
    strs.spread <- c('sd', 'IQR', 'mad')
    strs.range <- c('min', 'max')
    strs.pos <- c('first', 'last')
    strs.count <- c('n_distinct')

    # Grouping of Statistics
    if (missing(str.stats.specify)) {
        if (str.stats.group == 'main') {
            strs.all <- c('mean', 'min', 'max', 'sd')
        }
        if (str.stats.group == 'all') {
            strs.all <- c(strs.center, strs.spread, strs.range, strs.pos, strs.count)
        }
```

```
    } else {
        strs.all <- str.stats.specify
    }

    # Start Transform
    df <- df %>% drop_na() %>% mutate(!!(var.numeric) := as.numeric(!!sym(var.numeric)))

    # Overall Statistics
    if (boo.overall.stats) {
        df.overall.stats <- df %>% summarize_at(vars(var.numeric), funs(!!!strs.all))
        if (length(strs.all) == 1) {
            # give it a name, otherwise if only one stat, name of stat not saved
            df.overall.stats <- df.overall.stats %>% rename(!!strs.all := !!sym(var.numeric))
        }
        names(df.overall.stats) <- paste0(var.numeric, '.', names(df.overall.stats))
    }

    # Group Sort
    df.select <- df %>%
                  group_by(!!!syms(vars.group)) %>%
                  arrange(!!!syms(c(vars.group, var.numeric)))

    # Table of Statistics
    df.table.grp.stats <- df.select %>% summarize_at(vars(var.numeric), funs(!!!strs.all))

    # Add Stat Name
    if (length(strs.all) == 1) {
        # give it a name, otherwise if only one stat, name of stat not saved
        df.table.grp.stats <- df.table.grp.stats %>% rename(!!strs.all := !!sym(var.numeric))
    }

    # Row of Statistics
    str.vars.group.combine <- paste0(vars.group, collapse='_')
    if (length(vars.group) == 1) {
        df.row.grp.stats <- df.table.grp.stats %>%
                mutate(!!(str.vars.group.combine) := paste0(var.numeric, '.',
                                                       vars.group, '.g',
                                                       (!!!syms(vars.group)))) %>%
                gather(variable, value, -one_of(vars.group)) %>%
                unite(str.vars.group.combine, c(str.vars.group.combine, 'variable')) %>%
                spread(str.vars.group.combine, value)
    } else {
        df.row.grp.stats <- df.table.grp.stats %>%
          mutate(vars.groups.combine := paste0(paste0(vars.group, collapse='.')),
                 !!(str.vars.group.combine) := paste0(interaction(!!!(syms(vars.group))))) %>%
          mutate(!!(str.vars.group.combine) := paste0(var.numeric, '.', vars.groups.combine, '.',
                                                  (!!sym(str.vars.group.combine)))) %>%
          ungroup() %>%
          select(-vars.groups.combine, -one_of(vars.group)) %>%
          gather(variable, value, -one_of(str.vars.group.combine)) %>%
          unite(str.vars.group.combine, c(str.vars.group.combine, 'variable')) %>%
          spread(str.vars.group.combine, value)
    }

    # Clean up name strings
    names(df.table.grp.stats) <-
      gsub(x = names(df.table.grp.stats),pattern = "_", replacement = "\\.")
```

```
    names(df.row.grp.stats) <-
      gsub(x = names(df.row.grp.stats),pattern = "_", replacement = "\\.")

    # Return
    list.return <-
      list(df_table_grp_stats = df.table.grp.stats, df_row_grp_stats = df.row.grp.stats)

    # Overall Statistics, without grouping
    if (boo.overall.stats) {
        df.row.stats.all <- c(df.row.grp.stats, df.overall.stats)
        list.return <- append(list.return, list(df_overall_stats = df.overall.stats,
                                                df_row_stats_all = df.row.stats.all))
    }

    # Return
    return(list.return)

}
```

2.3.2.2 Test

Load data and test

```
# Library
library(tidyverse)

# Load Sample Data
setwd('C:/Users/fan/R4Econ/_data/')
df <- read_csv('height_weight.csv')
```

```
## Parsed with column specification:
## cols(
##   S.country = col_character(),
##   vil.id = col_double(),
##   indi.id = col_double(),
##   sex = col_character(),
##   svymthRound = col_double(),
##   momEdu = col_double(),
##   wealthIdx = col_double(),
##   hgt = col_double(),
##   wgt = col_double(),
##   hgt0 = col_double(),
##   wgt0 = col_double(),
##   prot = col_double(),
##   cal = col_double(),
##   p.A.prot = col_double(),
##   p.A.nProt = col_double()
## )
```

2.3.2.2.1 Function Testing By Gender Groups Need two variables, a group variable that is a factor, and a numeric

```
vars.group <- 'sex'
var.numeric <- 'hgt'
```

```
df.select <- df %>% select(one_of(vars.group, var.numeric)) %>% drop_na()
```

Main Statistics:

```r
# Single Variable Group Statistics
ff_summ_by_group_summ_one(
  df.select, vars.group = vars.group, var.numeric = var.numeric,
  str.stats.group = 'main')
```

```
## $df_table_grp_stats
## # A tibble: 2 x 5
##   sex     mean   min   max    sd
##   <chr>  <dbl> <dbl> <dbl> <dbl>
## 1 Female  82.8  41.2  171.  29.8
## 2 Male    84.7  41.3  183.  31.8
##
## $df_row_grp_stats
## # A tibble: 1 x 8
##   hgt.sex.gFemale.max hgt.sex.gFemale.mean hgt.sex.gFemale.min hgt.sex.gFemale.sd hgt.sex.gMale.m
##                 <dbl>                <dbl>               <dbl>              <dbl>            <db
## 1                171.                 82.8                41.2               29.8             18
##
## $df_overall_stats
## # A tibble: 1 x 4
##   hgt.mean hgt.min hgt.max hgt.sd
##      <dbl>   <dbl>   <dbl>  <dbl>
## 1     83.8    41.2    183.   30.9
##
## $df_row_stats_all
## $df_row_stats_all$hgt.sex.gFemale.max
## [1] 170.6
##
## $df_row_stats_all$hgt.sex.gFemale.mean
## [1] 82.81198
##
## $df_row_stats_all$hgt.sex.gFemale.min
## [1] 41.2
##
## $df_row_stats_all$hgt.sex.gFemale.sd
## [1] 29.79351
##
## $df_row_stats_all$hgt.sex.gMale.max
## [1] 182.9
##
## $df_row_stats_all$hgt.sex.gMale.mean
## [1] 84.68152
##
## $df_row_stats_all$hgt.sex.gMale.min
## [1] 41.3
##
## $df_row_stats_all$hgt.sex.gMale.sd
## [1] 31.75037
##
## $df_row_stats_all$hgt.mean
## [1] 83.80921
##
## $df_row_stats_all$hgt.min
## [1] 41.2
##
## $df_row_stats_all$hgt.max
## [1] 182.9
##
```

```
## $df_row_stats_all$hgt.sd
## [1] 30.86631
```

Specify Two Specific Statistics:

```
ff_summ_by_group_summ_one(
  df.select, vars.group = vars.group, var.numeric = var.numeric,
  str.stats.specify = c('mean', 'sd'))
```

```
## $df_table_grp_stats
## # A tibble: 2 x 3
##   sex     mean    sd
##   <chr>  <dbl> <dbl>
## 1 Female  82.8  29.8
## 2 Male    84.7  31.8
##
## $df_row_grp_stats
## # A tibble: 1 x 4
##   hgt.sex.gFemale.mean hgt.sex.gFemale.sd hgt.sex.gMale.mean hgt.sex.gMale.sd
##                  <dbl>              <dbl>              <dbl>            <dbl>
## 1                 82.8               29.8               84.7             31.8
##
## $df_overall_stats
## # A tibble: 1 x 2
##   hgt.mean hgt.sd
##      <dbl>  <dbl>
## 1     83.8   30.9
##
## $df_row_stats_all
## $df_row_stats_all$hgt.sex.gFemale.mean
## [1] 82.81198
##
## $df_row_stats_all$hgt.sex.gFemale.sd
## [1] 29.79351
##
## $df_row_stats_all$hgt.sex.gMale.mean
## [1] 84.68152
##
## $df_row_stats_all$hgt.sex.gMale.sd
## [1] 31.75037
##
## $df_row_stats_all$hgt.mean
## [1] 83.80921
##
## $df_row_stats_all$hgt.sd
## [1] 30.86631
```

Specify One Specific Statistics:

```
ff_summ_by_group_summ_one(
  df.select, vars.group = vars.group, var.numeric = var.numeric,
  str.stats.specify = c('mean'))
```

```
## $df_table_grp_stats
## # A tibble: 2 x 2
##   sex     mean
##   <chr>  <dbl>
## 1 Female  82.8
## 2 Male    84.7
##
## $df_row_grp_stats
```

```
## # A tibble: 1 x 2
##   hgt.sex.gFemale.mean hgt.sex.gMale.mean
##                  <dbl>              <dbl>
## 1                 82.8               84.7
##
## $df_overall_stats
## # A tibble: 1 x 1
##   hgt.mean
##      <dbl>
## 1     83.8
##
## $df_row_stats_all
## $df_row_stats_all$hgt.sex.gFemale.mean
## [1] 82.81198
##
## $df_row_stats_all$hgt.sex.gMale.mean
## [1] 84.68152
##
## $df_row_stats_all$hgt.mean
## [1] 83.80921
```

2.3.2.2.2 Function Testing By Country and Gender Groups

Need two variables, a group variable that is a factor, and a numeric. Now joint grouping variables.

```
vars.group <- c('S.country', 'sex')
var.numeric <- 'hgt'
```

```
df.select <- df %>% select(one_of(vars.group, var.numeric)) %>% drop_na()
```

Main Statistics:

```
ff_summ_by_group_summ_one(
  df.select, vars.group = vars.group, var.numeric = var.numeric,
  str.stats.group = 'main')
```

```
## $df_table_grp_stats
## # A tibble: 4 x 6
## # Groups:   S.country [2]
##   S.country sex     mean   min   max    sd
##   <chr>     <chr>  <dbl> <dbl> <dbl> <dbl>
## 1 Cebu      Female  84.6  41.3  171.  32.5
## 2 Cebu      Male    87.0  41.3  183.  35.0
## 3 Guatemala Female  76.6  41.2  120.  15.7
## 4 Guatemala Male    77.0  41.5  125.  15.1
##
## $df_row_grp_stats
## # A tibble: 1 x 16
##   hgt.S.country.s~ hgt.S.country.s~ hgt.S.country.s~ hgt.S.country.s~ hgt.S.country.s~ hgt.S.coun
##              <dbl>            <dbl>            <dbl>            <dbl>            <dbl>
## 1             171.             84.6             41.3             32.5             183.
## # ... with 7 more variables: hgt.S.country.sex.Guatemala.Female.mean <dbl>, hgt.S.country.sex.Gua
## #   hgt.S.country.sex.Guatemala.Female.sd <dbl>, hgt.S.country.sex.Guatemala.Male.max <dbl>, hgt.
## #   hgt.S.country.sex.Guatemala.Male.min <dbl>, hgt.S.country.sex.Guatemala.Male.sd <dbl>
##
## $df_overall_stats
## # A tibble: 1 x 4
##   hgt.mean hgt.min hgt.max hgt.sd
##      <dbl>   <dbl>   <dbl>  <dbl>
## 1     83.8    41.2    183.   30.9
##
```

```
## $df_row_stats_all
## $df_row_stats_all$hgt.S.country.sex.Cebu.Female.max
## [1] 170.6
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Female.mean
## [1] 84.61326
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Female.min
## [1] 41.3
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Female.sd
## [1] 32.53651
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Male.max
## [1] 182.9
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Male.mean
## [1] 87.02836
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Male.min
## [1] 41.3
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Male.sd
## [1] 34.9909
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Female.max
## [1] 119.9
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Female.mean
## [1] 76.58771
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Female.min
## [1] 41.2
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Female.sd
## [1] 15.71801
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Male.max
## [1] 124.7
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Male.mean
## [1] 77.0471
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Male.min
## [1] 41.5
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Male.sd
## [1] 15.11444
##
## $df_row_stats_all$hgt.mean
## [1] 83.80921
##
## $df_row_stats_all$hgt.min
## [1] 41.2
##
## $df_row_stats_all$hgt.max
## [1] 182.9
##
```

```
## $df_row_stats_all$hgt.sd
## [1] 30.86631
```

Specify Two Specific Statistics:

```
ff_summ_by_group_summ_one(
  df.select, vars.group = vars.group, var.numeric = var.numeric,
  str.stats.specify = c('mean', 'sd'))
```

```
## $df_table_grp_stats
## # A tibble: 4 x 4
## # Groups:   S.country [2]
##   S.country sex     mean    sd
##   <chr>     <chr>  <dbl> <dbl>
## 1 Cebu      Female  84.6  32.5
## 2 Cebu      Male    87.0  35.0
## 3 Guatemala Female  76.6  15.7
## 4 Guatemala Male    77.0  15.1
##
## $df_row_grp_stats
## # A tibble: 1 x 8
##   hgt.S.country.sex.~ hgt.S.country.sex.~ hgt.S.country.sex.~ hgt.S.country.sex~ hgt.S.country.se
##                 <dbl>               <dbl>               <dbl>              <dbl>              <db
## 1                84.6                32.5                87.0               35.0               76
##
## $df_overall_stats
## # A tibble: 1 x 2
##   hgt.mean hgt.sd
##      <dbl>  <dbl>
## 1     83.8   30.9
##
## $df_row_stats_all
## $df_row_stats_all$hgt.S.country.sex.Cebu.Female.mean
## [1] 84.61326
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Female.sd
## [1] 32.53651
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Male.mean
## [1] 87.02836
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Male.sd
## [1] 34.9909
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Female.mean
## [1] 76.58771
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Female.sd
## [1] 15.71801
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Male.mean
## [1] 77.0471
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Male.sd
## [1] 15.11444
##
## $df_row_stats_all$hgt.mean
## [1] 83.80921
##
```

```
## $df_row_stats_all$hgt.sd
## [1] 30.86631
```

Specify One Specific Statistics:

```
ff_summ_by_group_summ_one(
  df.select, vars.group = vars.group, var.numeric = var.numeric, str.stats.specify = c('mean'))
```

```
## $df_table_grp_stats
## # A tibble: 4 x 3
## # Groups:   S.country [2]
##   S.country sex     mean
##   <chr>     <chr>  <dbl>
## 1 Cebu      Female  84.6
## 2 Cebu      Male    87.0
## 3 Guatemala Female  76.6
## 4 Guatemala Male    77.0
##
## $df_row_grp_stats
## # A tibble: 1 x 4
##   hgt.S.country.sex.Cebu.Female.mean hgt.S.country.sex.Cebu.Male.mean hgt.S.country.sex.Guatemala
##                                <dbl>                            <dbl>
## 1                               84.6                             87.0
##
## $df_overall_stats
## # A tibble: 1 x 1
##   hgt.mean
##      <dbl>
## 1     83.8
##
## $df_row_stats_all
## $df_row_stats_all$hgt.S.country.sex.Cebu.Female.mean
## [1] 84.61326
##
## $df_row_stats_all$hgt.S.country.sex.Cebu.Male.mean
## [1] 87.02836
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Female.mean
## [1] 76.58771
##
## $df_row_stats_all$hgt.S.country.sex.Guatemala.Male.mean
## [1] 77.0471
##
## $df_row_stats_all$hgt.mean
## [1] 83.80921
```

2.3.3 Nested within Group Stats

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

By Multiple within Individual Groups Variables, Averages for All Numeric Variables within All Groups of All Group Variables (Long to very Wide). Suppose you have an individual level final outcome. The individual is observed for N periods, where each period the inputs differ. What inputs impacted the final outcome?

Suppose we can divide N periods in which the individual is in the data into a number of years, a number of semi-years, a number of quarters, or uneven-staggered lengths. We might want to generate averages across individuals and within each of these different possible groups averages of inputs.

Then we want to version of the data where each row is an individual, one of the variables is the final outcome, and the other variables are these different averages: averages for the 1st, 2nd, 3rd year in which indivdiual is in data, averages for 1st, …, final quarter in which indivdiual is in data.

2.3.3.1 Build Function

This function takes as inputs:

1. **vars.not.groups2avg**: a list of variables that are not the within-indivdiual or across-individual grouping variables, but the variables we want to average over. Withnin indivdiual grouping averages will be calculated for these variables using the not-listed variables as within indivdiual groups (excluding vars.indi.grp groups).
2. **vars.indi.grp**: a list or individual variables, and also perhaps villages, province, etc id variables that are higher than individual ID. Note the groups are are ACROSS individual higher level group variables.
3. the remaining variables are all within individual grouping variables.

the function output is a dataframe:

1. each row is an individual
2. initial variables individual ID and across individual groups from *vars.indi.grp*.
3. other variables are all averages for the variables in *vars.not.groups2avg*
    - if there are 2 within individual group variables, and the first has 3 groups (years), the second has 6 groups (semi-years), then there would be 9 average variables.
    - each average variables has the original variable name from vars.not.groups2avg plus the name of the within individual grouping variable, and at the end 'c_x', where x is a integer representing the category within the group (if 3 years, x=1, 2, 3)

```
# Data Function
# https://fanwangecon.github.io/R4Econ/summarize/summ/ByGroupsSummWide.html
f.by.groups.summ.wide <- function(df.groups.to.average,
                                  vars.not.groups2avg,
                                  vars.indi.grp = c('S.country','ID'),
                                  display=TRUE) {

# 1. generate categoricals for full year (m.12), half year (m.6), quarter year (m.4)
# 2. generate categoricals also for uneven years (m12t14) using stagger (+2 rather than -1)
# 3. reshape wide to long, so that all categorical date groups appear in var=value,
    # and categories in var=variable
# 4. calculate mean for all numeric variables for all date groups
# 5. combine date categorical variable and value, single var:
    # m.12.c1= first year average from m.12 averaging

######## ######## ######## ######## #######
# Step 1
######## ######## ######## ######## #######
# 1. generate categoricals for full year (m.12), half year (m.6), quarter year (m.4)
# 2. generate categoricals also for uneven years (m12t14) using stagger (+2 rather than -1)

######## ######## ######## ######## #######
# S2: reshape wide to long, so that all categorical date groups appear in var=value,
    # and categories in var=variable; calculate mean for all numeric variables for all date groups
######## ######## ######## ######## #######
df.avg.long <- df.groups.to.average %>%
       gather(variable, value, -one_of(c(vars.indi.grp,
                                         vars.not.groups2avg))) %>%
       group_by(!!!syms(vars.indi.grp), variable, value) %>%
       summarise_if(is.numeric, funs(mean(., na.rm = TRUE)))

if (display){
  dim(df.avg.long)
```

```
  options(repr.matrix.max.rows=10, repr.matrix.max.cols=20)
  print(df.avg.long)
}

######## ######## ######## ######## #######
# S3 combine date categorical variable and value, single var:
#    m.12.c1= first year average from m.12 averaging; to do this make data even longer first
######## ######## ######## ######## #######

# We already have the averages, but we want them to show up as variables,
    # mean for each group of each variable.
df.avg.allvars.wide <- df.avg.long %>%
   ungroup() %>%
   mutate(all_m_cate = paste0(variable, '_c', value)) %>%
   select(all_m_cate, everything(), -variable, -value) %>%
   gather(variable, value, -one_of(vars.indi.grp), -all_m_cate) %>%
   unite('var_mcate', variable, all_m_cate) %>%
   spread(var_mcate, value)

if (display){
  dim(df.avg.allvars.wide)
  options(repr.matrix.max.rows=10, repr.matrix.max.cols=10)
  print(df.avg.allvars.wide)
}

return(df.avg.allvars.wide)
}
```

2.3.3.2 Test Program

In our sample dataset, the number of nutrition/height/income etc information observed within each country and month of age group are different. We have a panel dataset for children observed over different months of age.

We have two key grouping variables: 1. country: data are observed for guatemala and cebu 2. month-age (survey month round=svymthRound): different months of age at which each individual child is observed

A child could be observed for many months, or just a few months. A child's height information could be observed for more months-of-age than nutritional intake information. We eventually want to run regressions where the outcome is height/weight and the input is nutrition. The regressions will be at the month-of-age level. We need to know how many times different variables are observed at the month-of-age level.

```
# Library
library(tidyverse)

# Load Sample Data
setwd('C:/Users/fan/R4Econ/_data/')
df <- read_csv('height_weight.csv')
```

```
## Parsed with column specification:
## cols(
##   S.country = col_character(),
##   vil.id = col_double(),
##   indi.id = col_double(),
##   sex = col_character(),
##   svymthRound = col_double(),
##   momEdu = col_double(),
##   wealthIdx = col_double(),
##   hgt = col_double(),
```

```
##   wgt = col_double(),
##   hgt0 = col_double(),
##   wgt0 = col_double(),
##   prot = col_double(),
##   cal = col_double(),
##   p.A.prot = col_double(),
##   p.A.nProt = col_double()
## )
```

2.3.3.2.1 Generate Within Individual Groups In the data, children are observed for different number of months since birth. We want to calculate quarterly, semi-year, annual, etc average nutritional intakes. First generate these within-individual grouping variables. We can also generate uneven-staggered calendar groups as shown below.

```
mth.var <- 'svymthRound'
df.groups.to.average<- df %>%
        filter(!!sym(mth.var) >= 0 & !!sym(mth.var) <= 24)  %>%
        mutate(m12t24=(floor((!!sym(mth.var) - 12) %/% 14) + 1),
               m8t24=(floor((!!sym(mth.var) - 8) %/% 18) + 1),
               m12 = pmax((floor((!!sym(mth.var)-1) %/% 12) + 1), 1),
               m6 = pmax((floor((!!sym(mth.var)-1) %/% 6) + 1), 1),
               m3 = pmax((floor((!!sym(mth.var)-1) %/% 3) + 1), 1))
```

```
# Show Results
options(repr.matrix.max.rows=30, repr.matrix.max.cols=20)
vars.arrange <- c('S.country','indi.id','svymthRound')
vars.groups.within.indi <- c('m12t24', 'm8t24', 'm12', 'm6', 'm3')
as.tibble(df.groups.to.average %>%
          group_by(!!!syms(vars.arrange)) %>%
          arrange(!!!syms(vars.arrange)) %>%
          select(!!!syms(vars.arrange), !!!syms(vars.groups.within.indi)))
```

```
## # A tibble: 23,603 x 8
##    S.country indi.id svymthRound m12t24 m8t24   m12    m6    m3
##    <chr>       <dbl>       <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Cebu            1           0      0     0     1     1     1
##  2 Cebu            1           2      0     0     1     1     1
##  3 Cebu            1           4      0     0     1     1     2
##  4 Cebu            1           6      0     0     1     1     2
##  5 Cebu            1           8      0     1     1     2     3
##  6 Cebu            1          10      0     1     1     2     4
##  7 Cebu            1          12      1     1     1     2     4
##  8 Cebu            1          14      1     1     2     3     5
##  9 Cebu            1          16      1     1     2     3     6
## 10 Cebu            1          18      1     1     2     3     6
## # ... with 23,593 more rows
```

2.3.3.2.2 Within Group Averages With the within-group averages created, we can generate averages for all variables within these groups.

```
vars.not.groups2avg <- c('prot', 'cal')
vars.indi.grp <- c('S.country', 'indi.id')
vars.groups.within.indi <- c('m12t24', 'm8t24', 'm12', 'm6', 'm3')

df.groups.to.average.select <- df.groups.to.average %>%
                        select(one_of(c(vars.indi.grp,
                                        vars.not.groups2avg,
                                        vars.groups.within.indi)))
df.avg.allvars.wide <- f.by.groups.summ.wide(df.groups.to.average.select,
```

```r
                                             vars.not.groups2avg,
                                             vars.indi.grp, display=TRUE)
```

```
## # A tibble: 36,414 x 6
## # Groups:   S.country, indi.id, variable [10,115]
##    S.country indi.id variable value   prot   cal
##    <chr>       <dbl> <chr>    <dbl>  <dbl> <dbl>
##  1 Cebu            1 m12          1   5.36 132.
##  2 Cebu            1 m12          2 NaN    NaN
##  3 Cebu            1 m12t24       0   4.37  97.1
##  4 Cebu            1 m12t24       1  11.3  343.
##  5 Cebu            1 m3           1   0.65   9.1
##  6 Cebu            1 m3           2   3.65  95.5
##  7 Cebu            1 m3           3   2.6   85.3
##  8 Cebu            1 m3           4  13.2  315.
##  9 Cebu            1 m3           5 NaN    NaN
## 10 Cebu            1 m3           6 NaN    NaN
## # ... with 36,404 more rows
## # A tibble: 2,023 x 38
##    S.country indi.id cal_m12_c1 cal_m12_c2 cal_m12t24_c0 cal_m12t24_c1 cal_m3_c1 cal_m3_c2 cal_m3
##    <chr>       <dbl>      <dbl>      <dbl>         <dbl>         <dbl>     <dbl>     <dbl>     <d
##  1 Cebu            1      132.        NaN           97.1          343.       9.1      95.5      8
##  2 Cebu            2       90.7       256.          81.5          240.      83.4      12.3     15
##  3 Cebu            3       96.8       659.          31.6          634.       0.5      28.8      5
##  4 Cebu            4       27.5       372.          24.6          325.       4.5      26.0      3
##  5 Cebu            5      101.       1081.          79.2          960.      14.1     144.       7
##  6 Cebu            6      185.        522.         162.           493.      23.8     185.      16
##  7 Cebu            7      157.        571.         146.           514.       8.3     138.      40
##  8 Cebu            8      472.        845.         379.           871.     159.      423       41
##  9 Cebu            9       32.3       415.          16.6          374.       5.05      10.4      1
## 10 Cebu           10       67.2       395.          68.6          347.       9.55      26.4     16
## # ... with 2,013 more rows, and 24 more variables: cal_m6_c1 <dbl>, cal_m6_c2 <dbl>, cal_m6_c3 <d
## #   cal_m8t24_c1 <dbl>, prot_m12_c1 <dbl>, prot_m12_c2 <dbl>, prot_m12t24_c0 <dbl>, prot_m12t24_c
## #   prot_m3_c3 <dbl>, prot_m3_c4 <dbl>, prot_m3_c5 <dbl>, prot_m3_c6 <dbl>, prot_m3_c7 <dbl>, pro
## #   prot_m6_c3 <dbl>, prot_m6_c4 <dbl>, prot_m8t24_c0 <dbl>, prot_m8t24_c1 <dbl>
```

This is the tabular version of results

```r
dim(df.avg.allvars.wide)
```

```
## [1] 2023   38
```

```r
names(df.avg.allvars.wide)
```

```
##  [1] "S.country"      "indi.id"        "cal_m12_c1"     "cal_m12_c2"     "cal_m12t24_c0"  "cal_m1
##  [9] "cal_m3_c3"      "cal_m3_c4"      "cal_m3_c5"      "cal_m3_c6"      "cal_m3_c7"      "cal_m3
## [17] "cal_m6_c3"      "cal_m6_c4"      "cal_m8t24_c0"   "cal_m8t24_c1"   "prot_m12_c1"    "prot_m
## [25] "prot_m3_c1"     "prot_m3_c2"     "prot_m3_c3"     "prot_m3_c4"     "prot_m3_c5"     "prot_m
## [33] "prot_m6_c1"     "prot_m6_c2"     "prot_m6_c3"     "prot_m6_c4"     "prot_m8t24_c0"  "prot_m
```

```r
options(repr.matrix.max.rows=30, repr.matrix.max.cols=12)
df.avg.allvars.wide
```

```
## # A tibble: 2,023 x 38
##    S.country indi.id cal_m12_c1 cal_m12_c2 cal_m12t24_c0 cal_m12t24_c1 cal_m3_c1 cal_m3_c2 cal_m3
##    <chr>       <dbl>      <dbl>      <dbl>         <dbl>         <dbl>     <dbl>     <dbl>     <d
##  1 Cebu            1      132.        NaN           97.1          343.       9.1      95.5      8
##  2 Cebu            2       90.7       256.          81.5          240.      83.4      12.3     15
##  3 Cebu            3       96.8       659.          31.6          634.       0.5      28.8      5
##  4 Cebu            4       27.5       372.          24.6          325.       4.5      26.0      3
##  5 Cebu            5      101.       1081.          79.2          960.      14.1     144.       7
```

```
##   6 Cebu            6       185.         522.          162.            493.     23.8       185.       16
##   7 Cebu            7       157.         571.          146.            514.      8.3       138.       40
##   8 Cebu            8       472.         845.          379.            871.    159.        423        41
##   9 Cebu            9        32.3        415.           16.6           374.      5.05       10.4       1
##  10 Cebu           10        67.2        395.           68.6           347.      9.55       26.4      16
## # ... with 2,013 more rows, and 24 more variables: cal_m6_c1 <dbl>, cal_m6_c2 <dbl>, cal_m6_c3 <d
## #   cal_m8t24_c1 <dbl>, prot_m12_c1 <dbl>, prot_m12_c2 <dbl>, prot_m12t24_c0 <dbl>, prot_m12t24_c
## #   prot_m3_c3 <dbl>, prot_m3_c4 <dbl>, prot_m3_c5 <dbl>, prot_m3_c6 <dbl>, prot_m3_c7 <dbl>, pro
## #   prot_m6_c3 <dbl>, prot_m6_c4 <dbl>, prot_m8t24_c0 <dbl>, prot_m8t24_c1 <dbl>
```

2.4 Distributional Statistics

2.4.1 Histogram

2.4.1.1 Generate Test Score Dataset

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

- r generate text string as csv
- r tibble matrix hand input

First, we will generate a test score dataset, directly from string. Below we type line by line a dataset with four variables in comma separated (csv) format, where the first row includes the variables names. These texts could be stored in a separate file, or they could be directly included in code and read in as csv

```
ar_test_scores_ec3 <- c(107.72,101.28,105.92,109.31,104.27,110.27,91.92846154,81.8,109.0071429,103.0
ar_test_scores_ec1 <- c(101.72,101.28,99.92,103.31,100.27,104.27,90.23615385,77.8,103.4357143,97.07,
mt_test_scores <- cbind(ar_test_scores_ec1, ar_test_scores_ec3)
ar_st_varnames <- c('course_total_ec1p','course_total_ec3p')
tb_final_twovar <- as_tibble(mt_test_scores) %>% rename_all(~c(ar_st_varnames))
summary(tb_final_twovar)
```

2.4.1.1.1 A Dataset with only Two Continuous Variable

```
##  course_total_ec1p course_total_ec3p
##  Min.   : 40.48    Min.   : 44.23
##  1st Qu.: 76.46    1st Qu.: 79.91
##  Median : 86.35    Median : 89.28
##  Mean   : 83.88    Mean   : 87.90
##  3rd Qu.: 95.89    3rd Qu.:100.75
##  Max.   :104.27    Max.   :112.22
```

```
ff_summ_percentiles(df = tb_final_twovar, bl_statsasrows = TRUE, col2varname = FALSE)
```

```
## # A tibble: 17 x 3
##    stats   course.total.ec1p course.total.ec3p
##    <chr>   <chr>             <chr>
##  1 n       46                46
##  2 NAobs   0                 0
##  3 ZEROobs 0                 0
##  4 mean    83.87572          87.90239
##  5 sd      15.87272          16.76041
##  6 cv      0.1892409         0.1906706
##  7 min     40.475            44.225
##  8 p01     42.14434          45.82202
##  9 p05     56.9650           57.1575
## 10 p10     63.05462          66.07500
## 11 p25     76.45616          79.90500
```

```
## 12 p50     86.35236         89.27923
## 13 p75     " 95.89054"      100.75250
## 14 p90     100.8137         106.8200
## 15 p95     102.9125         109.2343
## 16 p99     103.8946         111.3439
## 17 max     104.2700         112.2225
```

```
ar_final_scores <- c(94.28442509,95.68817475,97.25219512,77.89268293,95.08795497,93.27380863,92.3,84
mt_test_scores <- cbind(seq(1,length(ar_final_scores)), ar_final_scores)
ar_st_varnames <- c('index', 'course_final')
tb_onevar <- as_tibble(mt_test_scores) %>% rename_all(~c(ar_st_varnames))
summary(tb_onevar)
```

2.4.1.1.2 A Dataset with one Continuous Variable and Histogram

```
##      index       course_final
##  Min.   : 1.0   Min.   :  2.293
##  1st Qu.:12.5   1st Qu.: 76.372
##  Median :24.0   Median : 86.959
##  Mean   :24.0   Mean   : 82.415
##  3rd Qu.:35.5   3rd Qu.: 94.686
##  Max.   :47.0   Max.   :100.898
```

```
ff_summ_percentiles(df = tb_onevar, bl_statsasrows = TRUE, col2varname = FALSE)
```

```
## # A tibble: 17 x 3
##    stats   course.final index
##    <chr>   <chr>        <chr>
##  1 n       47           47
##  2 NAobs   0            0
##  3 ZEROobs 0            0
##  4 mean    82.41501     24.00000
##  5 sd      18.35476     13.71131
##  6 cv      0.2227113    0.5713046
##  7 min     2.292683     1.000000
##  8 p01     18.67401     " 1.46000"
##  9 p05     49.72075     " 3.30000"
## 10 p10     66.28051     " 5.60000"
## 11 p25     76.37177     12.50000
## 12 p50     86.95932     24.00000
## 13 p75     94.68619     35.50000
## 14 p90     97.52332     42.40000
## 15 p95     99.47459     44.70000
## 16 p99     100.5244     " 46.5400"
## 17 max     100.898      " 47.000"
```

```
#load in data empirically by hand
txt_test_data <- "init_prof, later_prof, class_id, exam_score
 'SW', 'SW', 1, 102
 'SW', 'SW', 1, 102
 'SW', 'SW', 1, 101
 'SW', 'SW', 1, 100
 'SW', 'SW', 1, 100
 'SW', 'SW', 1, 99
 'SW', 'SW', 1, 98.5
 'SW', 'SW', 1, 98.5
 'SW', 'SW', 1, 97
```

```
'SW', 'SW', 1, 95
'SW', 'SW', 1, 94
'SW', 'SW', 1, 91
'SW', 'SW', 1, 91
'SW', 'SW', 1, 90
'SW', 'SW', 1, 89
'SW', 'SW', 1, 88.5
'SW', 'SW', 1, 88
'SW', 'SW', 1, 87
'SW', 'SW', 1, 87
'SW', 'SW', 1, 87
'SW', 'SW', 1, 86
'SW', 'SW', 1, 86
'SW', 'SW', 1, 84
'SW', 'SW', 1, 82
'SW', 'SW', 1, 78.5
'SW', 'SW', 1, 76
'SW', 'SW', 1, 72
'SW', 'SW', 1, 70.5
'SW', 'SW', 1, 67.5
'SW', 'SW', 1, 67.5
'SW', 'SW', 1, 67
'SW', 'SW', 1, 63.5
'SW', 'SW', 1, 60
'SW', 'SW', 1, 59
'SW', 'SW', 1, 44.5
'SW', 'SW', 1, 44
'SW', 'SW', 1, 42.5
'SW', 'SW', 1, 40.5
'SW', 'SW', 1, 40.5
'SW', 'SW', 1, 36.5
'SW', 'SW', 1, 35.5
'SW', 'SW', 1, 21.5
'SW', 'SW', 1, 4
'MP', 'MP', 2, 105
'MP', 'MP', 2, 103
'MP', 'MP', 2, 102
'MP', 'MP', 2, 101
'MP', 'MP', 2, 101
'MP', 'MP', 2, 100.5
'MP', 'MP', 2, 100
'MP', 'MP', 2, 99
'MP', 'MP', 2, 97
'MP', 'MP', 2, 97
'MP', 'MP', 2, 97
'MP', 'MP', 2, 97
'MP', 'MP', 2, 96
'MP', 'MP', 2, 95
'MP', 'MP', 2, 91
'MP', 'MP', 2, 89
'MP', 'MP', 2, 85
'MP', 'MP', 2, 84
'MP', 'MP', 2, 84
'MP', 'MP', 2, 84
'MP', 'MP', 2, 83.5
'MP', 'MP', 2, 82.5
'MP', 'MP', 2, 81.5
'MP', 'MP', 2, 80.5
```

```
'MP', 'MP', 2, 80
'MP', 'MP', 2, 77
'MP', 'MP', 2, 77
'MP', 'MP', 2, 75
'MP', 'MP', 2, 75
'MP', 'MP', 2, 71
'MP', 'MP', 2, 70
'MP', 'MP', 2, 68
'MP', 'MP', 2, 63
'MP', 'MP', 2, 56
'MP', 'MP', 2, 56
'MP', 'MP', 2, 55.5
'MP', 'MP', 2, 49.5
'MP', 'MP', 2, 48.5
'MP', 'MP', 2, 47.5
'MP', 'MP', 2, 44.5
'MP', 'MP', 2, 34.5
'MP', 'MP', 2, 29.5
'CA', 'MP', 3, 103
'CA', 'MP', 3, 103
'CA', 'MP', 3, 101
'CA', 'MP', 3, 96.5
'CA', 'MP', 3, 93.5
'CA', 'MP', 3, 93
'CA', 'MP', 3, 93
'CA', 'MP', 3, 92
'CA', 'MP', 3, 90
'CA', 'MP', 3, 90
'CA', 'MP', 3, 89
'CA', 'MP', 3, 86.5
'CA', 'MP', 3, 84.5
'CA', 'MP', 3, 83
'CA', 'MP', 3, 83
'CA', 'MP', 3, 82
'CA', 'MP', 3, 78
'CA', 'MP', 3, 75
'CA', 'MP', 3, 74.5
'CA', 'MP', 3, 70
'CA', 'MP', 3, 54.5
'CA', 'MP', 3, 52
'CA', 'MP', 3, 50
'CA', 'MP', 3, 42
'CA', 'MP', 3, 36.5
'CA', 'MP', 3, 28
'CA', 'MP', 3, 26
'CA', 'MP', 3, 11
'CA', 'SN', 4, 103
'CA', 'SN', 4, 103
'CA', 'SN', 4, 102
'CA', 'SN', 4, 102
'CA', 'SN', 4, 101
'CA', 'SN', 4, 100
'CA', 'SN', 4, 98
'CA', 'SN', 4, 98
'CA', 'SN', 4, 98
'CA', 'SN', 4, 95
'CA', 'SN', 4, 95
'CA', 'SN', 4, 92.5
```

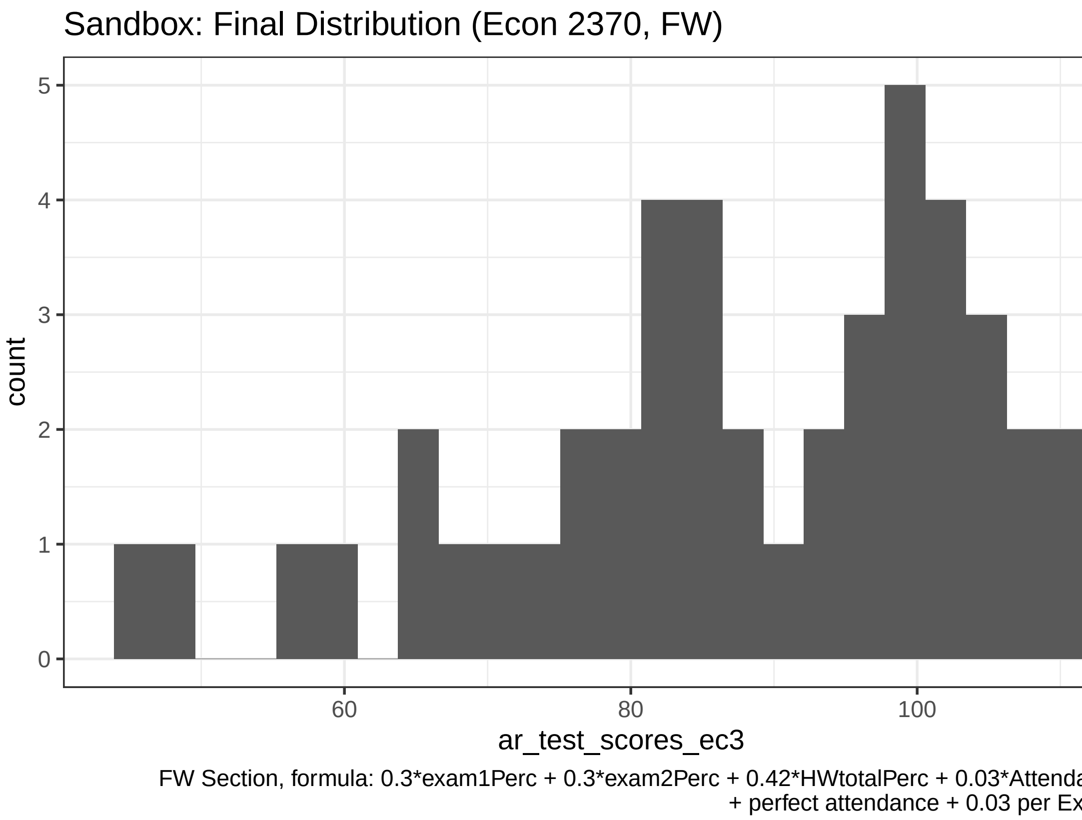
```
 'CA', 'SN', 4, 92
 'CA', 'SN', 4, 91
 'CA', 'SN', 4, 90
 'CA', 'SN', 4, 85.5
 'CA', 'SN', 4, 84
 'CA', 'SN', 4, 82.5
 'CA', 'SN', 4, 81
 'CA', 'SN', 4, 77.5
 'CA', 'SN', 4, 77
 'CA', 'SN', 4, 72
 'CA', 'SN', 4, 71.5
 'CA', 'SN', 4, 69
 'CA', 'SN', 4, 68.5
 'CA', 'SN', 4, 68
 'CA', 'SN', 4, 67
 'CA', 'SN', 4, 65.5
 'CA', 'SN', 4, 62.5
 'CA', 'SN', 4, 62
 'CA', 'SN', 4, 61.5
 'CA', 'SN', 4, 61
 'CA', 'SN', 4, 57.5
 'CA', 'SN', 4, 54
 'CA', 'SN', 4, 52.5
 'CA', 'SN', 4, 51
 'CA', 'SN', 4, 50.5
 'CA', 'SN', 4, 50
 'CA', 'SN', 4, 49
 'CA', 'SN', 4, 43
 'CA', 'SN', 4, 39.5
 'CA', 'SN', 4, 32.5
 'CA', 'SN', 4, 25.5
 'CA', 'SN', 4, 18"

csv_test_data = read.csv(text=txt_test_data, header=TRUE)
ar_st_varnames <- c('first_half_professor', 'second_half_professor', 'course_id', 'exam_score')
tb_test_data <- as_tibble(csv_test_data) %>% rename_all(~c(ar_st_varnames))
summary(tb_test_data)
```

2.4.1.1.3 A Dataset with Multiple Variables

```
##  first_half_professor second_half_professor   course_id       exam_score
##   'CA':72              'MP':70              Min.   :1.000   Min.   :  4.00
##   'MP':42              'SN':44              1st Qu.:1.000   1st Qu.: 60.00
##   'SW':43              'SW':43              Median :2.000   Median : 82.00
##                                             Mean   :2.465   Mean   : 75.08
##                                             3rd Qu.:4.000   3rd Qu.: 94.00
##                                             Max.   :4.000   Max.   :105.00
```

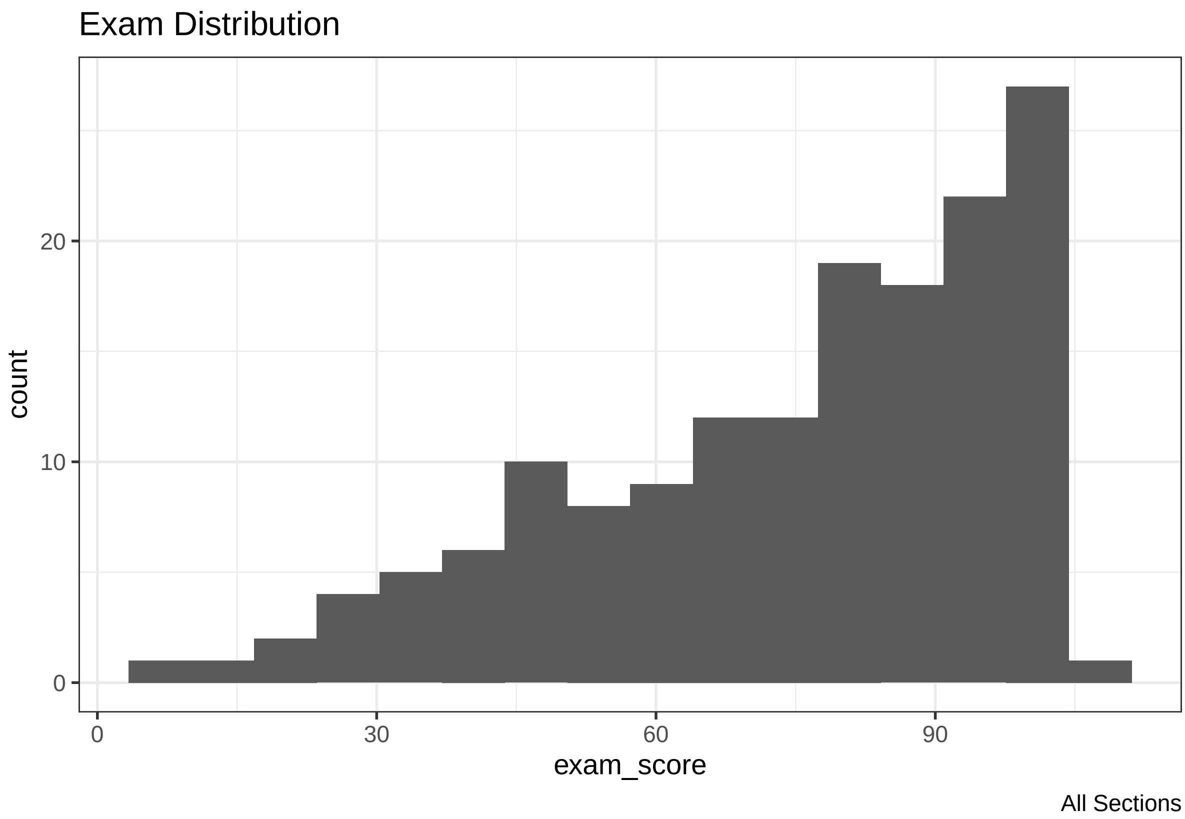
2.4.1.2 Test Score Distributions

```
ggplot(tb_final_twovar, aes(x=ar_test_scores_ec3)) +
  geom_histogram(bins=25) +
  labs(title = paste0('Sandbox: Final Distribution (Econ 2370, FW)'),
       caption = 'FW Section, formula: 0.3*exam1Perc + 0.3*exam2Perc + 0.42*HWtotalPerc + 0.03*Atten
  theme_bw()
```



2.4.1.2.1 Histogram

```
ggplot(tb_test_data, aes(x=exam_score)) +
  geom_histogram(bins=16) +
  labs(title = paste0('Exam Distribution'),
       caption = 'All Sections') +
  theme_bw()
```



2.4.2 Joint Quantiles from Continuous

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

There are multiple or a single continuous variables. Find which quantile each observation belongs to for each of the variables. Then also generate a joint/interaction variable of all combinations of quantiles from different variables.

The program has these features:

1. Quantiles breaks are generated based on group_by characteristics, meaning quantiles for individual level characteristics when data is panel
2. Quantiles variables apply to full panel at within-group observation levels.
3. Robust to non-unique breaks for quantiles (non-unique grouped together)
4. Quantile categories have detailed labeling (specifying which non-unique groupings belong to quantile)

When joining multiple quantile variables together:

1. First check if only calculate quantiles at observations where all quantile base variables are not null
2. Calculate Quantiles for each variable, with different quantile levels for sub-groups of variables
3. Summary statistics by mulltiple quantile-categorical variables, summary

2.4.2.1 Build Program

```
# Quantiles for any variable
gen_quantiles <- function(var, df, prob=c(0.25, 0.50, 0.75)) {
    enframe(quantile(as.numeric(df[[var]]), prob, na.rm=TRUE), 'quant.perc', var)
}
# Support Functions for Variable Suffix
f_Q_suffix <- function(seq.quantiles) {
    quantile.suffix <- paste0('Qs', min(seq.quantiles),
```

```
                              'e', max(seq.quantiles),
                              'n', (length(seq.quantiles)-1))
}
# Support Functions for Quantile Labeling
f_Q_label <- function(arr.quantiles,
                      arr.sort.unique.quantile,
                      seq.quantiles) {
    paste0('(',
           paste0(which(arr.quantiles %in% arr.sort.unique.quantile), collapse=','),
           ') of ', f_Q_suffix(seq.quantiles))
}
# Generate New Variable Names with Quantile Suffix
f_var_rename <- function(name, seq.quantiles) {
    quantile.suffix <- paste0('_', f_Q_suffix(seq.quantiles))
    return(sub('_q', quantile.suffix, name))
}

# Check Are Values within Group By Unique? If not, STOP
f_check_distinct_ingroup <- function(df, vars.group_by, vars.values_in_group) {

    df.uniqus.in.group <- df %>% group_by(!!!syms(vars.group_by)) %>%
            mutate(quant_vars_paste = paste(!!!(syms(vars.values_in_group)), sep='-')) %>%
            mutate(unique_in_group = n_distinct(quant_vars_paste)) %>%
            slice(1L) %>%
            ungroup() %>%
            group_by(unique_in_group) %>%
            summarise(n=n())

    if (sum(df.uniqus.in.group$unique_in_group) > 1) {
        print(df.uniqus.in.group)
        print(paste('vars.values_in_group', vars.values_in_group, sep=':'))
        print(paste('vars.group_by', vars.group_by, sep=':'))
        stop("The variables for which quantiles are to be taken are not identical within the group v
    }
}
```

2.4.2.1.1 Support Functions

2.4.2.1.2 Data Slicing and Quantile Generation

- Function 1: generate quantiles based on group-specific characteristics. the groups could be at the panel observation level as well.

```
# First Step, given groups, generate quantiles based on group characteristics
# vars.cts2quantile <- c('wealthIdx', 'hgt0', 'wgt0')
# seq.quantiles <- c(0, 0.3333, 0.6666, 1.0)
# vars.group_by <- c('indi.id')
# vars.arrange <- c('indi.id', 'svymthRound')
# vars.continuous <- c('wealthIdx', 'hgt0', 'wgt0')
df_sliced_quantiles <- function(df, vars.cts2quantile, seq.quantiles,
                                vars.group_by, vars.arrange) {

    # Slicing data
    df.grp.L1 <- df %>% group_by(!!!syms(vars.group_by)) %>% arrange(!!!syms(vars.arrange)) %>% slic

    # Quantiles based on sliced data
    df.sliced.quantiles <- lapply(vars.cts2quantile, gen_quantiles, df=df.grp.L1, prob=seq.quantiles

    return(list(df.sliced.quantiles=df.sliced.quantiles,
```

```
                df.grp.L1=df.grp.L1))
}
```

2.4.2.1.3 Data Cutting

- Function 2: cut groups for full panel dataframe based on group-specific characteristics quantiles.

```r
# Cutting Function, Cut Continuous Variables into Quantiles with labeing
f_cut <- function(var, df.sliced.quantiles, seq.quantiles, include.lowest=TRUE, fan.labels=TRUE, pri

    # unparsed string variable name
    var.str <- substitute(var)

    # Breaks
    arr.quantiles <- df.sliced.quantiles[[var.str]]
    arr.sort.unique.quantiles <- sort(unique(arr.quantiles))
    if (print) {
        print(arr.sort.unique.quantiles)
    }

    # Regular cutting With Standard Labels
    # TRUE, means the lowest group has closed bracket left and right
    var.quantile <- cut(var, breaks=arr.sort.unique.quantiles, include.lowest=include.lowest)

    # Use my custom labels
    if (fan.labels) {
        levels.suffix <- lapply(arr.sort.unique.quantiles[1:(length(arr.sort.unique.quantiles)-1)],
                                f_Q_label,
                                arr.quantiles=arr.quantiles,
                                seq.quantiles=seq.quantiles)
        if (print) {
            print(levels.suffix)
        }
        levels(var.quantile) <- paste0(levels(var.quantile), '; ', levels.suffix)
    }

    # Return
    return(var.quantile)
}
```

```r
# Combo Quantile Function
# vars.cts2quantile <- c('wealthIdx', 'hgt0', 'wgt0')
# seq.quantiles <- c(0, 0.3333, 0.6666, 1.0)
# vars.group_by <- c('indi.id')
# vars.arrange <- c('indi.id', 'svymthRound')
# vars.continuous <- c('wealthIdx', 'hgt0', 'wgt0')
df_cut_by_sliced_quantiles <- function(df, vars.cts2quantile, seq.quantiles,
                                       vars.group_by, vars.arrange) {

    # Check Are Values within Group By Unique? If not, STOP
    f_check_distinct_ingroup(df, vars.group_by, vars.values_in_group=vars.cts2quantile)

    # First Step Slicing
    df.sliced <- df_sliced_quantiles(df, vars.cts2quantile, seq.quantiles, vars.group_by, vars.arran

    # Second Step Generate Categorical Variables of Quantiles
    df.with.cut.quant <- df %>% mutate_at(vars.cts2quantile,
                              funs(q=f_cut(., df.sliced$df.sliced.quantiles,
                                           seq.quantiles=seq.quantiles,
```

```
                                                    include.lowest=TRUE, fan.labels=TRUE)))

    if (length(vars.cts2quantile) > 1) {
        df.with.cut.quant <- df.with.cut.quant %>%
                              rename_at(vars(contains('_q')),
                                        funs(f_var_rename(., seq.quantiles=seq.quantiles)))
    } else {
        new.var.name <- paste0(vars.cts2quantile[1], '_', f_Q_suffix(seq.quantiles))
        df.with.cut.quant <- df.with.cut.quant %>% rename(!!new.var.name := q)
    }

    # Newly Generated Quantile-Cut Variables
    vars.quantile.cut <- df.with.cut.quant %>%
                select(matches(paste0(vars.cts2quantile, collapse='|'))) %>%
                select(matches(f_Q_suffix(seq.quantiles)))

    # Return
    return(list(df.with.cut.quant = df.with.cut.quant,
                df.sliced.quantiles=df.sliced$df.sliced.quantiles,
                df.grp.L1=df.sliced$df.grp.L1,
                vars.quantile.cut=vars.quantile.cut))

}
```

2.4.2.1.4 Different Vars Different Probabilities Joint Quantiles

- Accomondate multiple continuousv ariables
- Different percentiles
- list of lists
- generate joint categorical variables
- keep only values that exist for all quantile base vars

```
# Function to handle list inputs with different quantiles vars and probabilities
df_cut_by_sliced_quantiles_grps <- function(quantile.grp.list, df, vars.group_by, vars.arrange) {
   vars.cts2quantile <- quantile.grp.list$vars
   seq.quantiles <- quantile.grp.list$prob
   return(df_cut_by_sliced_quantiles(df, vars.cts2quantile, seq.quantiles, vars.group_by, vars.arran
}
# Show Results
df_cut_by_sliced_quantiles_joint_results_grped <- function(df.with.cut.quant.all, vars.cts2quantile,
                                                           vars.quantile.cut.all, var.qjnt.grp.idx)
    # Show ALL
    df.group.panel.cnt.mean <- df.with.cut.quant.all %>% group_by(!!!syms(vars.quantile.cut.all), !!
            summarise_at(vars.cts2quantile, funs(mean, n()))

    # Show Based on SLicing first
    df.group.slice1.cnt.mean <- df.with.cut.quant.all %>% group_by(!!!syms(vars.group_by)) %>% arran
            group_by(!!!syms(vars.quantile.cut.all), !!sym(var.qjnt.grp.idx)) %>%
            summarise_at(vars.cts2quantile, funs(mean, n()))

    return(list(df.group.panel.cnt.mean=df.group.panel.cnt.mean,
                df.group.slice1.cnt.mean=df.group.slice1.cnt.mean))
}
```

```
# # Joint Quantile Group Name
# var.qjnt.grp.idx <- 'group.index'
# # Generate Categorical Variables of Quantiles
# vars.group_by <- c('indi.id')
# vars.arrange <- c('indi.id', 'svymthRound')
```

```
# # Quantile Variables and Quantiles
# vars.cts2quantile.wealth <- c('wealthIdx')
# seq.quantiles.wealth <- c(0, .5, 1.0)
# vars.cts2quantile.wgthgt <- c('hgt0', 'wgt0')
# seq.quantiles.wgthgt <- c(0, .3333, 0.6666, 1.0)
# drop.any.quantile.na <- TRUE
# # collect to list
# list.cts2quantile <- list(list(vars=vars.cts2quantile.wealth,
#                                prob=seq.quantiles.wealth),
#                           list(vars=vars.cts2quantile.wgthgt,
#                                prob=seq.quantiles.wgthgt))

df_cut_by_sliced_quantiles_joint <- function(df, var.qjnt.grp.idx,
                                             list.cts2quantile,
                                             vars.group_by, vars.arrange,
                                             drop.any.quantile.na = TRUE,
                                             toprint = TRUE) {

  #  Original dimensions
  if(toprint) {
   print(dim(df))
  }

  # All Continuous Variables from lists
  vars.cts2quantile <- unlist(lapply(list.cts2quantile, function(elist) elist$vars))
  vars.cts2quantile

  # Keep only if not NA for all Quantile variables
  if (drop.any.quantile.na) {
   df.select <- df %>% drop_na(c(vars.group_by, vars.arrange, vars.cts2quantile))
  } else {
   df.select <- df
  }

  if(toprint) {
   print(dim(df.select))
  }

  # Apply qunatile function to all elements of list of list
  df.cut.list <- lapply(list.cts2quantile, df_cut_by_sliced_quantiles_grps,
                        df=df.select, vars.group_by=vars.group_by, vars.arrange=vars.arrange)

  # Reduce Resulting Core Panel Matrix Together
  df.with.cut.quant.all <- lapply(df.cut.list, function(elist) elist$df.with.cut.quant) %>% reduce(l
  df.sliced.quantiles.all <- lapply(df.cut.list, function(elist) elist$df.sliced.quantiles)

  if(toprint) {
    print(dim(df.with.cut.quant.all))
  }

  # Obrain Newly Created Quantile Group Variables
  vars.quantile.cut.all <- unlist(lapply(df.cut.list, function(elist) names(elist$vars.quantile.cut)
  if(toprint) {
    print(vars.quantile.cut.all)
    print(summary(df.with.cut.quant.all %>% select(one_of(vars.quantile.cut.all))))
  }

  # Generate Joint Quantile Index Variable
```

```
  df.with.cut.quant.all <- df.with.cut.quant.all %>% mutate(!!var.qjnt.grp.idx := group_indices(., !

  # Quantile Groups
  arr.group.idx <- t(sort(unique(df.with.cut.quant.all[[var.qjnt.grp.idx]])))

  # Results Display
  df.group.print <- df_cut_by_sliced_quantiles_joint_results_grped(df.with.cut.quant.all, vars.cts2q
                                                      vars.group_by, vars.arrange,
                                                      vars.quantile.cut.all, var.qjnt.grp.idx)

  # list to Return
  # These returns are the same as returns earlier: df_cut_by_sliced_quantiles
  # Except that they are combined together
  return(list(df.with.cut.quant = df.with.cut.quant.all,
              df.sliced.quantiles = df.sliced.quantiles.all,
              df.grp.L1 = (df.cut.list[[1]])$df.grp.L1,
              vars.quantile.cut = vars.quantile.cut.all,
              df.group.panel.cnt.mean = df.group.print$df.group.panel.cnt.mean,
              df.group.slice1.cnt.mean = df.group.print$df.group.slice1.cnt.mean))

}
```

2.4.2.2 Program Testing

Load Data

```
# Library
library(tidyverse)

# Load Sample Data
setwd('C:/Users/fan/R4Econ/_data/')
df <- read_csv('height_weight.csv')
```

```
## Parsed with column specification:
## cols(
##   S.country = col_character(),
##   vil.id = col_double(),
##   indi.id = col_double(),
##   sex = col_character(),
##   svymthRound = col_double(),
##   momEdu = col_double(),
##   wealthIdx = col_double(),
##   hgt = col_double(),
##   wgt = col_double(),
##   hgt0 = col_double(),
##   wgt0 = col_double(),
##   prot = col_double(),
##   cal = col_double(),
##   p.A.prot = col_double(),
##   p.A.nProt = col_double()
## )
```

```
# Joint Quantile Group Name
var.qjnt.grp.idx <- 'group.index'
list.cts2quantile <- list(list(vars=c('hgt0'), prob=c(0, .3333, 0.6666, 1.0)))
results <- df_cut_by_sliced_quantiles_joint(df, var.qjnt.grp.idx, list.cts2quantile,
                                            vars.group_by = c('indi.id'), vars.arrange = c('indi.id'
```

```r
                                    drop.any.quantile.na = TRUE, toprint = FALSE)
# Show Results
results$df.group.slice1.cnt.mean
```

2.4.2.2.1 Hgt0 3 Groups

```
## # A tibble: 3 x 4
## # Groups:   hgt0_Qs0e1n3 [3]
##   hgt0_Qs0e1n3               group.index  mean     n
##   <fct>                            <int> <dbl> <int>
## 1 [40.6,48.5]; (1) of Qs0e1n3          1  47.0   580
## 2 (48.5,50.2]; (2) of Qs0e1n3          2  49.4   561
## 3 (50.2,58]; (3) of Qs0e1n3            3  51.7   568
```

```r
# Joint Quantile Group Name
var.qjnt.grp.idx <- 'wltQuintle.index'
list.cts2quantile <- list(list(vars=c('wealthIdx'), prob=seq(0, 1.0, 0.20)))
results <- df_cut_by_sliced_quantiles_joint((df %>% filter(S.country == 'Guatemala')),
                                    var.qjnt.grp.idx, list.cts2quantile,
                                    vars.group_by = c('indi.id'), vars.arrange = c('indi.id'
                                    drop.any.quantile.na = TRUE, toprint = FALSE)
# Show Results
results$df.group.slice1.cnt.mean
```

2.4.2.2.2 Wealth 5 Groups Guatemala

```
## # A tibble: 5 x 4
## # Groups:   wealthIdx_Qs0e1n5 [5]
##   wealthIdx_Qs0e1n5         wltQuintle.index  mean     n
##   <fct>                                <int> <dbl> <int>
## 1 [1,1.6]; (1) of Qs0e1n5                  1  1.25   151
## 2 (1.6,2.1]; (2) of Qs0e1n5                2  1.82   139
## 3 (2.1,2.3]; (3) of Qs0e1n5                3  2.25   139
## 4 (2.3,2.9]; (4) of Qs0e1n5                4  2.70   134
## 5 (2.9,6.6]; (5) of Qs0e1n5                5  3.77   111
```

```r
# Joint Quantile Group Name
var.qjnt.grp.idx <- 'group.index'
list.cts2quantile <- list(list(vars=c('hgt0', 'wgt0'), prob=c(0, .5, 1.0)))
results <- df_cut_by_sliced_quantiles_joint(df, var.qjnt.grp.idx, list.cts2quantile,
                                    vars.group_by = c('indi.id'), vars.arrange = c('indi.id'
                                    drop.any.quantile.na = TRUE, toprint = FALSE)
```

2.4.2.2.3 Hgt0 2 groups, Wgt0 2 groups too

```
## Joining, by = "quant.perc"
```

```r
# Show Results
results$df.group.slice1.cnt.mean
```

```
## # A tibble: 4 x 7
## # Groups:   hgt0_Qs0e1n2, wgt0_Qs0e1n2 [4]
##   hgt0_Qs0e1n2               wgt0_Qs0e1n2                         group.index hgt0_mean wgt0_mean
##   <fct>                      <fct>                                      <int>     <dbl>     <dbl>
## 1 [40.6,49.4]; (1) of Qs0e1n2 [1.4e+03,3.01e+03]; (1) of Qs0e1n2           1      47.4     2650.
## 2 [40.6,49.4]; (1) of Qs0e1n2 (3.01e+03,5.49e+03]; (2) of Qs0e1n2          2      48.5     3244.
## 3 (49.4,58]; (2) of Qs0e1n2   [1.4e+03,3.01e+03]; (1) of Qs0e1n2           3      50.4     2829.
## 4 (49.4,58]; (2) of Qs0e1n2   (3.01e+03,5.49e+03]; (2) of Qs0e1n2          4      51.3     3483.
```

```
# Joint Quantile Group Name
var.qjnt.grp.idx <- 'group.index'
list.cts2quantile <- list(list(vars=c('wealthIdx'), prob=c(0, .5, 1.0)), list(vars=c('hgt0'), prob=c
results <- df_cut_by_sliced_quantiles_joint((df %>% filter(S.country == 'Cebu')),
                                             var.qjnt.grp.idx, list.cts2quantile,
                                             vars.group_by = c('indi.id'), vars.arrange = c('indi.id
                                             drop.any.quantile.na = TRUE, toprint = FALSE)
```

2.4.2.2.4 Hgt0 2 groups, Wealth 2 groups, Cebu Only

```
## Joining, by = c("S.country", "vil.id", "indi.id", "sex", "svymthRound", "momEdu", "wealthIdx", "h
## "p.A.nProt")
```

```
# Show Results
results$df.group.slice1.cnt.mean
```

```
## # A tibble: 6 x 7
## # Groups:   wealthIdx_Qs0e1n2, hgt0_Qs0e1n3 [6]
##   wealthIdx_Qs0e1n2          hgt0_Qs0e1n3              group.index wealthIdx_mean hgt0_mean wea
##   <fct>                      <fct>                           <int>          <dbl>     <dbl>
## 1 [5.2,8.3]; (1) of Qs0e1n2  [41.1,48.4]; (1) of Qs0e1n3         1           7.15      46.9
## 2 [5.2,8.3]; (1) of Qs0e1n2  (48.4,50.1]; (2) of Qs0e1n3         2           7.18      49.2
## 3 [5.2,8.3]; (1) of Qs0e1n2  (50.1,58]; (3) of Qs0e1n3           3           7.13      51.3
## 4 (8.3,19.3]; (2) of Qs0e1n2 [41.1,48.4]; (1) of Qs0e1n3         4          11.1       47.2
## 5 (8.3,19.3]; (2) of Qs0e1n2 (48.4,50.1]; (2) of Qs0e1n3         5          11.2       49.3
## 6 (8.3,19.3]; (2) of Qs0e1n2 (50.1,58]; (3) of Qs0e1n3           6          11.6       51.7
```

2.4.2.2.5 Results of income + Wgt0 + Hgt0 joint Gruops in Cebu

Weight at month 0 below and above median, height at month zero into three terciles.

```
# Joint Quantile Group Name
var.qjnt.grp.idx <- 'wltHgt0Wgt0.index'
list.cts2quantile <- list(list(vars=c('wealthIdx'), prob=c(0, .5, 1.0)), list(vars=c('hgt0', 'wgt0')
results <- df_cut_by_sliced_quantiles_joint((df %>% filter(S.country == 'Cebu')),
                                            var.qjnt.grp.idx, list.cts2quantile,
                                            vars.group_by = c('indi.id'), vars.arrange = c('indi.id'
                                            drop.any.quantile.na = TRUE, toprint = FALSE)
```

```
## Joining, by = "quant.perc"Joining, by = c("S.country", "vil.id", "indi.id", "sex", "svymthRound",
## "prot", "cal", "p.A.prot", "p.A.nProt")
```

```
# Show Results
results$df.group.slice1.cnt.mean
```

```
## # A tibble: 8 x 10
## # Groups:   wealthIdx_Qs0e1n2, hgt0_Qs0e1n2, wgt0_Qs0e1n2 [8]
##   wealthIdx_Qs0e1n2      hgt0_Qs0e1n2           wgt0_Qs0e1n2                  wltHgt0Wgt0.ind~ w
##   <fct>                  <fct>                  <fct>                                    <int>
## 1 [5.2,8.3]; (1) of Qs0e~ [41.1,49.2]; (1) of Qs~ [1.4e+03,2.98e+03]; (1) of ~               1
## 2 [5.2,8.3]; (1) of Qs0e~ [41.1,49.2]; (1) of Qs~ (2.98e+03,5.49e+03]; (2) of~               2
## 3 [5.2,8.3]; (1) of Qs0e~ (49.2,58]; (2) of Qs0e~ [1.4e+03,2.98e+03]; (1) of ~               3
## 4 [5.2,8.3]; (1) of Qs0e~ (49.2,58]; (2) of Qs0e~ (2.98e+03,5.49e+03]; (2) of~               4
## 5 (8.3,19.3]; (2) of Qs0~ [41.1,49.2]; (1) of Qs~ [1.4e+03,2.98e+03]; (1) of ~               5
## 6 (8.3,19.3]; (2) of Qs0~ [41.1,49.2]; (1) of Qs~ (2.98e+03,5.49e+03]; (2) of~               6
## 7 (8.3,19.3]; (2) of Qs0~ (49.2,58]; (2) of Qs0e~ [1.4e+03,2.98e+03]; (1) of ~               7
## 8 (8.3,19.3]; (2) of Qs0~ (49.2,58]; (2) of Qs0e~ (2.98e+03,5.49e+03]; (2) of~               8
```

2.4.2.3 Line by Line–Quantiles Var by Var

The idea of the function is to generate quantiles levels first, and then use those to generate the categories based on quantiles. Rather than doing this in one step. These are done in two steps, to increase clarity in the quantiles used for quantile category generation. And a dataframe with these quantiles are saved as a separate output of the function.

2.4.2.3.1 Dataframe of Variables' Group-by Level Quantiles Quantiles from Different Variables. Note that these variables are specific to the individual, not individual/month. So we need to first slick the data, so that we only get the first rows.

Do this in several steps to clarify group_by level. No speed loss.

```
# Selected Variables, many Percentiles
vars.group_by <- c('indi.id')
vars.arrange <- c('indi.id', 'svymthRound')
vars.cts2quantile <- c('wealthIdx', 'hgt0', 'wgt0')
seq.quantiles <- c(0, 0.3333, 0.6666, 1.0)
df.sliced <- df_sliced_quantiles(df, vars.cts2quantile, seq.quantiles, vars.group_by, vars.arrange)
```

```
## Joining, by = "quant.perc"Joining, by = "quant.perc"
```

```
df.sliced.quantiles <- df.sliced$df.sliced.quantiles
df.grp.L1 <- df.sliced$df.grp.L1
```

```
df.sliced.quantiles
```

```
## # A tibble: 4 x 4
##   quant.perc wealthIdx  hgt0  wgt0
##   <chr>          <dbl> <dbl> <dbl>
## 1 0%               1    40.6 1402.
## 2 33.33%           5.2  48.5 2843.
## 3 66.66%           8.3  50.2 3209.
## 4 100%            19.3  58   5494.
```

```
# Quantiles all Variables
suppressMessages(lapply(names(df), gen_quantiles, df=df.grp.L1, prob=seq(0.1,0.9,0.10)) %>% reduce(f
```

```
## Warning in quantile(as.numeric(df[[var]]), prob, na.rm = TRUE): NAs introduced by coercion

## Warning in quantile(as.numeric(df[[var]]), prob, na.rm = TRUE): NAs introduced by coercion
```

```
## # A tibble: 9 x 16
##   quant.perc S.country vil.id indi.id   sex svymthRound momEdu wealthIdx   hgt   wgt  hgt0  wgt0
##   <chr>          <dbl>  <dbl>   <dbl> <dbl>       <dbl>  <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 10%               NA      3    203.    NA           0    5.7       1.7  46.3 1397.  46.6 2500.
## 2 20%               NA      4    405.    NA           0    6.9       2.3  47.3 1840.  47.7 2686.
## 3 30%               NA      6    608.    NA           0    7.7       3.3  48   2272.  48.3 2804.
## 4 40%               NA      8    810.    NA           0    8.6       6.3  48.7 2669.  48.8 2910.
## 5 50%               NA      9   1012     NA           0    9.3       7.3  49.4 3050.  49.4 3013
## 6 60%               NA     13   1214.    NA           0   10.4       8.3  49.9 3440.  49.9 3126.
## 7 70%               NA     14   1416.    NA           0   11.4       8.3  50.5 3857.  50.4 3250.
## 8 80%               NA     17   1619.    NA           0   12.7       9.3  51.2 4258.  51.0 3418.
## 9 90%               NA     26   1821.    NA           0   14.6      11.3  52.3 4704.  52   3683.
```

2.4.2.3.2 Cut Quantile Categorical Variables Using the Quantiles we have generate, cut the continuous variables to generate categorical quantile variables in the full dataframe.

Note that we can only cut based on unique breaks, but sometimes quantile break-points are the same if some values are often observed, and also if there are too few observations with respect to quantile groups.

To resolve this issue, we only look at unique quantiles.

We need several support Functions: 1. support functions to generate suffix for quantile variables based on quantile cuts 2. support for labeling variables of resulting quantiles beyond bracketing

```
# Function Testing
arr.quantiles <- df.sliced.quantiles[[substitute('wealthIdx')]]
arr.quantiles
```

```
## [1]  1.0  5.2  8.3 19.3
```

```
arr.sort.unique.quantiles <- sort(unique(df.sliced.quantiles[[substitute('wealthIdx')]]))
arr.sort.unique.quantiles
```

```
## [1]  1.0  5.2  8.3 19.3
```

```
f_Q_label(arr.quantiles, arr.sort.unique.quantiles[1], seq.quantiles)
```

```
## [1] "(1) of Qs0e1n3"
```

```
f_Q_label(arr.quantiles, arr.sort.unique.quantiles[2], seq.quantiles)
```

```
## [1] "(2) of Qs0e1n3"
```

```
lapply(arr.sort.unique.quantiles[1:(length(arr.sort.unique.quantiles)-1)],
       f_Q_label,
       arr.quantiles=arr.quantiles,
       seq.quantiles=seq.quantiles)
```

```
## [[1]]
## [1] "(1) of Qs0e1n3"
##
## [[2]]
## [1] "(2) of Qs0e1n3"
##
## [[3]]
## [1] "(3) of Qs0e1n3"
```

```
# Generate Categorical Variables of Quantiles
vars.group_by <- c('indi.id')
vars.arrange <- c('indi.id', 'svymthRound')
vars.cts2quantile <- c('wealthIdx', 'hgt0', 'wgt0')
seq.quantiles <- c(0, 0.3333, 0.6666, 1.0)
df.cut <- df_cut_by_sliced_quantiles(df, vars.cts2quantile, seq.quantiles, vars.group_by, vars.arran
```

```
## Joining, by = "quant.perc"Joining, by = "quant.perc"
```

```
vars.quantile.cut <- df.cut$vars.quantile.cut
df.with.cut.quant <- df.cut$df.with.cut.quant
df.grp.L1 <- df.cut$df.grp.L1
```

```
# Cut Variables Generated
names(vars.quantile.cut)
```

```
## [1] "wealthIdx_Qs0e1n3" "hgt0_Qs0e1n3"      "wgt0_Qs0e1n3"
```

```
summary(vars.quantile.cut)
```

```
##                    wealthIdx_Qs0e1n3                      hgt0_Qs0e1n3
##  [1,5.2]; (1) of Qs0e1n3   :10958   [40.6,48.5]; (1) of Qs0e1n3:10232   [1.4e+03,2.84e+03]; (1) o
##  (5.2,8.3]; (2) of Qs0e1n3 :13812   (48.5,50.2]; (2) of Qs0e1n3: 9895   (2.84e+03,3.21e+03]; (2)
##  (8.3,19.3]; (3) of Qs0e1n3:10295   (50.2,58]; (3) of Qs0e1n3  : 9908   (3.21e+03,5.49e+03]; (3)
##                                     NA's                       : 5030   NA's
```

```
# options(repr.matrix.max.rows=50, repr.matrix.max.cols=20)
# df.with.cut.quant
```

```r
# Group By Results
f.count <- function(df, var.cts, seq.quantiles) {
    df %>% select(S.country, indi.id, svymthRound, matches(paste0(var.cts, collapse='|'))) %>%
        group_by(!!sym(f_var_rename(paste0(var.cts,'_q'), seq.quantiles))) %>%
        summarise_all(funs(n=n()))
}
```

```r
# Full Panel Results
lapply(vars.cts2quantile, f.count, df=df.with.cut.quant, seq.quantiles=seq.quantiles)
```

2.4.2.3.3 Individual Variables' Quantile Cuts Review Results

```
## Warning: Factor `hgt0_Qs0e1n3` contains implicit NA, consider using `forcats::fct_explicit_na`

## Warning: Factor `wgt0_Qs0e1n3` contains implicit NA, consider using `forcats::fct_explicit_na`

## [[1]]
## # A tibble: 3 x 5
##   wealthIdx_Qs0e1n3          S.country_n indi.id_n svymthRound_n wealthIdx_n
##   <fct>                            <int>     <int>         <int>       <int>
## 1 [1,5.2]; (1) of Qs0e1n3          10958     10958         10958       10958
## 2 (5.2,8.3]; (2) of Qs0e1n3        13812     13812         13812       13812
## 3 (8.3,19.3]; (3) of Qs0e1n3       10295     10295         10295       10295
##
## [[2]]
## # A tibble: 4 x 5
##   hgt0_Qs0e1n3                 S.country_n indi.id_n svymthRound_n hgt0_n
##   <fct>                              <int>     <int>         <int>  <int>
## 1 [40.6,48.5]; (1) of Qs0e1n3        10232     10232         10232  10232
## 2 (48.5,50.2]; (2) of Qs0e1n3         9895      9895          9895   9895
## 3 (50.2,58]; (3) of Qs0e1n3           9908      9908          9908   9908
## 4 <NA>                                5030      5030          5030   5030
##
## [[3]]
## # A tibble: 4 x 5
##   wgt0_Qs0e1n3                             S.country_n indi.id_n svymthRound_n wgt0_n
##   <fct>                                          <int>     <int>         <int>  <int>
## 1 [1.4e+03,2.84e+03]; (1) of Qs0e1n3             10105     10105         10105  10105
## 2 (2.84e+03,3.21e+03]; (2) of Qs0e1n3            10056     10056         10056  10056
## 3 (3.21e+03,5.49e+03]; (3) of Qs0e1n3             9858      9858          9858   9858
## 4 <NA>                                            5046      5046          5046   5046
```

```r
# Results Individual Slice
lapply(vars.cts2quantile, f.count,
       df=(df.with.cut.quant %>% group_by(!!!syms(vars.group_by)) %>% arrange(!!!syms(vars.arrange))),
       seq.quantiles = seq.quantiles)
```

```
## Warning: Factor `hgt0_Qs0e1n3` contains implicit NA, consider using `forcats::fct_explicit_na`

## Warning: Factor `wgt0_Qs0e1n3` contains implicit NA, consider using `forcats::fct_explicit_na`

## [[1]]
## # A tibble: 3 x 5
##   wealthIdx_Qs0e1n3          S.country_n indi.id_n svymthRound_n wealthIdx_n
##   <fct>                            <int>     <int>         <int>       <int>
## 1 [1,5.2]; (1) of Qs0e1n3            683       683           683         683
## 2 (5.2,8.3]; (2) of Qs0e1n3          768       768           768         768
## 3 (8.3,19.3]; (3) of Qs0e1n3         572       572           572         572
##
## [[2]]
```

```
## # A tibble: 4 x 5
##   hgt0_Qs0e1n3                S.country_n indi.id_n svymthRound_n hgt0_n
##   <fct>                             <int>     <int>         <int>  <int>
## 1 [40.6,48.5]; (1) of Qs0e1n3         580       580           580    580
## 2 (48.5,50.2]; (2) of Qs0e1n3         561       561           561    561
## 3 (50.2,58]; (3) of Qs0e1n3           568       568           568    568
## 4 <NA>                                314       314           314    314
##
## [[3]]
## # A tibble: 4 x 5
##   wgt0_Qs0e1n3                        S.country_n indi.id_n svymthRound_n wgt0_n
##   <fct>                                     <int>     <int>         <int>  <int>
## 1 [1.4e+03,2.84e+03]; (1) of Qs0e1n3          569       569           569    569
## 2 (2.84e+03,3.21e+03]; (2) of Qs0e1n3         569       569           569    569
## 3 (3.21e+03,5.49e+03]; (3) of Qs0e1n3         570       570           570    570
## 4 <NA>                                        315       315           315    315
```

2.4.2.4 Differential Quantiles for Different Variables Then Combine to Form New Groups

Collect together different quantile base variables and their percentile cuttings quantile rules. Input Parameters.

```r
# Generate Categorical Variables of Quantiles
vars.group_by <- c('indi.id')
vars.arrange <- c('indi.id', 'svymthRound')
```

```r
# Quantile Variables and Quantiles
vars.cts2quantile.wealth <- c('wealthIdx')
seq.quantiles.wealth <- c(0, .5, 1.0)
vars.cts2quantile.wgthgt <- c('hgt0', 'wgt0')
seq.quantiles.wgthgt <- c(0, .3333, 0.6666, 1.0)
drop.any.quantile.na <- TRUE
# collect to list
list.cts2quantile <- list(list(vars=vars.cts2quantile.wealth,
                               prob=seq.quantiles.wealth),
                          list(vars=vars.cts2quantile.wgthgt,
                               prob=seq.quantiles.wgthgt))
```

2.4.2.5 Check if Within Group Variables Are The Same

Need to make sure quantile variables are unique within groups

```r
vars.cts2quantile <- unlist(lapply(list.cts2quantile, function(elist) elist$vars))
f_check_distinct_ingroup(df, vars.group_by, vars.values_in_group=vars.cts2quantile)
```

```r


# Original dimensions
dim(df)
```

2.4.2.5.1 Keep only non-NA for all Quantile Variables

```
## [1] 35065    15
```

```r
# All Continuous Variables from lists
vars.cts2quantile <- unlist(lapply(list.cts2quantile, function(elist) elist$vars))
vars.cts2quantile
```

```
## [1] "wealthIdx" "hgt0"      "wgt0"
```

```r
# Keep only if not NA for all Quantile variables
if (drop.any.quantile.na) {
    df.select <- df %>% drop_na(c(vars.group_by, vars.arrange, vars.cts2quantile))
```

```
}
dim(df.select)
```

```
## [1] 30019    15
```

```
# Dealing with a list of quantile variables
df.cut.wealth <- df_cut_by_sliced_quantiles(df.select, vars.cts2quantile.wealth, seq.quantiles.wealt
summary(df.cut.wealth$vars.quantile.cut)
```

2.4.2.5.2 Apply Quantiles for Each Quantile Variable

```
##                  wealthIdx_Qs0e1n2
##  [1,7.3]; (1) of Qs0e1n2   :14936
##  (7.3,19.3]; (2) of Qs0e1n2:15083
```

```
# summary((df.cut.wealth$df.with.cut.quant)[['wealthIdx_Qs0e1n2']])
# df.cut.wealth$df.with.cut.quant %>% filter(is.na(wealthIdx_Qs0e1n2))
# df.cut.wealth$df.with.cut.quant %>% filter(indi.id == 500)
```

```
df.cut.wgthgt <- df_cut_by_sliced_quantiles(df.select, vars.cts2quantile.wgthgt, seq.quantiles.wgthg
```

```
## Joining, by = "quant.perc"
```

```
summary(df.cut.wgthgt$vars.quantile.cut)
```

```
##                     hgt0_Qs0e1n3                               wgt0_Qs0e1n3
##  [40.6,48.5]; (1) of Qs0e1n3:10216   [1.4e+03,2.84e+03]; (1) of Qs0e1n3 :10105
##  (48.5,50.2]; (2) of Qs0e1n3: 9895   (2.84e+03,3.21e+03]; (2) of Qs0e1n3:10056
##  (50.2,58]; (3) of Qs0e1n3  : 9908   (3.21e+03,5.49e+03]; (3) of Qs0e1n3: 9858
```

```
# Function to handle list inputs with different quantiles vars and probabilities
df_cut_by_sliced_quantiles_grps <- function(quantile.grp.list, df, vars.group_by, vars.arrange) {
    vars.cts2quantile <- quantile.grp.list$vars
    seq.quantiles <- quantile.grp.list$prob
    return(df_cut_by_sliced_quantiles(df, vars.cts2quantile, seq.quantiles, vars.group_by, vars.arra
}
```

```
# Apply function
df.cut.list <- lapply(list.cts2quantile, df_cut_by_sliced_quantiles_grps,
                      df=df.select, vars.group_by=vars.group_by, vars.arrange=vars.arrange)
```

2.4.2.5.3 Apply Quantiles Functionally

```
## Joining, by = "quant.perc"
```

```
# Reduce Resulting Matrixes Together
df.with.cut.quant.all <- lapply(df.cut.list, function(elist) elist$df.with.cut.quant) %>% reduce(lef
```

```
## Joining, by = c("S.country", "vil.id", "indi.id", "sex", "svymthRound", "momEdu", "wealthIdx", "h
## "p.A.nProt")
```

```
dim(df.with.cut.quant.all)
```

```
## [1] 30019    18
```

```
# Obrain Newly Created Quantile Group Variables
vars.quantile.cut.all <- unlist(lapply(df.cut.list, function(elist) names(elist$vars.quantile.cut)))
vars.quantile.cut.all
```

```
## [1] "wealthIdx_Qs0e1n2" "hgt0_Qs0e1n3"      "wgt0_Qs0e1n3"
```

2.4.2.5.4 Summarize by Groups

Summarize by all groups.

```
summary(df.with.cut.quant.all %>% select(one_of(vars.quantile.cut.all)))
```

```
##                   wealthIdx_Qs0e1n2                        hgt0_Qs0e1n3
##  [1,7.3]; (1) of Qs0e1n2   :14936   [40.6,48.5]; (1) of Qs0e1n3:10216   [1.4e+03,2.84e+03]; (1) o
##  (7.3,19.3]; (2) of Qs0e1n2:15083   (48.5,50.2]; (2) of Qs0e1n3: 9895   (2.84e+03,3.21e+03]; (2)
##                                     (50.2,58]; (3) of Qs0e1n3  : 9908   (3.21e+03,5.49e+03]; (3)
```

```
# df.with.cut.quant.all %>%
#     group_by(!!!syms(vars.quantile.cut.all)) %>%
#     summarise_at(vars.cts2quantile, funs(mean, n()))
```

```
# Generate Joint Quantile Index Variable
var.qjnt.grp.idx <- 'group.index'
df.with.cut.quant.all <- df.with.cut.quant.all %>% mutate(!!var.qjnt.grp.idx := group_indices(., !!!
```

```
arr.group.idx <- t(sort(unique(df.with.cut.quant.all[[var.qjnt.grp.idx]])))
arr.group.idx
```

2.4.2.5.5 Generate Joint Quantile Vars Unique Groups

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17]
## [1,]    1    2    3    4    5    6    7    8    9    10    11    12    13    14    15    16    17
```

```
df.with.cut.quant.all %>% group_by(!!!syms(vars.quantile.cut.all), !!sym(var.qjnt.grp.idx)) %>%
        summarise_at(vars.cts2quantile, funs(mean, n()))
```

```
## # A tibble: 18 x 10
## # Groups:   wealthIdx_Qs0e1n2, hgt0_Qs0e1n3, wgt0_Qs0e1n3 [18]
##    wealthIdx_Qs0e1n2         hgt0_Qs0e1n3             wgt0_Qs0e1n3                    group.index w
##    <fct>                     <fct>                    <fct>                                 <int>
##  1 [1,7.3]; (1) of Qs0e1n2  [40.6,48.5]; (1) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~           1
##  2 [1,7.3]; (1) of Qs0e1n2  [40.6,48.5]; (1) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~           2
##  3 [1,7.3]; (1) of Qs0e1n2  [40.6,48.5]; (1) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~           3
##  4 [1,7.3]; (1) of Qs0e1n2  (48.5,50.2]; (2) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~           4
##  5 [1,7.3]; (1) of Qs0e1n2  (48.5,50.2]; (2) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~           5
##  6 [1,7.3]; (1) of Qs0e1n2  (48.5,50.2]; (2) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~           6
##  7 [1,7.3]; (1) of Qs0e1n2  (50.2,58]; (3) of Qs0e1~ [1.4e+03,2.84e+03]; (1) of Qs~           7
##  8 [1,7.3]; (1) of Qs0e1n2  (50.2,58]; (3) of Qs0e1~ (2.84e+03,3.21e+03]; (2) of Q~           8
##  9 [1,7.3]; (1) of Qs0e1n2  (50.2,58]; (3) of Qs0e1~ (3.21e+03,5.49e+03]; (3) of Q~           9
## 10 (7.3,19.3]; (2) of Qs0e~ [40.6,48.5]; (1) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~          10
## 11 (7.3,19.3]; (2) of Qs0e~ [40.6,48.5]; (1) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~          11
## 12 (7.3,19.3]; (2) of Qs0e~ [40.6,48.5]; (1) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~          12
## 13 (7.3,19.3]; (2) of Qs0e~ (48.5,50.2]; (2) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~          13
## 14 (7.3,19.3]; (2) of Qs0e~ (48.5,50.2]; (2) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~          14
## 15 (7.3,19.3]; (2) of Qs0e~ (48.5,50.2]; (2) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~          15
## 16 (7.3,19.3]; (2) of Qs0e~ (50.2,58]; (3) of Qs0e1~ [1.4e+03,2.84e+03]; (1) of Qs~          16
## 17 (7.3,19.3]; (2) of Qs0e~ (50.2,58]; (3) of Qs0e1~ (2.84e+03,3.21e+03]; (2) of Q~          17
## 18 (7.3,19.3]; (2) of Qs0e~ (50.2,58]; (3) of Qs0e1~ (3.21e+03,5.49e+03]; (3) of Q~          18
```

```
df.with.cut.quant.all  %>% group_by(!!!syms(vars.group_by)) %>% arrange(!!!syms(vars.arrange)) %>% s
        group_by(!!!syms(vars.quantile.cut.all), !!sym(var.qjnt.grp.idx)) %>%
        summarise_at(vars.cts2quantile, funs(mean, n()))
```

```
## # A tibble: 18 x 10
## # Groups:   wealthIdx_Qs0e1n2, hgt0_Qs0e1n3, wgt0_Qs0e1n3 [18]
##    wealthIdx_Qs0e1n2         hgt0_Qs0e1n3             wgt0_Qs0e1n3                    group.index w
##    <fct>                     <fct>                    <fct>                                 <int>
##  1 [1,7.3]; (1) of Qs0e1n2  [40.6,48.5]; (1) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~           1
```

```
##  2 [1,7.3]; (1) of Qs0e1n2  [40.6,48.5]; (1) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~          2
##  3 [1,7.3]; (1) of Qs0e1n2  [40.6,48.5]; (1) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~          3
##  4 [1,7.3]; (1) of Qs0e1n2  (48.5,50.2]; (2) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~          4
##  5 [1,7.3]; (1) of Qs0e1n2  (48.5,50.2]; (2) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~          5
##  6 [1,7.3]; (1) of Qs0e1n2  (48.5,50.2]; (2) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~          6
##  7 [1,7.3]; (1) of Qs0e1n2  (50.2,58]; (3) of Qs0e1~ [1.4e+03,2.84e+03]; (1) of Qs~          7
##  8 [1,7.3]; (1) of Qs0e1n2  (50.2,58]; (3) of Qs0e1~ (2.84e+03,3.21e+03]; (2) of Q~          8
##  9 [1,7.3]; (1) of Qs0e1n2  (50.2,58]; (3) of Qs0e1~ (3.21e+03,5.49e+03]; (3) of Q~          9
## 10 (7.3,19.3]; (2) of Qs0e~ [40.6,48.5]; (1) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~         10
## 11 (7.3,19.3]; (2) of Qs0e~ [40.6,48.5]; (1) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~         11
## 12 (7.3,19.3]; (2) of Qs0e~ [40.6,48.5]; (1) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~         12
## 13 (7.3,19.3]; (2) of Qs0e~ (48.5,50.2]; (2) of Qs0~ [1.4e+03,2.84e+03]; (1) of Qs~         13
## 14 (7.3,19.3]; (2) of Qs0e~ (48.5,50.2]; (2) of Qs0~ (2.84e+03,3.21e+03]; (2) of Q~         14
## 15 (7.3,19.3]; (2) of Qs0e~ (48.5,50.2]; (2) of Qs0~ (3.21e+03,5.49e+03]; (3) of Q~         15
## 16 (7.3,19.3]; (2) of Qs0e~ (50.2,58]; (3) of Qs0e1~ [1.4e+03,2.84e+03]; (1) of Qs~         16
## 17 (7.3,19.3]; (2) of Qs0e~ (50.2,58]; (3) of Qs0e1~ (2.84e+03,3.21e+03]; (2) of Q~         17
## 18 (7.3,19.3]; (2) of Qs0e~ (50.2,58]; (3) of Qs0e1~ (3.21e+03,5.49e+03]; (3) of Q~         18
```

2.4.2.5.6 Change values Based on Index

Index from 1 to 18, change input values based on index

```r
# arr.group.idx.subsidy <- arr.group.idx*2 - ((arr.group.idx)^2)*0.01
arr.group.idx.subsidy <- arr.group.idx*2
df.with.cut.quant.all %>%
        mutate(more_prot = prot + arr.group.idx.subsidy[!!sym(var.qjnt.grp.idx)]) %>%
        group_by(!!!syms(vars.quantile.cut.all), !!sym(var.qjnt.grp.idx))  %>%
        summarise_at(c('more_prot', 'prot'), funs(mean(., na.rm=TRUE)))
```

```
## # A tibble: 18 x 6
## # Groups:   wealthIdx_Qs0e1n2, hgt0_Qs0e1n3, wgt0_Qs0e1n3 [18]
##    wealthIdx_Qs0e1n2          hgt0_Qs0e1n3               wgt0_Qs0e1n3                     gro
##    <fct>                      <fct>                      <fct>
##  1 [1,7.3]; (1) of Qs0e1n2    [40.6,48.5]; (1) of Qs0e1n3 [1.4e+03,2.84e+03]; (1) of Qs0e1n3
##  2 [1,7.3]; (1) of Qs0e1n2    [40.6,48.5]; (1) of Qs0e1n3 (2.84e+03,3.21e+03]; (2) of Qs0e1n3
##  3 [1,7.3]; (1) of Qs0e1n2    [40.6,48.5]; (1) of Qs0e1n3 (3.21e+03,5.49e+03]; (3) of Qs0e1n3
##  4 [1,7.3]; (1) of Qs0e1n2    (48.5,50.2]; (2) of Qs0e1n3 [1.4e+03,2.84e+03]; (1) of Qs0e1n3
##  5 [1,7.3]; (1) of Qs0e1n2    (48.5,50.2]; (2) of Qs0e1n3 (2.84e+03,3.21e+03]; (2) of Qs0e1n3
##  6 [1,7.3]; (1) of Qs0e1n2    (48.5,50.2]; (2) of Qs0e1n3 (3.21e+03,5.49e+03]; (3) of Qs0e1n3
##  7 [1,7.3]; (1) of Qs0e1n2    (50.2,58]; (3) of Qs0e1n3   [1.4e+03,2.84e+03]; (1) of Qs0e1n3
##  8 [1,7.3]; (1) of Qs0e1n2    (50.2,58]; (3) of Qs0e1n3   (2.84e+03,3.21e+03]; (2) of Qs0e1n3
##  9 [1,7.3]; (1) of Qs0e1n2    (50.2,58]; (3) of Qs0e1n3   (3.21e+03,5.49e+03]; (3) of Qs0e1n3
## 10 (7.3,19.3]; (2) of Qs0e1n2 [40.6,48.5]; (1) of Qs0e1n3 [1.4e+03,2.84e+03]; (1) of Qs0e1n3
## 11 (7.3,19.3]; (2) of Qs0e1n2 [40.6,48.5]; (1) of Qs0e1n3 (2.84e+03,3.21e+03]; (2) of Qs0e1n3
## 12 (7.3,19.3]; (2) of Qs0e1n2 [40.6,48.5]; (1) of Qs0e1n3 (3.21e+03,5.49e+03]; (3) of Qs0e1n3
## 13 (7.3,19.3]; (2) of Qs0e1n2 (48.5,50.2]; (2) of Qs0e1n3 [1.4e+03,2.84e+03]; (1) of Qs0e1n3
## 14 (7.3,19.3]; (2) of Qs0e1n2 (48.5,50.2]; (2) of Qs0e1n3 (2.84e+03,3.21e+03]; (2) of Qs0e1n3
## 15 (7.3,19.3]; (2) of Qs0e1n2 (48.5,50.2]; (2) of Qs0e1n3 (3.21e+03,5.49e+03]; (3) of Qs0e1n3
## 16 (7.3,19.3]; (2) of Qs0e1n2 (50.2,58]; (3) of Qs0e1n3   [1.4e+03,2.84e+03]; (1) of Qs0e1n3
## 17 (7.3,19.3]; (2) of Qs0e1n2 (50.2,58]; (3) of Qs0e1n3   (2.84e+03,3.21e+03]; (2) of Qs0e1n3
## 18 (7.3,19.3]; (2) of Qs0e1n2 (50.2,58]; (3) of Qs0e1n3   (3.21e+03,5.49e+03]; (3) of Qs0e1n3
```

2.5 Summarize Multiple Variables

2.5.1 Generate Replace Variables

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

2.5.1.1 Replace NA for Multiple Variables

Replace some variables NA by some values, and other variables' NAs by other values.

```
# Define
it_N <- 3
it_M <- 5
svr_id <- 'date'

# NA dataframe
df_NA <- as_tibble(matrix(NA, nrow=it_N, ncol=it_M)) %>%
  rowid_to_column(var = svr_id) %>%
  rename_at(vars(starts_with("V")),
            funs(str_replace(., "V", "var")))
kable(df_NA) %>%
  kable_styling_fc()
```

date	var1	var2	var3	var4	var5
1	NA	NA	NA	NA	NA
2	NA	NA	NA	NA	NA
3	NA	NA	NA	NA	NA

```
# Replace NA
df_NA_replace <- df_NA %>%
  mutate_at(vars(one_of(c('var1', 'var2'))), list(~replace_na(., 0))) %>%
  mutate_at(vars(one_of(c('var3', 'var5'))), list(~replace_na(., 99)))
kable(df_NA_replace) %>%
  kable_styling_fc()
```

date	var1	var2	var3	var4	var5
1	0	0	99	NA	99
2	0	0	99	NA	99
3	0	0	99	NA	99

2.5.1.2 Cumulative Sum Multiple Variables

Each row is a different date, each column is the profit a firms earns on a date, we want to compute cumulatively how much a person is earning. Also renames variable names below jointly.

```
# Define
it_N <- 3
it_M <- 5
svr_id <- 'date'

# random dataframe, daily profit of firms
# dp_fx: daily profit firm ID something
set.seed(123)
df_daily_profit <- as_tibble(matrix(rnorm(it_N*it_M), nrow=it_N, ncol=it_M)) %>%
  rowid_to_column(var = svr_id) %>%
  rename_at(vars(starts_with("V")),
            funs(str_replace(., "V", "dp_f")))
kable(df_daily_profit) %>%
  kable_styling_fc_wide()
```

```
# cumulative sum with suffix
df_cumu_profit_suffix <- df_daily_profit %>%
  mutate_at(vars(contains('dp_f')), .funs = list(cumu = ~cumsum(.)))
kable(df_cumu_profit_suffix) %>%
  kable_styling_fc_wide()
```

date	dp_f1	dp_f2	dp_f3	dp_f4	dp_f5
1	-0.5604756	0.0705084	0.4609162	-0.4456620	0.4007715
2	-0.2301775	0.1292877	-1.2650612	1.2240818	0.1106827
3	1.5587083	1.7150650	-0.6868529	0.3598138	-0.5558411

date	dp_f1	dp_f2	dp_f3	dp_f4	dp_f5	dp_f1_cumu	dp_f2_cumu	dp_f3_cumu	dp_f4_cumu	dp_f5_cumu
1	-0.5604756	0.0705084	0.4609162	-0.4456620	0.4007715	-0.5604756	0.0705084	0.4609162	-0.4456620	0.4007715
2	-0.2301775	0.1292877	-1.2650612	1.2240818	0.1106827	-0.7906531	0.1997961	-0.8041450	0.7784198	0.5114542
3	1.5587083	1.7150650	-0.6868529	0.3598138	-0.5558411	0.7680552	1.9148611	-1.4909979	1.1382337	-0.0443870

```r
# cumulative sum variables naming to prefix
df_cumu_profit <- df_cumu_profit_suffix %>%
  rename_at(vars(contains( "_cumu") ), list(~paste("cp_f", gsub("_cumu", "", .), sep = ""))) %>%
  rename_at(vars(contains( "cp_f") ), list(~gsub("dp_f", "", .)))
kable(df_cumu_profit) %>%
  kable_styling_fc_wide()
```

date	dp_f1	dp_f2	dp_f3	dp_f4	dp_f5	cp_f1	cp_f2	cp_f3	cp_f4	cp_f5
1	-0.5604756	0.0705084	0.4609162	-0.4456620	0.4007715	-0.5604756	0.0705084	0.4609162	-0.4456620	0.4007715
2	-0.2301775	0.1292877	-1.2650612	1.2240818	0.1106827	-0.7906531	0.1997961	-0.8041450	0.7784198	0.5114542
3	1.5587083	1.7150650	-0.6868529	0.3598138	-0.5558411	0.7680552	1.9148611	-1.4909979	1.1382337	-0.0443870

Chapter 3

Functions

3.1 Dataframe Mutate

3.1.1 Row Input Functions

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

We want evaluate nonlinear function f(Q_i, y_i, ar_x, ar_y, c, d), where c and d are constants, and ar_x and ar_y are arrays, both fixed. x_i and y_i vary over each row of matrix. We would like to evaluate this nonlinear function concurrently across N individuals. The eventual goal is to find the i specific Q that solves the nonlinear equations.

This is a continuation of R use Apply, Sapply and dplyr Mutate to Evaluate one Function Across Rows of a Matrix

3.1.1.1 Set up Input Arrays

There is a function that takes $M = Q + P$ inputs, we want to evaluate this function N times. Each time, there are M inputs, where all but Q of the M inputs, meaning P of the M inputs, are the same. In particular, $P = Q * N$.

$$M = Q + P = Q + Q * N$$

```
# it_child_count = N, the number of children
it_N_child_cnt = 5
# it_heter_param = Q, number of parameters that are heterogeneous across children
it_Q_hetpa_cnt = 2

# P fixed parameters, nN is N dimensional, nP is P dimensional
ar_nN_A = seq(-2, 2, length.out = it_N_child_cnt)
ar_nN_alpha = seq(0.1, 0.9, length.out = it_N_child_cnt)
ar_nP_A_alpha = c(ar_nN_A, ar_nN_alpha)
ar_nN_N_choice = seq(1,it_N_child_cnt)/sum(seq(1,it_N_child_cnt))

# N by Q varying parameters
mt_nN_by_nQ_A_alpha = cbind(ar_nN_A, ar_nN_alpha, ar_nN_N_choice)
# Show
kable(mt_nN_by_nQ_A_alpha) %>%
  kable_styling_fc()
```

3.1.1.2 Testing Function

Test non-linear Equation.

ar_nN_A	ar_nN_alpha	ar_nN_N_choice
-2	0.1	0.0666667
-1	0.3	0.1333333
0	0.5	0.2000000
1	0.7	0.2666667
2	0.9	0.3333333

```r
# Test Parameters
fl_N_agg = 100
fl_rho = -1
fl_N_q = ar_nN_N_choice[4]*fl_N_agg
ar_A_alpha = mt_nN_by_nQ_A_alpha[4,]
# Apply Function
ar_p1_s1 = exp((ar_A_alpha[1] - ar_nN_A)*fl_rho)
ar_p1_s2 = (ar_A_alpha[2]/ar_nN_alpha)
ar_p1_s3 = (1/(ar_nN_alpha*fl_rho - 1))
ar_p1 = (ar_p1_s1*ar_p1_s2)^ar_p1_s3
ar_p2 = fl_N_q^((ar_A_alpha[2]*fl_rho-1)/(ar_nN_alpha*fl_rho-1))
ar_overall = ar_p1*ar_p2
fl_overall = fl_N_agg - sum(ar_overall)
print(fl_overall)
```

```
## [1] -598.2559
```

Implement the non-linear problem's evaluation using apply over all N individuals.

```r
# Define Implicit Function
ffi_nonlin_dplyrdo <- function(fl_A, fl_alpha, fl_N, ar_A, ar_alpha, fl_N_agg, fl_rho){
  # ar_A_alpha[1] is A
  # ar_A_alpha[2] is alpha

  # # Test Parameters
  # fl_N = 100
  # fl_rho = -1
  # fl_N_q = 10

  # Apply Function
  ar_p1_s1 = exp((fl_A - ar_A)*fl_rho)
  ar_p1_s2 = (fl_alpha/ar_alpha)
  ar_p1_s3 = (1/(ar_alpha*fl_rho - 1))
  ar_p1 = (ar_p1_s1*ar_p1_s2)^ar_p1_s3
  ar_p2 = fl_N^((fl_alpha*fl_rho-1)/(ar_alpha*fl_rho-1))
  ar_overall = ar_p1*ar_p2
  fl_overall = fl_N_agg - sum(ar_overall)

  return(fl_overall)
}

# Parameters
fl_rho = -1

# Evaluate Function
print(ffi_nonlin_dplyrdo(mt_nN_by_nQ_A_alpha[1,1],
                         mt_nN_by_nQ_A_alpha[1,2],
                         mt_nN_by_nQ_A_alpha[1,3]*fl_N_agg,
                         ar_nN_A, ar_nN_alpha, fl_N_agg, fl_rho))
```

```
## [1] 81.86645
```

```
for (i in seq(1,dim(mt_nN_by_nQ_A_alpha)[1])){
  fl_eval = ffi_nonlin_dplyrdo(mt_nN_by_nQ_A_alpha[i,1],
                               mt_nN_by_nQ_A_alpha[i,2],
                               mt_nN_by_nQ_A_alpha[i,3]*fl_N_agg,
                               ar_nN_A, ar_nN_alpha, fl_N_agg, fl_rho)
  print(fl_eval)
}
```

```
## [1] 81.86645
## [1] 54.48885
## [1] -65.5619
## [1] -598.2559
## [1] -3154.072
```

3.1.1.3 Evaluate Nonlinear Function using dplyr mutate

```
# Convert Matrix to Tibble
ar_st_col_names = c('fl_A', 'fl_alpha', 'fl_N')
tb_nN_by_nQ_A_alpha <- as_tibble(mt_nN_by_nQ_A_alpha) %>% rename_all(~c(ar_st_col_names))

# Define Implicit Function
ffi_nonlin_dplyrdo <- function(fl_A, fl_alpha, fl_N, ar_A, ar_alpha, fl_N_agg, fl_rho){

  # Test Parameters
  # ar_A = ar_nN_A
  # ar_alpha = ar_nN_alpha
  # fl_N = 100
  # fl_rho = -1
  # fl_N_q = 10

  # Apply Function
  ar_p1_s1 = exp((fl_A - ar_A)*fl_rho)
  ar_p1_s2 = (fl_alpha/ar_alpha)
  ar_p1_s3 = (1/(ar_alpha*fl_rho - 1))
  ar_p1 = (ar_p1_s1*ar_p1_s2)^ar_p1_s3
  ar_p2 = (fl_N*fl_N_agg)^((fl_alpha*fl_rho-1)/(ar_alpha*fl_rho-1))
  ar_overall = ar_p1*ar_p2
  fl_overall = fl_N_agg - sum(ar_overall)

  return(fl_overall)
}

# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
tb_nN_by_nQ_A_alpha = tb_nN_by_nQ_A_alpha %>% rowwise() %>%
                        mutate(dplyr_eval = ffi_nonlin_dplyrdo(fl_A, fl_alpha, fl_N,
                                                               ar_nN_A, ar_nN_alpha,
                                                               fl_N_agg, fl_rho))
# Show
kable(tb_nN_by_nQ_A_alpha) %>%
  kable_styling_fc()
```

fl_A	fl_alpha	fl_N	dplyr_eval
-2	0.1	0.0666667	81.86645
-1	0.3	0.1333333	54.48885
0	0.5	0.2000000	-65.56190
1	0.7	0.2666667	-598.25595
2	0.9	0.3333333	-3154.07226

3.1.2 Evaluate Choices Across States

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

See the ff_opti_bisect_pmap_multi function from Fan's *REconTools* Package, which provides a resuable function based on the algorithm worked out here.

We want evaluate linear function $0 = f(z_{ij}, x_i, y_i, \mathbf{X}, \mathbf{Y}, c, d)$. There are i functions that have i specific x and y. For each i function, we evaluate along a grid of feasible values for z, over $j \in J$ grid points, potentially looking for the j that is closest to the root. $\mathbf{X}$ and $\mathbf{Y}$ are arrays common across the i equations, and c and d are constants.

The evaluation strategy is the following, given min and max for z that are specific for each j, and given common number of grid points, generate a matrix of z_{ij}. Suppose there the number of i is I, and the number of grid points for j is J.

1. Generate a $J \cdot I$ by 3 matrix where the columns are z, x, y as tibble
2. Follow this Mutate to evaluate the $f(\cdot)$ function.
3. Add two categorical columns for grid levels and wich i, i and j index. Plot Mutate output evaluated column categorized by i as color and j as x-axis.

3.1.2.1 Set up Input Arrays

There is a function that takes $M = Q + P$ inputs, we want to evaluate this function N times. Each time, there are M inputs, where all but Q of the M inputs, meaning P of the M inputs, are the same. In particular, $P = Q * N$.

$$M = Q + P = Q + Q * N$$

Now we need to expand this by the number of choice grid. Each row, representing one equation, is expanded by the number of choice grids. We are graphically searching, or rather brute force searching, which means if we have 100 individuals, we want to plot out the nonlinear equation for each of these lines, and show graphically where each line crosses zero. We achieve this, by evaluating the equation for each of the 100 individuals along a grid of feasible choices.

In this problem here, the feasible choices are shared across individuals.

```
# Parameters
fl_rho = 0.20
svr_id_var = 'INDI_ID'

# it_child_count = N, the number of children
it_N_child_cnt = 4
# it_heter_param = Q, number of parameters that are heterogeneous across children
it_Q_hetpa_cnt = 2

# P fixed parameters, nN is N dimensional, nP is P dimensional
ar_nN_A = seq(-2, 2, length.out = it_N_child_cnt)
ar_nN_alpha = seq(0.1, 0.9, length.out = it_N_child_cnt)
ar_nP_A_alpha = c(ar_nN_A, ar_nN_alpha)

# N by Q varying parameters
mt_nN_by_nQ_A_alpha = cbind(ar_nN_A, ar_nN_alpha)

# Choice Grid for nutritional feasible choices for each
fl_N_agg = 100
fl_N_min = 0
it_N_choice_cnt_ttest = 3
it_N_choice_cnt_dense = 100
ar_N_choices_ttest = seq(fl_N_min, fl_N_agg, length.out = it_N_choice_cnt_ttest)
```

```r
ar_N_choices_dense = seq(fl_N_min, fl_N_agg, length.out = it_N_choice_cnt_dense)

# Mesh Expand
tb_states_choices <- as_tibble(mt_nN_by_nQ_A_alpha) %>% rowid_to_column(var=svr_id_var)
tb_states_choices_ttest <- tb_states_choices %>% expand_grid(choices = ar_N_choices_ttest)
tb_states_choices_dense <- tb_states_choices %>% expand_grid(choices = ar_N_choices_dense)

# display
summary(tb_states_choices_dense)
```

```
##     INDI_ID        ar_nN_A     ar_nN_alpha     choices
##  Min.   :1.00   Min.   :-2   Min.   :0.1   Min.   :  0
##  1st Qu.:1.75   1st Qu.:-1   1st Qu.:0.3   1st Qu.: 25
##  Median :2.50   Median : 0   Median :0.5   Median : 50
##  Mean   :2.50   Mean   : 0   Mean   :0.5   Mean   : 50
##  3rd Qu.:3.25   3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.: 75
##  Max.   :4.00   Max.   : 2   Max.   :0.9   Max.   :100
```

```r
kable(tb_states_choices_ttest) %>%
  kable_styling_fc()
```

INDI_ID	ar_nN_A	ar_nN_alpha	choices
1	-2.0000000	0.1000000	0
1	-2.0000000	0.1000000	50
1	-2.0000000	0.1000000	100
2	-0.6666667	0.3666667	0
2	-0.6666667	0.3666667	50
2	-0.6666667	0.3666667	100
3	0.6666667	0.6333333	0
3	0.6666667	0.6333333	50
3	0.6666667	0.6333333	100
4	2.0000000	0.9000000	0
4	2.0000000	0.9000000	50
4	2.0000000	0.9000000	100

3.1.2.2 Apply Same Function all Rows, Some Inputs Row-specific, other Shared

There are two types of inputs, row-specific inputs, and inputs that should be applied for each row. The Function just requires all of these inputs, it does not know what is row-specific and what is common for all row. Dplyr recognizes which parameter inputs already existing in the piped dataframe/tibble, given rowwise, those will be row-specific inputs. Additional function parameters that do not exist in dataframe as variable names, but that are pre-defined scalars or arrays will be applied to all rows.

- **?** string variable name of input where functions are evaluated, these are already contained in the dataframe, existing variable names, row specific, rowwise computation over these, each rowwise calculation using different rows: *fl_A*, *fl_alpha*, *fl_N*
- **?** scalar and array values that are applied to every rowwise calculation, all rowwise calculations using the same scalars and arrays:*ar_A*, *ar_alpha*, *fl_N_agg*, *fl_rho*
- **?** string output variable name

The function looks within group, finds min/max etc that are relevant.

```r
# Convert Matrix to Tibble
ar_st_col_names = c(svr_id_var,'fl_A', 'fl_alpha')
tb_states_choices <- tb_states_choices %>% rename_all(~c(ar_st_col_names))
ar_st_col_names = c(svr_id_var,'fl_A', 'fl_alpha', 'fl_N')
tb_states_choices_ttest <- tb_states_choices_ttest %>% rename_all(~c(ar_st_col_names))
```

```
tb_states_choices_dense <- tb_states_choices_dense %>% rename_all(~c(ar_st_col_names))

# Define Implicit Function
ffi_nonlin_dplyrdo <- function(fl_A, fl_alpha, fl_N, ar_A, ar_alpha, fl_N_agg, fl_rho){
  # scalar value that are row-specific, in dataframe already: *fl_A*, *fl_alpha*, *fl_N*
  # array and scalars not in dataframe, common all rows: *ar_A*, *ar_alpha*, *fl_N_agg*, *fl_rho*

  # Test Parameters
  # ar_A = ar_nN_A
  # ar_alpha = ar_nN_alpha
  # fl_N = 100
  # fl_rho = -1
  # fl_N_q = 10

  # Apply Function
  ar_p1_s1 = exp((fl_A - ar_A)*fl_rho)
  ar_p1_s2 = (fl_alpha/ar_alpha)
  ar_p1_s3 = (1/(ar_alpha*fl_rho - 1))
  ar_p1 = (ar_p1_s1*ar_p1_s2)^ar_p1_s3
  ar_p2 = fl_N^((fl_alpha*fl_rho-1)/(ar_alpha*fl_rho-1))
  ar_overall = ar_p1*ar_p2
  fl_overall = fl_N_agg - sum(ar_overall)

  return(fl_overall)
}
```

3.1.2.2.1 3 Points and Denser Dataframs and Define Function

3.1.2.2.2 Evaluate at Three Choice Points and Show Table

In the example below, just show results evaluating over three choice points and show table.

```
# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
tb_states_choices_ttest_eval = tb_states_choices_ttest %>% rowwise() %>%
                               mutate(dplyr_eval = ffi_nonlin_dplyrdo(fl_A, fl_alpha, fl_N,
                                                                      ar_nN_A, ar_nN_alpha,
                                                                      fl_N_agg, fl_rho))
# Show
kable(tb_states_choices_ttest_eval) %>%
  kable_styling_fc()
```

INDI_ID	fl_A	fl_alpha	fl_N	dplyr_eval
1	-2.0000000	0.1000000	0	100.00000
1	-2.0000000	0.1000000	50	-5666.95576
1	-2.0000000	0.1000000	100	-12880.28392
2	-0.6666667	0.3666667	0	100.00000
2	-0.6666667	0.3666667	50	-595.73454
2	-0.6666667	0.3666667	100	-1394.70698
3	0.6666667	0.6333333	0	100.00000
3	0.6666667	0.6333333	50	-106.51058
3	0.6666667	0.6333333	100	-323.94216
4	2.0000000	0.9000000	0	100.00000
4	2.0000000	0.9000000	50	22.55577
4	2.0000000	0.9000000	100	-51.97161

3.1.2.2.3 Evaluate at Many Choice Points and Show Graphically

Same as above, but now we evaluate the function over the individuals at many choice points so that we can graph things out.

```
# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
tb_states_choices_dense_eval = tb_states_choices_dense %>% rowwise() %>%
                        mutate(dplyr_eval = ffi_nonlin_dplyrdo(fl_A, fl_alpha, fl_N,
                                                               ar_nN_A, ar_nN_alpha,
                                                               fl_N_agg, fl_rho))
```

```
# Show
dim(tb_states_choices_dense_eval)
```

```
## [1] 400   5
```

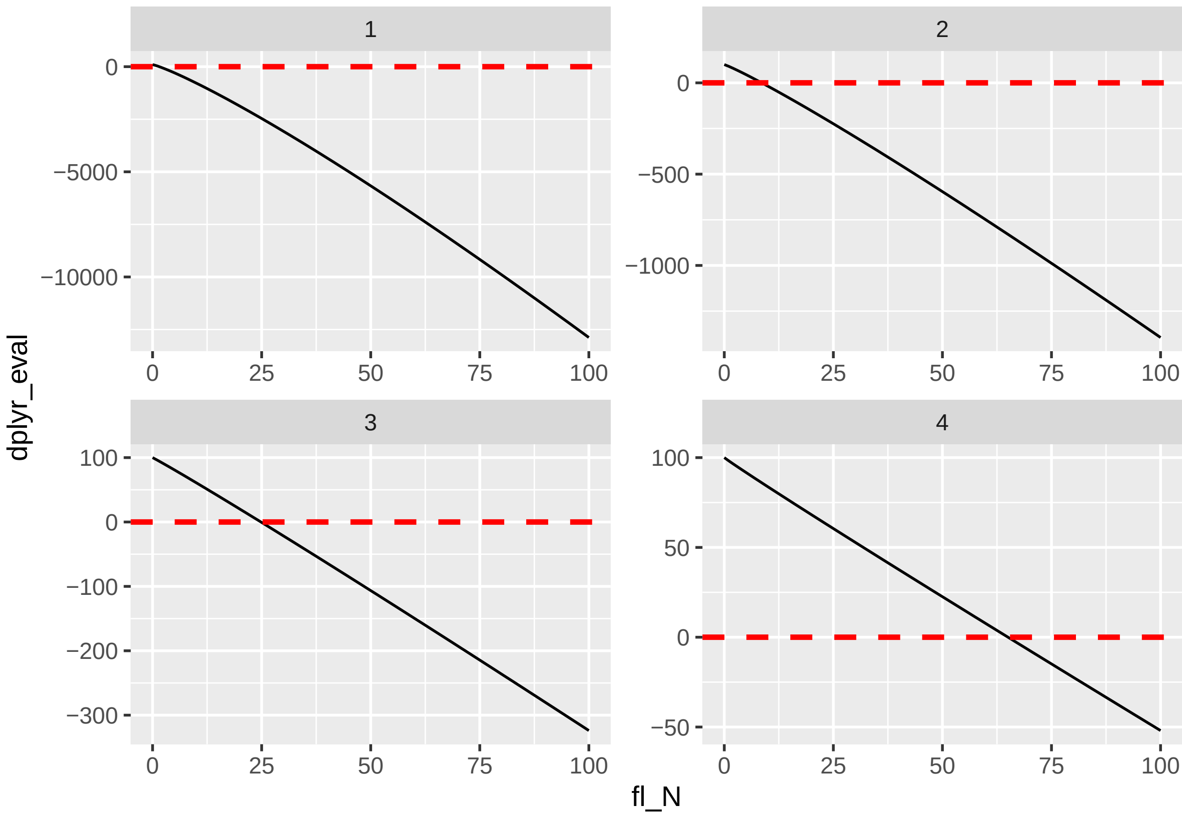
```
summary(tb_states_choices_dense_eval)
```

```
##     INDI_ID          fl_A       fl_alpha         fl_N        dplyr_eval
##  Min.   :1.00   Min.   :-2   Min.   :0.1   Min.   :  0   Min.   :-12880.28
##  1st Qu.:1.75   1st Qu.:-1   1st Qu.:0.3   1st Qu.: 25   1st Qu.: -1167.29
##  Median :2.50   Median : 0   Median :0.5   Median : 50   Median :  -202.42
##  Mean   :2.50   Mean   : 0   Mean   :0.5   Mean   : 50   Mean   : -1645.65
##  3rd Qu.:3.25   3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.: 75   3rd Qu.:     0.96
##  Max.   :4.00   Max.   : 2   Max.   :0.9   Max.   :100   Max.   :   100.00
```

```
lineplot <- tb_states_choices_dense_eval %>%
    ggplot(aes(x=fl_N, y=dplyr_eval)) +
        geom_line() +
        facet_wrap( . ~ INDI_ID, scales = "free") +
        geom_hline(yintercept=0, linetype="dashed",
               color = "red", size=1)
        labs(title = 'Evaluate Non-Linear Functions to Search for Roots',
             x = 'X values',
             y = 'f(x)',
             caption = 'Evaluating the Function')
```

```
## $x
## [1] "X values"
##
## $y
## [1] "f(x)"
##
## $title
## [1] "Evaluate Non-Linear Functions to Search for Roots"
##
## $caption
## [1] "Evaluating the Function"
##
## attr(,"class")
## [1] "labels"
```

```
print(lineplot)
```



3.2 Dataframe Do Anything

3.2.1 MxQ to MxP Rows

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

3.2.1.1 MxQ to Mx1 Rows: Within Group Gini

There is a Panel with M individuals and each individual has Q records/rows. A function generate an individual specific outcome given the Q individual specific inputs, along with shared parameters and arrays across the M individuals.

For example, suppose we have a dataframe of individual wage information from different countries, each row is an individual from one country. We want to generate country specific gini based on the individual data for each country in the dataframe. But additionally, perhaps the gini formula requires not just individual income but some additional parameters or shared dataframes as inputs.

Given the within m income observations, we can compute gini statistics that are individual specific based on the observed distribution of incomes. For this, we will use the ff_dist_gini_vector_pos.html function from REconTools.

To make this more interesting, we will generate large dataframe with more M and more Q each m.

3.2.1.1.1 Large Dataframe There are up to ten thousand income observation per person. And there are ten people.

```
# Parameter Setups
it_M <- 10
it_Q_max <- 10000
fl_rnorm_mu <- 1
ar_rnorm_sd <- seq(0.01, 0.2, length.out=it_M)
ar_it_q <- sample.int(it_Q_max, it_M, replace=TRUE)
```

```r
# N by Q varying parameters
mt_data = cbind(ar_it_q, ar_rnorm_sd)
tb_M <- as_tibble(mt_data) %>% rowid_to_column(var = "ID") %>%
                rename(sd = ar_rnorm_sd, Q = ar_it_q) %>%
                mutate(mean = fl_rnorm_mu)
```

3.2.1.1.2 Compute Group specific gini, NORMAL

There is only one input for the gini function *ar_pos*. Note that the gini are not very large even with large SD, because these are normal distributions. By Construction, most peple are in the middle. So with almost zero standard deviation, we have perfect equality, as standard deviation increases, inequality increases, but still pretty equal overall, there is no fat upper tail.

Note that there are three ways of referring to variable names with dot, which are all shown below:

1. We can explicitly refer to names
2. We can use the dollar dot structure to use string variable names in do anything.
3. We can use dot bracket, this is the only option that works with string variable names

```r
# A. Normal Draw Expansion, Explicitly Name
set.seed('123')
tb_income_norm_dot_dollar <- tb_M %>% group_by(ID) %>%
  do(income = rnorm(.$Q,
                    mean=.$mean,
                    sd=.$sd)) %>%
  unnest(c(income)) %>%
  left_join(tb_M, by="ID")

# Normal Draw Expansion again, dot dollar differently with string variable name
set.seed('123')
tb_income_norm_dollar_dot <- tb_M %>% group_by(ID) %>%
  do(income = rnorm(`$`(., 'Q'),
                    mean = `$`(., 'mean'),
                    sd = `$`(., 'sd'))) %>%
  unnest(c(income)) %>%
  left_join(tb_M, by="ID")

# Normal Draw Expansion again, dot double bracket
set.seed('123')
svr_mean <- 'mean'
svr_sd <- 'sd'
svr_Q <- 'Q'
tb_income_norm_dot_bracket_db <- tb_M %>% group_by(ID) %>%
  do(income = rnorm(.[[svr_Q]],
                    mean = .[[svr_mean]],
                    sd = .[[svr_sd]])) %>%
  unnest(c(income)) %>%
  left_join(tb_M, by="ID")

# display
sum(sum(tb_income_norm_dollar_dot - tb_income_norm_dot_dollar - tb_income_norm_dot_bracket_db))
```

```
## [1] -463785175
```

```r
# display
head(tb_income_norm_dot_dollar, 20)
```

```
## # A tibble: 20 x 5
##        ID income     Q    sd  mean
##     <int>  <dbl> <dbl> <dbl> <dbl>
```

```
## 1     1  0.994  9982  0.01      1
## 2     1  0.998  9982  0.01      1
## 3     1  1.02   9982  0.01      1
## 4     1  1.00   9982  0.01      1
## 5     1  1.00   9982  0.01      1
## 6     1  1.02   9982  0.01      1
## 7     1  1.00   9982  0.01      1
## 8     1  0.987  9982  0.01      1
## 9     1  0.993  9982  0.01      1
## 10    1  0.996  9982  0.01      1
## 11    1  1.01   9982  0.01      1
## 12    1  1.00   9982  0.01      1
## 13    1  1.00   9982  0.01      1
## 14    1  1.00   9982  0.01      1
## 15    1  0.994  9982  0.01      1
## 16    1  1.02   9982  0.01      1
## 17    1  1.00   9982  0.01      1
## 18    1  0.980  9982  0.01      1
## 19    1  1.01   9982  0.01      1
## 20    1  0.995  9982  0.01      1
```

```
# Gini by Group
tb_gini_norm <- tb_income_norm_dollar_dot %>% group_by(ID) %>%
  do(inc_gini_norm = ff_dist_gini_vector_pos(.$income)) %>%
  unnest(c(inc_gini_norm)) %>%
  left_join(tb_M, by="ID")
```

```
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
## see REconTools for formula: DIST GINI--Compute Gini Inequality Coefficient Given Data Vector (One
```

```
# display
kable(tb_gini_norm) %>%
  kable_styling_fc()
```

ID	inc_gini_norm	Q	sd	mean
1	0.0056337	9982	0.0100000	1
2	0.0175280	2980	0.0311111	1
3	0.0293986	1614	0.0522222	1
4	0.0422304	555	0.0733333	1
5	0.0535146	4469	0.0944444	1
6	0.0653938	9359	0.1155556	1
7	0.0769135	7789	0.1366667	1
8	0.0894165	9991	0.1577778	1
9	0.1010982	9097	0.1788889	1
10	0.1124019	1047	0.2000000	1

3.2.2 Mx1 to MxQ Rows

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

Case One: There is a dataframe with M rows, based on these m specific information, generate dataframes for each m. Stack these indivdiual dataframes together and merge original m specific information in as well. The number of rows for each m is Q_m, each m could have different number of expansion rows.

Generate a panel with M individuals, each individual is observed for different spans of times (*uncount*). Before expanding, generate individual specific normal distribution standard deviation. All individuals share the same mean, but have increasing standard deviations.

3.2.2.1 Generate Dataframe with M Rows.

This is the first step, generate M rows of data, to be expanded. Each row contains the number of normal draws to make and the mean and the standard deviation for normal daraws that are m specific.

```
# Parameter Setups
it_M <- 3
it_Q_max <- 5
fl_rnorm_mu <- 1000
ar_rnorm_sd <- seq(0.01, 200, length.out=it_M)
ar_it_q <- sample.int(it_Q_max, it_M, replace=TRUE)

# N by Q varying parameters
mt_data = cbind(ar_it_q, ar_rnorm_sd)
tb_M <- as_tibble(mt_data) %>% rowid_to_column(var = "ID") %>%
                rename(sd = ar_rnorm_sd, Q = ar_it_q) %>%
                mutate(mean = fl_rnorm_mu)

# display
kable(tb_M) %>%
  kable_styling_fc()
```

ID	Q	sd	mean
1	3	0.010	1000
2	3	100.005	1000
3	1	200.000	1000

3.2.2.2 Random Normal Draw Expansion

The steps are:

1. do anything
2. use ".$" sign to refer to variable names, or [['name']]
3. unnest
4. left_join expanded and original

Note these all give the same results

Use dot dollar to get variables

```
# Generate $Q_m$ individual specific incomes, expanded different number of times for each m
tb_income <- tb_M %>% group_by(ID) %>%
  do(income = rnorm(.$Q, mean=.$mean, sd=.$sd)) %>%
  unnest(c(income))

# Merge back with tb_M
tb_income_full_dd <- tb_income %>%
  left_join(tb_M)
```

```
## Joining, by = "ID"
```

```r
# display
kable(tb_income) %>%
  kable_styling_fc()
```

ID	income
1	1000.0183
1	999.9943
1	999.9822
2	1033.7465
2	1093.1374
2	862.1896
3	988.7742

```r
kable(tb_income_full_dd) %>%
  kable_styling_fc()
```

ID	income	Q	sd	mean
1	1000.0183	3	0.010	1000
1	999.9943	3	0.010	1000
1	999.9822	3	0.010	1000
2	1033.7465	3	100.005	1000
2	1093.1374	3	100.005	1000
2	862.1896	3	100.005	1000
3	988.7742	1	200.000	1000

3.3 Apply and pmap

3.3.1 Apply, Sapply, Mutate

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

- r apply matrix to function row by row
- r evaluate function on grid
- Apply a function to every row of a matrix or a data frame
- r apply
- r sapply
- sapply over matrix row by row
- apply dplyr vectorize
- function as parameters using formulas
- do

We want evaluate linear function f(x_i, y_i, ar_x, ar_y, c, d), where c and d are constants, and ar_x and ar_y are arrays, both fixed. x_i and y_i vary over each row of matrix. More specifically, we have a functions, this function takes inputs that are individual specific. We would like to evaluate this function concurrently across N individuals.

The function is such that across the N individuals, some of the function parameter inputs are the same, but others are different. If we are looking at demand for a particular product, the prices of all products enter the demand equation for each product, but the product's own price enters also in a different way.

The objective is either to just evaluate this function across N individuals, or this is a part of a nonlinear solution system.

What is the relationship between apply, lapply and vectorization? see Is the "*apply" family really not vectorized?.

3.3.1.1 Set up Input Arrays

There is a function that takes $M = Q + P$ inputs, we want to evaluate this function N times. Each time, there are M inputs, where all but Q of the M inputs, meaning P of the M inputs, are the same. In particular, $P = Q * N$.

$$M = Q + P = Q + Q * N$$

```
# it_child_count = N, the number of children
it_N_child_cnt = 5
# it_heter_param = Q, number of parameters that are heterogeneous across children
it_Q_hetpa_cnt = 2

# P fixed parameters, nN is N dimensional, nP is P dimensional
ar_nN_A = seq(-2, 2, length.out = it_N_child_cnt)
ar_nN_alpha = seq(0.1, 0.9, length.out = it_N_child_cnt)
ar_nP_A_alpha = c(ar_nN_A, ar_nN_alpha)

# N by Q varying parameters
mt_nN_by_nQ_A_alpha = cbind(ar_nN_A, ar_nN_alpha)

# display
kable(mt_nN_by_nQ_A_alpha) %>%
  kable_styling_fc()
```

ar_nN_A	ar_nN_alpha
-2	0.1
-1	0.3
0	0.5
1	0.7
2	0.9

3.3.1.2 Using apply

3.3.1.2.1 Apply with Named Function First we use the apply function, we have to hard-code the arrays that are fixed for each of the N individuals. Then apply allows us to loop over the matrix that is N by Q, each row one at a time, from 1 to N.

```
# Define Implicit Function
ffi_linear_hardcode <- function(ar_A_alpha){
  # ar_A_alpha[1] is A
  # ar_A_alpha[2] is alpha

  fl_out = sum(ar_A_alpha[1]*ar_nN_A + 1/(ar_A_alpha[2] + 1/ar_nN_alpha))

  return(fl_out)
}

# Evaluate function row by row
ar_func_apply = apply(mt_nN_by_nQ_A_alpha, 1, ffi_linear_hardcode)
```

3.3.1.2.2 Apply using Anonymous Function

- apply over matrix

Apply with anonymous function generating a list of arrays of different lengths. In the example below, we want to drawn N sets of random uniform numbers, but for each set the number of draws we want to have is Q_i. Furthermore, we want to rescale the random uniform draws so that they all become proportions that sum u pto one for each i, but then we multply each row's values by the row specific aggregates.

The anonymous function has hard coded parameters. Using an anonymous function here allows for parameters to be provided inside the function that are shared across each looped evaluation. This is perhaps more convenient than sapply with additional parameters.

```
set.seed(1039)

# Define the number of draws each row and total amount
it_N <- 4
fl_unif_min <- 1
fl_unif_max <- 2
mt_draw_define <- cbind(seq(it_N),runif(it_N, min=1,max=10))
print(mt_draw_define)
```

```
##      [,1]     [,2]
## [1,]    1 2.131008
## [2,]    2 7.016820
## [3,]    3 4.774441
## [4,]    4 5.023006
```

```
# apply row by row, anonymous function has hard coded min and max
ls_ar_draws_shares_lvls = apply(cbind(seq(it_N),runif(it_N, min=1,max=10)),
                                1,
                                function(row, min, max) {
                                 it_draw <- row[1]
                                 fl_sum <- row[2]
                                 ar_unif <- runif(it_draw,
                                                  min=fl_unif_min,
                                                  max=fl_unif_max)
                                 ar_share <- ar_unif/sum(ar_unif)
                                 ar_levels <- ar_share*fl_sum
                                 return(list(ar_share=ar_share,
                                             ar_levels=ar_levels))
                                })

# Show Results
print(ls_ar_draws_shares_lvls)
```

```
## [[1]]
## [[1]]$ar_share
## [1] 1
##
## [[1]]$ar_levels
## [1] 5.361378
##
##
## [[2]]
## [[2]]$ar_share
## [1] 0.4428811 0.5571189
##
## [[2]]$ar_levels
## [1] 3.388957 4.263112
##
##
## [[3]]
## [[3]]$ar_share
## [1] 0.4233740 0.2913644 0.2852616
##
## [[3]]$ar_levels
## [1] 4.052625 2.789002 2.730584
##
```

```
##
## [[4]]
## [[4]]$ar_share
## [1] 0.3082076 0.2913433 0.2012986 0.1991505
##
## [[4]]$ar_levels
## [1] 2.965381 2.803123 1.936769 1.916102
```

3.3.1.3 Using sapply

3.3.1.3.1 sapply with named function

- r convert matrix to list
- Convert a matrix to a list of vectors in R

Sapply allows us to not have tohard code in the A and alpha arrays. But Sapply works over List or Vector, not Matrix. So we have to convert the N by Q matrix to a N element list Now update the function with sapply.

```
ls_ar_nN_by_nQ_A_alpha = as.list(data.frame(t(mt_nN_by_nQ_A_alpha)))

# Define Implicit Function
ffi_linear_sapply <- function(ar_A_alpha, ar_A, ar_alpha){
  # ar_A_alpha[1] is A
  # ar_A_alpha[2] is alpha

  fl_out = sum(ar_A_alpha[1]*ar_nN_A + 1/(ar_A_alpha[2] + 1/ar_nN_alpha))

  return(fl_out)
}

# Evaluate function row by row
ar_func_sapply = sapply(ls_ar_nN_by_nQ_A_alpha, ffi_linear_sapply,
                        ar_A=ar_nN_A, ar_alpha=ar_nN_alpha)
```

3.3.1.3.2 sapply using anonymous function

- sapply anonymous function
- r anoymous function multiple lines

Sapply with anonymous function generating a list of arrays of different lengths. In the example below, we want to drawn N sets of random uniform numbers, but for each set the number of draws we want to have is Q_i. Furthermore, we want to rescale the random uniform draws so that they all become proportions that sum u pto one for each i.

```
it_N <- 4
fl_unif_min <- 1
fl_unif_max <- 2

# Generate using runif without anonymous function
set.seed(1039)
ls_ar_draws = sapply(seq(it_N),
                     runif,
                     min=fl_unif_min, max=fl_unif_max)
print(ls_ar_draws)
```

```
## [[1]]
## [1] 1.125668
##
## [[2]]
## [1] 1.668536 1.419382
```

```
##
## [[3]]
## [1] 1.447001 1.484598 1.739119
##
## [[4]]
## [1] 1.952468 1.957931 1.926995 1.539678
```

```
# Generate Using Anonymous Function
set.seed(1039)
ls_ar_draws_shares = sapply(seq(it_N),
                            function(n, min, max) {
                              ar_unif <- runif(n,min,max)
                              ar_share <- ar_unif/sum(ar_unif)
                              return(ar_share)
                            },
                            min=fl_unif_min, max=fl_unif_max)
# Print Share
print(ls_ar_draws_shares)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 0.5403432 0.4596568
##
## [[3]]
## [1] 0.3098027 0.3178522 0.3723451
##
## [[4]]
## [1] 0.2646671 0.2654076 0.2612141 0.2087113
```

```
# Sapply with anonymous function to check sums
sapply(seq(it_N), function(x) {sum(ls_ar_draws[[x]])})
```

```
## [1] 1.125668 3.087918 4.670717 7.377071
```

```
sapply(seq(it_N), function(x) {sum(ls_ar_draws_shares[[x]])})
```

```
## [1] 1 1 1 1
```

3.3.1.4 Using dplyr mutate rowwise

- dplyr mutate own function
- dplyr all row function
- dplyr do function
- apply function each row dplyr
- applying a function to every row of a table using dplyr
- dplyr rowwise

```
# Convert Matrix to Tibble
ar_st_col_names = c('fl_A', 'fl_alpha')
tb_nN_by_nQ_A_alpha <- as_tibble(mt_nN_by_nQ_A_alpha) %>% rename_all(~c(ar_st_col_names))
# Show
kable(tb_nN_by_nQ_A_alpha) %>%
  kable_styling_fc()
```

```
# Define Implicit Function
ffi_linear_dplyrdo <- function(fl_A, fl_alpha, ar_nN_A, ar_nN_alpha){
  # ar_A_alpha[1] is A
  # ar_A_alpha[2] is alpha

  print(paste0('cur row, fl_A=', fl_A, ', fl_alpha=', fl_alpha))
```

fl_A	fl_alpha
-2	0.1
-1	0.3
0	0.5
1	0.7
2	0.9

```
  fl_out = sum(fl_A*ar_nN_A + 1/(fl_alpha + 1/ar_nN_alpha))

  return(fl_out)
}

# Evaluate function row by row of tibble
# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
tb_nN_by_nQ_A_alpha_show <- tb_nN_by_nQ_A_alpha %>% rowwise() %>%
                    mutate(dplyr_eval = ffi_linear_dplyrdo(fl_A, fl_alpha, ar_nN_A, ar_nN_alpha))
```

```
## [1] "cur row, fl_A=-2, fl_alpha=0.1"
## [1] "cur row, fl_A=-1, fl_alpha=0.3"
## [1] "cur row, fl_A=0, fl_alpha=0.5"
## [1] "cur row, fl_A=1, fl_alpha=0.7"
## [1] "cur row, fl_A=2, fl_alpha=0.9"
```

```
# Show
kable(tb_nN_by_nQ_A_alpha_show) %>%
  kable_styling_fc()
```

fl_A	fl_alpha	dplyr_eval
-2	0.1	2.346356
-1	0.3	2.094273
0	0.5	1.895316
1	0.7	1.733708
2	0.9	1.599477

same as before, still rowwise, but hard code some inputs:

```
# Define function, fixed inputs are not parameters, but defined earlier as a part of the function
# ar_nN_A, ar_nN_alpha are fixed, not parameters
ffi_linear_dplyrdo_func <- function(fl_A, fl_alpha){
  fl_out <- sum(fl_A*ar_nN_A + 1/(fl_alpha + 1/ar_nN_alpha))
  return(fl_out)
}

# Evaluate function row by row of tibble
tbfunc_A_nN_by_nQ_A_alpha_rowwise = tb_nN_by_nQ_A_alpha %>% rowwise() %>%
                    mutate(dplyr_eval = ffi_linear_dplyrdo_func(fl_A, fl_alpha))
# Show
kable(tbfunc_A_nN_by_nQ_A_alpha_rowwise) %>%
  kable_styling_fc()
```

fl_A	fl_alpha	dplyr_eval
-2	0.1	2.346356
-1	0.3	2.094273
0	0.5	1.895316
1	0.7	1.733708
2	0.9	1.599477

3.3.1.5 Using Dplyr Mutate with Pmap

Apparently *rowwise()* is not a good idea, and *pmap* should be used, below is the *pmap* solution to the problem. Which does seem nicer. Crucially, don't have to define input parameter names, automatically I think they are matching up to the names in the function

- dplyr mutate pass function
- r function quosure string multiple
- r function multiple parameters as one string
- dplyr mutate anonymous function
- quosure style lambda
- pmap tibble rows
- dplyr pwalk

```
# Define function, fixed inputs are not parameters, but defined earlier as a part of the function
# Rorate fl_alpha and fl_A name compared to before to make sure pmap tracks by names
ffi_linear_dplyrdo_func <- function(fl_alpha, fl_A){
  fl_out <- sum(fl_A*ar_nN_A + 1/(fl_alpha + 1/ar_nN_alpha))
  return(fl_out)
}

# Evaluate a function row by row of dataframe, generate list, then to vecotr
tb_nN_by_nQ_A_alpha %>% pmap(ffi_linear_dplyrdo_func) %>% unlist()
```

```
## [1] 2.346356 2.094273 1.895316 1.733708 1.599477
```

```
# Same as above, but in line line and save output as new column in dataframe
# note this ONLY works if the tibble only has variables that are inputs for the function
# if tibble contains additional variables, those should be droppd, or only the ones needed
# selected, inside the pmap call below.
tbfunc_A_nN_by_nQ_A_alpha_pmap <- tb_nN_by_nQ_A_alpha %>%
          mutate(dplyr_eval_pmap =
                   unlist(
                     pmap(tb_nN_by_nQ_A_alpha, ffi_linear_dplyrdo_func)
                     )
                 )

# Show
kable(tbfunc_A_nN_by_nQ_A_alpha_pmap) %>%
  kable_styling_fc()
```

fl_A	fl_alpha	dplyr_eval_pmap
-2	0.1	2.346356
-1	0.3	2.094273
0	0.5	1.895316
1	0.7	1.733708
2	0.9	1.599477

3.3.1.6 DPLYR Three Types of Inputs ROWWISE

Now, we have three types of parameters, for something like a bisection type calculation. We will supply the program with a function with some hard-coded value inside, and as parameters, we will have one parameter which is a row in the current matrix, and another parameter which is a sclar values. The three types of parameters are dealt with sparately:

1. parameters that are fixed for all bisection iterations, but differ for each row
   - these are hard-coded into the function
2. parameters that are fixed for all bisection iterations, but are shared across rows
   - these are the first parameter of the function, a list
3. parameters that differ for each iteration, but differ acoss iterations

    - second scalar value parameter for the function

- dplyr mutate function applow to each row dot notation
- note rowwise might be bad according to Hadley, should use pmap?

```
ffi_linear_dplyrdo_fdot <- function(ls_row, fl_param){
  # Type 1 Param = ar_nN_A, ar_nN_alpha
  # Type 2 Param = ls_row$fl_A, ls_row$fl_alpha
  # Type 3 Param = fl_param

  fl_out <- (sum(ls_row$fl_A*ar_nN_A + 1/(ls_row$fl_alpha + 1/ar_nN_alpha))) + fl_param
  return(fl_out)
}

cur_func <- ffi_linear_dplyrdo_fdot
fl_param <- 0
dplyr_eval_flex <- tb_nN_by_nQ_A_alpha %>% rowwise() %>%
                              do(dplyr_eval_flex = cur_func(., fl_param)) %>%
                              unnest(dplyr_eval_flex)
tbfunc_B_nN_by_nQ_A_alpha <- tb_nN_by_nQ_A_alpha %>% add_column(dplyr_eval_flex)
# Show
kable(tbfunc_B_nN_by_nQ_A_alpha) %>%
  kable_styling_fc()
```

fl_A	fl_alpha	dplyr_eval_flex
-2	0.1	2.346356
-1	0.3	2.094273
0	0.5	1.895316
1	0.7	1.733708
2	0.9	1.599477

3.3.1.7 Compare Apply and Mutate Results

```
# Show overall Results
mt_results <- cbind(ar_func_apply, ar_func_sapply,
                    tb_nN_by_nQ_A_alpha_show['dplyr_eval'],
                    tbfunc_A_nN_by_nQ_A_alpha_rowwise['dplyr_eval'],
                    tbfunc_A_nN_by_nQ_A_alpha_pmap['dplyr_eval_pmap'],
                    tbfunc_B_nN_by_nQ_A_alpha['dplyr_eval_flex'],
                    mt_nN_by_nQ_A_alpha)
colnames(mt_results) <- c('eval_lin_apply', 'eval_lin_sapply',
                          'eval_dplyr_mutate',
                          'eval_dplyr_mutate_hcode',
                          'eval_dplyr_mutate_pmap',
                          'eval_dplyr_mutate_flex',
                          'A_child', 'alpha_child')
kable(mt_results) %>%
  kable_styling_fc_wide()
```

	eval_lin_apply	eval_lin_sapply	eval_dplyr_mutate	eval_dplyr_mutate_hcode	eval_dplyr_mutate_pmap	eval_dplyr_mutate_flex	A_child	alpha_child
X1	2.346356	2.346356	2.346356	2.346356	2.346356	2.346356	-2	0.1
X2	2.094273	2.094273	2.094273	2.094273	2.094273	2.094273	-1	0.3
X3	1.895316	1.895316	1.895316	1.895316	1.895316	1.895316	0	0.5
X4	1.733708	1.733708	1.733708	1.733708	1.733708	1.733708	1	0.7
X5	1.599477	1.599477	1.599477	1.599477	1.599477	1.599477	2	0.9

Chapter 4

Panel

4.1 Generate and Join

4.1.1 Generate Panel Structure

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

4.1.1.1 Balanced Panel Skeleton

There are N individuals, each could be observed M times. In the example below, there are 3 students, each observed over 4 dates. This just uses the uncount function from *tidyr*.

```
# Define
it_N <- 3
it_M <- 5
svr_id <- 'student_id'
svr_date <- 'class_day'

# dataframe
df_panel_skeleton <- as_tibble(matrix(it_M, nrow=it_N, ncol=1)) %>%
  rowid_to_column(var = svr_id) %>%
  uncount(V1) %>%
  group_by(!!sym(svr_id)) %>% mutate(!!sym(svr_date) := row_number()) %>%
  ungroup()

# Print
kable(df_panel_skeleton) %>%
  kable_styling_fc()
```

4.1.2 Join Datasets

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

4.1.2.1 Join Panel with Multiple Keys

We have two datasets, one for student enrollment, panel over time, but some students do not show up on some dates. The other is a skeleton panel with all student ID and all dates. Often we need to join dataframes together, and we need to join by the student ID and the panel time Key at the same time. When students show up, there is a quiz score for that day, so the joined panel should have as data column quiz score

student_id	class_day
1	1
1	2
1	3
1	4
1	5
2	1
2	2
2	3
2	4
2	5
3	1
3	2
3	3
3	4
3	5

Student count is N, total dates are M. First we generate two panels below, then we join by both keys using *left_join*. First, define dataframes:

```
# Define
it_N <- 4
it_M <- 3
svr_id <- 'sid'
svr_date <- 'classday'
svr_attend <- 'date_in_class'

# Panel Skeleton
df_panel_balanced_skeleton <- as_tibble(matrix(it_M, nrow=it_N, ncol=1)) %>%
  rowid_to_column(var = svr_id) %>%
  uncount(V1) %>%
  group_by(!!sym(svr_id)) %>% mutate(!!sym(svr_date) := row_number()) %>%
  ungroup()
# Print
kable(df_panel_balanced_skeleton) %>%
  kable_styling_fc()
```

sid	classday
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3
4	1
4	2
4	3

```
# Smaller Panel of Random Days in School
set.seed(456)
df_panel_attend <- as_tibble(matrix(it_M, nrow=it_N, ncol=1)) %>%
  rowid_to_column(var = svr_id) %>%
  uncount(V1) %>%
  group_by(!!sym(svr_id)) %>% mutate(!!sym(svr_date) := row_number()) %>%
```

```
  ungroup() %>% mutate(in_class = case_when(rnorm(n(),mean=0,sd=1) < 0 ~ 1, TRUE ~ 0)) %>%
  filter(in_class == 1) %>% select(!!sym(svr_id), !!sym(svr_date)) %>%
  rename(!!sym(svr_attend) := !!sym(svr_date)) %>%
  mutate(dayquizscore = rnorm(n(),mean=80,sd=10))
# Print
kable(df_panel_attend) %>%
  kable_styling_fc()
```

sid	date_in_class	dayquizscore
1	1	89.88726
2	1	96.53929
2	2	65.59195
2	3	99.47356
4	2	97.36936

Second, now join dataframes:

```
# Join with explicit names
df_quiz_joined_multikey <- df_panel_balanced_skeleton %>%
  left_join(df_panel_attend,
            by=(c('sid'='sid', 'classday'='date_in_class')))

# Join with setname strings
df_quiz_joined_multikey_setnames <- df_panel_balanced_skeleton %>%
  left_join(df_panel_attend, by=setNames(c('sid', 'date_in_class'), c('sid', 'classday')))

# Print
kable(df_quiz_joined_multikey) %>%
  kable_styling_fc()
```

sid	classday	dayquizscore
1	1	89.88726
1	2	NA
1	3	NA
2	1	96.53929
2	2	65.59195
2	3	99.47356
3	1	NA
3	2	NA
3	3	NA
4	1	NA
4	2	97.36936
4	3	NA

```
kable(df_quiz_joined_multikey_setnames) %>%
  kable_styling_fc()
```

4.1.2.2 Stack Panel Frames Together

There are multiple panel dataframe, each for different subsets of dates. All variable names and units of observations are identical. Use DPLYR bind_rows.

```
# Define
it_N <- 2 # Number of individuals
it_M <- 3 # Number of Months
svr_id <- 'sid'
svr_date <- 'date'
```

sid	classday	dayquizscore
1	1	89.88726
1	2	NA
1	3	NA
2	1	96.53929
2	2	65.59195
2	3	99.47356
3	1	NA
3	2	NA
3	3	NA
4	1	NA
4	2	97.36936
4	3	NA

```
# Panel First Half of Year
df_panel_m1tom3 <- as_tibble(matrix(it_M, nrow=it_N, ncol=1)) %>%
  rowid_to_column(var = svr_id) %>%
  uncount(V1) %>%
  group_by(!!sym(svr_id)) %>% mutate(!!sym(svr_date) := row_number()) %>%
  ungroup()

# Panel Second Half of Year
df_panel_m4tom6 <- as_tibble(matrix(it_M, nrow=it_N, ncol=1)) %>%
  rowid_to_column(var = svr_id) %>%
  uncount(V1) %>%
  group_by(!!sym(svr_id)) %>% mutate(!!sym(svr_date) := row_number() + 3) %>%
  ungroup()

# Bind Rows
df_panel_m1tm6 <- bind_rows(df_panel_m1tom3, df_panel_m4tom6) %>% arrange(!!!syms(c(svr_id, svr_date

# Print
kable(df_panel_m1tom3) %>%
  kable_styling_fc()
```

sid	date
1	1
1	2
1	3
2	1
2	2
2	3

```
kable(df_panel_m4tom6) %>%
  kable_styling_fc()
```

sid	date
1	4
1	5
1	6
2	4
2	5
2	6

```
kable(df_panel_m1tm6) %>%
  kable_styling_fc()
```

sid	date
1	1
1	2
1	3
1	4
1	5
1	6
2	1
2	2
2	3
2	4
2	5
2	6

4.2 Wide and Long

4.2.1 Long to Wide

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

Using the pivot_wider function in tidyr to reshape panel or other data structures

4.2.1.1 Panel Long Attendance Roster to Wide

There are N students in class, but only a subset of them attend class each day. If student id_i is in class on day Q, the teacher records on a sheet the date and the student ID. So if the student has been in class 10 times, the teacher has ten rows of recorded data for the student with two columns: column one is the student ID, and column two is the date on which the student was in class. Suppose there were 50 students, who on average attended exactly 10 classes each during the semester, this means we have $10 \cdot 50$ rows of data, with differing numbers of rows for each student. This is shown as *df_panel_attend_date* generated below.

Now we want to generate a new dataframe, where each row is a date, and each column is a student. The values in the new dataframe shows, at the Q^{th} day, how many classes student i has attended so far. The following results is also in a REconTools Function. This is shown as *df_attend_cumu_by_day* generated below.

First, generate the raw data structure, *df_panel_attend_date*:

```
# Define
it_N <- 3
it_M <- 5
svr_id <- 'student_id'

# from : support/rand/fs_rand_draws.Rmd
set.seed(222)
df_panel_attend_date <- as_tibble(matrix(it_M, nrow=it_N, ncol=1)) %>%
  rowid_to_column(var = svr_id) %>%
  uncount(V1) %>%
  group_by(!!sym(svr_id)) %>% mutate(date = row_number()) %>%
  ungroup() %>% mutate(in_class = case_when(rnorm(n(),mean=0,sd=1) < 0 ~ 1, TRUE ~ 0)) %>%
  filter(in_class == 1) %>% select(!!sym(svr_id), date) %>%
  rename(date_in_class = date)
```

```
# Print
kable(df_panel_attend_date) %>%
  kable_styling_fc()
```

student_id	date_in_class
1	2
1	4
2	1
2	2
2	5
3	2
3	3
3	5

Second, generate wider data structure, *df_attend_cumu_by_day*:

```
# Define
svr_id <- 'student_id'
svr_date <- 'date_in_class'
st_idcol_prefix <- 'sid_'

# Generate cumulative enrollment counts by date
df_panel_attend_date_addone <- df_panel_attend_date %>% mutate(attended = 1)
kable(df_panel_attend_date_addone) %>%
  kable_styling_fc()
```

student_id	date_in_class	attended
1	2	1
1	4	1
2	1	1
2	2	1
2	5	1
3	2	1
3	3	1
3	5	1

```
# Pivot Wide
df_panel_attend_date_wider <- df_panel_attend_date_addone %>%
  pivot_wider(names_from = svr_id,
              values_from = attended)
kable(df_panel_attend_date_wider) %>%
  kable_styling_fc()
```

date_in_class	1	2	3
2	1	1	1
4	1	NA	NA
1	NA	1	NA
5	NA	1	1
3	NA	NA	1

```
# Sort and rename
# rename see: https://fanwangecon.github.io/R4Econ/amto/tibble/fs_tib_basics.html
ar_unique_ids <- sort(unique(df_panel_attend_date %>% pull(!!sym(svr_id))))
df_panel_attend_date_wider_sort <- df_panel_attend_date_wider %>%
    arrange(!!sym(svr_date)) %>%
    rename_at(vars(num_range('',ar_unique_ids))
              , list(~paste0(st_idcol_prefix, . , '')))
```

```r
              )
kable(df_panel_attend_date_wider_sort) %>%
  kable_styling_fc()
```

date_in_class	sid_1	sid_2	sid_3
1	NA	1	NA
2	1	1	1
3	NA	NA	1
4	1	NA	NA
5	NA	1	1

```r
# replace NA and cumusum again
# see: R4Econ/support/function/fs_func_multivar for renaming and replacing
df_attend_cumu_by_day <- df_panel_attend_date_wider_sort %>%
  mutate_at(vars(contains(st_idcol_prefix)), list(~replace_na(., 0))) %>%
  mutate_at(vars(contains(st_idcol_prefix)), list(~cumsum(.)))

kable(df_attend_cumu_by_day) %>%
  kable_styling_fc()
```

date_in_class	sid_1	sid_2	sid_3
1	0	1	0
2	1	2	1
3	1	2	2
4	2	2	2
5	2	3	3

The structure above is also a function in Fan's REconTools Package, here the function is tested:

```r
# Parameters
df <- df_panel_attend_date
svr_id_i <- 'student_id'
svr_id_t <- 'date_in_class'
st_idcol_prefix <- 'sid_'

# Invoke Function
ls_df_rosterwide <- ff_panel_expand_longrosterwide(df, svr_id_t, svr_id_i, st_idcol_prefix)
df_roster_wide_func <- ls_df_rosterwide$df_roster_wide
df_roster_wide_cumu_func <- ls_df_rosterwide$df_roster_wide_cumu

# Print
print(df_roster_wide_func)
```

```
## # A tibble: 5 x 4
##   date_in_class sid_1 sid_2 sid_3
##           <int> <dbl> <dbl> <dbl>
## 1             1    NA     1    NA
## 2             2     1     1     1
## 3             3    NA    NA     1
## 4             4     1    NA    NA
## 5             5    NA     1     1
```

```r
print(df_roster_wide_cumu_func)
```

```
## # A tibble: 5 x 4
##   date_in_class sid_1 sid_2 sid_3
##           <int> <dbl> <dbl> <dbl>
## 1             1     0     1     0
## 2             2     1     2     1
```

```
## 3                 3      1      2      2
## 4                 4      2      2      2
## 5                 5      2      3      3
```

Chapter 5

Linear Regression

5.1 OLS and IV

Back to **Fan**'s R4Econ Homepage **Table of Content**

5.1.1 OLS and IV Regression

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

IV regression using AER package. Option to store all results in dataframe row for combining results from other estimations together. Produce Row Statistics.

5.1.1.1 Construct Program

```
# IV regression function
# The code below uses the AER library's regresison function
# All results are stored in a single row as data_frame
# This functoin could work with dplyr do
# var.y is single outcome, vars.x, vars.c and vars.z are vectors of endogenous variables, controls a
regf.iv <- function(var.y, vars.x, vars.c, vars.z, df, transpose=TRUE) {

#     print(length(vars.z))

    # A. Set-Up Equation
    str.vars.x <- paste(vars.x, collapse='+')
    str.vars.c <- paste(vars.c, collapse='+')

    df <- df %>% select(one_of(var.y, vars.x, vars.c, vars.z)) %>% drop_na() %>% filter_all(all_vars

    if (length(vars.z) >= 1) {
        #     library(AER)
            str.vars.z <- paste(vars.z, collapse='+')
            equa.iv <- paste(var.y,
                             paste(paste(str.vars.x, str.vars.c, sep='+'),
                                   paste(str.vars.z, str.vars.c, sep='+'),
                                   sep='|'),
                             sep='~')
        #     print(equa.iv)

        # B. IV Regression
        ivreg.summ <- summary(ivreg(as.formula(equa.iv), data=df),
                              vcov = sandwich, df = Inf, diagnostics = TRUE)
```

```r
        # C. Statistics from IV Regression
    #     ivreg.summ$coef
    #     ivreg.summ$diagnostics

        # D. Combine Regression Results into a Matrix
        df.results <- suppressMessages(as_tibble(ivreg.summ$coef, rownames='rownames') %>%
            full_join(as_tibble(ivreg.summ$diagnostics, rownames='rownames')) %>%
            full_join(tibble(rownames=c('vars'),
                             var.y=var.y,
                             vars.x=str.vars.x,
                             vars.z=str.vars.z,
                             vars.c=str.vars.c)))
    } else {

        # OLS regression
        equa.ols <- paste(var.y,
                          paste(paste(vars.x, collapse='+'),
                                paste(vars.c, collapse='+'), sep='+'),
                          sep='~')

        lmreg.summ <- summary(lm(as.formula(equa.ols), data=df))

        lm.diagnostics <- as_tibble(list(df1=lmreg.summ$df[[1]],
                                         df2=lmreg.summ$df[[2]],
                                         df3=lmreg.summ$df[[3]],
                                         sigma=lmreg.summ$sigma,
                                         r.squared=lmreg.summ$r.squared,
                                         adj.r.squared=lmreg.summ$adj.r.squared)) %>%
                                         gather(variable, value) %>%
                                         rename(rownames = variable) %>%
                                         rename(v = value)

        df.results <- suppressMessages(as_tibble(lmreg.summ$coef, rownames='rownames') %>%
            full_join(lm.diagnostics) %>%
            full_join(tibble(rownames=c('vars'),
                             var.y=var.y,
                             vars.x=str.vars.x,
                             vars.c=str.vars.c)))
    }

    # E. Flatten Matrix, All IV results as a single tibble row to be combined with other IV results
    df.row.results <- df.results %>%
        gather(variable, value, -rownames) %>%
        drop_na() %>%
        unite(esti.val, rownames, variable) %>%
        mutate(esti.val = gsub(' ', '', esti.val))

    if (transpose) {
      df.row.results <- df.row.results %>% spread(esti.val, value)
    }

    # F. Return
    return(data.frame(df.row.results))
}
```

5.1.1.2 Program Testing

Load Data

```
# Library
library(tidyverse)
library(AER)

# Load Sample Data
setwd('C:/Users/fan/R4Econ/_data/')
df <- read_csv('height_weight.csv')
```

```
## Parsed with column specification:
## cols(
##   S.country = col_character(),
##   vil.id = col_double(),
##   indi.id = col_double(),
##   sex = col_character(),
##   svymthRound = col_double(),
##   momEdu = col_double(),
##   wealthIdx = col_double(),
##   hgt = col_double(),
##   wgt = col_double(),
##   hgt0 = col_double(),
##   wgt0 = col_double(),
##   prot = col_double(),
##   cal = col_double(),
##   p.A.prot = col_double(),
##   p.A.nProt = col_double()
## )
```

```
# Setting
options(repr.matrix.max.rows=50, repr.matrix.max.cols=50)
```

```
# One Instrucments
var.y <- c('hgt')
vars.x <- c('prot')
vars.z <- NULL
vars.c <- c('sex', 'hgt0', 'wgt0')
# Regression
regf.iv(var.y, vars.x, vars.c, vars.z, df, transpose=FALSE)
```

5.1.1.2.1 Example No Instrument, OLS

```
##                  esti.val                value
## 1   (Intercept)_Estimate     52.1186286658651
## 2          prot_Estimate    0.374472386357917
## 3       sexMale_Estimate    0.611043720578292
## 4          hgt0_Estimate    0.148513781160842
## 5          wgt0_Estimate  0.00150560230505631
## 6  (Intercept)_Std.Error     1.57770483608693
## 7         prot_Std.Error  0.00418121191133815
## 8      sexMale_Std.Error    0.118396259120659
## 9         hgt0_Std.Error   0.0393807494783186
## 10        wgt0_Std.Error 0.000187123663624397
## 11    (Intercept)_tvalue     33.0344608660332
## 12           prot_tvalue     89.5607288744356
## 13        sexMale_tvalue     5.16100529794248
## 14           hgt0_tvalue     3.77122790013449
```

```
## 15           wgt0_tvalue      8.04602836377991
## 16  (Intercept)_Pr(>|t|) 9.92126150975783e-233
## 17         prot_Pr(>|t|)                     0
## 18      sexMale_Pr(>|t|)  2.48105505495642e-07
## 19         hgt0_Pr(>|t|)  0.000162939618371183
## 20         wgt0_Pr(>|t|)  9.05257561534111e-16
## 21                 df1_v                     5
## 22                 df2_v                 18958
## 23                 df3_v                     5
## 24               sigma_v      8.06197784622979
## 25           r.squared_v     0.319078711001325
## 26       adj.r.squared_v     0.318935041565942
## 27          vars_var.y                     hgt
## 28          vars_vars.x                   prot
## 29          vars_vars.c           sex+hgt0+wgt0
```

```r
# One Instrucments
var.y <- c('hgt')
vars.x <- c('prot')
vars.z <- c('momEdu')
vars.c <- c('sex', 'hgt0', 'wgt0')
# Regression
regf.iv(var.y, vars.x, vars.c, vars.z, df, transpose=FALSE)
```

5.1.1.2.2 Example 1 Insturment

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
##                      esti.val                 value
## 1       (Intercept)_Estimate      43.4301969117558
## 2              prot_Estimate     0.130833343849446
## 3           sexMale_Estimate     0.868121847262411
## 4              hgt0_Estimate     0.412093881817148
## 5              wgt0_Estimate  0.000858630042617921
## 6      (Intercept)_Std.Error      1.82489550971182
## 7             prot_Std.Error    0.0192036220809189
## 8          sexMale_Std.Error      0.13373016700542
## 9             hgt0_Std.Error    0.0459431912927002
## 10            wgt0_Std.Error   0.00022691057702563
## 11        (Intercept)_zvalue       23.798730766023
## 12               prot_zvalue      6.81295139521853
## 13            sexMale_zvalue      6.49159323361366
## 14               hgt0_zvalue      8.96963990141069
## 15               wgt0_zvalue       3.7840018472164
## 16      (Intercept)_Pr(>|z|)  3.4423766196876e-125
## 17             prot_Pr(>|z|)  9.56164541643828e-12
## 18          sexMale_Pr(>|z|)  8.49333228172763e-11
## 19             hgt0_Pr(>|z|)  2.97485394526792e-19
## 20             wgt0_Pr(>|z|)  0.000154326676608523
## 21       Weakinstruments_df1                     1
## 22           Wu-Hausman_df1                      1
## 23               Sargan_df1                      0
## 24       Weakinstruments_df2                 16394
## 25           Wu-Hausman_df2                  16393
## 26 Weakinstruments_statistic      935.817456612075
## 27      Wu-Hausman_statistic      123.595856606729
## 28   Weakinstruments_p-value 6.39714929178024e-200
```

```
## 29        Wu-Hausman_p-value  1.30703637796748e-28
## 30                vars_var.y                   hgt
## 31               vars_vars.x                  prot
## 32               vars_vars.z                momEdu
## 33               vars_vars.c         sex+hgt0+wgt0
```

```
# Multiple Instrucments
var.y <- c('hgt')
vars.x <- c('prot')
vars.z <- c('momEdu', 'wealthIdx', 'p.A.prot', 'p.A.nProt')
vars.c <- c('sex', 'hgt0', 'wgt0')
# Regression
regf.iv(var.y, vars.x, vars.c, vars.z, df, transpose=FALSE)
```

5.1.1.2.3 Example Multiple Instrucments

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
##                     esti.val                                 value
## 1        (Intercept)_Estimate                      42.2437613555242
## 2               prot_Estimate                      0.26699945194704
## 3            sexMale_Estimate                     0.695548488812932
## 4               hgt0_Estimate                     0.424954881263031
## 5               wgt0_Estimate                  0.000486951420329484
## 6       (Intercept)_Std.Error                      1.85356686789642
## 7              prot_Std.Error                    0.0154939347964083
## 8           sexMale_Std.Error                     0.133157977814374
## 9              hgt0_Std.Error                    0.0463195803786233
## 10             wgt0_Std.Error                  0.000224867994873235
## 11         (Intercept)_zvalue                      22.7905246296649
## 12                prot_zvalue                      17.2325142357597
## 13             sexMale_zvalue                      5.22348341593581
## 14                hgt0_zvalue                      9.17441129192849
## 15                wgt0_zvalue                      2.16549901022595
## 16        (Intercept)_Pr(>|z|)                  5.69294074735747e-115
## 17               prot_Pr(>|z|)                   1.51424021931607e-66
## 18            sexMale_Pr(>|z|)                   1.75588197502565e-07
## 19               hgt0_Pr(>|z|)                   4.54048595587756e-20
## 20               wgt0_Pr(>|z|)                     0.030349491114332
## 21        Weakinstruments_df1                                     4
## 22             Wu-Hausman_df1                                     1
## 23                 Sargan_df1                                     3
## 24        Weakinstruments_df2                                 14914
## 25             Wu-Hausman_df2                                 14916
## 26 Weakinstruments_statistic                      274.147084958343
## 27       Wu-Hausman_statistic                      17.7562545747101
## 28           Sargan_statistic                      463.729664547249
## 29   Weakinstruments_p-value                  8.61731956233366e-228
## 30         Wu-Hausman_p-value                   2.52567249124181e-05
## 31             Sargan_p-value                  3.45452874915475e-100
## 32                 vars_var.y                                   hgt
## 33                vars_vars.x                                  prot
## 34                vars_vars.z momEdu+wealthIdx+p.A.prot+p.A.nProt
## 35                vars_vars.c                         sex+hgt0+wgt0
```

```
# Multiple Instrucments
```

```
var.y <- c('hgt')
vars.x <- c('prot', 'cal')
vars.z <- c('momEdu', 'wealthIdx', 'p.A.prot', 'p.A.nProt')
vars.c <- c('sex', 'hgt0', 'wgt0')
# Regression
regf.iv(var.y, vars.x, vars.c, vars.z, df, transpose=FALSE)
```

5.1.1.2.4 Example Multiple Endogenous Variables

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
##                                esti.val                              value
## 1              (Intercept)_Estimate                   44.0243196254297
## 2                     prot_Estimate                   -1.4025623247106
## 3                      cal_Estimate                  0.065104895750151
## 4                  sexMale_Estimate                  0.120832787571818
## 5                     hgt0_Estimate                  0.286525437984517
## 6                     wgt0_Estimate               0.000850481389651033
## 7             (Intercept)_Std.Error                   2.75354847244082
## 8                    prot_Std.Error                  0.198640060273635
## 9                     cal_Std.Error                0.00758881298880996
## 10               sexMale_Std.Error                  0.209984580636303
## 11                  hgt0_Std.Error                 0.0707828182888255
## 12                  wgt0_Std.Error                0.00033711210444429
## 13              (Intercept)_zvalue                   15.9882130516502
## 14                     prot_zvalue                   -7.06082309267581
## 15                      cal_zvalue                   8.57906181719737
## 16                  sexMale_zvalue                  0.575436478267434
## 17                     hgt0_zvalue                   4.04795181812859
## 18                     wgt0_zvalue                   2.52284441418383
## 19             (Intercept)_Pr(>|z|)               1.54396598126854e-57
## 20                    prot_Pr(>|z|)               1.65519210848649e-12
## 21                     cal_Pr(>|z|)               9.56500648203187e-18
## 22                 sexMale_Pr(>|z|)                  0.564996139463599
## 23                    hgt0_Pr(>|z|)               5.16677787108928e-05
## 24                    wgt0_Pr(>|z|)                 0.0116409892837831
## 25         Weakinstruments(prot)_df1                                  4
## 26          Weakinstruments(cal)_df1                                  4
## 27                    Wu-Hausman_df1                                  2
## 28                        Sargan_df1                                  2
## 29         Weakinstruments(prot)_df2                              14914
## 30          Weakinstruments(cal)_df2                              14914
## 31                    Wu-Hausman_df2                              14914
## 32 Weakinstruments(prot)_statistic                   274.147084958343
## 33  Weakinstruments(cal)_statistic                   315.036848606231
## 34              Wu-Hausman_statistic                   94.7020085425169
## 35                  Sargan_statistic                   122.081979628898
## 36   Weakinstruments(prot)_p-value              8.61731956233366e-228
## 37    Weakinstruments(cal)_p-value              1.18918641220866e-260
## 38                Wu-Hausman_p-value               1.35024050408262e-41
## 39                    Sargan_p-value               3.09196773720398e-27
## 40                        vars_var.y                                hgt
## 41                       vars_vars.x                           prot+cal
## 42                       vars_vars.z momEdu+wealthIdx+p.A.prot+p.A.nProt
## 43                       vars_vars.c                      sex+hgt0+wgt0
```

5.1.1.2.5 Examples Line by Line The examples are just to test the code with different types of variables.

```r
# Selecting Variables
var.y <- c('hgt')
vars.x <- c('prot', 'cal')
vars.z <- c('momEdu', 'wealthIdx', 'p.A.prot', 'p.A.nProt')
vars.c <- c('sex', 'hgt0', 'wgt0')
```

```r
# A. create Equation
str.vars.x <- paste(vars.x, collapse='+')
str.vars.c <- paste(vars.c, collapse='+')
str.vars.z <- paste(vars.z, collapse='+')
print(str.vars.x)
```

```
## [1] "prot+cal"
```

```r
print(str.vars.c)
```

```
## [1] "sex+hgt0+wgt0"
```

```r
print(str.vars.z)
```

```
## [1] "momEdu+wealthIdx+p.A.prot+p.A.nProt"
```

```r
equa.iv <- paste(var.y,
                 paste(paste(str.vars.x, str.vars.c, sep='+'),
                       paste(str.vars.z, str.vars.c, sep='+'),
                       sep='|'),
                 sep='~')
print(equa.iv)
```

```
## [1] "hgt~prot+cal+sex+hgt0+wgt0|momEdu+wealthIdx+p.A.prot+p.A.nProt+sex+hgt0+wgt0"
```

```r
# B. regression
res.ivreg <- ivreg(as.formula(equa.iv), data=df)
coef(res.ivreg)
```

```
##   (Intercept)          prot           cal       sexMale          hgt0          wgt0
## 44.0243196254 -1.4025623247  0.0651048958  0.1208327876  0.2865254380  0.0008504814
```

```r
# C. Regression Summary
ivreg.summ <- summary(res.ivreg, vcov = sandwich, df = Inf, diagnostics = TRUE)

ivreg.summ$coef
```

```
##                  Estimate    Std. Error    z value     Pr(>|z|)
## (Intercept) 44.0243196254 2.7535484724 15.9882131 1.543966e-57
## prot        -1.4025623247 0.1986400603 -7.0608231 1.655192e-12
## cal          0.0651048958 0.0075888130  8.5790618 9.565006e-18
## sexMale      0.1208327876 0.2099845806  0.5754365 5.649961e-01
## hgt0         0.2865254380 0.0707828183  4.0479518 5.166778e-05
## wgt0         0.0008504814 0.0003371121  2.5228444 1.164099e-02
## attr(,"df")
## [1] 0
```

```r
ivreg.summ$diagnostics
```

```
##                           df1   df2 statistic       p-value
## Weak instruments (prot)    4 14914 274.14708 8.617320e-228
## Weak instruments (cal)     4 14914 315.03685 1.189186e-260
## Wu-Hausman                 2 14914  94.70201  1.350241e-41
## Sargan                     2    NA 122.08198  3.091968e-27
```

```r
# D. Combine Regression Results into a Matrix
df.results <- suppressMessages(as_tibble(ivreg.summ$coef, rownames='rownames') %>%
```

```
    full_join(as_tibble(ivreg.summ$diagnostics, rownames='rownames')) %>%
    full_join(tibble(rownames=c('vars'),
                     var.y=var.y,
                     vars.x=str.vars.x,
                     vars.z=str.vars.z,
                     vars.c=str.vars.c)))
# E. Flatten Matrix, All IV results as a single tibble row to be combined with other IV results
df.row.results <- df.results %>%
    gather(variable, value, -rownames) %>%
    drop_na() %>%
    unite(esti.val, rownames, variable) %>%
    mutate(esti.val = gsub(' ', '', esti.val))
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
# F. Results as Single Colum
df.row.results
```

```
## # A tibble: 43 x 2
##    esti.val               value
##    <chr>                  <chr>
##  1 (Intercept)_Estimate   44.0243196254297
##  2 prot_Estimate          -1.4025623247106
##  3 cal_Estimate           0.065104895750151
##  4 sexMale_Estimate       0.120832787571818
##  5 hgt0_Estimate          0.286525437984517
##  6 wgt0_Estimate          0.000850481389651033
##  7 (Intercept)_Std.Error 2.75354847244082
##  8 prot_Std.Error         0.198640060273635
##  9 cal_Std.Error          0.00758881298880996
## 10 sexMale_Std.Error      0.209984580636303
## # ... with 33 more rows
```

```
# G. Results as Single Row
df.row.results
```

```
## # A tibble: 43 x 2
##    esti.val               value
##    <chr>                  <chr>
##  1 (Intercept)_Estimate   44.0243196254297
##  2 prot_Estimate          -1.4025623247106
##  3 cal_Estimate           0.065104895750151
##  4 sexMale_Estimate       0.120832787571818
##  5 hgt0_Estimate          0.286525437984517
##  6 wgt0_Estimate          0.000850481389651033
##  7 (Intercept)_Std.Error 2.75354847244082
##  8 prot_Std.Error         0.198640060273635
##  9 cal_Std.Error          0.00758881298880996
## 10 sexMale_Std.Error      0.209984580636303
## # ... with 33 more rows
```

```
df.row.results %>% spread(esti.val, value)
```

```
## # A tibble: 1 x 43
##   `(Intercept)_Es~ `(Intercept)_Pr~ `(Intercept)_St~ `(Intercept)_zv~ cal_Estimate `cal_Pr(>|z|)`
##   <chr>            <chr>            <chr>            <chr>            <chr>        <chr>
## 1 44.0243196254297 1.5439659812685~ 2.75354847244082 15.9882130516502 0.065104895~ 9.56500648203~
## # ... with 33 more variables: hgt0_Std.Error <chr>, hgt0_zvalue <chr>, prot_Estimate <chr>, `prot
## #   Sargan_df1 <chr>, `Sargan_p-value` <chr>, Sargan_statistic <chr>, sexMale_Estimate <chr>, `se
```

```
## #   sexMale_zvalue <chr>, vars_var.y <chr>, vars_vars.c <chr>, vars_vars.x <chr>, vars_vars.z <ch
## #   `Weakinstruments(cal)_df2` <chr>, `Weakinstruments(cal)_p-value` <chr>, `Weakinstruments(cal)
## #   `Weakinstruments(prot)_df2` <chr>, `Weakinstruments(prot)_p-value` <chr>, `Weakinstruments(pr
## #   `wgt0_Pr(>|z|)` <chr>, wgt0_Std.Error <chr>, wgt0_zvalue <chr>, `Wu-Hausman_df1` <chr>, `Wu-H
## #   `Wu-Hausman_statistic` <chr>
```

5.1.2 IV Loop over RHS

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

Regression with a Variety of Outcome Variables and Right Hand Side Variables. There are M outcome variables, and there are N alternative right hand side variables. Regress each M outcome variable and each N alternative right hand side variable, with some common sets of controls and perhaps shared instruments. The output file is a M by N matrix of coefficients, with proper variable names and row names. The matrix stores coefficients for this key endogenous variable.

- Dependency: *R4Econ/linreg/ivreg/ivregdfrow.R*

5.1.2.1 Construct Program

The program relies on double lapply. lapply is used for convenience, not speed.

```
ff_reg_mbyn <- function(list.vars.y, list.vars.x,
                        vars.c, vars.z, df,
                        return_all = FALSE,
                        stats_ends = 'value', time = FALSE) {

    # regf.iv() function is from C:\Users\fan\R4Econ\linreg\ivreg\ivregdfrow.R
    if (time) {
        start_time <- Sys.time()
    }

    if (return_all) {
        df.reg.out.all <- bind_rows(lapply(list.vars.x,
                              function(x) (
                                  bind_rows(lapply(list.vars.y, regf.iv, vars.x=x, vars.c=vars.c, va
                              )))

    } else {
        df.reg.out.all <- (lapply(list.vars.x,
                              function(x) (
                                  bind_rows(lapply(list.vars.y, regf.iv, vars.x=x, vars.c=vars.c, va
                                      select(vars_var.y, starts_with(x)) %>%
                                      select(vars_var.y, ends_with(stats_ends))
                              ))) %>% reduce(full_join)
    }

    if (time) {
        end_time <- Sys.time()
        print(paste0('Estimation for all ys and xs took (seconds):', end_time - start_time))
    }

    return(df.reg.out.all)
}
```

5.1.2.2 Prepare Data

```
# Library
library(tidyverse)
```

```
library(AER)

# Load Sample Data
setwd('C:/Users/fan/R4Econ/_data/')
df <- read_csv('height_weight.csv')
```

```
## Parsed with column specification:
## cols(
##   S.country = col_character(),
##   vil.id = col_double(),
##   indi.id = col_double(),
##   sex = col_character(),
##   svymthRound = col_double(),
##   momEdu = col_double(),
##   wealthIdx = col_double(),
##   hgt = col_double(),
##   wgt = col_double(),
##   hgt0 = col_double(),
##   wgt0 = col_double(),
##   prot = col_double(),
##   cal = col_double(),
##   p.A.prot = col_double(),
##   p.A.nProt = col_double()
## )
```

```
# Source Dependency
source('C:/Users/fan/R4Econ/linreg/ivreg/ivregdfrow.R')

# Setting
options(repr.matrix.max.rows=50, repr.matrix.max.cols=50)
```

Parameters.

```
var.y1 <- c('hgt')
var.y2 <- c('wgt')
var.y3 <- c('vil.id')
list.vars.y <- c(var.y1, var.y2, var.y3)

var.x1 <- c('prot')
var.x2 <- c('cal')
var.x3 <- c('wealthIdx')
var.x4 <- c('p.A.prot')
var.x5 <- c('p.A.nProt')
list.vars.x <- c(var.x1, var.x2, var.x3, var.x4, var.x5)

vars.z <- c('indi.id')
vars.c <- c('sex', 'wgt0', 'hgt0', 'svymthRound')
```

5.1.2.3 Program Testing

```
vars.z <- NULL
suppressMessages(ff_reg_mbyn(list.vars.y, list.vars.x,
                             vars.c, vars.z, df,
                             return_all = FALSE,
                             stats_ends = 'value'))
```

5.1.2.3.1 Test Program OLS Z-Stat

```
##   vars_var.y       prot_tvalue        cal_tvalue wealthIdx_tvalue  p.A.prot_tvalue p.A.nProt_tval
```

```
## 1        hgt  18.8756010031786  23.4421863484661  13.508899618216 3.83682180045518 32.54482575548
## 2        wgt  16.3591125056062  17.3686031309332 14.1390521528113 1.36958319982295 12.09615579114
## 3     vil.id -14.9385580468907 -19.6150110809452 34.0972558327347 8.45943342783186 17.78014224214
```

```
vars.z <- c('indi.id')
suppressMessages(ff_reg_mbyn(list.vars.y, list.vars.x,
                             vars.c, vars.z, df,
                             return_all = FALSE,
                             stats_ends = 'value'))
```

5.1.2.3.2 Test Program IV T-stat

```
## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

##   vars_var.y       prot_zvalue        cal_zvalue  wealthIdx_zvalue   p.A.prot_zvalue  p.A.nProt_z
```

```
## 1        hgt  8.87674929300964  12.0739764947235  4.62589553677969  26.6373587567312  32.11621923
## 2        wgt  5.60385871756365   6.1225187008946  5.17869536991717  11.9295584469998  12.35093070
## 3     vil.id -9.22106223347162 -13.0586007975839 -51.5866689219593 -29.9627476577329 -38.35288946
```

```
vars.z <- NULL
suppressMessages(ff_reg_mbyn(list.vars.y, list.vars.x,
                             vars.c, vars.z, df,
                             return_all = FALSE,
                             stats_ends = 'Estimate'))
```

5.1.2.3.3 Test Program OLS Coefficient

```
##   vars_var.y       prot_Estimate         cal_Estimate wealthIdx_Estimate    p.A.prot_Estimate  p.
## 1        hgt   0.049431093806755  0.00243408846205622   0.21045655488185 3.86952250259526e-05 0.0
## 2        wgt    16.5557424523585    0.699072500364623   106.678721085969  0.00521731297924587    0
## 3     vil.id -0.0758835879205584 -0.00395676177098486  0.451733304543324 0.000149388430455142 0.0
```

```
vars.z <- c('indi.id')
suppressMessages(ff_reg_mbyn(list.vars.y, list.vars.x,
                             vars.c, vars.z, df,
                             return_all = FALSE,
                             stats_ends = 'Estimate'))
```

5.1.2.3.4 Test Program IV coefficient

```
## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

##   vars_var.y     prot_Estimate       cal_Estimate wealthIdx_Estimate    p.A.prot_Estimate  p.A.nP
## 1        hgt 0.859205733632614 0.0238724384575419  0.144503490136948  0.00148073028434642  0.0141
## 2        wgt  98.9428234201406   2.71948246216953   69.1816142883022    0.221916473012486    2.11
## 3     vil.id -6.02451379136132 -0.168054407187466  -1.91414470908345 -0.00520794333267238 -0.0494
```

```
vars.z <- NULL
ff_reg_mbyn(list.vars.y, list.vars.x,
            vars.c, vars.z, df,
            return_all = TRUE,
            stats_ends = 'Estimate')
```

5.1.2.3.5 Test Program OLS Return All

```
##    X.Intercept._Estimate X.Intercept._Pr...t.. X.Intercept._Std.Error X.Intercept._tvalue    adj.
## 1       27.3528514188608 5.68247182214952e-231      0.831272666092284    32.9047886867776  0.8142
## 2        99.873884728925      0.75529705553815       320.450650378664    0.31166697465244   0.607
## 3       31.4646660224049  6.78164655340399e-84       1.61328519718754     19.503474077155 0.03732
## 4       27.9038445914729 8.24252673989353e-242      0.828072565159449    33.6973421962119   0.816
## 5       219.626705179399     0.493216914827181       320.522532223672   0.685214557790078  0.6078
## 6       30.5103987898551  1.62608789535248e-79       1.60831193651104    18.9704485163756 0.04534
## 7       35.7840188807906 2.26726906489443e-145       1.38461348429899    25.8440491058106  0.9350
## 8      -2662.74787734003  7.13318862990131e-05       670.301542938561   -3.97246270039407   0.921
## 9       29.2381039651127 1.53578035267873e-124       1.22602177264147    23.8479483950102  0.0595
## 10      23.9948407749744 2.11912344053336e-165      0.866581042166672    27.6890903532576  0.8146
## 11     -547.959546430028    0.0941551350855875       327.343126852912   -1.6739607509042  0.6173
## 12      22.3367814226238  3.04337266226599e-49       1.5098937308759     14.7936116071335 0.02611
## 13      24.4904444950827 2.34941965806705e-181      0.843371070670838    29.0387533397398  0.8245
## 14     -476.703973630552     0.143844033032183       326.132837036936   -1.46168652614567  0.6202
## 15      22.7781908464511  9.58029450711211e-52       1.5004526558957     15.1808794212527 0.03854
##              hgt0_Pr...t..     hgt0_Std.Error       hgt0_tvalue        prot_Estimate        prot_Pr.
## 1  1.14533314566771e-183 0.0206657538633713  29.2231378249683   0.049431093806755 9.5476932230464
## 2    1.52417506966835e-12    7.96735224000553   7.0770314931977    16.5557424523585 9.6120337322218
## 3    1.40290395213743e-13 0.0401060913799595 -7.40147890309685 -0.0758835879205584 3.5639609356233
## 4  7.79174951119325e-177 0.0205836398278421  28.6561486875877                 <NA>
## 5    3.05720143843395e-11    7.96822145797115  6.64774497790599                 <NA>
## 6    8.49149153665126e-12 0.0399777363511633 -6.83428417151858                 <NA>
## 7    2.71000479249152e-36 0.0348701896610764  12.6002885423502                 <NA>
## 8     0.00520266507060071    16.8823489375743  2.79445531182864                 <NA>
## 9    2.41020063623865e-31 0.0307984635553859  -11.659076407325                 <NA>
## 10 1.31914432912869e-220 0.0213841849324282  32.1391351404584                 <NA>
## 11   4.78613024244006e-19    8.07744906400683  8.92677379355593                 <NA>
## 12    0.0034801146146182 0.0372288594891345 -2.92217281443323                 <NA>
## 13 1.11511327164938e-190 0.0208846437570215  29.8015803204665                 <NA>
## 14   8.38546282719268e-15    8.07589192978212  7.76801157994423                 <NA>
## 15   2.13723119924676e-05 0.0371223237183417 -4.25112470577158                 <NA>
##           r.squared_v    sexMale_Estimate     sexMale_Pr...t..  sexMale_Std.Error    sexMale_tvalu
```

```
## 1   0.814298005954592  0.935177182449406 2.36432111724607e-51 0.0618482294097262  15.120516648166
## 2   0.607272921412825   415.163616765357 2.48252880290814e-67   23.8518341439675  17.405940954455
## 3  0.0375780335372857 -0.254089999175318   0.0343768259467621  0.120093045309631 -2.1157761344148
## 4   0.816137722617266  0.893484662055608 2.08765935335877e-47 0.0616078355613525  14.502776374375
## 5    0.60796705182314   405.534891838028 2.51355675686752e-64   23.8567507583516  16.998747899315
## 6  0.0456010419476623 -0.181389489610951    0.129768754080748   0.11972270545355 -1.5150801088547
## 7    0.93502787877066   1.80682463132073 1.26527362032354e-66  0.104475287357902  17.294277690101
## 8   0.921952383432195   999.926876716707 2.64630894140004e-86   50.5879876531386  19.766093159759
## 9  0.0596997716363463  -0.33436777751525 0.000311174554787706 0.0927193334338799 -3.6062357777161
## 10  0.814740639193486  0.932686930233136 7.90489020586094e-47 0.0647209948973267  14.410886787397
## 11  0.617403496088206   397.141948675354 6.19449742677662e-59   24.4473730956481  16.244769821345
## 12 0.0263714328556815 -0.445232370681998 7.93666802281971e-05  0.112797805327952 -3.9471722821868
## 13  0.824589538985803   0.96466980500711 1.24556615236597e-52 0.0629827627260302   15.31640981205
## 14  0.620352835549783    401.59056368102 1.18469030741261e-60   24.3549086073387  16.489101649102
## 15 0.0387987636986586 -0.423829627017582  0.00015644693636154  0.112083516545945 -3.7813733908308
##    svymthRound_Pr...t.. svymthRound_Std.Error svymthRound_tvalue vars_var.y            vars_va
## 1                     0   0.00387681209575621   224.840892330022        hgt sex+wgt0+hgt0+svymthR
## 2                     0       1.4955473831309   126.403823119306        wgt sex+wgt0+hgt0+svymthR
## 3    0.0397984032097113   0.00752730297891317  -2.05597660181154     vil.id sex+wgt0+hgt0+svymthR
## 4                     0   0.00411253488213795   207.168832400006        hgt sex+wgt0+hgt0+svymthR
## 5                     0      1.59266949679221   116.357025971267        wgt sex+wgt0+hgt0+svymthR
## 6    0.0117151185126433   0.00799217807522278   2.52085521254888     vil.id sex+wgt0+hgt0+svymthR
## 7                     0  0.000728323735328998   594.262183761197        hgt sex+wgt0+hgt0+svymthR
## 8                     0     0.352701518968252   538.353209678558        wgt sex+wgt0+hgt0+svymthR
## 9  0.000447277200167272  0.000612792699568233   3.51088227277012     vil.id sex+wgt0+hgt0+svymthR
## 10                    0   0.00331108017589107   277.738571133786        hgt sex+wgt0+hgt0+svymthR
## 11                    0      1.25083486490652   164.368128386085        wgt sex+wgt0+hgt0+svymthR
## 12 1.37139389802397e-18   0.00578476859618168  -8.80889965139067     vil.id sex+wgt0+hgt0+svymthR
## 13                    0   0.00317113547025635   290.714194782148        hgt sex+wgt0+hgt0+svymthR
## 14                    0      1.22639878616071   167.926734460268        wgt sex+wgt0+hgt0+svymthR
## 15 7.79141497751766e-23   0.00565696328562864  -9.84988636256528     vil.id sex+wgt0+hgt0+svymthR
##           wgt0_Pr...t..       wgt0_Std.Error       wgt0_tvalue         cal_Estimate          cal_
## 1     0.136011583497549 9.79994437486573e-05 -1.49087260496811                 <NA>
## 2  2.96480083692757e-63   0.0378027371614794  16.8512547316329                 <NA>
## 3  2.05763549729273e-06 0.000190221503167431 -4.74915073475531                 <NA>
## 4     0.230228828649018 9.74307633896921e-05 -1.19980821193398  0.00243408846205622 8.01672708877
## 5  7.43034302413852e-66    0.037739875283113  17.2071051836606    0.699072500364623  4.7133190088
## 6  6.66901196231733e-07 0.000189270503626621 -4.97244448929308 -0.00395676177098486  7.9464612402
## 7  1.22269348058816e-13 0.000164767846917989  7.41843614592224                 <NA>
## 8  6.75367630221077e-62   0.0798131859486402  16.6477281392748                 <NA>
## 9  4.32675510884621e-09 0.000144040382619518   -5.872926128913                 <NA>
## 10 7.77000489086602e-07 9.90410500454311e-05 -4.94274682926991                 <NA>
## 11 7.42419220783427e-54   0.0374185042114355  15.5009805428138                 <NA>
## 12 1.40362012201826e-19 0.000172365145002826  -9.0619777654873                 <NA>
## 13    0.740027016459552 9.75208524392668e-05 0.331822524275644                 <NA>
## 14 4.09082062947785e-67   0.0377202854835204  17.3782370584956                 <NA>
## 15 2.75472781728448e-11 0.000173241059789276 -6.66312732777158                 <NA>
##    wealthIdx_Estimate    wealthIdx_Pr...t.. wealthIdx_Std.Error wealthIdx_tvalue    p.A.prot_Esti
## 1                <NA>                  <NA>                <NA>             <NA>
## 2                <NA>                  <NA>                <NA>             <NA>
## 3                <NA>                  <NA>                <NA>             <NA>
## 4                <NA>                  <NA>                <NA>             <NA>
## 5                <NA>                  <NA>                <NA>             <NA>
## 6                <NA>                  <NA>                <NA>             <NA>
## 7    0.21045655488185  1.93494257274268e-41  0.0155791042075745  13.508899618216
## 8    106.678721085969   3.2548345535026e-45    7.54496977117083 14.1390521528113
## 9   0.451733304543324 4.82890644822007e-250  0.0132483771350785 34.0972558327347
## 10               <NA>                  <NA>                <NA>             <NA> 3.86952250259526
```

```
## 11              <NA>                 <NA>               <NA>            <NA>  0.0052173129792
## 12              <NA>                 <NA>               <NA>            <NA> 0.00014938843045
## 13              <NA>                 <NA>               <NA>            <NA>
## 14              <NA>                 <NA>               <NA>            <NA>
## 15              <NA>                 <NA>               <NA>            <NA>
##     p.A.prot_tvalue  p.A.nProt_Estimate    p.A.nProt_Pr...t..  p.A.nProt_Std.Error p.A.nProt_tval
## 1              <NA>                 <NA>                  <NA>                 <NA>             <N
## 2              <NA>                 <NA>                  <NA>                 <NA>             <N
## 3              <NA>                 <NA>                  <NA>                 <NA>             <N
## 4              <NA>                 <NA>                  <NA>                 <NA>             <N
## 5              <NA>                 <NA>                  <NA>                 <NA>             <N
## 6              <NA>                 <NA>                  <NA>                 <NA>             <N
## 7              <NA>                 <NA>                  <NA>                 <NA>             <N
## 8              <NA>                 <NA>                  <NA>                 <NA>             <N
## 9              <NA>                 <NA>                  <NA>                 <NA>             <N
## 10 3.83682180045518                 <NA>                  <NA>                 <NA>             <N
## 11 1.36958319982295                 <NA>                  <NA>                 <NA>             <N
## 12 8.45943342783186                 <NA>                  <NA>                 <NA>             <N
## 13             <NA> 0.00542428867316449 5.25341325077391e-226 0.000166671307872964 32.54482575548
## 14             <NA>   0.779514232050632  1.47950939943836e-33     0.06444313759758 12.09615579114
## 15             <NA> 0.00526237555581024   3.7685780281174e-70 0.000295969260771016 17.78014224214
```

```r
vars.z <- c('indi.id')
ff_reg_mbyn(list.vars.y, list.vars.x,
            vars.c, vars.z, df,
            return_all = TRUE,
            stats_ends = 'Estimate')
```

5.1.2.3.6 Test Program IV Return All

```
## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
```

```
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped
##    X.Intercept._Estimate X.Intercept._Pr...z.. X.Intercept._Std.Error X.Intercept._zvalue     hgt
## 1       40.2173991882938  3.69748206920405e-59       2.47963650430699    16.2190704639323 0.40313
## 2        1408.1626637032   0.00217397545504963       459.377029874119    3.06537456626657  35.576
## 3       -64.490636067872  0.000109756271656929        16.673099250727   -3.86794531107106  1.2099
## 4       39.6732302990235 1.30030240177373e-103       1.83545587849039    21.6149190857443 0.35797
## 5       1325.54736576331   0.00138952700443324       414.645900526211    3.19681772828602  31.017
## 6      -59.8304089440729  3.75547414421179e-07       11.7754321198995   -5.08095230263053   1.503
## 7       35.5561817357046 2.01357089467444e-142       1.39936229104453    25.4088465605032 0.46043
## 8        -2791.221534909  1.95034793045284e-05       653.605248808641   -4.27050048939585  59.154
## 9       21.8005242861645  1.17899313785408e-34       1.77547715237629    12.2786847788984 0.41251
## 10      24.3009261707644  1.97968607369592e-84        1.2481331128579    19.4698193008609 0.51579
## 11     -499.067024090554      0.155922992163314       351.723712333143   -1.41891776582254  46.259
## 12      21.4632286881661  1.84405333738942e-09       3.57067054655531    6.01097984491234 0.52081
## 13       25.299209739617 1.29388565624566e-157      0.945826571474308     26.748254386829 0.51086
## 14     -352.278518334717      0.287184942021997       330.990098562619    -1.0643173915611  45.565
## 15      17.9359211844992  1.13855583530306e-12       2.52170174723203    7.11262590993832 0.53436
##          hgt0_zvalue     prot_Estimate        prot_Pr...z..    prot_Std.Error       prot_zvalue S
## 1  7.41136089709158 0.859205733632614 6.88427338202428e-19 0.0967928354481331  8.87674929300964
## 2  3.51137048180512  98.9428234201406 2.09631602352917e-08   17.6561952052848  5.60385871756365
## 3  3.29876072644971 -6.02451379136132 2.94171378745816e-20  0.653342710289155 -9.22106223347162
## 4  8.45373003027063              <NA>                 <NA>               <NA>              <NA>
## 5  3.21377335801252              <NA>                 <NA>               <NA>              <NA>
## 6  5.50460248701607              <NA>                 <NA>               <NA>              <NA>
## 7  12.7533216258548              <NA>                 <NA>               <NA>              <NA>
## 8  3.45880859967647              <NA>                 <NA>               <NA>              <NA>
## 9  9.21816552325528              <NA>                 <NA>               <NA>              <NA>
## 10 16.1673191711084              <NA>                 <NA>               <NA>              <NA>
## 11 5.13270005180026              <NA>                 <NA>               <NA>              <NA>
## 12 5.71448149208973              <NA>                 <NA>               <NA>              <NA>
## 13 21.4658243761363              <NA>                 <NA>               <NA>              <NA>
## 14 5.40878275196011              <NA>                 <NA>               <NA>              <NA>
## 15  8.4310762436216              <NA>                 <NA>               <NA>              <NA>
##     sexMale_Std.Error    sexMale_zvalue svymthRound_Estimate  svymthRound_Pr...z.. svymthRound_St
## 1   0.178475271469781  0.86310792817082     0.20990165085783   0.00846239710392287    0.079718317
## 2    33.0216035385405  10.1085242471545      121.78985943172  5.96047652813855e-17      14.557708
## 3    1.19371921154418  4.53352366774387     4.84745570027424  2.07373887977152e-19     0.53805014
## 4   0.132821186086547 0.800381017440976    0.322893837128574  9.66146445882893e-11    0.049889691
## 5    30.5174257711927  10.8283251459136     135.494858749214  4.48931446042076e-34       11.13348
## 6   0.847955715223327  6.87676174970095     4.07024693316581  5.64723572160763e-36     0.32504334
## 7   0.105343525210948   17.113904962338    0.433164820953121                     0   0.0012047281
## 8    49.7632792630648  20.0498764266063      190.07735139541                     0     0.73926987
## 9   0.132754263303719 -3.41102322376347   0.0137438264666969  1.57416908709431e-66  0.00079765593
```

```
## 10 0.0945646985181925  10.8646912458831     1.00582859923509                     0   0.0074686771
## 11   26.4822313532216  15.5336574870174     218.549980922774                     0       1.931571
## 12  0.276250047248363 -2.85655126226267   -0.369567838754916 2.42696379701225e-102    0.017205698
## 13 0.0675715533063635  15.0964658352764    0.929266902426869                     0   0.0053933063
## 14   24.5920104216267  16.6647907361992     207.078222946319                     0      1.4616785
## 15   0.18692145837209 -3.99115565898846  -0.0985678389223824  1.84569897952709e-27   0.0090786748
##                   vars_vars.c vars_vars.x vars_vars.z Weakinstruments_df1 Weakinstruments_df2 Weak
## 1  sex+wgt0+hgt0+svymthRound        prot     indi.id                   1               18957    1
## 2  sex+wgt0+hgt0+svymthRound        prot     indi.id                   1               18962    4
## 3  sex+wgt0+hgt0+svymthRound        prot     indi.id                   1               18999    5
## 4  sex+wgt0+hgt0+svymthRound         cal     indi.id                   1               18957    1
## 5  sex+wgt0+hgt0+svymthRound         cal     indi.id                   1               18962    4
## 6  sex+wgt0+hgt0+svymthRound         cal     indi.id                   1               18999    5
## 7  sex+wgt0+hgt0+svymthRound   wealthIdx     indi.id                   1               25092
## 8  sex+wgt0+hgt0+svymthRound   wealthIdx     indi.id                   1               25102
## 9  sex+wgt0+hgt0+svymthRound   wealthIdx     indi.id                   1               30013
## 10 sex+wgt0+hgt0+svymthRound    p.A.prot     indi.id                   1               18587
## 11 sex+wgt0+hgt0+svymthRound    p.A.prot     indi.id                   1               18591
## 12 sex+wgt0+hgt0+svymthRound    p.A.prot     indi.id                   1               18845
## 13 sex+wgt0+hgt0+svymthRound   p.A.nProt     indi.id                   1               18587
## 14 sex+wgt0+hgt0+svymthRound   p.A.nProt     indi.id                   1               18591
## 15 sex+wgt0+hgt0+svymthRound   p.A.nProt     indi.id                   1               18845
##             wgt0_Estimate        wgt0_Pr...z..       wgt0_Std.Error       wgt0_zvalue Wu.Hausman_
## 1   -0.00163274724538111  4.88365163639597e-08  0.00029928487659495 -5.45549532591606
## 2      0.492582112313709  2.33136555228405e-20   0.0532753838702833  9.24596082710666
## 3    0.00999798623641602  7.95432753711715e-07  0.00202532507408065  4.93648469787221
## 4  -0.000658938519302931   0.00032843149807424 0.000183457551985601 -3.59177647456371
## 5      0.601258436431587   2.0921134733036e-48   0.0411255751282477  14.6200614716414
## 6    0.00326074237566435   0.00667886646012294  0.00120214094164169  2.71244598924594
## 7    0.00112485055604169  2.26123807446765e-11 0.000168187467853553  6.68807593334564
## 8       1.27282038539707  6.67525280062144e-56     0.08080475140115  15.7518012657231
## 9   -0.00512158791392237 6.51923753120087e-127 0.000213715312589078 -23.9645341827701
## 10  0.000716628918444932  2.43477572076212e-06 0.000152036990658929  4.71351685756907
## 11     0.761704518610475  8.2201479288098e-69   0.0434474820359048   17.531614789115
## 12  -0.00601345031606092  5.19751747217521e-44  0.00043218241369976 -13.9141485757875
## 13  0.000922100117259348  1.68237436753105e-15  0.00011580150512068  7.96276452796019
## 14     0.792700893714085  4.81415543564975e-82   0.0413159097814445  19.1863351892132
## 15  -0.00668277875606482 2.54848840100353e-105 0.000306609919182859 -21.7957030675165
##    Wu.Hausman_statistic       cal_Estimate          cal_Pr...z..       cal_Std.Error        cal_zv
## 1      543.467268879953               <NA>                  <NA>                <NA>
## 2      30.6481856102772               <NA>                  <NA>                <NA>
## 3      5652.51924792859               <NA>                  <NA>                <NA>
## 4      494.955883488045 0.0238724384575419 1.44956616452661e-33 0.00197718112735887  12.073976494
## 5      24.4605456760994   2.71948246216953 9.21076021290446e-10   0.444177077282291   6.122518700
## 6      5583.56513052781 -0.168054407187466 5.67614501764414e-39  0.0128692506794877 -13.058600797
## 7      5.23078768861684               <NA>                  <NA>                <NA>
## 8       6.6473469952822               <NA>                  <NA>                <NA>
## 9      25949.7118056025               <NA>                  <NA>                <NA>
## 10     1119.87022468742               <NA>                  <NA>                <NA>
## 11     154.793296861581               <NA>                  <NA>                <NA>
## 12     4826.92242730041               <NA>                  <NA>                <NA>
## 13     494.903094649183               <NA>                  <NA>                <NA>
## 14      72.530787010352               <NA>                  <NA>                <NA>
## 15     7607.83405438193               <NA>                  <NA>                <NA>
##    wealthIdx_Std.Error  wealthIdx_zvalue    p.A.prot_Estimate     p.A.prot_Pr...z..   p.A.prot_St
## 1                 <NA>              <NA>                 <NA>                  <NA>
## 2                 <NA>              <NA>                 <NA>                  <NA>
## 3                 <NA>              <NA>                 <NA>                  <NA>
```

```
## 4                <NA>               <NA>                 <NA>                  <NA>
## 5                <NA>               <NA>                 <NA>                  <NA>
## 6                <NA>               <NA>                 <NA>                  <NA>
## 7   0.0312379492766376   4.62589553677969                 <NA>                  <NA>
## 8      13.358888551386   5.17869536991717                 <NA>                  <NA>
## 9   0.0371054140359243 -51.5866689219593                 <NA>                  <NA>
## 10               <NA>               <NA>  0.00148073028434642 2.50759287066563e-156 5.55884799941
## 11               <NA>               <NA>     0.221916473012486  8.30126393398654e-33    0.018602236
## 12               <NA>               <NA> -0.00520794333267238 3.00201194005694e-197 0.00017381394
## 13               <NA>               <NA>                 <NA>                  <NA>
## 14               <NA>               <NA>                 <NA>                  <NA>
## 15               <NA>               <NA>                 <NA>                  <NA>
##       p.A.nProt_Pr...z..  p.A.nProt_Std.Error  p.A.nProt_zvalue
## 1                   <NA>                 <NA>              <NA>
## 2                   <NA>                 <NA>              <NA>
## 3                   <NA>                 <NA>              <NA>
## 4                   <NA>                 <NA>              <NA>
## 5                   <NA>                 <NA>              <NA>
## 6                   <NA>                 <NA>              <NA>
## 7                   <NA>                 <NA>              <NA>
## 8                   <NA>                 <NA>              <NA>
## 9                   <NA>                 <NA>              <NA>
## 10                  <NA>                 <NA>              <NA>
## 11                  <NA>                 <NA>              <NA>
## 12                  <NA>                 <NA>              <NA>
## 13 2.61782083774363e-226 0.000440019589949091  32.1162192385744
## 14  4.81511329043196e-35     0.17153115470458  12.3509307017263
## 15                     0  0.00128926108222202 -38.3528894620707
```

5.1.2.4 Program Line by Line

Set Up Parameters

```
vars.z <- c('indi.id')
vars.z <- NULL
vars.c <- c('sex', 'wgt0', 'hgt0', 'svymthRound')
```

```

df.reg.out <- as_tibble(bind_rows(lapply(list.vars.y, regf.iv, vars.x=var.x1, vars.c=vars.c, vars.z=
```

5.1.2.4.1 Lapply

```

lapply(list.vars.y, function(y) (mean(df[[var.x1]], na.rm=TRUE) + mean(df[[y]], na.rm=TRUE)))
```

5.1.2.4.2 Nested Lapply Test

```
## [[1]]
## [1] 98.3272
##
## [[2]]
## [1] 13626.51
##
## [[3]]
## [1] 26.11226
```

```
lapplytwice <- lapply(list.vars.x, function(x) (lapply(list.vars.y, function(y) (mean(df[[x]], na.rm
lapplytwice
```

```
## [[1]]
```

```
## [[1]][[1]]
## [1] 98.3272
##
## [[1]][[2]]
## [1] 13626.51
##
## [[1]][[3]]
## [1] 26.11226
##
##
## [[2]]
## [[2]][[1]]
## [1] 525.4708
##
## [[2]][[2]]
## [1] 14053.65
##
## [[2]][[3]]
## [1] 453.2558
##
##
## [[3]]
## [[3]][[1]]
## [1] 90.69287
##
## [[3]][[2]]
## [1] 13618.87
##
## [[3]][[3]]
## [1] 18.47793
##
##
## [[4]]
## [[4]][[1]]
## [1] 2095.3
##
## [[4]][[2]]
## [1] 15623.48
##
## [[4]][[3]]
## [1] 2023.085
##
##
## [[5]]
## [[5]][[1]]
## [1] 271.2886
##
## [[5]][[2]]
## [1] 13799.47
##
## [[5]][[3]]
## [1] 199.0737
```

```
df.reg.out.all <- bind_rows(lapply(list.vars.x,
                    function(x) (
                        bind_rows(lapply(list.vars.y, regf.iv, vars.x=x, vars.c=vars.c, vars.z=var
                    )))
```

```
df.reg.out.all
```

5.1.2.4.3 Nested Lapply All

```
##    X.Intercept._Estimate X.Intercept._Pr...t.. X.Intercept._Std.Error X.Intercept._tvalue    adj.
## 1       27.3528514188608 5.68247182214952e-231      0.831272666092284    32.9047886867776  0.8142
## 2        99.873884728925      0.75529705553815       320.450650378664    0.31166697465244   0.607
## 3       31.4646660224049  6.78164655340399e-84       1.61328519718754     19.503474077155 0.03732
## 4       27.9038445914729 8.24252673989353e-242      0.828072565159449    33.6973421962119   0.816
## 5       219.626705179399     0.493216914827181       320.522532223672   0.685214557790078  0.6078
## 6       30.5103987898551  1.62608789535248e-79       1.60831193651104    18.9704485163756 0.04534
## 7       35.7840188807906 2.26726906489443e-145       1.38461348429899    25.8440491058106  0.9350
## 8      -2662.74787734003  7.13318862990131e-05       670.301542938561   -3.97246270039407   0.921
## 9       29.2381039651127 1.53578035267873e-124       1.22602177264147    23.8479483950102  0.0595
## 10      23.9948407749744 2.11912344053336e-165      0.86658104216672     27.6890903532576  0.8146
## 11     -547.959546430028    0.0941551350855875       327.343126852912    -1.6739607509042  0.6173
## 12      22.3367814226238  3.04337266226599e-49       1.5098937308759     14.7936116071335 0.02611
## 13      24.4904444950827 2.34941965806705e-181      0.843371070670838    29.0387533397398  0.8245
## 14     -476.703973630552     0.143844033032183       326.132837036936   -1.46168652614567  0.6202
## 15      22.7781908464511  9.58029450711211e-52       1.5004526558957     15.1808794212527 0.03854
##            hgt0_Pr...t..     hgt0_Std.Error       hgt0_tvalue       prot_Estimate         prot_Pr.
## 1  1.14533314566771e-183 0.0206657538633713  29.2231378249683   0.049431093806755 9.5476932230464
## 2   1.52417506966835e-12   7.96735224000553   7.0770314931977    16.5557424523585 9.6120337322218
## 3   1.40290395213743e-13 0.0401060913799595 -7.40147890309685 -0.0758835879205584 3.5639609356233
## 4  7.79174951119325e-177 0.0205836398278421  28.6561486875877                <NA>
## 5   3.05720143843395e-11   7.96822145797115  6.64774497790599                <NA>
## 6   8.49149153665126e-12 0.0399777363511633 -6.83428417151858                <NA>
## 7   2.71000479249152e-36 0.0348701896610764  12.6002885423502                <NA>
## 8    0.00520266507060071   16.8823489375743  2.79445531182864                <NA>
## 9   2.41020063623865e-31 0.0307984635553859  -11.659076407325                <NA>
## 10 1.31914432912869e-220 0.0213841849324282  32.1391351404584                <NA>
## 11  4.78613024244006e-19   8.07744906400683  8.92677379355593                <NA>
## 12    0.0034801146146182 0.0372288594891345 -2.92217281443323                <NA>
## 13 1.11511327164938e-190 0.0208846437570215  29.8015803204665                <NA>
## 14  8.38546282719268e-15   8.07589192978212  7.76801157994423                <NA>
## 15  2.13723119924676e-05 0.0371223237183417 -4.25112470577158                <NA>
##            r.squared_v   sexMale_Estimate     sexMale_Pr...t..  sexMale_Std.Error    sexMale_tvalu
## 1   0.814298005954592  0.935177182449406 2.36432111724607e-51 0.0618482294097262  15.120516648166
## 2   0.607272921412825   415.163616765357 2.48252880290814e-67   23.8518341439675  17.405940954455
## 3  0.0375780335372857 -0.254089999175318   0.0343768259467621  0.120093045309631 -2.1157761344148
## 4   0.816137722617266  0.893484662055608 2.08765935335877e-47 0.0616078355613525  14.502776374375
## 5    0.60796705182314   405.534891838028 2.51355675686752e-64   23.8567507583516  16.998747899315
## 6  0.0456010419476623 -0.181389489610951    0.129768754080748   0.11972270545355 -1.5150801088547
## 7    0.93502787877066   1.80682463132073 1.26527362032354e-66  0.104475287357902  17.294277690101
## 8   0.921952383432195   999.926876716707 2.64630894140004e-86   50.5879876531386  19.766093159759
## 9  0.0596997716363463  -0.33436777751525 0.000311174554787706 0.0927193334338799 -3.6062357777161
## 10  0.814740639193486  0.932686930233136 7.90489020586094e-47 0.0647209948973267  14.410886787397
## 11  0.617403496088206   397.141948675354 6.19449742677662e-59   24.4473730956481  16.244769821345
## 12 0.0263714328556815 -0.445232370681998 7.93666802281971e-05  0.112797805327952 -3.9471722821868
## 13  0.824589538985803   0.96466980500711 1.24556615236597e-52 0.0629827627260302   15.31640981205
## 14  0.620352835549783    401.59056368102 1.18469030741261e-60   24.3549086073387  16.489101649102
## 15 0.0387987636986586 -0.423829627017582  0.00015644693636154  0.112083516545945 -3.7813733908308
##    svymthRound_Pr...t.. svymthRound_Std.Error svymthRound_tvalue vars_var.y             vars_va
## 1                     0    0.00387681209575621   224.840892330022        hgt sex+wgt0+hgt0+svymthR
## 2                     0        1.4955473831309   126.403823119306        wgt sex+wgt0+hgt0+svymthR
## 3    0.0397984032097113    0.00752730297891317  -2.05597660181154     vil.id sex+wgt0+hgt0+svymthR
## 4                     0    0.00411253488213795   207.168832400006        hgt sex+wgt0+hgt0+svymthR
```

```
## 5                     0      1.59266949679221   116.357025971267        wgt sex+wgt0+hgt0+svymthR
## 6    0.0117151185126433   0.00799217807522278   2.52085521254888     vil.id sex+wgt0+hgt0+svymthR
## 7                     0  0.000728323735328998   594.262183761197        hgt sex+wgt0+hgt0+svymthR
## 8                     0     0.352701518968252   538.353209678558        wgt sex+wgt0+hgt0+svymthR
## 9  0.000447277200167272  0.000612792699568233   3.51088227277012     vil.id sex+wgt0+hgt0+svymthR
## 10                    0   0.00331108017589107   277.738571133786        hgt sex+wgt0+hgt0+svymthR
## 11                    0      1.25083486490652   164.368128386085        wgt sex+wgt0+hgt0+svymthR
## 12 1.37139389802397e-18   0.00578476859618168  -8.80889965139067     vil.id sex+wgt0+hgt0+svymthR
## 13                    0   0.00317113547025635   290.714194782148        hgt sex+wgt0+hgt0+svymthR
## 14                    0      1.22639878616071   167.926734460268        wgt sex+wgt0+hgt0+svymthR
## 15 7.79141497751766e-23   0.00565696328562864  -9.84988636256528     vil.id sex+wgt0+hgt0+svymthR
##           wgt0_Pr...t..        wgt0_Std.Error       wgt0_tvalue         cal_Estimate         cal_
## 1     0.136011583497549 9.79994437486573e-05 -1.49087260496811                 <NA>
## 2  2.96480083692757e-63   0.0378027371614794  16.8512547316329                 <NA>
## 3  2.05763549729273e-06 0.000190221503167431 -4.74915073475531                 <NA>
## 4     0.230228828649018 9.74307633896921e-05 -1.19980821193398  0.00243408846205622 8.01672708877
## 5  7.43034302413852e-66    0.037739875283113  17.2071051836606    0.699072500364623  4.7133190088
## 6  6.66901196231733e-07 0.000189270503626621 -4.97244448929308 -0.00395676177098486  7.9464612402
## 7  1.22269348058816e-13 0.000164767846917989  7.41843614592224                 <NA>
## 8  6.75367630221077e-62   0.0798131859486402  16.6477281392748                 <NA>
## 9  4.32675510884621e-09 0.000144040382619518   -5.872926128913                 <NA>
## 10 7.77000489086602e-07 9.90410500454311e-05 -4.94274682926991                 <NA>
## 11 7.42419220783427e-54   0.0374185042114355  15.5009805428138                 <NA>
## 12 1.40362012201826e-19 0.000172365145002826  -9.0619777654873                 <NA>
## 13    0.740027016459552 9.75208524392668e-05 0.331822524275644                 <NA>
## 14 4.09082062947785e-67   0.0377202854835204  17.3782370584956                 <NA>
## 15 2.75472781728448e-11 0.000173241059789276 -6.66312732777158                 <NA>
##    wealthIdx_Estimate    wealthIdx_Pr...t.. wealthIdx_Std.Error wealthIdx_tvalue    p.A.prot_Esti
## 1                <NA>                  <NA>                <NA>             <NA>
## 2                <NA>                  <NA>                <NA>             <NA>
## 3                <NA>                  <NA>                <NA>             <NA>
## 4                <NA>                  <NA>                <NA>             <NA>
## 5                <NA>                  <NA>                <NA>             <NA>
## 6                <NA>                  <NA>                <NA>             <NA>
## 7    0.21045655488185  1.93494257274268e-41  0.0155791042075745  13.508899618216
## 8    106.678721085969   3.2548345535026e-45    7.54496977117083 14.1390521528113
## 9   0.451733304543324 4.82890644822007e-250  0.0132483771350785 34.0972558327347
## 10               <NA>                  <NA>                <NA>             <NA> 3.86952250259526
## 11               <NA>                  <NA>                <NA>             <NA>  0.0052173129792
## 12               <NA>                  <NA>                <NA>             <NA> 0.00014938843045
## 13               <NA>                  <NA>                <NA>             <NA>
## 14               <NA>                  <NA>                <NA>             <NA>
## 15               <NA>                  <NA>                <NA>             <NA>
##     p.A.prot_tvalue  p.A.nProt_Estimate    p.A.nProt_Pr...t..  p.A.nProt_Std.Error p.A.nProt_tval
## 1              <NA>                <NA>                  <NA>                 <NA>             <N
## 2              <NA>                <NA>                  <NA>                 <NA>             <N
## 3              <NA>                <NA>                  <NA>                 <NA>             <N
## 4              <NA>                <NA>                  <NA>                 <NA>             <N
## 5              <NA>                <NA>                  <NA>                 <NA>             <N
## 6              <NA>                <NA>                  <NA>                 <NA>             <N
## 7              <NA>                <NA>                  <NA>                 <NA>             <N
## 8              <NA>                <NA>                  <NA>                 <NA>             <N
## 9              <NA>                <NA>                  <NA>                 <NA>             <N
## 10 3.83682180045518                <NA>                  <NA>                 <NA>             <N
## 11 1.36958319982295                <NA>                  <NA>                 <NA>             <N
## 12 8.45943342783186                <NA>                  <NA>                 <NA>             <N
## 13             <NA> 0.00542428867316449 5.25341325077391e-226 0.000166671307872964 32.54482575548
## 14             <NA>   0.779514232050632  1.47950939943836e-33     0.06444313759758 12.09615579114
```

```
## 15           <NA> 0.00526237555581024   3.7685780281174e-70 0.000295969260771016 17.78014224214
```

```
df.reg.out.all <- (lapply(list.vars.x,
                      function(x) (
                          bind_rows(lapply(list.vars.y, regf.iv, vars.x=x, vars.c=vars.c, vars.z=var
                              select(vars_var.y, starts_with(x)) %>%
                              select(vars_var.y, ends_with('value'))
                      ))) %>% reduce(full_join)
```

5.1.2.4.4 Nested Lapply Select

```
## Joining, by = "vars_var.y"Joining, by = "vars_var.y"Joining, by = "vars_var.y"Joining, by = "vars
```

```
df.reg.out.all
```

```
##   vars_var.y       prot_tvalue        cal_tvalue wealthIdx_tvalue  p.A.prot_tvalue p.A.nProt_tval
## 1        hgt  18.8756010031786  23.4421863484661  13.508899618216 3.83682180045518 32.54482575548
## 2        wgt  16.3591125056062  17.3686031309332 14.1390521528113 1.36958319982295 12.09615579114
## 3     vil.id -14.9385580468907 -19.6150110809452 34.0972558327347 8.45943342783186 17.78014224214
```

5.2 Decomposition

5.2.1 Decompose RHS

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

One runs a number of regressions. With different outcomes, and various right hand side variables.

What is the remaining variation in the left hand side variable if right hand side variable one by one is set to the average of the observed values.

- Dependency: *R4Econ/linreg/ivreg/ivregdfrow.R*

The code below does not work with categorical variables (except for dummies). Dummy variable inputs need to be converted to zero/one first.

5.2.1.1 Decomposition Program

```
ff_lr_decompose <- function(df, vars.y, vars.x, vars.c, vars.z, vars.other.keep,
                            list.vars.tomean, list.vars.tomean.name.suffix,
                            df.reg.out = NULL,
                            graph=FALSE, graph.nrow=2) {

    vars.xc <- c(vars.x, vars.c)

    # Regressions
    # regf.iv from C:\Users\fan\R4Econ\linreg\ivreg\ivregdfrow.R
    if(is.null(df.reg.out)) {
      df.reg.out <- as_tibble(
        bind_rows(lapply(vars.y, regf.iv,
                         vars.x=vars.x, vars.c=vars.c, vars.z=vars.z, df=df)))
    }

    # Select Variables
    str.esti.suffix <- '_Estimate'
    arr.esti.name <- paste0(vars.xc, str.esti.suffix)
    str.outcome.name <- 'vars_var.y'
    arr.columns2select <- c(arr.esti.name, str.outcome.name)
    # arr.columns2select
```

```
  # Generate dataframe for coefficients
  df.coef <- df.reg.out[,c(arr.columns2select)] %>%
    mutate_at(vars(arr.esti.name), as.numeric) %>% column_to_rownames(str.outcome.name)
  # df.coef
  # str(df.coef)

  # Decomposition Step 1: gather
  df.decompose <- df %>%
    filter(svymthRound %in% c(12, 18, 24)) %>%
    select(one_of(c(vars.other.keep, vars.xc, vars.y))) %>%
    drop_na() %>%
    gather(variable, value, -one_of(c(vars.other.keep, vars.xc)))

  # Decomposition Step 2: mutate_at(vars, funs(mean = mean(.)))
  # the xc averaging could have taken place earlier, no difference in mean across variables
  df.decompose <- df.decompose %>%
    group_by(variable) %>%
    mutate_at(vars(c(vars.xc, 'value')), funs(mean = mean(.))) %>%
    ungroup()

  # Decomposition Step 3 With Loop
  for (i in 1:length(list.vars.tomean)) {
      var.decomp.cur <- (paste0('value', list.vars.tomean.name.suffix[[i]]))
      vars.tomean <- list.vars.tomean[[i]]
      var.decomp.cur
      df.decompose <- df.decompose %>%
        mutate((!!var.decomp.cur) :=
                 ff_lr_decompose_valadj(., df.coef, vars.tomean, str.esti.suffix))
  }

  # Additional Statistics
  df.decompose.var.frac <- df.decompose %>%
          select(variable, contains('value')) %>%
          group_by(variable) %>%
          summarize_all(funs(mean = mean, var = var)) %>%
          select(variable, matches('value')) %>% select(variable, ends_with("_var")) %>%
          mutate_if(is.numeric, funs( frac = (./value_var))) %>%
          mutate_if(is.numeric, round, 3)

  # Graph
  g.graph.dist <- NULL
  if (graph) {
    g.graph.dist <- df.decompose %>%
        select(variable, contains('value'), -value_mean) %>%
        rename(outcome = variable) %>%
        gather(variable, value, -outcome) %>%
        ggplot(aes(x=value, color = variable, fill = variable)) +
            geom_line(stat = "density") +
            facet_wrap(~ outcome, scales='free', nrow=graph.nrow)
  }

  # Return
  return(list(dfmain = df.decompose,
              dfsumm = df.decompose.var.frac,
              graph = g.graph.dist))

}
```

```
# Support Function
ff_lr_decompose_valadj <- function(df, df.coef, vars.tomean, str.esti.suffix) {
    new_value <- (df$value +
                  rowSums((df[paste0(vars.tomean, '_mean')] - df[vars.tomean])
                          *df.coef[df$variable, paste0(vars.tomean, str.esti.suffix)]))
    return(new_value)
}
```

5.2.1.2 Prepare Decomposition Data

```
# Library
library(tidyverse)
library(AER)

# Load Sample Data
setwd('C:/Users/fan/R4Econ/_data/')
df <- read_csv('height_weight.csv')
```

```
## Parsed with column specification:
## cols(
##   S.country = col_character(),
##   vil.id = col_double(),
##   indi.id = col_double(),
##   sex = col_character(),
##   svymthRound = col_double(),
##   momEdu = col_double(),
##   wealthIdx = col_double(),
##   hgt = col_double(),
##   wgt = col_double(),
##   hgt0 = col_double(),
##   wgt0 = col_double(),
##   prot = col_double(),
##   cal = col_double(),
##   p.A.prot = col_double(),
##   p.A.nProt = col_double()
## )
```

```
# Source Dependency
source('C:/Users/fan/R4Econ/linreg/ivreg/ivregdfrow.R')

# Setting
options(repr.matrix.max.rows=50, repr.matrix.max.cols=50)
```

Data Cleaning.

```
# Convert Variable for Sex which is categorical to Numeric
df <- df
df$male <- (as.numeric(factor(df$sex)) - 1)
summary(factor(df$sex))
```

```
## Female   Male
##  16446  18619
```

```
summary(df$male)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   1.000   0.531   1.000   1.000
```

Parameters.

```
var.y1 <- c('hgt')
var.y2 <- c('wgt')
vars.y <- c(var.y1, var.y2)
vars.x <- c('prot')
vars.c <- c('male', 'wgt0', 'hgt0', 'svymthRound')
vars.other.keep <- c('S.country', 'vil.id', 'indi.id', 'svymthRound')

# Decompose sequence
vars.tomean.first <- c('male', 'hgt0')
var.tomean.first.name.suffix <- '_A'
vars.tomean.third <- c(vars.tomean.first, 'prot')
var.tomean.third.name.suffix <- '_B'
vars.tomean.fourth <- c(vars.tomean.third, 'svymthRound')
var.tomean.fourth.name.suffix <- '_C'
list.vars.tomean = list(vars.tomean.first,
                        vars.tomean.third,
                        vars.tomean.fourth)
list.vars.tomean.name.suffix <- list(var.tomean.first.name.suffix,
                                     var.tomean.third.name.suffix,
                                     var.tomean.fourth.name.suffix)
```

5.2.1.3 Example Guatemala OLS

```
df.use <- df %>% filter(S.country == 'Guatemala') %>%
  filter(svymthRound %in% c(12, 18, 24))
vars.z <- NULL
list.out <-
  ff_lr_decompose(df=df.use, vars.y, vars.x, vars.c, vars.z, vars.other.keep,
                  list.vars.tomean, list.vars.tomean.name.suffix,
                  graph=TRUE, graph.nrow=1)
options(repr.matrix.max.rows=10, repr.matrix.max.cols=50)
list.out$dfmain
```

```
## # A tibble: 1,382 x 19
##    S.country vil.id indi.id svymthRound  prot  male  wgt0  hgt0 variable value prot_mean male_mea
##    <chr>      <dbl>   <dbl>       <dbl> <dbl> <dbl> <dbl> <dbl> <chr>    <dbl>     <dbl>     <dbl
##  1 Guatemala      3    1352          18  13.3     1 2545.  47.4 hgt       70.2      20.6     0.55
##  2 Guatemala      3    1352          24  46.3     1 2545.  47.4 hgt       75.8      20.6     0.55
##  3 Guatemala      3    1354          12   1       1 3634.  51.2 hgt       66.3      20.6     0.55
##  4 Guatemala      3    1354          18   9.8     1 3634.  51.2 hgt       69.2      20.6     0.55
##  5 Guatemala      3    1354          24  15.4     1 3634.  51.2 hgt       75.3      20.6     0.55
##  6 Guatemala      3    1356          12   8.6     1 3912.  51.9 hgt       68.1      20.6     0.55
##  7 Guatemala      3    1356          18  17.8     1 3912.  51.9 hgt       74.1      20.6     0.55
##  8 Guatemala      3    1356          24  30.5     1 3912.  51.9 hgt       77.1      20.6     0.55
##  9 Guatemala      3    1357          12   1       1 3791.  52.6 hgt       71.5      20.6     0.55
## 10 Guatemala      3    1357          18  12.7     1 3791.  52.6 hgt       77.8      20.6     0.55
## # ... with 1,372 more rows, and 2 more variables: value_B <dbl>, value_C <dbl>
```

```
options(repr.plot.width = 10, repr.plot.height = 4)
list.out$dfsumm
```

```
## # A tibble: 2 x 11
##   variable value_var value_mean_var value_A_var value_B_var value_C_var value_var_frac value_mean
##   <chr>        <dbl>          <dbl>       <dbl>       <dbl>       <dbl>          <dbl>
## 1 hgt           21.9             NA        20.3        18.4        8.40              1
## 2 wgt      2965693.              NA    2863501.    2659434.    2346297.              1
```

5.2.1.4 Example Guatemala IV = vil.id

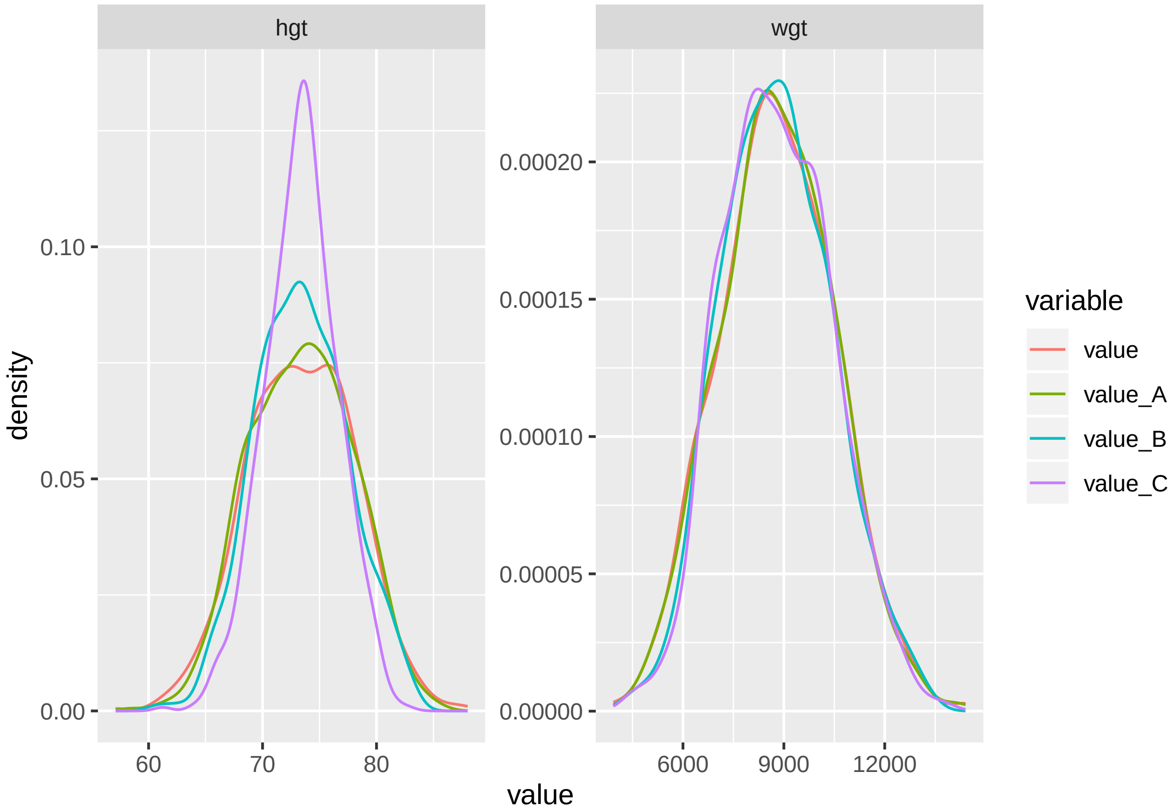
```
df.use <- df %>% filter(S.country == 'Guatemala') %>%
  filter(svymthRound %in% c(12, 18, 24))
vars.z <- c('vil.id')
list.out <- ff_lr_decompose(
  df=df.use, vars.y, vars.x, vars.c, vars.z, vars.other.keep,
  list.vars.tomean, list.vars.tomean.name.suffix,
  graph=TRUE, graph.nrow=1)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
list.out$dfsumm
```

```
## # A tibble: 2 x 11
##   variable value_var value_mean_var value_A_var value_B_var value_C_var value_var_frac value_mean
##   <chr>         <dbl>          <dbl>       <dbl>       <dbl>       <dbl>          <dbl>
## 1 hgt            21.9             NA        20.2        16.3        10.0              1
## 2 wgt       2965693.              NA    2876683.    2676220.    2583301.              1
```

```
options(repr.plot.width = 10, repr.plot.height = 2)
list.out$graph
```



5.2.1.5 Example Cebu OLS

```
df.use <- df %>% filter(S.country == 'Cebu') %>%
  filter(svymthRound %in% c(12, 18, 24))
vars.z <- NULL
```

```
list.out <- ff_lr_decompose(
  df=df.use, vars.y, vars.x, vars.c, vars.z, vars.other.keep,
  list.vars.tomean, list.vars.tomean.name.suffix,
  graph=TRUE, graph.nrow=1)
options(repr.matrix.max.rows=10, repr.matrix.max.cols=50)
list.out$dfmain
```

```
## # A tibble: 7,262 x 19
##    S.country vil.id indi.id svymthRound  prot  male  wgt0  hgt0 variable value prot_mean male_mea
##    <chr>      <dbl>   <dbl>       <dbl> <dbl> <dbl> <dbl> <dbl> <chr>    <dbl>     <dbl>     <dbl
##  1 Cebu           1       1          12  11.3     1 2044.  44.2 hgt       70.8      17.0     0.52
##  2 Cebu           1       2          12   5.9     0 2840.  49.7 hgt       72.2      17.0     0.52
##  3 Cebu           1       2          18   0.5     0 2840.  49.7 hgt       76.5      17.0     0.52
##  4 Cebu           1       2          24  14.1     0 2840.  49.7 hgt       79.2      17.0     0.52
##  5 Cebu           1       3          12  21.4     0 3446.  51.7 hgt       68        17.0     0.52
##  6 Cebu           1       3          18  23.6     0 3446.  51.7 hgt       71.6      17.0     0.52
##  7 Cebu           1       3          24  20.6     0 3446.  51.7 hgt       76.7      17.0     0.52
##  8 Cebu           1       4          12   0.7     0 3091.  50.2 hgt       69.1      17.0     0.52
##  9 Cebu           1       4          18   7.2     0 3091.  50.2 hgt       74.3      17.0     0.52
## 10 Cebu           1       4          24  10.3     0 3091.  50.2 hgt       78.1      17.0     0.52
## # ... with 7,252 more rows, and 2 more variables: value_B <dbl>, value_C <dbl>
```

```
options(repr.plot.width = 10, repr.plot.height = 4)
list.out$dfsumm
```

```
## # A tibble: 2 x 11
##   variable value_var value_mean_var value_A_var value_B_var value_C_var value_var_frac value_mean
##   <chr>        <dbl>          <dbl>       <dbl>       <dbl>       <dbl>          <dbl>
## 1 hgt           24.4             NA        22.6        21.3        10.0              1
## 2 wgt      3337461.              NA   3218987.    3039514.    2558514.               1
```

5.2.1.6 Example Cebu IV

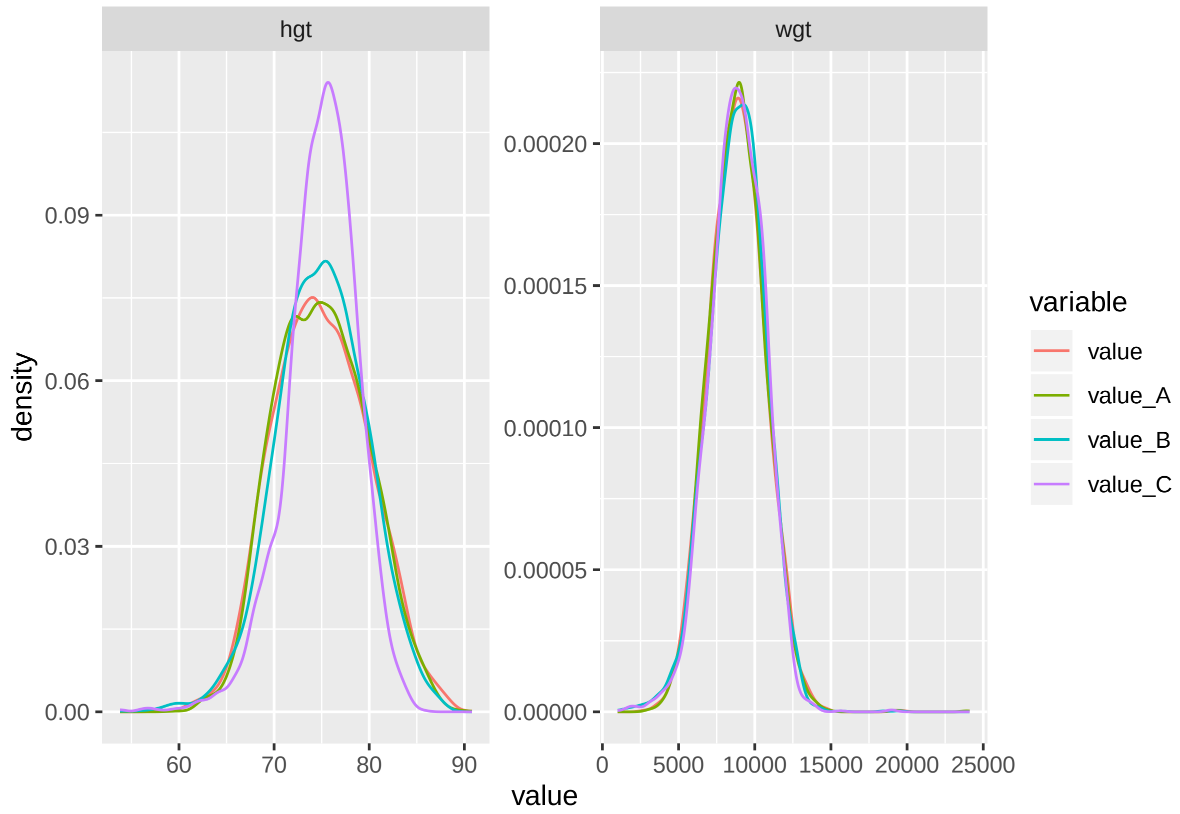
```
df.use <- df %>% filter(S.country == 'Cebu') %>%
  filter(svymthRound %in% c(12, 18, 24))
vars.z <- c('wealthIdx')
list.out <- ff_lr_decompose(
  df=df.use, vars.y, vars.x, vars.c, vars.z, vars.other.keep,
  list.vars.tomean, list.vars.tomean.name.suffix,
  graph=TRUE, graph.nrow=1)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
list.out$dfsumm
```

```
## # A tibble: 2 x 11
##   variable value_var value_mean_var value_A_var value_B_var value_C_var value_var_frac value_mean
##   <chr>        <dbl>          <dbl>       <dbl>       <dbl>       <dbl>          <dbl>
## 1 hgt           24.4             NA        22.6        22.2        14.4              1
## 2 wgt      3337461.              NA   3237415.    3385815.    3158659.               1
```

```
options(repr.plot.width = 10, repr.plot.height = 2)
list.out$graph
```



5.2.1.7 Examples Line by Line

The examples are just to test the code with different types of variables.

```
df.use <- df %>% filter(S.country == 'Guatemala') %>%
  filter(svymthRound %in% c(12, 18, 24))
dim(df.use)
```

```
## [1] 2022   16
```

Setting Up Parameters.

```
# Define Left Hand Side Variables
var.y1 <- c('hgt')
var.y2 <- c('wgt')
vars.y <- c(var.y1, var.y2)
# Define Right Hand Side Variables
vars.x <- c('prot')
vars.c <- c('male', 'wgt0', 'hgt0', 'svymthRound')
# vars.z <- c('p.A.prot')
vars.z <- c('vil.id')
# vars.z <- NULL
vars.xc <- c(vars.x, vars.c)

# Other variables to keep
vars.other.keep <- c('S.country', 'vil.id', 'indi.id', 'svymthRound')

# Decompose sequence
vars.tomean.first <- c('male', 'hgt0')
var.tomean.first.name.suffix <- '_mh02m'
vars.tomean.second <- c(vars.tomean.first, 'hgt0', 'wgt0')
var.tomean.second.name.suffix <- '_mh0me2m'
vars.tomean.third <- c(vars.tomean.second, 'prot')
```

```
var.tomean.third.name.suffix <- '_mh0mep2m'
vars.tomean.fourth <- c(vars.tomean.third, 'svymthRound')
var.tomean.fourth.name.suffix <- '_mh0mepm2m'
list.vars.tomean = list(
#                          vars.tomean.first,
                         vars.tomean.second,
                         vars.tomean.third,
                         vars.tomean.fourth
                         )
list.vars.tomean.name.suffix <- list(
#                                      var.tomean.first.name.suffix,
                                     var.tomean.second.name.suffix,
                                     var.tomean.third.name.suffix,
                                     var.tomean.fourth.name.suffix
                                    )
```

```
# Regressions
# regf.iv from C:\Users\fan\R4Econ\linreg\ivreg\ivregdfrow.R
df.reg.out <- as_tibble(
  bind_rows(lapply(vars.y, regf.iv,
                   vars.x=vars.x, vars.c=vars.c, vars.z=vars.z, df=df)))
```

5.2.1.7.1 Obtain Regression Coefficients from somewhere

```
## Warning: attributes are not identical across measure variables;
## they will be dropped

## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
# Regressions
# reg1 <- regf.iv(var.y = var.y1, vars.x, vars.c, vars.z, df.use)
# reg2 <- regf.iv(var.y = var.y2, vars.x, vars.c, vars.z, df.use)
# df.reg.out <- as_tibble(bind_rows(reg1, reg2))
```

```
options(repr.matrix.max.rows=50, repr.matrix.max.cols=50)
df.reg.out
```

```
## # A tibble: 2 x 37
##   X.Intercept._Es~ X.Intercept._Pr~ X.Intercept._St~ X.Intercept._zv~ hgt0_Estimate hgt0_Pr...z..
##   <chr>            <chr>            <chr>            <chr>            <chr>         <chr>
## 1 22.2547168993562 8.9088080511633~ 1.21637209166939 18.2959778934199 0.6834853337~ 4.5575874740~
## 2 -1101.090058068~ 0.0051062029326~ 393.210441213089 -2.800256408938~ 75.486789661~ 3.0043362381~
## # ... with 27 more variables: male_Std.Error <chr>, male_zvalue <chr>, prot_Estimate <chr>, prot_
## #   Sargan_df1 <chr>, svymthRound_Estimate <chr>, svymthRound_Pr...z.. <chr>, svymthRound_Std.Err
## #   vars_vars.c <chr>, vars_vars.x <chr>, vars_vars.z <chr>, Weakinstruments_df1 <chr>, Weakinstr
## #   Weakinstruments_statistic <chr>, wgt0_Estimate <chr>, wgt0_Pr...z.. <chr>, wgt0_Std.Error <ch
## #   Wu.Hausman_df2 <chr>, Wu.Hausman_p.value <chr>, Wu.Hausman_statistic <chr>
```

```
# Select Variables
str.esti.suffix <- '_Estimate'
arr.esti.name <- paste0(vars.xc, str.esti.suffix)
str.outcome.name <- 'vars_var.y'
arr.columns2select <- c(arr.esti.name, str.outcome.name)
arr.columns2select
```

```
## [1] "prot_Estimate"        "male_Estimate"        "wgt0_Estimate"        "hgt0_Estimate"        "
```

```
# Generate dataframe for coefficients
df.coef <- df.reg.out[,c(arr.columns2select)] %>% mutate_at(vars(arr.esti.name), as.numeric) %>% col
```

```
df.coef
```

```
##     prot_Estimate male_Estimate wgt0_Estimate hgt0_Estimate svymthRound_Estimate
## hgt    -0.2714772      1.244735  0.0004430418     0.6834853             1.133919
## wgt   -59.0727542    489.852902  0.7696158110    75.4867897           250.778883
```

```
str(df.coef)
```

```
## 'data.frame':    2 obs. of  5 variables:
##  $ prot_Estimate       : num  -0.271 -59.073
##  $ male_Estimate       : num  1.24 489.85
##  $ wgt0_Estimate       : num  0.000443 0.769616
##  $ hgt0_Estimate       : num  0.683 75.487
##  $ svymthRound_Estimate: num  1.13 250.78
```

```
# Decomposition Step 1: gather
df.decompose_step1 <- df.use %>%
                        filter(svymthRound %in% c(12, 18, 24)) %>%
                        select(one_of(c(vars.other.keep, vars.xc, vars.y))) %>%
                        drop_na() %>%
                        gather(variable, value, -one_of(c(vars.other.keep, vars.xc)))
options(repr.matrix.max.rows=20, repr.matrix.max.cols=20)
dim(df.decompose_step1)
```

5.2.1.7.2 Decomposition Step 1

```
## [1] 1382   10
```

```
df.decompose_step1
```

```
## # A tibble: 1,382 x 10
##    S.country vil.id indi.id svymthRound  prot  male  wgt0  hgt0 variable value
##    <chr>      <dbl>   <dbl>       <dbl> <dbl> <dbl> <dbl> <dbl> <chr>    <dbl>
##  1 Guatemala      3    1352          18  13.3     1 2545.  47.4 hgt       70.2
##  2 Guatemala      3    1352          24  46.3     1 2545.  47.4 hgt       75.8
##  3 Guatemala      3    1354          12   1       1 3634.  51.2 hgt       66.3
##  4 Guatemala      3    1354          18   9.8     1 3634.  51.2 hgt       69.2
##  5 Guatemala      3    1354          24  15.4     1 3634.  51.2 hgt       75.3
##  6 Guatemala      3    1356          12   8.6     1 3912.  51.9 hgt       68.1
##  7 Guatemala      3    1356          18  17.8     1 3912.  51.9 hgt       74.1
##  8 Guatemala      3    1356          24  30.5     1 3912.  51.9 hgt       77.1
##  9 Guatemala      3    1357          12   1       1 3791.  52.6 hgt       71.5
## 10 Guatemala      3    1357          18  12.7     1 3791.  52.6 hgt       77.8
## # ... with 1,372 more rows
```

```
# Decomposition Step 2: mutate_at(vars, funs(mean = mean(.)))
# the xc averaging could have taken place earlier, no difference in mean across variables
df.decompose_step2 <- df.decompose_step1 %>%
                        group_by(variable) %>%
                        mutate_at(vars(c(vars.xc, 'value')), funs(mean = mean(.))) %>%
                        ungroup()

options(repr.matrix.max.rows=20, repr.matrix.max.cols=20)
dim(df.decompose_step2)
```

5.2.1.7.3 Decomposition Step 2

```
## [1] 1382   16
```

```
df.decompose_step2
```

```
## # A tibble: 1,382 x 16
##    S.country vil.id indi.id svymthRound  prot  male  wgt0  hgt0 variable value prot_mean male_mea
##    <chr>      <dbl>   <dbl>       <dbl> <dbl> <dbl> <dbl> <dbl> <chr>    <dbl>     <dbl>     <dbl
##  1 Guatemala      3    1352          18  13.3     1 2545.  47.4 hgt       70.2      20.6     0.55
##  2 Guatemala      3    1352          24  46.3     1 2545.  47.4 hgt       75.8      20.6     0.55
##  3 Guatemala      3    1354          12   1       1 3634.  51.2 hgt       66.3      20.6     0.55
##  4 Guatemala      3    1354          18   9.8     1 3634.  51.2 hgt       69.2      20.6     0.55
##  5 Guatemala      3    1354          24  15.4     1 3634.  51.2 hgt       75.3      20.6     0.55
##  6 Guatemala      3    1356          12   8.6     1 3912.  51.9 hgt       68.1      20.6     0.55
##  7 Guatemala      3    1356          18  17.8     1 3912.  51.9 hgt       74.1      20.6     0.55
##  8 Guatemala      3    1356          24  30.5     1 3912.  51.9 hgt       77.1      20.6     0.55
##  9 Guatemala      3    1357          12   1       1 3791.  52.6 hgt       71.5      20.6     0.55
## 10 Guatemala      3    1357          18  12.7     1 3791.  52.6 hgt       77.8      20.6     0.55
## # ... with 1,372 more rows
```

```
ff_lr_decompose_valadj <- function(df, df.coef, vars.tomean, str.esti.suffix) {
    new_value <- (df$value +
                  rowSums((df[paste0(vars.tomean, '_mean')] - df[vars.tomean])
                          *df.coef[df$variable, paste0(vars.tomean, str.esti.suffix)]))
    return(new_value)
}

# # Decomposition Step 3: mutate_at(vars, funs(mean = mean(.)))
# var.decomp.one <- (paste0('value', list.vars.tomean.name.suffix[[1]]))
# var.decomp.two <- (paste0('value', list.vars.tomean.name.suffix[[2]]))
# var.decomp.thr <- (paste0('value', list.vars.tomean.name.suffix[[3]]))
# df.decompose_step3 <- df.decompose_step2 %>%
#                          mutate((!!var.decomp.one) := f_decompose_here(., df.coef, list.vars.tomean
#                                 (!!var.decomp.two) := f_decompose_here(., df.coef, list.vars.tomean
#                                 (!!var.decomp.thr) := f_decompose_here(., df.coef, list.vars.tomean

# options(repr.matrix.max.rows=10, repr.matrix.max.cols=20)
# dim(df.decompose_step3)
# df.decompose_step3
```

5.2.1.7.4 Decomposition Step 3 Non-Loop

```
df.decompose_step3 <- df.decompose_step2
for (i in 1:length(list.vars.tomean)) {
    var.decomp.cur <- (paste0('value', list.vars.tomean.name.suffix[[i]]))
    vars.tomean <- list.vars.tomean[[i]]
    var.decomp.cur
    df.decompose_step3 <- df.decompose_step3 %>%
      mutate((!!var.decomp.cur) :=
               ff_lr_decompose_valadj(., df.coef, vars.tomean, str.esti.suffix))

}
options(repr.matrix.max.rows=10, repr.matrix.max.cols=20)
dim(df.decompose_step3)
```

5.2.1.7.5 Decomposition Step 3 With Loop

```
## [1] 1382   19
```

```r
df.decompose_step3
```

```
## # A tibble: 1,382 x 19
##    S.country vil.id indi.id svymthRound  prot  male  wgt0  hgt0 variable value prot_mean male_mea
##    <chr>      <dbl>   <dbl>       <dbl> <dbl> <dbl> <dbl> <dbl> <chr>    <dbl>     <dbl>     <dbl
##  1 Guatemala      3    1352          18  13.3     1 2545.  47.4 hgt       70.2      20.6     0.55
##  2 Guatemala      3    1352          24  46.3     1 2545.  47.4 hgt       75.8      20.6     0.55
##  3 Guatemala      3    1354          12   1       1 3634.  51.2 hgt       66.3      20.6     0.55
##  4 Guatemala      3    1354          18   9.8     1 3634.  51.2 hgt       69.2      20.6     0.55
##  5 Guatemala      3    1354          24  15.4     1 3634.  51.2 hgt       75.3      20.6     0.55
##  6 Guatemala      3    1356          12   8.6     1 3912.  51.9 hgt       68.1      20.6     0.55
##  7 Guatemala      3    1356          18  17.8     1 3912.  51.9 hgt       74.1      20.6     0.55
##  8 Guatemala      3    1356          24  30.5     1 3912.  51.9 hgt       77.1      20.6     0.55
##  9 Guatemala      3    1357          12   1       1 3791.  52.6 hgt       71.5      20.6     0.55
## 10 Guatemala      3    1357          18  12.7     1 3791.  52.6 hgt       77.8      20.6     0.55
## # ... with 1,372 more rows, and 3 more variables: value_mh0me2m <dbl>, value_mh0mep2m <dbl>, valu
```

```r
df.decompose_step3 %>%
        select(variable, contains('value')) %>%
        group_by(variable) %>%
        summarize_all(funs(mean = mean, var = var)) %>%
        select(matches('value')) %>% select(ends_with("_var")) %>%
        mutate_if(is.numeric, funs( frac = (./value_var))) %>%
        mutate_if(is.numeric, round, 3)
```

5.2.1.7.6 Decomposition Step 4 Variance

```
## # A tibble: 2 x 10
##   value_var value_mean_var value_mh0me2m_v~ value_mh0mep2m_~ value_mh0mepm2m~ value_var_frac valu
##       <dbl>          <dbl>            <dbl>            <dbl>            <dbl>          <dbl>
## 1      21.9             NA             25.4             49.0             23.1              1
## 2 2965693.              NA         2949188.         4192770.         3147507.              1
## # ... with 1 more variable: value_mh0mepm2m_var_frac <dbl>
```

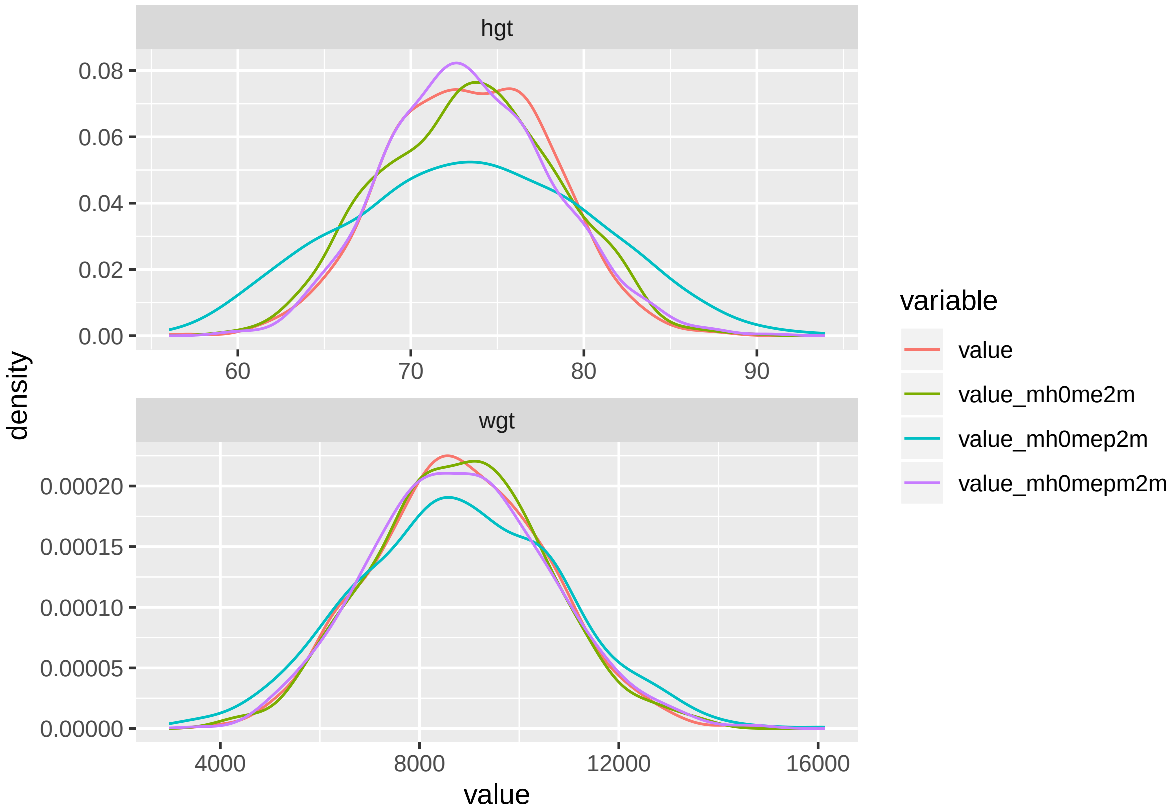
5.2.1.7.7 Graphical Results Graphically, difficult to pick up exact differences in variance, a 50 percent reduction in variance visually does not look like 50 percent. Intuitively, we are kind of seeing standard deviation, not variance on the graph if we think abou the x-scale.

```r
df.decompose_step3 %>%
    select(variable, contains('value'), -value_mean)
```

```
## # A tibble: 1,382 x 5
##    variable value value_mh0me2m value_mh0mep2m value_mh0mepm2m
##    <chr>    <dbl>         <dbl>          <dbl>           <dbl>
##  1 hgt       70.2          73.2           71.2            71.7
##  2 hgt       75.8          78.8           85.8            79.4
##  3 hgt       66.3          63.6           58.3            65.6
##  4 hgt       69.2          66.5           63.6            64.1
##  5 hgt       75.3          72.6           71.2            64.9
##  6 hgt       68.1          64.3           61.1            68.4
##  7 hgt       74.1          70.3           69.6            70.0
##  8 hgt       77.1          73.3           76.0            69.7
##  9 hgt       71.5          66.8           61.5            68.8
## 10 hgt       77.8          73.1           71.0            71.5
## # ... with 1,372 more rows
```

```r
options(repr.plot.width = 10, repr.plot.height = 4)
df.decompose_step3 %>%
    select(variable, contains('value'), -value_mean) %>%
```

```r
    rename(outcome = variable) %>%
    gather(variable, value, -outcome) %>%
    ggplot(aes(x=value, color = variable, fill = variable)) +
        geom_line(stat = "density") +
        facet_wrap(~ outcome, scales='free', nrow=2)
```



5.2.1.8 Additional Decomposition Testings

```r
head(df.decompose_step2[vars.tomean.first],3)
```

```
## # A tibble: 3 x 2
##    male  hgt0
##   <dbl> <dbl>
## 1     1  47.4
## 2     1  47.4
## 3     1  51.2
```

```r
head(df.decompose_step2[paste0(vars.tomean.first, '_mean')], 3)
```

```
## # A tibble: 3 x 2
##   male_mean hgt0_mean
##       <dbl>     <dbl>
## 1     0.550      49.8
## 2     0.550      49.8
## 3     0.550      49.8
```

```r
head(df.coef[df.decompose_step2$variable,
             paste0(vars.tomean.first, str.esti.suffix)], 3)
```

```
##        male_Estimate hgt0_Estimate
## hgt         1.244735     0.6834853
## hgt.1       1.244735     0.6834853
```

```
## hgt.2      1.244735     0.6834853
```

```
df.decompose.tomean.first <- df.decompose_step2 %>%
    mutate(pred_new = df.decompose_step2$value +
        rowSums((df.decompose_step2[paste0(vars.tomean.first, '_mean')]
                 - df.decompose_step2[vars.tomean.first])
             *df.coef[df.decompose_step2$variable,
                     paste0(vars.tomean.first, str.esti.suffix)])) %>%
        select(variable, value, pred_new)
head(df.decompose.tomean.first, 10)
```

```
## # A tibble: 10 x 3
##    variable value pred_new
##    <chr>    <dbl>    <dbl>
##  1 hgt       70.2     71.2
##  2 hgt       75.8     76.8
##  3 hgt       66.3     64.7
##  4 hgt       69.2     67.6
##  5 hgt       75.3     73.7
##  6 hgt       68.1     66.1
##  7 hgt       74.1     72.1
##  8 hgt       77.1     75.1
##  9 hgt       71.5     69.0
## 10 hgt       77.8     75.3
```

```
df.decompose.tomean.first %>%
        group_by(variable) %>%
        summarize_all(funs(mean = mean, sd = sd))
```

```
## # A tibble: 2 x 5
##   variable value_mean pred_new_mean value_sd pred_new_sd
##   <chr>         <dbl>         <dbl>    <dbl>       <dbl>
## 1 hgt            73.4          73.4     4.68        4.53
## 2 wgt          8808.         8808.   1722.       1695.
```

Note the r-square from regression above matches up with the 1 - ratio below. This is the proper decomposition method that is equivalent to r2.

```
df.decompose_step2 %>%
    mutate(pred_new = df.decompose_step2$value +
        rowSums((df.decompose_step2[paste0(vars.tomean.second, '_mean')]
                 - df.decompose_step2[vars.tomean.second])
             *df.coef[df.decompose_step2$variable,
                     paste0(vars.tomean.second, str.esti.suffix)])) %>%
        select(variable, value, pred_new) %>%
        group_by(variable) %>%
        summarize_all(funs(mean = mean, var = var)) %>%
        mutate(ratio = (pred_new_var/value_var))
```

```
## # A tibble: 2 x 6
##   variable value_mean pred_new_mean value_var pred_new_var ratio
##   <chr>         <dbl>         <dbl>     <dbl>        <dbl> <dbl>
## 1 hgt            73.4          73.4      21.9         25.4 1.16
## 2 wgt          8808.         8808.   2965693.     2949188.  0.994
```

Chapter 6

Nonlinear Regression

6.1 Logit Regression

6.1.1 Binary Logit

Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

Data Preparation

```
df_mtcars <- mtcars

# X-variables to use on RHS
ls_st_xs <- c('mpg', 'qsec')
ls_st_xs <- c('mpg')
ls_st_xs <- c('qsec')
ls_st_xs <- c('wt')
ls_st_xs <- c('mpg', 'wt', 'vs')

svr_binary <- 'hpLowHigh'
svr_binary_lb0 <- 'LowHP'
svr_binary_lb1 <- 'HighHP'
svr_outcome <- 'am'
sdt_name <- 'mtcars'

# Discretize hp
df_mtcars <- df_mtcars %>%
    mutate(!!sym(svr_binary) := cut(hp,
                                breaks=c(-Inf, 210, Inf),
                                labels=c(svr_binary_lb0, svr_binary_lb1)))
```

6.1.1.1 Logit Regresion and Prediction

logit regression with glm, and predict using estimation data. Prediction and estimation with one variable.

- LOGIT REGRESSION R DATA ANALYSIS EXAMPLES
- Generalized Linear Models

```
# Regress
rs_logit <- glm(as.formula(paste(svr_outcome, "~", paste(ls_st_xs, collapse="+")))
                ,data = df_mtcars, family = "binomial")
summary(rs_logit)
```

```
##
## Call:
```

```
## glm(formula = as.formula(paste(svr_outcome, "~", paste(ls_st_xs,
##     collapse = "+"))), family = "binomial", data = df_mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.73603  -0.25477  -0.04891   0.13402   1.90321
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 22.69008   13.95112   1.626   0.1039
## mpg         -0.01786    0.33957  -0.053   0.9581
## wt          -6.73804    3.01400  -2.236   0.0254 *
## vs          -4.44046    2.84247  -1.562   0.1182
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.230  on 31  degrees of freedom
## Residual deviance: 13.092  on 28  degrees of freedom
## AIC: 21.092
##
## Number of Fisher Scoring iterations: 7
```

```
# Predcit Using Regression Data
df_mtcars$p_mpg <- predict(rs_logit, newdata = df_mtcars, type = "response")
```

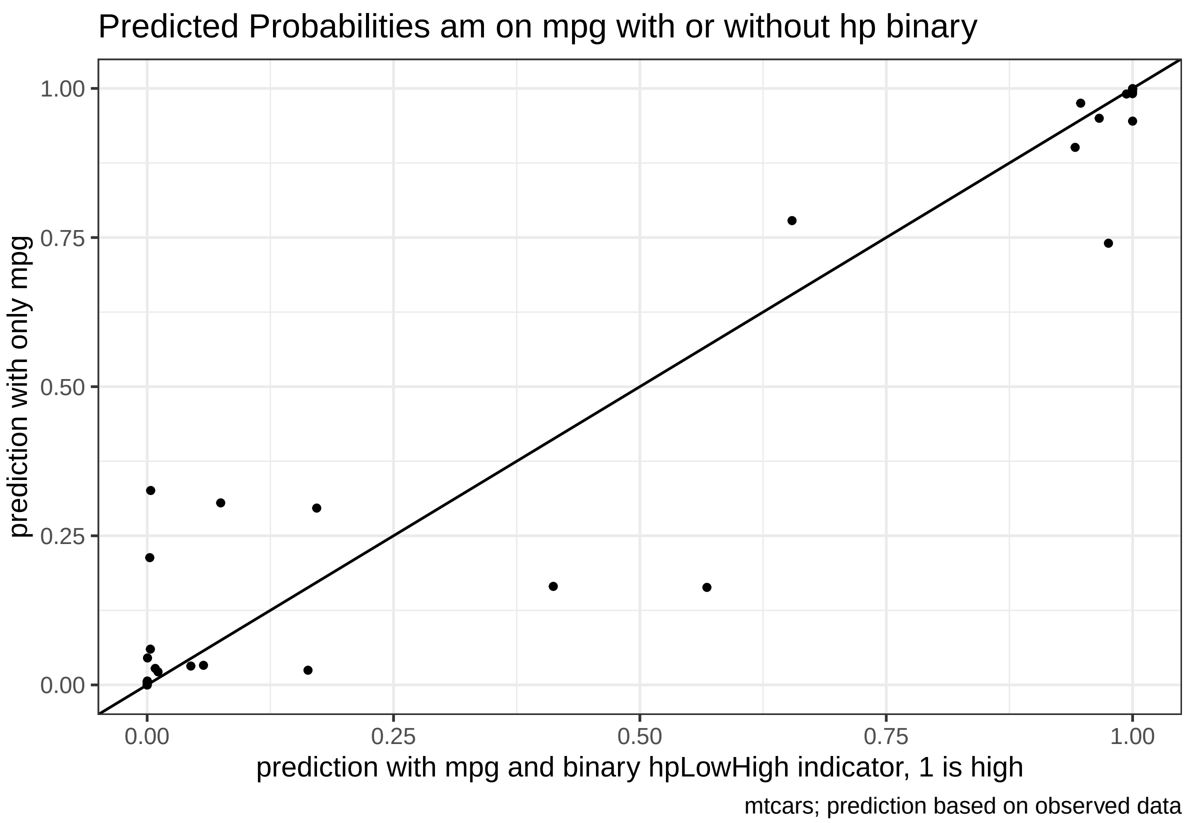
6.1.1.1.1 Prediction with Observed Binary Input Logit regression with a continuous variable and a binary variable. Predict outcome with observed continuous variable as well as observed binary input variable.

```
# Regress
rs_logit_bi <- glm(as.formula(paste(svr_outcome,
                                    "~ factor(", svr_binary,") + ",
                                    paste(ls_st_xs, collapse="+")))
                   , data = df_mtcars, family = "binomial")
summary(rs_logit_bi)
```

```
##
## Call:
## glm(formula = as.formula(paste(svr_outcome, "~ factor(", svr_binary,
##     ") + ", paste(ls_st_xs, collapse = "+"))), family = "binomial",
##     data = df_mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.45771  -0.09563  -0.00875   0.00555   1.87612
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 3.8285    18.0390   0.212   0.8319
## factor(hpLowHigh)HighHP     6.9907     5.5176   1.267   0.2052
## mpg                         0.8985     0.8906   1.009   0.3131
## wt                         -6.7291     3.3166  -2.029   0.0425 *
## vs                         -5.9206     4.1908  -1.413   0.1577
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 43.2297  on 31  degrees of freedom
## Residual deviance:  8.9777  on 27  degrees of freedom
## AIC: 18.978
##
## Number of Fisher Scoring iterations: 9
```

```
# Predcit Using Regresion Data
df_mtcars$p_mpg_hp <- predict(rs_logit_bi, newdata = df_mtcars, type = "response")

# Predicted Probabilities am on mgp with or without hp binary
scatter <- ggplot(df_mtcars, aes(x=p_mpg_hp, y=p_mpg)) +
      geom_point(size=1) +
      # geom_smooth(method=lm) + # Trend line
      geom_abline(intercept = 0, slope = 1) + # 45 degree line
      labs(title = paste0('Predicted Probabilities ', svr_outcome, ' on ', ls_st_xs, ' with or witho
           x = paste0('prediction with ', ls_st_xs, ' and binary ', svr_binary, ' indicator, 1 is hi
           y = paste0('prediction with only ', ls_st_xs),
           caption = 'mtcars; prediction based on observed data') +
      theme_bw()
print(scatter)
```



6.1.1.1.2 Prediction with Binary set to 0 and 1 Now generate two predictions. One set where binary input is equal to 0, and another where the binary inputs are equal to 1. Ignore whether in data binary input is equal to 0 or 1. Use the same regression results as what was just derived.

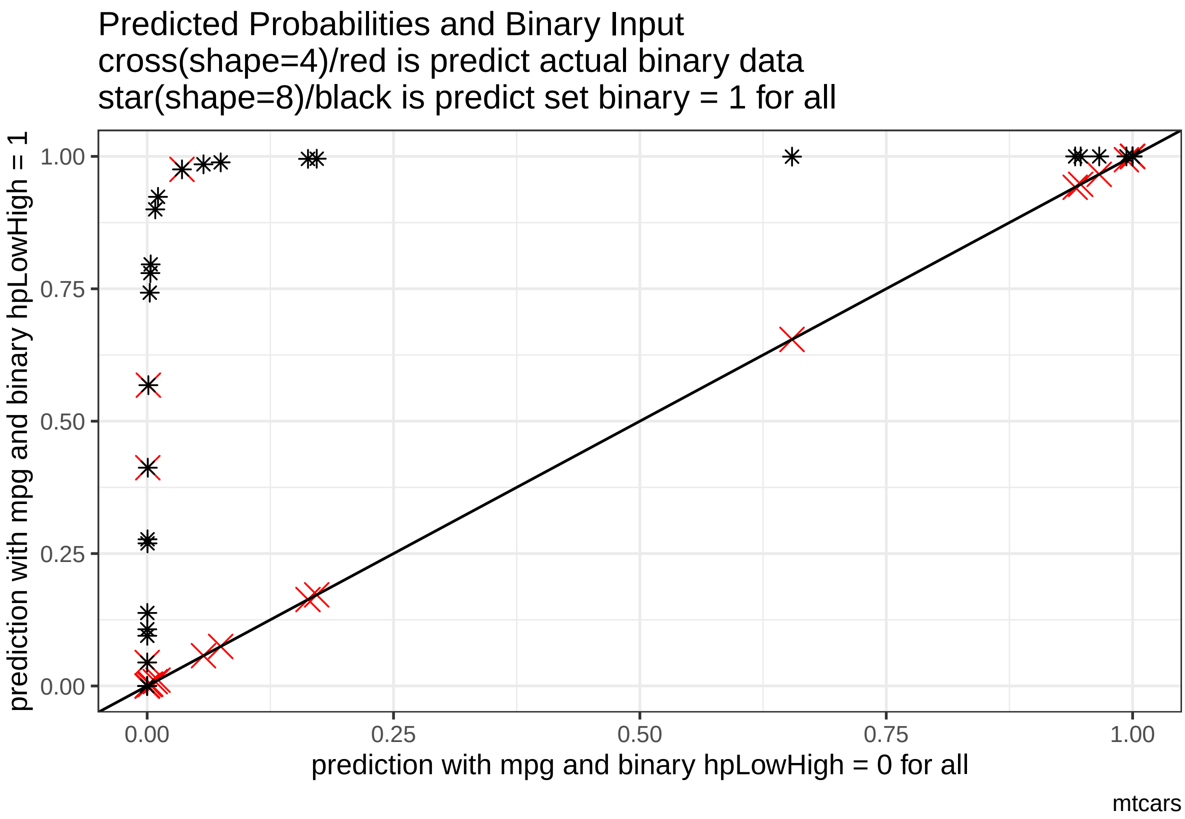
Note that given the example here, the probability changes a lot when we

```
# Previous regression results
summary(rs_logit_bi)
```

```
##
## Call:
```

```
## glm(formula = as.formula(paste(svr_outcome, "~ factor(", svr_binary,
##     ") + ", paste(ls_st_xs, collapse = "+"))), family = "binomial",
##     data = df_mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.45771  -0.09563  -0.00875   0.00555   1.87612
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)              3.8285    18.0390   0.212   0.8319
## factor(hpLowHigh)HighHP  6.9907     5.5176   1.267   0.2052
## mpg                      0.8985     0.8906   1.009   0.3131
## wt                      -6.7291     3.3166  -2.029   0.0425 *
## vs                      -5.9206     4.1908  -1.413   0.1577
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.2297  on 31  degrees of freedom
## Residual deviance:  8.9777  on 27  degrees of freedom
## AIC: 18.978
##
## Number of Fisher Scoring iterations: 9
```

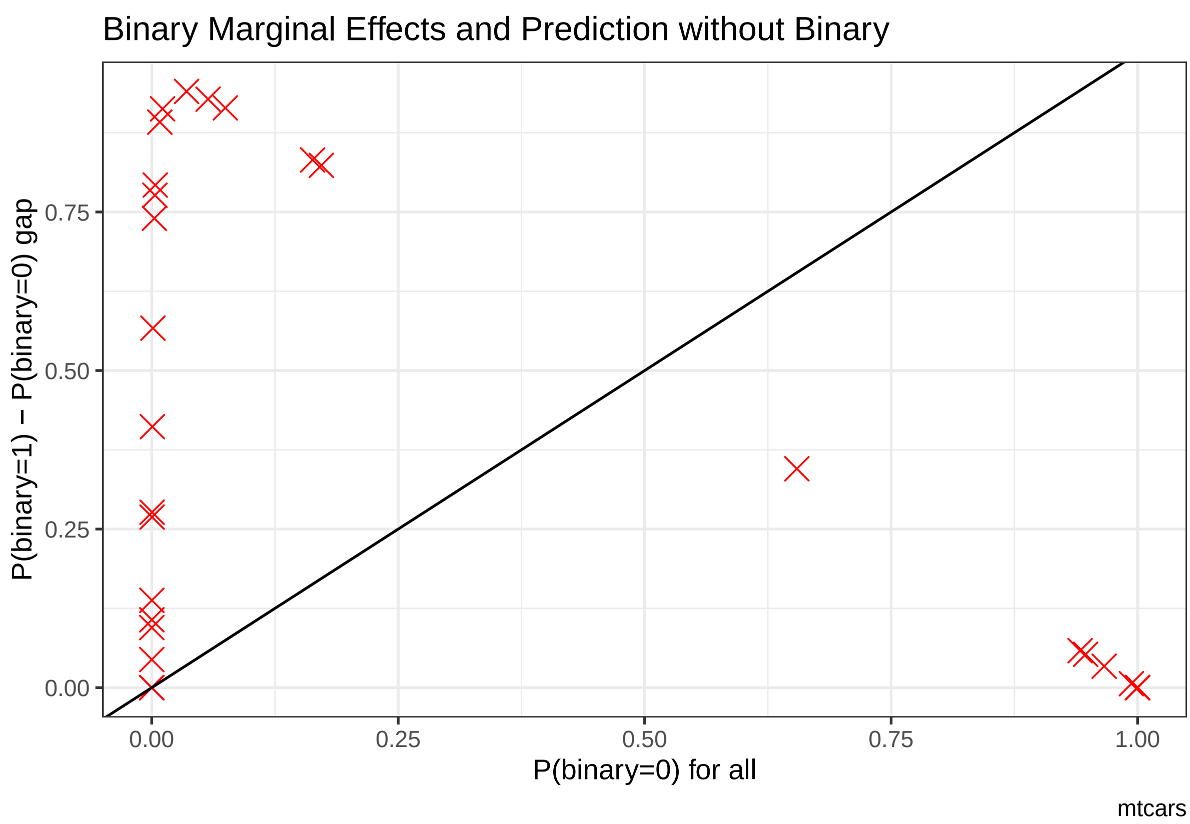
```r
# Two different dataframes, mutate the binary regressor
df_mtcars_bi0 <- df_mtcars %>% mutate(!!sym(svr_binary) := svr_binary_lb0)
df_mtcars_bi1 <- df_mtcars %>% mutate(!!sym(svr_binary) := svr_binary_lb1)

# Predcit Using Regresion Data
df_mtcars$p_mpg_hp_bi0 <- predict(rs_logit_bi, newdata = df_mtcars_bi0, type = "response")
df_mtcars$p_mpg_hp_bi1 <- predict(rs_logit_bi, newdata = df_mtcars_bi1, type = "response")

# Predicted Probabilities and Binary Input
scatter <- ggplot(df_mtcars, aes(x=p_mpg_hp_bi0)) +
      geom_point(aes(y=p_mpg_hp), size=4, shape=4, color="red") +
      geom_point(aes(y=p_mpg_hp_bi1), size=2, shape=8) +
      # geom_smooth(method=lm) + # Trend line
      geom_abline(intercept = 0, slope = 1) + # 45 degree line
      labs(title = paste0('Predicted Probabilities and Binary Input',
                          '\ncross(shape=4)/red is predict actual binary data',
                          '\nstar(shape=8)/black is predict set binary = 1 for all'),
            x = paste0('prediction with ', ls_st_xs, ' and binary ', svr_binary, ' = 0 for all'),
            y = paste0('prediction with ', ls_st_xs, ' and binary ', svr_binary, ' = 1'),
           caption = paste0(sdt_name)) +
      theme_bw()
print(scatter)
```



6.1.1.1.3 Prediction with Binary set to 0 and 1 Difference What is the difference in probability between binary $= 0$ vs binary $= 1$. How does that relate to the probability of outcome of interest when binary $= 0$ for all.

In the binary logit case, the relationship will be hump–shaped by construction between A_i and α_i. In the exponential wage cases, the relationship is convex upwards.

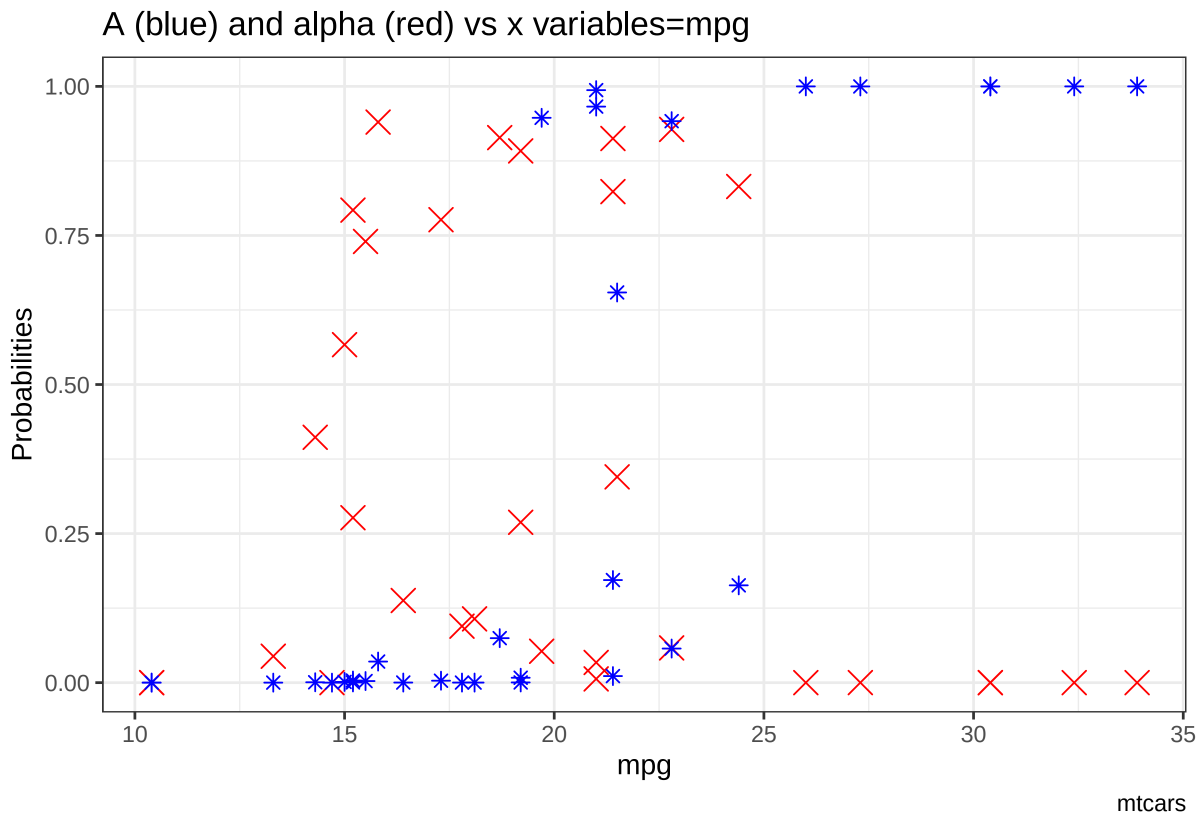
```
# Generate Gap Variable
df_mtcars <- df_mtcars %>% mutate(alpha_i = p_mpg_hp_bi1 - p_mpg_hp_bi0) %>%
                mutate(A_i = p_mpg_hp_bi0)

# Binary Marginal Effects and Prediction without Binary
scatter <- ggplot(df_mtcars, aes(x=A_i)) +
      geom_point(aes(y=alpha_i), size=4, shape=4, color="red") +
      geom_abline(intercept = 0, slope = 1) + # 45 degree line
      labs(title = paste0('Binary Marginal Effects and Prediction without Binary'),
           x = 'P(binary=0) for all',
           y = 'P(binary=1) - P(binary=0) gap',
           caption = paste0(sdt_name)) +
      theme_bw()
print(scatter)
```



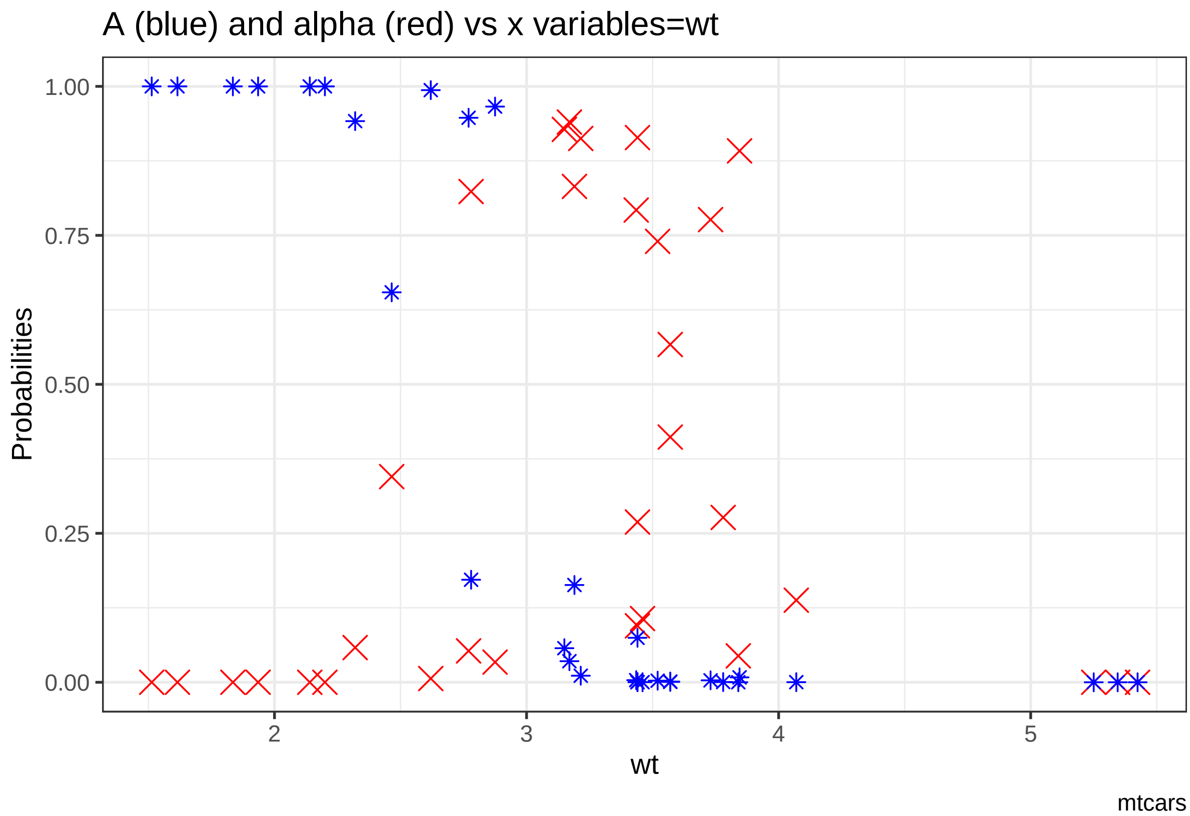
6.1.1.1.4 X variables and A and alpha Given the x-variables included in the logit regression, how do they relate to A_i and alpha_i

```
# Generate Gap Variable
df_mtcars <- df_mtcars %>% mutate(alpha_i = p_mpg_hp_bi1 - p_mpg_hp_bi0) %>%
                mutate(A_i = p_mpg_hp_bi0)

# Binary Marginal Effects and Prediction without Binary
ggplot.A.alpha.x <- function(svr_x, df,
                             svr_alpha = 'alpha_i', svr_A = "A_i"){

  scatter <- ggplot(df, aes(x=!!sym(svr_x))) +
        geom_point(aes(y=alpha_i), size=4, shape=4, color="red") +
        geom_point(aes(y=A_i), size=2, shape=8, color="blue") +
        geom_abline(intercept = 0, slope = 1) + # 45 degree line
        labs(title = paste0('A (blue) and alpha (red) vs x variables=', svr_x),
             x = svr_x,
             y = 'Probabilities',
             caption = paste0(sdt_name)) +
        theme_bw()

return(scatter)
}

# Plot over multiple
lapply(ls_st_xs,
       ggplot.A.alpha.x,
       df = df_mtcars)
```

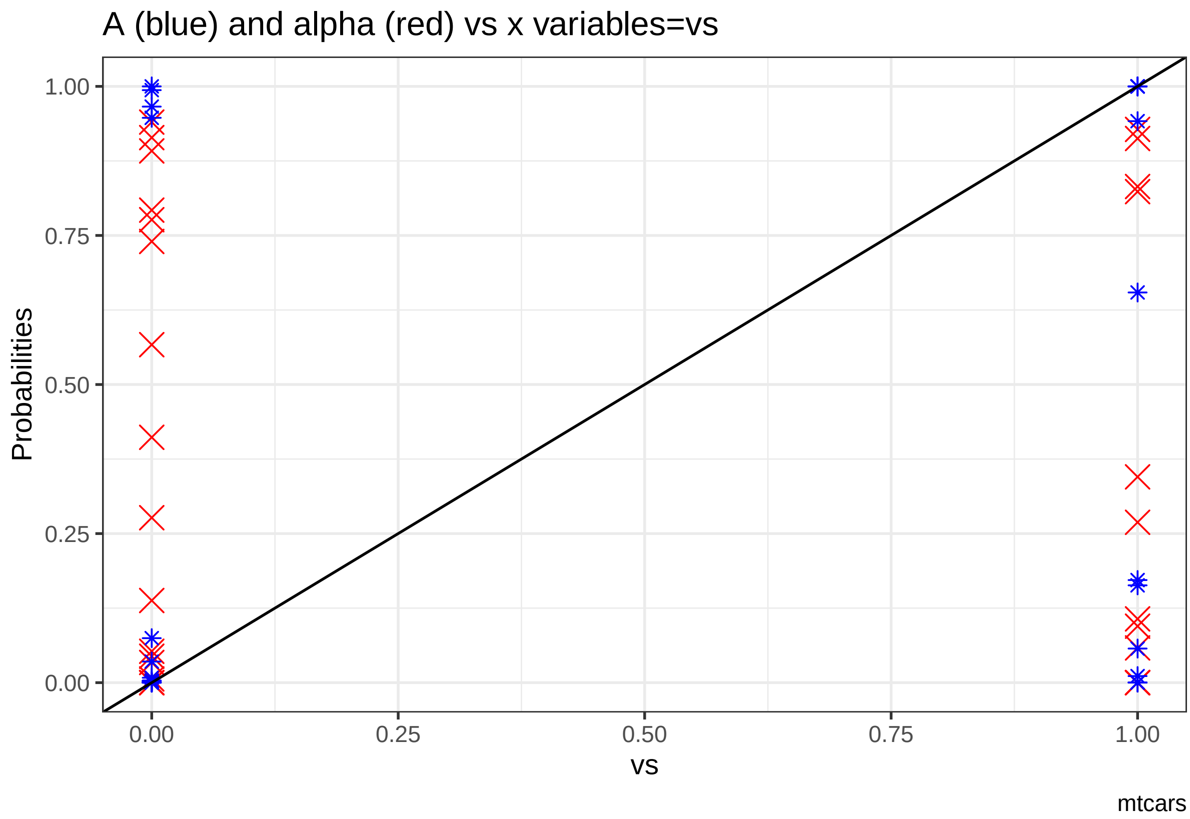
```
## [[1]]
```


A (blue) and alpha (red) vs x variables=mpg
Probabilities
1.00
0.75
0.50
0.25
0.00
10
15
20
25
30
35
mpg
mtcars

```
##
## [[2]]
```


A (blue) and alpha (red) vs x variables=wt
Probabilities
1.00
0.75
0.50
0.25
0.00
2
3
4
5
wt
mtcars

```
##
## [[3]]
```


A (blue) and alpha (red) vs x variables=vs
Probabilities
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
vs
mtcars

Chapter 7

Optimization

7.1 Bisection

7.1.1 Bisection

> Go back to fan's REconTools Package, R4Econ Repository (bookdown site), or Intro Stats with R Repository.

See the ff_opti_bisect_pmap_multi function from Fan's *REconTools* Package, which provides a resuable function based on the algorithm worked out here.

The bisection specific code does not need to do much.

- list variables in file for grouping, each group is an individual for whom we want to calculate optimal choice for using bisection.
- string variable name of input where functions are evaluated, these are already contained in the dataframe, existing variable names, row specific, rowwise computation over these, each rowwise calculation using different rows.
- scalar and array values that are applied to every rowwise calculation, all rowwise calculations using the same scalars and arrays.
- string output variable name

This is how I implement the bisection algorithm, when we know the bounding minimum and maximum to be below and above zero already.

1. Evaluate $f_a^0 = f(a^0)$ and $f_b^0 = f(b^0)$, min and max points.
2. Evaluate at $f_p^0 = f(p^0)$, where $p_0 = \frac{a^0+b^0}{2}$.
3. if $f_a^i \cdot f_p^i < 0$, then $b_{i+1} = p_i$, else, $a_{i+1} = p_i$ and $f_a^{i+1} = p_i$.
4. iteratre until convergence.

Generate New columns of a and b as we iteratre, do not need to store p, p is temporary. Evaluate the function below which we have already tested, but now, in the dataframe before generating all permutations, *tb_states_choices*, now the *fl_N* element will be changing with each iteration, it will be row specific. *fl_N* are first min and max, then each subsequent ps.

7.1.1.1 Initialize Matrix

First, initialize the matrix with a_0 and b_0, the initial min and max points:

```
# common prefix to make reshaping easier
st_bisec_prefix <- 'bisec_'
svr_a_lst <- paste0(st_bisec_prefix, 'a_0')
svr_b_lst <- paste0(st_bisec_prefix, 'b_0')
svr_fa_lst <- paste0(st_bisec_prefix, 'fa_0')
svr_fb_lst <- paste0(st_bisec_prefix, 'fb_0')
```

```
# Add initial a and b
tb_states_choices_bisec <- tb_states_choices %>%
  mutate(!!sym(svr_a_lst) := fl_N_min, !!sym(svr_b_lst) := fl_N_agg)

# Evaluate function f(a_0) and f(b_0)
tb_states_choices_bisec <- tb_states_choices_bisec %>%
  rowwise() %>%
  mutate(!!sym(svr_fa_lst) := ffi_nonlin_dplyrdo(fl_A, fl_alpha, !!sym(svr_a_lst),
                                                 ar_nN_A, ar_nN_alpha,
                                                 fl_N_agg, fl_rho),
         !!sym(svr_fb_lst) := ffi_nonlin_dplyrdo(fl_A, fl_alpha, !!sym(svr_b_lst),
                                                 ar_nN_A, ar_nN_alpha,
                                                 fl_N_agg, fl_rho))
# Summarize
dim(tb_states_choices_bisec)
```

```
## [1] 4 7
```

```
summary(tb_states_choices_bisec)
```

```
##     INDI_ID          fl_A        fl_alpha     bisec_a_0   bisec_b_0     bisec_fa_0    bisec_fb_0
##  Min.   :1.00   Min.   :-2   Min.   :0.1   Min.   :0   Min.   :100   Min.   :100   Min.   :-15057
##  1st Qu.:1.75   1st Qu.:-1   1st Qu.:0.3   1st Qu.:0   1st Qu.:100   1st Qu.:100   1st Qu.: -5011
##  Median :2.50   Median : 0   Median :0.5   Median :0   Median :100   Median :100   Median : -1033
##  Mean   :2.50   Mean   : 0   Mean   :0.5   Mean   :0   Mean   :100   Mean   :100   Mean   : -4301
##  3rd Qu.:3.25   3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.:0   3rd Qu.:100   3rd Qu.:100   3rd Qu.:  -322
##  Max.   :4.00   Max.   : 2   Max.   :0.9   Max.   :0   Max.   :100   Max.   :100   Max.   :   -81
```

7.1.1.2 Iterate and Solve for f(p), update f(a) and f(b)

Implement the DPLYR based Concurrent bisection algorithm.

```
# fl_tol = float tolerance criteria
# it_tol = number of interations to allow at most
fl_tol <- 10^-2
it_tol <- 100

# fl_p_dist2zr = distance to zero to initalize
fl_p_dist2zr <- 1000
it_cur <- 0
while (it_cur <= it_tol && fl_p_dist2zr >= fl_tol ) {

  it_cur <- it_cur + 1

  # New Variables
  svr_a_cur <- paste0(st_bisec_prefix, 'a_', it_cur)
  svr_b_cur <- paste0(st_bisec_prefix, 'b_', it_cur)
  svr_fa_cur <- paste0(st_bisec_prefix, 'fa_', it_cur)
  svr_fb_cur <- paste0(st_bisec_prefix, 'fb_', it_cur)

  # Evaluate function f(a_0) and f(b_0)
  # 1. generate p
  # 2. generate f_p
  # 3. generate f_p*f_a
  tb_states_choices_bisec <- tb_states_choices_bisec %>%
    rowwise() %>%
    mutate(p = ((!!sym(svr_a_lst) + !!sym(svr_b_lst))/2)) %>%
    mutate(f_p = ffi_nonlin_dplyrdo(fl_A, fl_alpha, p,
                                    ar_nN_A, ar_nN_alpha,
```

```
                                     fl_N_agg, fl_rho)) %>%
    mutate(f_p_t_f_a = f_p*!!sym(svr_fa_lst))
  # fl_p_dist2zr = sum(abs(p))
  fl_p_dist2zr <- mean(abs(tb_states_choices_bisec %>% pull(f_p)))

  # Update a and b
  tb_states_choices_bisec <- tb_states_choices_bisec %>%
    mutate(!!sym(svr_a_cur) :=
             case_when(f_p_t_f_a < 0 ~ !!sym(svr_a_lst),
                       TRUE ~ p)) %>%
    mutate(!!sym(svr_b_cur) :=
             case_when(f_p_t_f_a < 0 ~ p,
                       TRUE ~ !!sym(svr_b_lst)))
  # Update f(a) and f(b)
  tb_states_choices_bisec <- tb_states_choices_bisec %>%
    mutate(!!sym(svr_fa_cur) :=
             case_when(f_p_t_f_a < 0 ~ !!sym(svr_fa_lst),
                       TRUE ~ f_p)) %>%
    mutate(!!sym(svr_fb_cur) :=
             case_when(f_p_t_f_a < 0 ~ f_p,
                       TRUE ~ !!sym(svr_fb_lst)))
  # Save from last
  svr_a_lst <- svr_a_cur
  svr_b_lst <- svr_b_cur
  svr_fa_lst <- svr_fa_cur
  svr_fb_lst <- svr_fb_cur

  # Summar current round
  print(paste0('it_cur:', it_cur, ', fl_p_dist2zr:', fl_p_dist2zr))
  summary(tb_states_choices_bisec %>%
            select(one_of(svr_a_cur, svr_b_cur, svr_fa_cur, svr_fb_cur)))
}
```

```
## [1] "it_cur:1, fl_p_dist2zr:1884.20860322127"
## [1] "it_cur:2, fl_p_dist2zr:815.07213515036"
## [1] "it_cur:3, fl_p_dist2zr:346.193951089409"
## [1] "it_cur:4, fl_p_dist2zr:133.268318242343"
## [1] "it_cur:5, fl_p_dist2zr:52.0759336601643"
## [1] "it_cur:6, fl_p_dist2zr:8.2057326579422"
## [1] "it_cur:7, fl_p_dist2zr:12.7240911320081"
## [1] "it_cur:8, fl_p_dist2zr:4.10100732130902"
## [1] "it_cur:9, fl_p_dist2zr:1.19915237247596"
## [1] "it_cur:10, fl_p_dist2zr:1.46089191924225"
## [1] "it_cur:11, fl_p_dist2zr:0.261965457555881"
## [1] "it_cur:12, fl_p_dist2zr:0.462901483859291"
## [1] "it_cur:13, fl_p_dist2zr:0.166336071560483"
## [1] "it_cur:14, fl_p_dist2zr:0.011649263648799"
## [1] "it_cur:15, fl_p_dist2zr:0.0715183716517558"
## [1] "it_cur:16, fl_p_dist2zr:0.0299376539319738"
## [1] "it_cur:17, fl_p_dist2zr:0.0132655999120672"
## [1] "it_cur:18, fl_p_dist2zr:0.00317751042553027"
```

7.1.1.3 Reshape Wide to long to Wide

To view results easily, how iterations improved to help us find the roots, convert table from wide to long. Pivot twice. This allows us to easily graph out how bisection is working out iterationby iteration.

Here, we will first show what the raw table looks like, the wide only table, and then show the long version, and finally the version that is medium wide.

7.1.1.3.1 Table One–Very Wide

Show what the *tb_states_choices_bisec* looks like.

Variables are formatted like: *bisec_xx_yy*, where yy is the iteration indicator, and xx is either a, b, fa, or fb.

```
head(tb_states_choices_bisec, 10)
```

```
## Source: local data frame [4 x 82]
## Groups: <by row>
##
## # A tibble: 4 x 82
##   INDI_ID   fl_A fl_alpha bisec_a_0 bisec_b_0 bisec_fa_0 bisec_fb_0     p      f_p f_p_t_f_a bise
##     <int>  <dbl>    <dbl>     <dbl>     <dbl>      <dbl>      <dbl> <dbl>    <dbl>     <dbl>
## 1       1 -2        0.1           0       100        100    -15058.  1.32  1.02e-2   4.45e-4
## 2       2 -0.667    0.367         0       100        100     -1663.  7.29 -1.26e-3  -5.61e-6
## 3       3  0.667    0.633         0       100        100      -403.  20.9   6.18e-4   1.54e-6
## 4       4  2        0.9           0       100        100       -81.3 54.1  -6.09e-4  -4.46e-8
## # ... with 66 more variables: bisec_fa_2 <dbl>, bisec_fb_2 <dbl>, bisec_a_3 <dbl>, bisec_b_3 <dbl
## #   bisec_b_4 <dbl>, bisec_fa_4 <dbl>, bisec_fb_4 <dbl>, bisec_a_5 <dbl>, bisec_b_5 <dbl>, bisec_
## #   bisec_b_6 <dbl>, bisec_fa_6 <dbl>, bisec_fb_6 <dbl>, bisec_a_7 <dbl>, bisec_b_7 <dbl>, bisec_
## #   bisec_b_8 <dbl>, bisec_fa_8 <dbl>, bisec_fb_8 <dbl>, bisec_a_9 <dbl>, bisec_b_9 <dbl>, bisec_
## #   bisec_b_10 <dbl>, bisec_fa_10 <dbl>, bisec_fb_10 <dbl>, bisec_a_11 <dbl>, bisec_b_11 <dbl>, b
## #   bisec_b_12 <dbl>, bisec_fa_12 <dbl>, bisec_fb_12 <dbl>, bisec_a_13 <dbl>, bisec_b_13 <dbl>, b
## #   bisec_b_14 <dbl>, bisec_fa_14 <dbl>, bisec_fb_14 <dbl>, bisec_a_15 <dbl>, bisec_b_15 <dbl>, b
## #   bisec_b_16 <dbl>, bisec_fa_16 <dbl>, bisec_fb_16 <dbl>, bisec_a_17 <dbl>, bisec_b_17 <dbl>, b
## #   bisec_b_18 <dbl>, bisec_fa_18 <dbl>, bisec_fb_18 <dbl>
```

```
str(tb_states_choices_bisec)
```

```
## Classes 'rowwise_df', 'tbl_df', 'tbl' and 'data.frame':  4 obs. of  82 variables:
##  $ INDI_ID   : int  1 2 3 4
##  $ fl_A      : num  -2 -0.667 0.667 2
##  $ fl_alpha  : num  0.1 0.367 0.633 0.9
##  $ bisec_a_0 : num  0 0 0 0
##  $ bisec_b_0 : num  100 100 100 100
##  $ bisec_fa_0 : num  100 100 100 100
##  $ bisec_fb_0 : num  -15057.6 -1663.3 -403.1 -81.3
##  $ p         : num  1.32 7.29 20.89 54.1
##  $ f_p       : num  0.010225 -0.001259 0.000618 -0.000609
##  $ f_p_t_f_a : num  4.45e-04 -5.61e-06 1.54e-06 -4.46e-08
##  $ bisec_a_1 : num  0 0 0 50
##  $ bisec_b_1 : num  50 50 50 100
##  $ bisec_fa_1 : num  100 100 100 7.33
##  $ bisec_fb_1 : num  -6659.8 -723.7 -145.9 -81.3
##  $ bisec_a_2 : num  0 0 0 50
##  $ bisec_b_2 : num  25 25 25 75
##  $ bisec_fa_2 : num  100 100 100 7.33
##  $ bisec_fb_2 : num  -2917.6 -285.2 -20.3 -37.2
##  $ bisec_a_3 : num  0 0 12.5 50
##  $ bisec_b_3 : num  12.5 12.5 25 62.5
##  $ bisec_fa_3 : num  100 100 41.08 7.33
##  $ bisec_fb_3 : num  -1248.4 -80.3 -20.3 -15
##  $ bisec_a_4 : num  0 6.25 18.75 50
##  $ bisec_b_4 : num  6.25 12.5 25 56.25
##  $ bisec_fa_4 : num  100 15.52 10.54 7.33
##  $ bisec_fb_4 : num  -503.16 -80.3 -20.32 -3.85
##  $ bisec_a_5 : num  0 6.25 18.75 53.12
##  $ bisec_b_5 : num  3.12 9.38 21.88 56.25
##  $ bisec_fa_5 : num  100 15.52 10.54 1.74
##  $ bisec_fb_5 : num  -170.1 -31.61 -4.86 -3.85
```

```
##  $ bisec_a_6  : num  0 6.25 20.31 53.12
##  $ bisec_b_6  : num  1.56 7.81 21.88 54.69
##  $ bisec_fa_6 : num  100 15.52 2.85 1.74
##  $ bisec_fb_6 : num  -21.09 -7.82 -4.86 -1.06
##  $ bisec_a_7  : num  0.781 7.031 20.312 53.906
##  $ bisec_b_7  : num  1.56 7.81 21.09 54.69
##  $ bisec_fa_7 : num  45.65 3.909 2.853 0.338
##  $ bisec_fb_7 : num  -21.089 -7.822 -0.999 -1.059
##  $ bisec_a_8  : num  1.17 7.03 20.7 53.91
##  $ bisec_b_8  : num  1.56 7.42 21.09 54.3
##  $ bisec_fa_8 : num  13.174 3.909 0.928 0.338
##  $ bisec_fb_8 : num  -21.089 -1.942 -0.999 -0.36
##  $ bisec_a_9  : num  1.17 7.23 20.7 53.91
##  $ bisec_b_9  : num  1.37 7.42 20.9 54.1
##  $ bisec_fa_9 : num  13.174 0.988 0.928 0.338
##  $ bisec_fb_9 : num  -3.763 -1.9416 -0.0351 -0.0108
##  $ bisec_a_10 : num  1.27 7.23 20.8 54
##  $ bisec_b_10 : num  1.37 7.32 20.9 54.1
##  $ bisec_fa_10: num  4.757 0.988 0.446 0.164
##  $ bisec_fb_10: num  -3.763 -0.476 -0.0351 -0.0108
##  $ bisec_a_11 : num  1.32 7.28 20.85 54.05
##  $ bisec_b_11 : num  1.37 7.32 20.9 54.1
##  $ bisec_fa_11: num  0.5096 0.2561 0.2057 0.0765
##  $ bisec_fb_11: num  -3.763 -0.476 -0.0351 -0.0108
##  $ bisec_a_12 : num  1.32 7.28 20.87 54.08
##  $ bisec_b_12 : num  1.34 7.3 20.9 54.1
##  $ bisec_fa_12: num  0.5096 0.2561 0.0853 0.0328
##  $ bisec_fb_12: num  -1.6236 -0.1099 -0.0351 -0.0108
##  $ bisec_a_13 : num  1.32 7.29 20.89 54.09
##  $ bisec_b_13 : num  1.33 7.3 20.9 54.1
##  $ bisec_fa_13: num  0.5096 0.0731 0.0251 0.011
##  $ bisec_fb_13: num  -0.5562 -0.1099 -0.0351 -0.0108
##  $ bisec_a_14 : num  1.32 7.29 20.89 54.1
##  $ bisec_b_14 : num  1.32 7.29 20.89 54.1
##  $ bisec_fa_14: num  5.10e-01 7.31e-02 2.51e-02 7.33e-05
##  $ bisec_fb_14: num  -0.02308 -0.01842 -0.00503 -0.01084
##  $ bisec_a_15 : num  1.32 7.29 20.89 54.1
##  $ bisec_b_15 : num  1.32 7.29 20.89 54.1
##  $ bisec_fa_15: num  2.43e-01 2.73e-02 1.00e-02 7.33e-05
##  $ bisec_fb_15: num  -0.02308 -0.01842 -0.00503 -0.00538
##  $ bisec_a_16 : num  1.32 7.29 20.89 54.1
##  $ bisec_b_16 : num  1.32 7.29 20.89 54.1
##  $ bisec_fa_16: num  1.10e-01 4.46e-03 2.50e-03 7.33e-05
##  $ bisec_fb_16: num  -0.02308 -0.01842 -0.00503 -0.00266
##  $ bisec_a_17 : num  1.32 7.29 20.89 54.1
##  $ bisec_b_17 : num  1.32 7.29 20.89 54.1
##  $ bisec_fa_17: num  4.35e-02 4.46e-03 2.50e-03 7.33e-05
##  $ bisec_fb_17: num  -0.02308 -0.00698 -0.00126 -0.00129
##  $ bisec_a_18 : num  1.32 7.29 20.89 54.1
##  $ bisec_b_18 : num  1.32 7.29 20.89 54.1
##  $ bisec_fa_18: num  1.02e-02 4.46e-03 6.18e-04 7.33e-05
##  $ bisec_fb_18: num  -0.023082 -0.001259 -0.001264 -0.000609
```

7.1.1.3.2 Table Two–Very Wide to Very Long We want to treat the iteration count information that is the suffix of variable names as a variable by itself. Additionally, we want to treat the a,b,fa,fb as a variable. Structuring the data very long like this allows for easy graphing and other types of analysis. Rather than dealing with many many variables, we have only 3 core variables that store bisection iteration information.

Here we use the very nice *pivot_longer* function. Note that to achieve this, we put a common prefix in front of the variables we wanted to convert to long. THis is helpful, because we can easily identify which variables need to be reshaped.

```
# New variables
svr_bisect_iter <- 'biseciter'
svr_abfafb_long_name <- 'varname'
svr_number_col <- 'value'
svr_id_bisect_iter <- paste0(svr_id_var, '_bisect_ier')

# Pivot wide to very long
tb_states_choices_bisec_long <- tb_states_choices_bisec %>%
  pivot_longer(
    cols = starts_with(st_bisec_prefix),
    names_to = c(svr_abfafb_long_name, svr_bisect_iter),
    names_pattern = paste0(st_bisec_prefix, "(.*)_(.*)"),
    values_to = svr_number_col
  )

# Print
summary(tb_states_choices_bisec_long)
```

```
##     INDI_ID          fl_A      fl_alpha          p                f_p                  f_p_t_f_a
##  Min.   :1.00   Min.   :-2   Min.   :0.1   Min.   : 1.324   Min.   :-1.259e-03   Min.   :-5.614e-
##  1st Qu.:1.75   1st Qu.:-1   1st Qu.:0.3   1st Qu.: 5.800   1st Qu.:-7.714e-04   1st Qu.:-1.437e-
##  Median :2.50   Median : 0   Median :0.5   Median :14.092   Median : 4.364e-06   Median : 7.495e-
##  Mean   :2.50   Mean   : 0   Mean   :0.5   Mean   :20.901   Mean   : 2.244e-03   Mean   : 1.102e-
##  3rd Qu.:3.25   3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.:29.192   3rd Qu.: 3.019e-03   3rd Qu.: 1.124e-
##  Max.   :4.00   Max.   : 2   Max.   :0.9   Max.   :54.096   Max.   : 1.022e-02   Max.   : 4.451e-
##      value
##  Min.   :-15057.608
##  1st Qu.:     0.000
##  Median :     1.367
##  Mean   :   -82.350
##  3rd Qu.:    20.892
##  Max.   :   100.000
```

```
head(tb_states_choices_bisec_long %>% select(-one_of('p','f_p','f_p_t_f_a')), 30)
```

```
## # A tibble: 30 x 6
##    INDI_ID  fl_A fl_alpha varname biseciter   value
##      <int> <dbl>    <dbl> <chr>   <chr>       <dbl>
##  1       1    -2      0.1 a       0               0
##  2       1    -2      0.1 b       0             100
##  3       1    -2      0.1 fa      0             100
##  4       1    -2      0.1 fb      0          -15058.
##  5       1    -2      0.1 a       1               0
##  6       1    -2      0.1 b       1              50
##  7       1    -2      0.1 fa      1             100
##  8       1    -2      0.1 fb      1           -6660.
##  9       1    -2      0.1 a       2               0
## 10       1    -2      0.1 b       2              25
## # ... with 20 more rows
```

```
tail(tb_states_choices_bisec_long %>% select(-one_of('p','f_p','f_p_t_f_a')), 30)
```

```
## # A tibble: 30 x 6
##    INDI_ID  fl_A fl_alpha varname biseciter   value
##      <int> <dbl>    <dbl> <chr>   <chr>       <dbl>
##  1       4     2      0.9 fa      11         0.0765
```

```
## 2 4 2 0.9 fb 11 -0.0108
## 3 4 2 0.9 a 12 54.1
## 4 4 2 0.9 b 12 54.1
## 5 4 2 0.9 fa 12 0.0328
## 6 4 2 0.9 fb 12 -0.0108
## 7 4 2 0.9 a 13 54.1
## 8 4 2 0.9 b 13 54.1
## 9 4 2 0.9 fa 13 0.0110
## 10 4 2 0.9 fb 13 -0.0108
## # ... with 20 more rows
```

7.1.1.3.3 Table Two–Very Very Long to Wider Again But the previous results are too long, with the a, b, fa, and fb all in one column as different categories, they are really not different categories, they are in fact different types of variables. So we want to spread those four categories of this variable into four columns, each one representing the a, b, fa, and fb values. The rows would then be uniqly identified by the iteration counter and individual ID.

```
# Pivot wide to very long to a little wide
tb_states_choices_bisec_wider <- tb_states_choices_bisec_long %>%
  pivot_wider(
    names_from = !!sym(svr_abfafb_long_name),
    values_from = svr_number_col
  )

# Print
summary(tb_states_choices_bisec_wider)
```

```
##     INDI_ID          fl_A        fl_alpha         p                 f_p                  f_p_t_f_a
##  Min.   :1.00   Min.   :-2   Min.   :0.1   Min.   : 1.324   Min.   :-1.259e-03   Min.   :-5.614e-
##  1st Qu.:1.75   1st Qu.:-1   1st Qu.:0.3   1st Qu.: 5.800   1st Qu.:-7.714e-04   1st Qu.:-1.437e-
##  Median :2.50   Median : 0   Median :0.5   Median :14.092   Median : 4.364e-06   Median : 7.495e-
##  Mean   :2.50   Mean   : 0   Mean   :0.5   Mean   :20.901   Mean   : 2.244e-03   Mean   : 1.102e-
##  3rd Qu.:3.25   3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.:29.192   3rd Qu.: 3.019e-03   3rd Qu.: 1.124e-
##  Max.   :4.00   Max.   : 2   Max.   :0.9   Max.   :54.096   Max.   : 1.022e-02   Max.   : 4.451e-
##        fa                   fb
##  Min.   :  0.00007   Min.   :-15057.608
##  1st Qu.:  0.06570   1st Qu.:   -21.089
##  Median :  0.92799   Median :    -1.029
##  Mean   : 22.90627   Mean   :  -399.547
##  3rd Qu.: 15.51699   3rd Qu.:    -0.018
##  Max.   :100.00000   Max.   :    -0.001
```

```
print(tb_states_choices_bisec_wider %>% select(-one_of('p','f_p','f_p_t_f_a')))
```

```
## # A tibble: 76 x 8
##    INDI_ID  fl_A fl_alpha biseciter     a      b    fa        fb
##      <int> <dbl>    <dbl> <chr>     <dbl>  <dbl> <dbl>     <dbl>
##  1       1    -2      0.1 0         0     100    100   -15058.
##  2       1    -2      0.1 1         0      50    100    -6660.
##  3       1    -2      0.1 2         0      25    100    -2918.
##  4       1    -2      0.1 3         0      12.5  100    -1248.
##  5       1    -2      0.1 4         0       6.25 100     -503.
##  6       1    -2      0.1 5         0       3.12 100     -170.
##  7       1    -2      0.1 6         0       1.56 100      -21.1
##  8       1    -2      0.1 7         0.781   1.56  45.7    -21.1
##  9       1    -2      0.1 8         1.17    1.56  13.2    -21.1
## 10       1    -2      0.1 9         1.17    1.37  13.2     -3.76
## # ... with 66 more rows
```

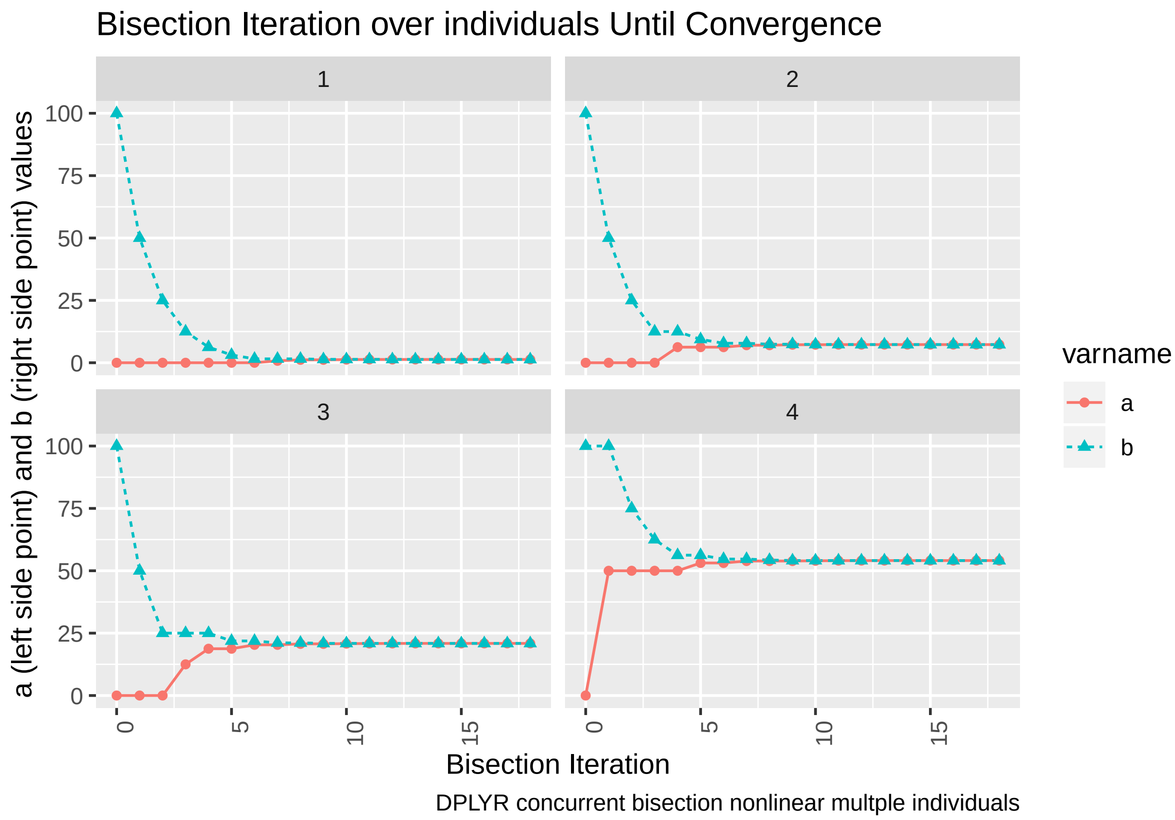
```
print(tb_states_choices_bisec_wider %>% select(-one_of('p','f_p','f_p_t_f_a')))
```

```
## # A tibble: 76 x 8
##    INDI_ID  fl_A fl_alpha biseciter     a      b    fa        fb
##      <int> <dbl>    <dbl> <chr>     <dbl>  <dbl> <dbl>     <dbl>
##  1       1    -2      0.1 0         0      100    100   -15058.
##  2       1    -2      0.1 1         0       50    100    -6660.
##  3       1    -2      0.1 2         0       25    100    -2918.
##  4       1    -2      0.1 3         0       12.5  100    -1248.
##  5       1    -2      0.1 4         0        6.25 100     -503.
##  6       1    -2      0.1 5         0        3.12 100     -170.
##  7       1    -2      0.1 6         0        1.56 100      -21.1
##  8       1    -2      0.1 7         0.781    1.56  45.7    -21.1
##  9       1    -2      0.1 8         1.17     1.56  13.2    -21.1
## 10       1    -2      0.1 9         1.17     1.37  13.2     -3.76
## # ... with 66 more rows
```

7.1.1.4 Graph Bisection Iteration Results

Actually we want to graph based on the long results, not the wider. Wider easier to view in table.

```
# Graph results
lineplot <- tb_states_choices_bisec_long %>%
    mutate(!!sym(svr_bisect_iter) := as.numeric(!!sym(svr_bisect_iter))) %>%
    filter(!!sym(svr_abfafb_long_name) %in% c('a', 'b')) %>%
    ggplot(aes(x=!!sym(svr_bisect_iter), y=!!sym(svr_number_col),
               colour=!!sym(svr_abfafb_long_name),
               linetype=!!sym(svr_abfafb_long_name),
               shape=!!sym(svr_abfafb_long_name))) +
        facet_wrap( ~ INDI_ID) +
        geom_line() +
        geom_point() +
        labs(title = 'Bisection Iteration over individuals Until Convergence',
             x = 'Bisection Iteration',
             y = 'a (left side point) and b (right side point) values',
             caption = 'DPLYR concurrent bisection nonlinear multple individuals') +
      theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(lineplot)
```


Bisection Iteration over individuals Until Convergence
1
2
3
4
100
75
50
25
0
0
5
10
15
a (left side point) and b (right side point) values
Bisection Iteration
varname
a
b
DPLYR concurrent bisection nonlinear multple individuals

Bibliography

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Wang, F. (2020). *REconTools: R Tools for Panel Data and Optimization*. R package version 0.0.0.9000.

Wickham, H. (2019). *tidyverse: Easily Install and Load the 'Tidyverse'*. R package version 1.3.0.

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.18.