Panel Data and Optimization with R

Fan Wang

2020-04-14

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)
```

Print Various objects generated above:

```
# 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 element from list:

```
# 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]]*
4. converted two dimensional named list to tibble/matrix

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))

# Convert to Matrix
tb_2d_flat_named <- as_tibble(ls_2d_flat_named) %>% unnest()
```

```
## Warning: `cols` is now required.
## Please use `cols = c(col1, col2, col3, col4)`
```

```
mt_2d_flat_named <- as.matrix(tb_2d_flat_named)
```

Print various objects generated above:

```
# 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
```

```
print('tb_2d_flat_named')
```

```
## [1] "tb_2d_flat_named"
```

```
print(tb_2d_flat_named)
```

```
## # A tibble: 3 x 4
##    col1  col2  col3  col4
##   <dbl> <dbl> <dbl> <dbl>
## 1     1     4     7    10
## 2     2     5     8    11
## 3     3     6     9    12
```

```
print('mt_2d_flat_named')
```

```
## [1] "mt_2d_flat_named"
```

```
print(mt_2d_flat_named)
```

```
##      col1 col2 col3 col4
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Select elements from list:

```
# 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

1.2.1.2.1 Remove Elements of Array Select elements with direct indexing, or with head and tail functions. Get the first two elements of three elements array.

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

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

```
# Use the head function to remove last element
head(vars.group.bydf, -1)
```

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

```
head(vars.group.bydf, 2)
```

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

Get last two elements of array.

```
# Remove first element of array
vars.group.bydf <- c('23','dfa', 'wer')
vars.group.bydf[2:length(vars.group.bydf)]
```

```
## [1] "dfa" "wer"
```

```
# Use Tail function
tail(vars.group.bydf, -1)
```

```
## [1] "dfa" "wer"
```

```
tail(vars.group.bydf, 2)
```

```
## [1] "dfa" "wer"
```

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 <- (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] "C:/Users/fan/R4Econ/amto/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

```
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

```
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

```
# 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
```

```r
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()
```

-0.9453838	0.7635994	1.2740310	0.3112232	0.8221379	0.0341220	0.3112232	0.8221379
1.6740934	1.0163546	-0.2660679	0.2556918	0.4832825	0.8689931	0.2556918	0.4832825
0.9906241	-1.4880550	1.3920327	0.8835966	0.2126780	0.5405587	0.8835966	0.2126780
-0.5932284	1.5907415	-0.2087109	0.3076419	0.1163842	0.2666046	0.3076419	0.1163842
2.4771568	0.2896318	0.8775317	0.1691353	0.1733660	0.5139974	0.1691353	0.1733660

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

```r
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
```

```r
print(ar_row_four)
```

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

```r
print(mt_out)
```

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

1.4 Variables in Dataframes

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	2.0737835	1.2469403	-0.7814914	-0.6592099	-0.7814914	-0.6592099
1	0.1636428	-0.2581466	1.5367323	1.8210046	1.5367323	1.8210046

```
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	2.0737835	1.2469403	-0.7814914	-0.6592099	-0.7814914	-0.6592099
2	1	0.1636428	-0.2581466	1.5367323	1.8210046	1.5367323	1.8210046

```
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	2.0737835	1.2469403	-0.7814914	-0.6592099	-0.7814914	-0.6592099
2	1	0.1636428	-0.2581466	1.5367323	1.8210046	1.5367323	1.8210046

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	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931
2	1	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670
3	-1	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153
4	1	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407
5	-1	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931

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

ID	var_one	rho1var	rho2var	rho3var	rho4var	rho5var	rho6var	rho7var	rho8var	rho9var	rho10var
1	-1	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931
2	1	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670
3	-1	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153
4	1	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407
5	-1	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931	-0.3204670	1.7420153	-0.0953407	-1.5841931

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 Factor Label and Combine

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

1.4.2.1 Factor, Label, Cross and Graph

Generate a Scatter plot with different colors representing different categories. There are multiple underlying factor/categorical variables, for example two binary variables. Generate scatter plot with colors for the combinations of these two binary variables.

We combine here the *vs* and *am* variables from the *mtcars* dataset. *vs* is engine shape, *am* is auto or manual shift. We will generate a scatter plot of *mpg* and *qsec* over four categories with different colors.

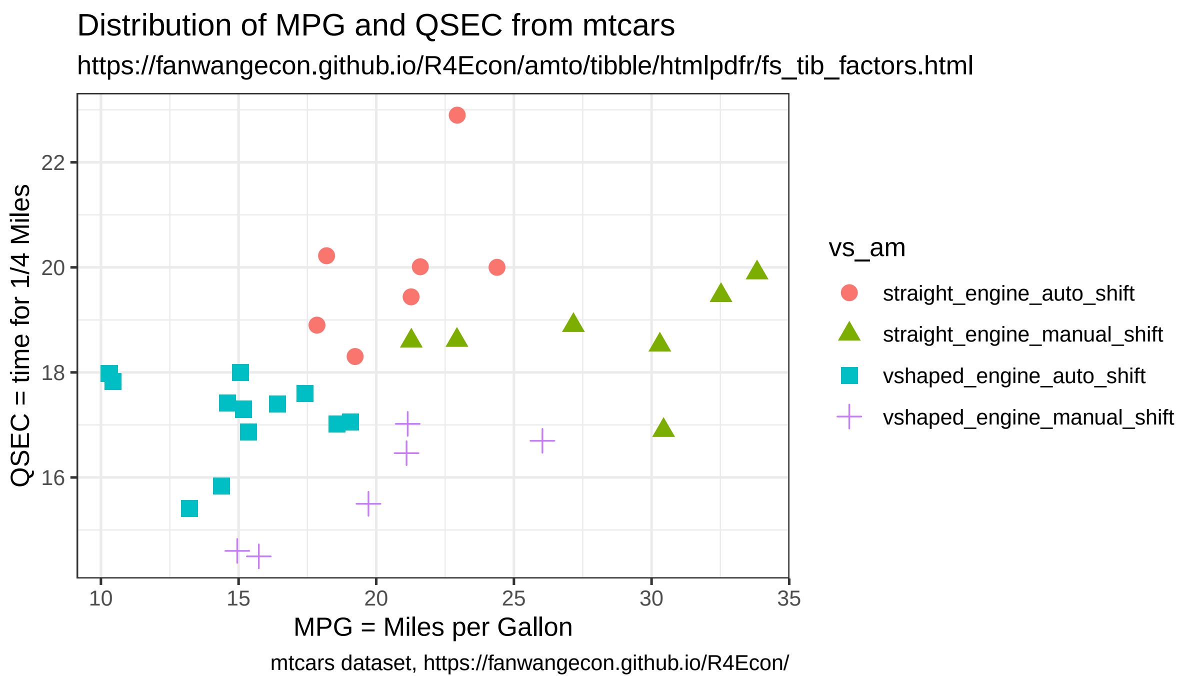
- *am*: Transmission (0 = automatic, 1 = manual)
- *vs*: Engine (0 = V-shaped, 1 = straight)
- *mpg*: miles per galon
- *qsec*: 1/4 mile time

```
# First make sure these are factors
tb_mtcars <- as_tibble(mtcars) %>%
  mutate(vs = as_factor(vs), am = as_factor(am))

# Second Label the Factors
am_levels <- c(auto_shift = "0", manual_shift = "1")
vs_levels <- c(vshaped_engine = "0", straight_engine = "1")
tb_mtcars <- tb_mtcars %>%
  mutate(vs = fct_recode(vs, !!!vs_levels),
         am = fct_recode(am, !!!am_levels))

# Third Combine Factors
tb_mtcars_selected <- tb_mtcars %>%
  mutate(vs_am = fct_cross(vs, am, sep='_', keep_empty = FALSE)) %>%
  select(mpg, qsec, vs_am)
print(tb_mtcars_selected)
```

```
## # A tibble: 32 x 3
##      mpg  qsec vs_am
##    <dbl> <dbl> <fct>
##  1  21    16.5 vshaped_engine_manual_shift
##  2  21    17.0 vshaped_engine_manual_shift
##  3  22.8  18.6 straight_engine_manual_shift
##  4  21.4  19.4 straight_engine_auto_shift
##  5  18.7  17.0 vshaped_engine_auto_shift
##  6  18.1  20.2 straight_engine_auto_shift
##  7  14.3  15.8 vshaped_engine_auto_shift
##  8  24.4  20   straight_engine_auto_shift
##  9  22.8  22.9 straight_engine_auto_shift
## 10  19.2  18.3 straight_engine_auto_shift
## # ... with 22 more rows
```

Now we generate scatter plot based on the combined factors

```r
# Labeling
st_title <- paste0('Distribution of MPG and QSEC from mtcars')
st_subtitle <- paste0('https://fanwangecon.github.io/',
                      'R4Econ/amto/tibble/htmlpdfr/fs_tib_factors.html')
st_caption <- paste0('mtcars dataset, ',
                     'https://fanwangecon.github.io/R4Econ/')
st_x_label <- 'MPG = Miles per Gallon'
st_y_label <- 'QSEC = time for 1/4 Miles'

# Graphing
plt_mtcars_scatter <-
  ggplot(tb_mtcars_selected,
         aes(x=mpg, y=qsec, colour=vs_am, shape=vs_am)) +
  geom_jitter(size=3, width = 0.15) +
  labs(title = st_title, subtitle = st_subtitle,
       x = st_x_label, y = st_y_label, caption = st_caption) +
  theme_bw()

# show
print(plt_mtcars_scatter)
```



1.4.3 Draw Random Rows

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

1.4.3.1 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.

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

```r
# 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

```r
kable(df_rand_sub_a) %>% 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

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

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

1.4.3.2 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.

```r
# 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

1.4.4 Variable NA 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.

```
# 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
```

1.4.5 String Values

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

1.4.5.1 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

```r
# 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:

```
# 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
```

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
```

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
```

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
```

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
```

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
```

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,
                                             vars.not.groups2avg,
                                             vars.indi.grp, display=FALSE)
```

This is the tabular version of results

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

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

```
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
```

```
df.avg.allvars.wide[1:20,] %>% kable() %>% kable_styling_fc_wide()
```

S.country	indi.id	cal_m12_c1	cal_m12_c2	cal_m12t24_c0	cal_m12t24_c1	cal_m3_c1	cal_m3_c2	cal_m3_c3	cal_m3_c4	cal_m3_c5	cal_m3_c6	cal_m3_c7	cal_m3_c8
Cebu	1	132.15714	NaN	97.08333	342.6000	9.10	95.50	85.3	315.30	NaN	NaN	NaN	NaN
Cebu	2	90.72857	255.6500	81.46667	240.0286	83.35	12.30	155.1	144.35	228.0	152.85	305.0	347.60
Cebu	3	96.80000	658.8167	31.56667	634.4429	0.50	28.85	57.0	280.95	459.3	549.95	612.0	890.85
Cebu	4	27.45714	371.7000	24.55000	325.0143	4.50	25.95	39.4	45.95	221.2	271.00	581.3	442.85
Cebu	5	101.34286	1080.8500	79.15000	959.9429	14.10	143.80	71.3	161.15	452.6	1345.20	1178.1	1082.00
Cebu	6	185.35714	521.5333	162.23333	493.3286	23.85	184.70	169.1	355.65	653.4	506.50	416.8	523.00
Cebu	7	157.25714	570.9800	145.50000	513.7833	8.30	137.80	407.8	200.40	390.6	637.10	688.1	569.55
Cebu	8	471.92857	844.8333	379.20000	871.0429	158.95	423.00	417.5	861.05	691.3	897.95	637.1	972.35
Cebu	9	32.27143	415.2167	16.58333	373.9571	5.05	10.40	15.1	89.95	142.4	203.60	753.2	594.25
Cebu	10	67.18571	395.2500	68.58333	347.1857	9.55	26.40	164.6	116.90	296.6	303.00	385.1	541.90
Cebu	11	14.90000	245.3833	11.80000	215.1143	0.50	5.20	30.0	31.45	126.2	223.05	239.6	330.20
Cebu	12	453.61429	745.6833	419.51667	733.1857	325.60	483.65	463.0	546.90	766.8	676.85	998.5	677.55
Cebu	13	47.51429	210.2500	36.81667	196.1714	17.45	40.00	28.5	94.60	216.9	195.15	281.3	186.50
Cebu	14	608.85714	924.5833	527.30000	949.3857	259.05	554.30	688.1	973.60	525.5	1039.60	800.0	1071.40
Cebu	15	74.67143	440.0833	64.21667	396.8429	62.40	39.60	80.1	119.30	292.5	237.45	607.8	632.65
Cebu	16	128.45714	519.9333	90.50000	496.5429	4.80	11.00	205.3	331.15	290.8	354.50	563.3	778.25
Cebu	17	130.78571	718.8667	97.48333	663.4000	5.50	7.80	249.9	319.50	774.6	892.90	551.9	600.45
Cebu	18	172.64000	497.9500	172.64000	497.9500	29.60	234.40	NaN	335.20	NaN	NaN	NaN	497.95
Cebu	19	74.45714	314.7333	80.10000	275.5714	3.65	95.70	171.3	75.60	131.3	350.50	304.6	375.75
Cebu	20	110.90000	583.2000	80.51667	541.7714	7.30	120.65	77.8	221.30	391.2	582.10	466.1	738.85

indi.id	cal_m6_c1	cal_m6_c2	cal_m6_c3	cal_m6_c4	cal_m8t24_c0	cal_m8t24_c1	prot_m12_c1	prot_m12_c2	prot_m12t24_c0	prot_m12t24_c1	prot_m3_c1	prot_m3_c2
1	52.300	238.63333	NaN	NaN	52.300	238.6333	5.3571429	NaN	4.3666667	11.300000	0.65	3.65
2	47.825	147.93333	177.9000	333.4000	47.825	219.7444	3.1857143	8.550000	2.7333333	8.171429	3.20	1.25
3	14.675	206.30000	519.7333	797.9000	14.675	507.9778	4.5000000	21.116667	1.6833333	21.157143	1.05	2.15
4	15.225	43.76667	254.4000	489.0000	15.225	262.3889	0.8714286	6.850000	0.9000000	5.971429	0.75	1.10
5	78.950	131.20000	1047.6667	1114.0333	78.950	764.3000	2.4000000	19.483333	2.3166667	17.114286	1.35	3.00
6	104.275	293.46667	555.4667	487.6000	104.275	445.5111	8.4000000	15.116667	7.3833333	15.028571	0.85	7.40
7	73.050	269.53333	513.8500	609.0667	73.050	457.9375	3.3000000	20.440000	2.7833333	18.100000	0.95	1.70
8	290.975	713.20000	829.0667	860.6000	290.975	800.9556	13.6857143	32.716667	11.0166667	32.285714	3.90	11.35
9	7.725	65.00000	183.2000	647.2333	7.725	298.4778	0.9571429	18.283333	0.9166667	15.842857	0.50	0.50
10	17.975	132.80000	300.8667	489.6333	17.975	307.7667	2.0428571	8.466667	1.9333333	7.642857	0.85	0.50
11	2.850	30.96667	190.7667	300.0000	2.850	173.9111	1.0285714	6.833333	1.1166667	5.928571	0.80	1.70
12	404.625	518.93333	706.8333	784.5333	404.625	670.1000	14.7714286	22.133333	14.1666667	21.600000	7.40	13.60
13	28.725	72.56667	202.4000	218.1000	28.725	164.3556	1.9571429	6.800000	1.6666667	6.357143	0.70	1.50
14	406.675	878.43333	868.2333	980.9333	406.675	909.2000	24.6714286	28.050000	23.0833333	28.928571	15.05	28.85
15	51.000	106.23333	255.8000	624.3667	51.000	328.8000	2.2571429	10.633333	1.6333333	9.971429	1.65	0.90
16	7.900	289.20000	333.2667	706.6000	7.900	443.0222	4.6857143	18.083333	4.1000000	16.671429	0.65	2.70
17	6.650	296.30000	853.4667	584.2667	6.650	578.0111	2.8142857	19.716667	2.2166667	17.814286	0.60	0.50
18	132.000	335.20000	NaN	497.9500	132.000	443.7000	11.7000000	19.100000	11.7000000	19.100000	2.95	17.90
19	49.675	107.50000	277.4333	352.0333	49.675	245.6556	2.5428571	10.466667	2.8833333	9.042857	0.50	2.95
20	63.975	173.46667	518.4667	647.9333	63.975	446.6222	3.2000000	16.966667	2.1833333	15.871429	0.50	3.85

indi.id	prot_m3_c3	prot_m3_c4	prot_m3_c5	prot_m3_c6	prot_m3_c7	prot_m3_c8	prot_m6_c1	prot_m6_c2	prot_m6_c3	prot_m6_c4	prot_m8t24_c0	prot_m8t24_c1
1	2.6	13.15	NaN	NaN	NaN	NaN	2.150	9.6333333	NaN	NaN	2.150	9.633333
2	5.2	4.10	5.4	5.15	7.7	13.95	2.225	4.4666667	5.233333	11.866667	2.225	7.188889
3	2.3	11.40	18.5	18.05	18.0	27.05	1.600	8.3666667	18.200000	24.033333	1.600	16.866667
4	1.2	0.60	1.8	4.85	10.1	9.75	0.925	0.8000000	3.833333	9.866667	0.925	4.833333
5	3.4	2.35	7.1	23.15	24.5	19.50	2.175	2.7000000	17.800000	21.166667	2.175	13.888889
6	9.8	16.25	26.8	14.10	11.4	12.15	4.125	14.1000000	18.333333	11.900000	4.125	14.777778
7	8.6	4.60	16.4	23.00	21.5	20.65	1.325	5.9333333	19.700000	20.933333	1.325	15.000000
8	10.8	27.25	42.7	26.45	25.8	37.45	7.625	21.7666667	31.866667	33.566667	7.625	29.066667
9	0.5	2.10	4.2	10.85	39.5	22.15	0.500	1.5666667	8.633333	27.933333	0.500	12.711111
10	4.9	3.35	7.5	6.05	9.2	11.00	0.675	3.8666667	6.533333	10.400000	0.675	6.933333
11	1.2	0.50	3.6	6.35	7.3	8.70	1.250	0.7333333	5.433333	8.233333	1.250	4.800000
12	26.6	17.40	18.6	20.35	25.6	23.95	10.500	20.4666667	19.766667	24.500000	10.500	21.577778
13	2.1	3.60	8.3	4.75	6.2	8.40	1.100	3.1000000	5.933333	7.666667	1.100	5.566667
14	32.6	26.15	12.7	29.75	29.5	33.30	21.950	28.3000000	24.066667	32.033333	21.950	28.133333
15	1.5	4.60	9.4	5.80	13.6	14.60	1.275	3.5666667	7.000000	14.266667	1.275	8.277778
16	8.7	8.70	7.8	12.95	25.9	24.45	1.675	8.7000000	11.233333	24.933333	1.675	14.955556
17	6.1	5.70	20.1	21.55	24.9	15.10	0.550	5.8333333	21.066667	18.366667	0.550	15.088889
18	NaN	16.80	NaN	NaN	NaN	19.10	10.425	16.8000000	NaN	19.100000	10.425	18.333333
19	6.7	2.10	3.4	11.40	12.3	12.15	1.725	3.6333333	8.733333	12.200000	1.725	8.188889
20	2.7	5.50	7.9	20.15	11.8	20.90	2.175	4.5666667	16.066667	17.866667	2.175	12.833333

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)
```

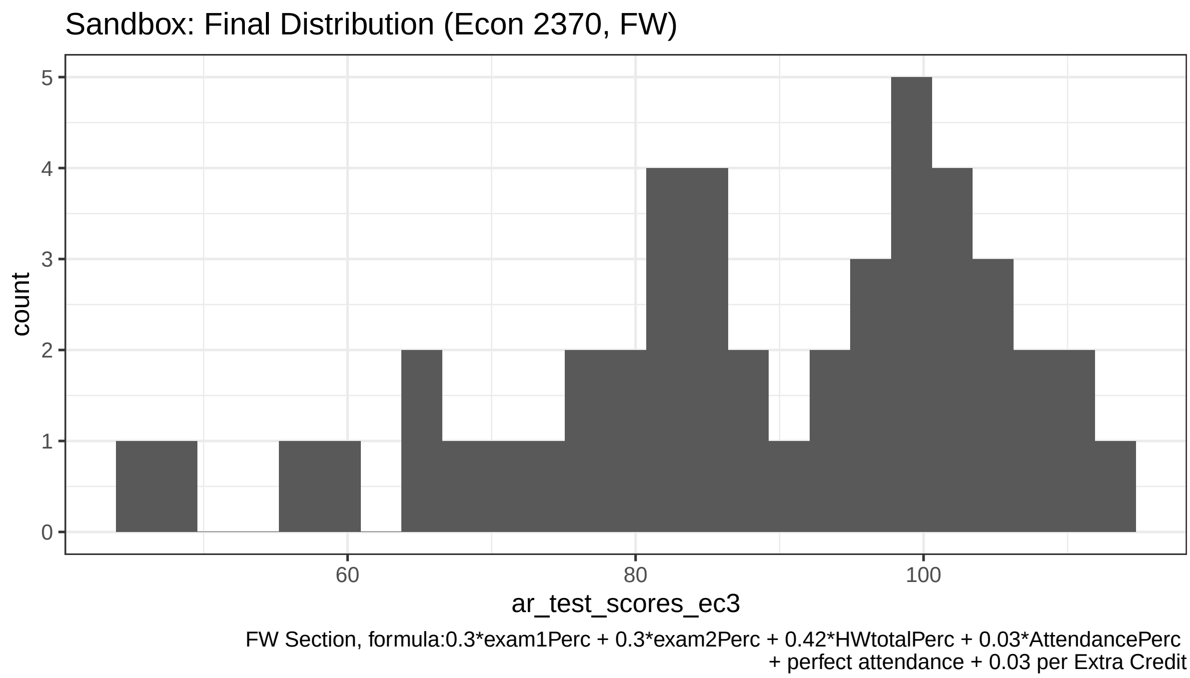
2.4.1.1.3 A Dataset with Multiple Variables

```
##  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
```

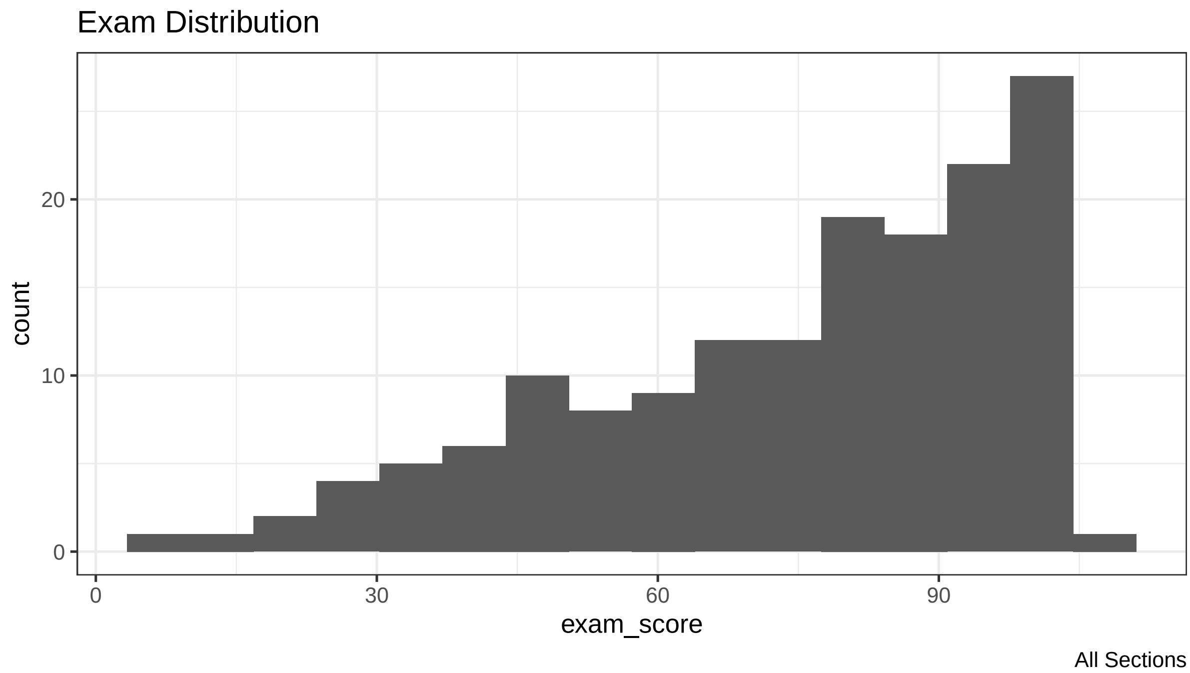
2.4.1.2 Test Score Distributions

```
ggplot(tb_final_twovar, aes(x=ar_test_scores_ec3)) +
  geom_histogram(bins=25) +
  labs(title = paste0('Sandbox: Final Distribution (Econ 2370, FW)'),
       caption = paste0('FW Section, formula:',
                        '0.3*exam1Perc + 0.3*exam2Perc + ',
                        '0.42*HWtotalPerc + 0.03*AttendancePerc \n',
                        '+ perfect attendance + 0.03 per Extra Credit')) +
  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()
```



2.5 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()
```

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

```
# 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	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

```
# 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

```
# 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.

```
# 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)
```

```
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
```

```
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.

```
# 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
```

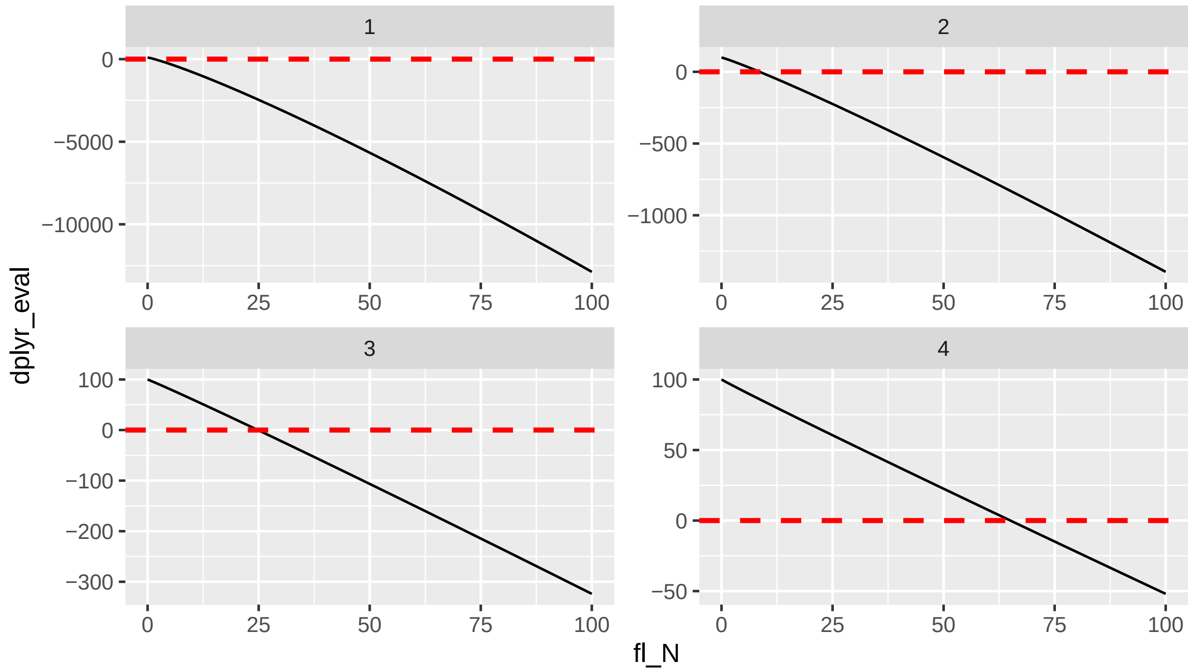
```
summary(tb_states_choices_dense_eval)
```

```
##     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.

```r
# 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)

# 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

```
# 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
```

```
# 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"
```

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

```
kable(tb_income_full_dd) %>%
  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

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
```

```r
it_N <- 4
fl_unif_min <- 1
fl_unif_max <- 2
mt_draw_define <- cbind(sample(it_N, it_N, replace=TRUE),
                        runif(it_N, min=1, max=10))
tb_draw_define <- as_tibble(mt_draw_define) %>%
  rowid_to_column(var = "draw_group")
print(tb_draw_define)
```

```
## # A tibble: 4 x 3
##   draw_group    V1    V2
##        <int> <dbl> <dbl>
## 1          1     4  5.36
## 2          2     2  7.65
## 3          3     1  9.57
## 4          4     3  9.62
```

```r
# apply row by row, anonymous function has hard
# coded min and max
ls_ar_draws_shares_lvls =
  apply(tb_draw_define,
        1,
        function(row) {
          it_draw <- row[2]
          fl_sum <- row[3]
          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]]$ar_share
## [1] 0.2783638 0.2224140 0.2797840 0.2194381
##
## [[1]]$ar_levels
## [1] 1.492414 1.192446 1.500028 1.176491
##
##
## [[2]]
## [[2]]$ar_share
## [1] 0.5052919 0.4947081
##
## [[2]]$ar_levels
## [1] 3.866528 3.785541
##
##
## [[3]]
## [[3]]$ar_share
## [1] 1
##
## [[3]]$ar_levels
##       V2
```

```
## 9.572211
##
##
## [[4]]
## [[4]]$ar_share
## [1] 0.4211426 0.2909812 0.2878762
##
## [[4]]$ar_levels
## [1] 4.051971 2.799640 2.769765
```

We will try to do the same thing as above, but now the output will be a stacked dataframe. Note that within each element of the apply row by row loop, we are generating two variables *ar_share* and *ar_levels*. We will not generate a dataframe with multiple columns, storing *ar_share*, *ar_levels* as well as information on *min*, *max*, number of draws and rescale total sum.

```r
set.seed(1039)
# apply row by row, anonymous function has hard coded min and max
ls_mt_draws_shares_lvls =
  apply(tb_draw_define, 1, function(row) {

    it_draw_group <- row[1]
    it_draw <- row[2]
    fl_sum <- row[3]

    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

    mt_all_res <- cbind(it_draw_group, it_draw, fl_sum,
                        ar_unif, ar_share, ar_levels)
    colnames(mt_all_res) <-
      c('draw_group', 'draw_count', 'sum',
        'unif_draw', 'share', 'rescale')
    rownames(mt_all_res) <- NULL

    return(mt_all_res)
  })
mt_draws_shares_lvls_all <- do.call(rbind, ls_mt_draws_shares_lvls)
# Show Results
kable(mt_draws_shares_lvls_all) %>% kable_styling_fc()
```

draw_group	draw_count	sum	unif_draw	share	rescale
1	4	5.361378	1.125668	0.1988606	1.066167
1	4	5.361378	1.668536	0.2947638	1.580340
1	4	5.361378	1.419382	0.2507483	1.344356
1	4	5.361378	1.447001	0.2556274	1.370515
2	2	7.652069	1.484598	0.4605236	3.523959
2	2	7.652069	1.739119	0.5394764	4.128110
3	1	9.572211	1.952468	1.0000000	9.572211
4	3	9.621375	1.957931	0.3609352	3.472693
4	3	9.621375	1.926995	0.3552324	3.417824
4	3	9.621375	1.539678	0.2838324	2.730858

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) {
```

```r
                              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
```

```r
# 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
```

```r
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

```r
# 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()
```

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

```r
# 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_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 vector
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, . , '')))
```

```
              )
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

```
# 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:

```
# 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
```

```
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) {

  # 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(!is.infinite(.)))

  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)

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

  # D. Combine Regression Results into a Matrix
  df.results <- suppressWarnings(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 <- suppressWarnings(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()
## )
```

```

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

5.1.1.2.1 Example No Instrument, OLS

```

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

esti.val	value
(Intercept)_Estimate	52.1186286658651
prot_Estimate	0.374472386357917
sexMale_Estimate	0.611043720578292
hgt0_Estimate	0.148513781160842
wgt0_Estimate	0.00150560230505631
(Intercept)_Std.Error	1.57770483608693
prot_Std.Error	0.00418121191133815
sexMale_Std.Error	0.118396259120659
hgt0_Std.Error	0.0393807494783186
wgt0_Std.Error	0.000187123663624397
(Intercept)_tvalue	33.0344608660332
prot_tvalue	89.5607288744356
sexMale_tvalue	5.16100529794248
hgt0_tvalue	3.77122790013449
wgt0_tvalue	8.04602836377991
(Intercept)_Pr(>|t|)	9.92126150975783e-233
prot_Pr(>|t|)	0
sexMale_Pr(>|t|)	2.48105505495642e-07
hgt0_Pr(>|t|)	0.000162939618371183
wgt0_Pr(>|t|)	9.05257561534111e-16
df1_v	5
df2_v	18958
df3_v	5
sigma_v	8.06197784622979
r.squared_v	0.319078711001325
adj.r.squared_v	0.318935041565942
vars_var.y	hgt
vars_vars.x	prot
vars_vars.c	sex+hgt0+wgt0

5.1.1.2.2 Example 1 Insturment

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

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

5.1.1.2.3 Example Multiple Instrucments

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

```
# Multiple Instrucments
var.y <- c('hgt')
vars.x <- c('prot', 'cal')
vars.z <- c('momEdu', 'wealthIdx', 'p.A.prot', 'p.A.nProt')
```

esti.val	value
(Intercept)_Estimate	43.4301969117558
prot_Estimate	0.130833343849446
sexMale_Estimate	0.868121847262411
hgt0_Estimate	0.412093881817148
wgt0_Estimate	0.000858630042617921
(Intercept)_Std.Error	1.82489550971182
prot_Std.Error	0.0192036220809189
sexMale_Std.Error	0.13373016700542
hgt0_Std.Error	0.0459431912927002
wgt0_Std.Error	0.00022691057702563
(Intercept)_zvalue	23.798730766023
prot_zvalue	6.81295139521853
sexMale_zvalue	6.49159323361366
hgt0_zvalue	8.96963990141069
wgt0_zvalue	3.7840018472164
(Intercept)_Pr(>|z|)	3.4423766196876e-125
prot_Pr(>|z|)	9.56164541643828e-12
sexMale_Pr(>|z|)	8.49333228172763e-11
hgt0_Pr(>|z|)	2.97485394526792e-19
wgt0_Pr(>|z|)	0.000154326676608523
Weakinstruments_df1	1
Wu-Hausman_df1	1
Sargan_df1	0
Weakinstruments_df2	16394
Wu-Hausman_df2	16393
Weakinstruments_statistic	935.817456612075
Wu-Hausman_statistic	123.595856606729
Weakinstruments_p-value	6.39714929178024e-200
Wu-Hausman_p-value	1.30703637796748e-28
vars_var.y	hgt
vars_vars.x	prot
vars_vars.z	momEdu
vars_vars.c	sex+hgt0+wgt0

```
vars.c <- c('sex', 'hgt0', 'wgt0')
# Regression
regf.iv(var.y, vars.x, vars.c, vars.z, df, transpose=FALSE) %>%
  kable() %>%
  kable_styling_fc()
```

5.1.1.2.4 Example Multiple Endogenous Variables

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

5.1.1.2.5 Examples Line by Line The examples are just to test the code with different types of variables.

```
# 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')
```

```
# A. create Equation
str.vars.x <- paste(vars.x, collapse='+')
```

esti.val	value
(Intercept)__Estimate	42.2437613555242
prot__Estimate	0.26699945194704
sexMale__Estimate	0.695548488812932
hgt0__Estimate	0.424954881263031
wgt0__Estimate	0.000486951420329484
(Intercept)__Std.Error	1.85356686789642
prot__Std.Error	0.0154939347964083
sexMale__Std.Error	0.133157977814374
hgt0__Std.Error	0.0463195803786233
wgt0__Std.Error	0.000224867994873235
(Intercept)__zvalue	22.7905246296649
prot__zvalue	17.2325142357597
sexMale__zvalue	5.22348341593581
hgt0__zvalue	9.17441129192849
wgt0__zvalue	2.16549901022595
(Intercept)__Pr(>|z|)	5.69294074735747e-115
prot__Pr(>|z|)	1.51424021931607e-66
sexMale__Pr(>|z|)	1.75588197502565e-07
hgt0__Pr(>|z|)	4.54048595587756e-20
wgt0__Pr(>|z|)	0.030349491114332
Weakinstruments__df1	4
Wu-Hausman__df1	1
Sargan__df1	3
Weakinstruments__df2	14914
Wu-Hausman__df2	14916
Weakinstruments__statistic	274.147084958343
Wu-Hausman__statistic	17.7562545747101
Sargan__statistic	463.729664547249
Weakinstruments__p-value	8.61731956233366e-228
Wu-Hausman__p-value	2.52567249124181e-05
Sargan__p-value	3.45452874915475e-100
vars__var.y	hgt
vars__vars.x	prot
vars__vars.z	momEdu+wealthIdx+p.A.prot+p.A.nProt
vars__vars.c	sex+hgt0+wgt0

```
str.vars.c <- paste(vars.c, collapse='+')
str.vars.z <- paste(vars.z, collapse='+')
print(str.vars.x)
```

```
## [1] "prot+cal"
```

```
print(str.vars.c)
```

```
## [1] "sex+hgt0+wgt0"
```

```
print(str.vars.z)
```

```
## [1] "momEdu+wealthIdx+p.A.prot+p.A.nProt"
```

```
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)
```

esti.val	value
(Intercept)_Estimate	44.0243196254297
prot_Estimate	-1.4025623247106
cal_Estimate	0.065104895750151
sexMale_Estimate	0.120832787571818
hgt0_Estimate	0.286525437984517
wgt0_Estimate	0.000850481389651033
(Intercept)_Std.Error	2.75354847244082
prot_Std.Error	0.198640060273635
cal_Std.Error	0.00758881298880996
sexMale_Std.Error	0.209984580636303
hgt0_Std.Error	0.0707828182888255
wgt0_Std.Error	0.000337112104444429
(Intercept)_zvalue	15.9882130516502
prot_zvalue	-7.06082309267581
cal_zvalue	8.57906181719737
sexMale_zvalue	0.575436478267434
hgt0_zvalue	4.04795181812859
wgt0_zvalue	2.52284441418383
(Intercept)_Pr(>|z|)	1.54396598126854e-57
prot_Pr(>|z|)	1.65519210848649e-12
cal_Pr(>|z|)	9.56500648203187e-18
sexMale_Pr(>|z|)	0.564996139463599
hgt0_Pr(>|z|)	5.16677787108928e-05
wgt0_Pr(>|z|)	0.0116409892837831
Weakinstruments(prot)_df1	4
Weakinstruments(cal)_df1	4
Wu-Hausman_df1	2
Sargan_df1	2
Weakinstruments(prot)_df2	14914
Weakinstruments(cal)_df2	14914
Wu-Hausman_df2	14914
Weakinstruments(prot)_statistic	274.147084958343
Weakinstruments(cal)_statistic	315.036848606231
Wu-Hausman_statistic	94.7020085425169
Sargan_statistic	122.081979628898
Weakinstruments(prot)_p-value	8.61731956233366e-228
Weakinstruments(cal)_p-value	1.18918641220866e-260
Wu-Hausman_p-value	1.35024050408262e-41
Sargan_p-value	3.09196773720398e-27
vars_var.y	hgt
vars_vars.x	prot+cal
vars_vars.z	momEdu+wealthIdx+p.A.prot+p.A.nProt
vars_vars.c	sex+hgt0+wgt0

```
## [1] "hgt~prot+cal+sex+hgt0+wgt0|momEdu+wealthIdx+p.A.prot+p.A.nProt+sex+hgt0+wgt0"
```

```
# 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
```

```
# 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
```

```
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
```

```
# 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
```

```
# G. Results as Single Row
# df.row.results
```

```
# t(df.row.results %>% spread(esti.val, value)) %>%
#   kable() %>%
#   kable_styling_fc_wide()
```

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, vars.z=vars.z, df=df))
                       )))

  } 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, vars.z=vars.z, df=df)) %>%
                  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
suppressWarnings(suppressMessages(
  ff_reg_mbyn(list.vars.y, list.vars.x,
              vars.c, vars.z, df,
              return_all = FALSE,
              stats_ends = 'value'))) %>%
  kable() %>%
  kable_styling_fc_wide()
```

vars_var.y	prot_tvalue	cal_tvalue	wealthIdx_tvalue	p.A.prot_tvalue	p.A.nProt_tvalue
hgt	18.8756010031786	23.4421863484661	13.508899618216	3.83682180045518	32.5448257554855
wgt	16.3591125056062	17.3686031309332	14.1390521528113	1.36958319982295	12.0961557911467
vil.id	-14.9385580468907	-19.6150110809452	34.0972558327347	8.45943342783186	17.7801422421419

5.1.2.3.1 Test Program OLS Z-Stat

```
vars.z <- c('indi.id')
suppressWarnings(suppressMessages(
  ff_reg_mbyn(list.vars.y, list.vars.x,
```

```r
              vars.c, vars.z, df,
              return_all = FALSE,
              stats_ends = 'value'))) %>%
  kable() %>%
  kable_styling_fc_wide()
```

vars_var.y	prot_zvalue	cal_zvalue	wealthIdx_zvalue	p.A.prot_zvalue	p.A.nProt_zvalue
hgt	8.87674929300964	12.0739764947235	4.62589553677969	26.6373587567312	32.1162192385744
wgt	5.60385871756365	6.1225187008946	5.17869536991717	11.9295584469998	12.3509307017263
vil.id	-9.22106223347162	-13.0586007975839	-51.5866689219593	-29.9627476577329	-38.3528894620707

5.1.2.3.2 Test Program IV T-stat

```r
vars.z <- NULL
suppressWarnings(suppressMessages(
  ff_reg_mbyn(list.vars.y, list.vars.x,
              vars.c, vars.z, df,
              return_all = FALSE,
              stats_ends = 'Estimate'))) %>%
  kable() %>%
  kable_styling_fc_wide()
```

vars_var.y	prot_Estimate	cal_Estimate	wealthIdx_Estimate	p.A.prot_Estimate	p.A.nProt_Estimate
hgt	0.049431093806755	0.00243408846205622	0.21045655488185	3.86952250259526e-05	0.00542428867316449
wgt	16.5557424523585	0.699072500364623	106.678721085969	0.00521731297924587	0.779514232050632
vil.id	-0.0758835879205584	-0.00395676177098486	0.451733304543324	0.000149388430455142	0.00526237555581024

5.1.2.3.3 Test Program OLS Coefficient

```r
vars.z <- c('indi.id')
suppressWarnings(suppressMessages(
  ff_reg_mbyn(list.vars.y, list.vars.x,
              vars.c, vars.z, df,
              return_all = FALSE,
              stats_ends = 'Estimate'))) %>%
  kable() %>%
  kable_styling_fc_wide()
```

vars_var.y	prot_Estimate	cal_Estimate	wealthIdx_Estimate	p.A.prot_Estimate	p.A.nProt_Estimate
hgt	0.859205733632614	0.0238724384575419	0.144503490136948	0.00148073028434642	0.0141317656200726
wgt	98.9428234201406	2.71948246216953	69.1816142883022	0.221916473012486	2.11856940494335
vil.id	-6.02451379136132	-0.168054407187466	-1.91414470908345	-0.00520794333267238	-0.0494468877742109

5.1.2.3.4 Test Program IV coefficient

```r
vars.z <- NULL
t(suppressWarnings(suppressMessages(
  ff_reg_mbyn(list.vars.y, list.vars.x,
             vars.c, vars.z, df,
             return_all = TRUE,
             stats_ends = 'Estimate')))) %>%
  kable() %>%
  kable_styling_fc_wide()
```

5.1.2.3.5 Test Program OLS Return All

X.Intercept._Estimate	27.3528514188608	99.873884728925	31.4646660224049	27.9038445914729	219.626705179399	30.5103987898551	35.7840188807906	-2662.74787734003	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
X.Intercept._Pr...t..	5.68247182214952e-231	0.75529705553815	6.78164655340399e-84	8.24252673989353e-242	0.493216914827181	1.62608789535248e-79	2.26726906489443e-145	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
X.Intercept._Std.Error	0.831272666092284	320.450650378664	1.61328519718754	0.828072565159449	320.522532223672	1.60831193651104	1.38461348429899	670.301542938561	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
X.Intercept._tvalue	32.9047886867776	0.31166697465244	19.503474077155	33.6973421962119	0.685214557790078	18.9704485163756	25.8440491058106	-3.97246270039407	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
adj.r.squared_v	0.814249026159781	0.60716936506893	0.0373247512680971	0.81608922805658	0.607863678511207	0.0453498711076042	0.935014931990565	0.92193683733695	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
df1_v	6	6	6	6	6	6	6	6	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
df2_v	18957	18962	18999	18957	18962	18999	25092	25102	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
df3_v	6	6	6	6	6	6	6	6	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
hgt0_Estimate	0.60391817340617	56.3852027199184	-0.296844389234445	0.589847843438394	52.9707041800704	-0.273219210757899	0.439374451256039	47.176969664749	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
hgt0_Pr...t..	1.14533314566771e-183	1.52417506966835e-12	1.40290395213743e-13	7.79174951119325e-177	3.05720143843395e-11	8.49149153665126e-12	2.71000479249152e-36	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
hgt0_Std.Error	0.0206657538633713	7.96735224000553	0.0401060913799595	0.0205836398278421	7.96822145797115	0.0399777363511633	0.0348701896610764	16.8823489375743	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
hgt0_tvalue	29.2231378249683	7.0770314931977	-7.40147890309685	28.6561486875877	6.64774497790599	-6.83428417151858	12.6002885423502	2.79445531182864	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
prot_Estimate	0.049431093806755	16.5557424523585	-0.0758835879205584	NA	NA	NA	NA	NA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
prot_Pr...t..	9.54769322304645e-79	9.61203373222183e-60	3.56396093562335e-50	NA	NA	NA	NA	NA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
prot_Std.Error	0.00261878251179557	1.01201959743751	0.00507971302734622	NA	NA	NA	NA	NA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
prot_tvalue	18.8756010031786	16.3591125056062	-14.9385580468907	NA	NA	NA	NA	NA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
r.squared_v	0.814298005954592	0.607272921412825	0.0375780335372857	0.816137722617266	0.60796705182314	0.0456010419476623	0.93502787877066	0.921952383432195	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
sexMale_Estimate	0.935177182449406	415.163616765357	-0.254089999175318	0.893484662055608	405.534891838028	-0.181389489610951	1.80682463132073	999.926876716707	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
sexMale_Pr...t..	2.36432111724607e-51	2.48252880290814e-67	0.0343768259467621	2.08765935335877e-47	2.51355675686752e-64	0.129768754080748	1.26527362032354e-66	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
sexMale_Std.Error	0.0618482294097262	23.8518341439675	0.120093045309631	0.0616078355613525	23.8567507583516	0.11972270545355	0.104475287357902	50.5879876531386	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
sexMale_tvalue	15.1205166481668	17.4059409544552	-2.11577613441484	14.5027763743757	16.9987478993157	-1.51508010885476	17.2942776901016	19.7660931597596	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
sigma_v	4.21029844914315	1623.77111076428	8.18491760066961	4.18939119979502	1622.33549880859	8.15073036560541	8.18607049768594	3964.45339913597	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
svymthRound_Estimate	0.87166589100565	189.04290688382	-0.0154759587993917	0.851989049736817	185.318286001897	0.0201471237605442	0.432815253441723	189.877994795061	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
svymthRound_Pr...t..	0	0	0.0397984032097113	0	0	0.0117151185126433	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
svymthRound_Std.Error	0.00387681209575621	1.4955473831309	0.00752730297891317	0.00411253488213795	1.59266949679221	0.00799217807522278	0.000728323735328998	0.352701518968252	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
svymthRound_tvalue	224.840892330022	126.403823119306	-2.05597660181154	207.168832400006	116.357025971267	2.52085521254888	594.262183761197	538.353209678558	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
vars_var.y	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
vars_vars.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
vars_vars.x	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wgt0_Estimate	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wgt0_Pr...t..	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wgt0_Std.Error	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wgt0_tvalue	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
cal_Estimate	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
cal_Pr...t..	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
cal_Std.Error	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
cal_tvalue	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wealthIdx_Estimate	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wealthIdx_Pr...t..	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wealthIdx_Std.Error	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
wealthIdx_tvalue	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.prot_Estimate	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.prot_Pr...t..	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.prot_Std.Error	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.prot_tvalue	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.nProt_Estimate	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.nProt_Pr...t..	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.nProt_Std.Error	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
p.A.nProt_tvalue	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
vars.z <- c('indi.id')
t(suppressWarnings(suppressMessages(
  ff_reg_mbyn(list.vars.y, list.vars.x,
              vars.c, vars.z, df,
              return_all = TRUE,
              stats_ends = 'Estimate')))) %>%
  kable() %>%
  kable_styling_fc_wide()
```

X.Intercept._Estimate	40.2173991882938	1408.1626637032	-64.490636067872	39.6732302990235	1325.54736576331	-59.8304089440729	35.5561817357046	-2791.221534909	21.8005242861645	24.3009261707644	-499.067024090554	21.4632286881661	25.299209739617	-352.278518334717	17.9359211844992
X.Intercept._Pr...z..	3.69748206920405e-59	0.00217397545504963	0.000109756271656929	1.30030240177373e-103	0.00138952700443324	3.75547414421179e-07	2.01357089467444e-142	1.95034793045284e-05	1.17899313785408e-34	1.97968607369592e-84	0.155922992163314	1.84405333738942e-09	1.29388565624566e-157	0.287184942021997	1.13855583530306e-12
X.Intercept._Std.Error	2.47963650430699	459.377029874119	16.673099250727	1.83545587849039	414.645900526211	11.7754321198995	1.39936229104453	653.605248808641	1.77547715237629	1.2481331128579	351.723712333143	3.57067054655531	0.945826571474308	330.990098562619	2.52170174723203
X.Intercept._zvalue	16.2190704639323	3.06537456626657	-3.86794531107106	21.6149190857443	3.19681772828602	-5.08095230263053	25.4088465605032	-4.27050048939585	12.2786847788984	19.4698193008609	-1.41891776582254	6.01097984491234	26.748254386829	-1.0643173915611	7.11262590993832
hgt0_Estimate	0.403139725681418	35.5765914326678	1.20995060148712	0.357976348180876	31.0172706497394	1.5037447089682	0.460434521499963	59.1545587745268	0.412512139031067	0.515794899569023	46.2591615803265	0.520812513246773	0.510868687340428	45.5654716961559	0.534362107844268
hgt0_Pr...z..	1.25009876641748e-13	0.000445802636381424	0.00097112649404847	2.82141265004339e-17	0.0013100303315764	3.70002169470828e-08	2.98739737280869e-37	0.000542570320022534	3.02226357947691e-20	8.57492956381676e-59	2.8561488738123e-07	1.10039023747789e-08	3.24936430168307e-102	6.3454545304127e-08	3.42500501176006e-17
hgt0_Std.Error	0.0543948312973965	10.1318250572006	0.366789440587685	0.0423453726223874	9.65135595900306	0.273179527952317	0.0361031059207763	17.1025823111635	0.0447499166716409	0.0319035514861838	9.01263684093548	0.0911390672920558	0.0237991645877977	8.42434865398195	0.0633800058773461
hgt0_zvalue	7.41136089709158	3.51137048180512	3.29876072644971	8.45373003027063	3.21377335801252	5.50460248701607	12.7533216258548	3.45880859967647	9.21816552325528	16.1673191711084	5.13270005180026	5.71448149208973	21.4658243761363	5.40878275196011	8.4310762436216
prot_Estimate	0.859205733632614	98.9428234201406	-6.02451379136132	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
prot_Pr...z..	6.88427338202428e-19	2.09631602352917e-08	2.94171378745816e-20	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
prot_Std.Error	0.0967928354481331	17.6561952052848	0.653342710289155	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
prot_zvalue	8.87674929300964	5.60385871756365	-9.22106223347162	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Sargan_df1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sexMale_Estimate	0.154043421788007	333.799680049259	5.41175429817609	0.106307556057668	330.452608866758	5.83118942788808	1.80283907885782	997.747599807148	-0.452827875182598	1.02741625216018	411.365911332896	-0.789122421167432	1.02009164592608	409.820707458838	-0.746032636368145
sexMale_Pr...z..	0.38807812932888	5.06413216642981e-24	5.80077629932476e-06	0.423490075745117	2.52735690930834e-27	6.12283824664132e-12	1.1689328480129e-65	2.02347084785411e-89	0.000647195788038449	1.69796551008584e-27	2.05327249429949e-54	0.00428270841484855	1.70848440093529e-51	2.36314216739034e-62	6.57521045473888e-05
sexMale_Std.Error	0.178475271469781	33.0216035385405	1.19371921154418	0.132821186086547	30.5174257711927	0.847955715223327	0.105343525210948	49.7632792630648	0.132754263303719	0.0945646985181925	26.4822313532216	0.276250047248363	0.0675715533063635	24.5920104216267	0.18692145837209
sexMale_zvalue	0.86310792817082	10.1085242471545	4.53352366774387	0.800381017440976	10.8283251459136	6.87676174970095	17.113904962338	20.0498764266063	-3.41102322376347	10.8646912458831	15.5336574870174	-2.85655126226267	15.0964658352764	16.6647907361992	-3.99115565898846
svymthRound_Estimate	0.20990165085783	121.78985943172	4.84745570027424	0.322893837128574	135.494858749214	4.07024693316581	0.433164820953121	190.07735139541	0.0137438264666969	1.00582859923509	218.549980922774	-0.369567838754916	0.929266902426869	207.078222946319	-0.0985678389223824
svymthRound_Pr...z..	0.00846239710392287	5.96047652813855e-17	2.07373887977152e-19	9.66146445882893e-11	4.48931446042076e-34	5.64723572160763e-36	0	0	1.57416908709431e-66	0	0	2.42696379701225e-102	0	0	1.84569897952709e-27
svymthRound_Std.Error	0.0797183179471441	14.5577085129475	0.538050140685815	0.0498896912188091	11.133488331472	0.325043349284718	0.00120472816008751	0.739269879490032	0.000797655931686456	0.00746867714609297	1.9315711781906	0.0172056989832505	0.00539330635998817	1.46167854745858	0.00907867488118012
svymthRound_zvalue	2.63304164291327	8.36600480930094	9.00930105527994	6.47215545416802	12.1700274626596	12.5221664806331	359.553993426746	257.11496798237	17.2302692435808	134.672925279848	113.146221785884	-21.4793853545086	172.300040161061	141.671520941705	-10.8570733297996
vars_var.y	hgt	wgt	vil.id	hgt	wgt	vil.id	hgt	wgt	vil.id	hgt	wgt	vil.id	hgt	wgt	vil.id
vars_vars.c	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound	sex+wgt0+hgt0+svymthRound
vars_vars.x	prot	prot	prot	cal	cal	cal	wealthIdx	wealthIdx	wealthIdx	p.A.prot	p.A.prot	p.A.prot	p.A.nProt	p.A.nProt	p.A.nProt
vars_vars.z	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id	indi.id
Weakinstruments_df1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Weakinstruments_df2	18957	18962	18999	18957	18962	18999	25092	25102	30013	18587	18591	18845	18587	18591	18845
Weakinstruments_p.value	1.42153759923994e-19	4.45734829676713e-19	5.72345606957941e-20	1.77770827184424e-37	4.03760292920738e-37	5.47447735093002e-38	0	0	0	0	0	0	0	0	0
Weakinstruments_statistic	82.0931934821266	79.8251182827386	83.8989817367586	164.392129625299	162.747072038429	166.75260665498	7029.47383089383	7038.38467113128	12942.6315513372	1710.98122418591	1715.15052113399	1725.71954882902	5097.88462603711	5110.7741807338	5136.55662964887
wgt0_Estimate	-0.00163274724538111	0.492582112313709	0.00999798623641602	-0.000658938519302931	0.601258436431587	0.00326074237566435	0.00112485055604169	1.27282038539707	-0.00512158791392237	0.000716628918444932	0.761704518610475	-0.00601345031606092	0.000922100117259348	0.792700893714085	-0.00668277875606482
wgt0_Pr...z..	4.88365163639597e-08	2.33136555228405e-20	7.95432753711715e-07	0.000328431149807424	2.0921134733036e-48	0.00667886646012294	2.26123807446765e-11	6.67525280062144e-56	6.51923753120087e-127	2.43477572076212e-06	8.2201479288098e-69	5.19751747217521e-44	1.68237436753105e-15	4.81415543564975e-82	2.54848840100353e-105
wgt0_Std.Error	0.00029928487659495	0.0532753838702833	0.00202532507408065	0.000183457551985601	0.0411255751282477	0.00120214094164169	0.000168187467853553	0.08080475140115	0.000213715312589078	0.000152036990658929	0.0434474820359048	0.00043218241369976	0.00011580150512068	0.0413159097814445	0.000306609919182859
wgt0_zvalue	-5.45549532591606	9.24596082710666	4.93648469787221	-3.59177647456371	14.6200614716414	2.71244598924594	6.68807593334564	15.7518012657231	-23.9645341827701	4.71351685756907	17.531614789115	-13.9141485757875	7.96276452796019	19.1863351892132	-21.7957030675165
Wu.Hausman_df1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Wu.Hausman_df2	18956	18961	18998	18956	18961	18998	25091	25101	30012	18586	18590	18844	18586	18590	18844
Wu.Hausman_p.value	1.53929570343279e-118	3.13415891402799e-08	0	2.88592507054107e-108	7.6495944085204e-07	0	0.0221987672063003	0.0099360023036833	0	1.80909125272768e-238	2.14946499922491e-35	0	3.15182965429765e-108	1.7681125741529e-17	0
Wu.Hausman_statistic	543.467268879953	30.6481856102772	5652.51924792859	494.955883488045	24.4605456760994	5583.56513052781	5.23078768861684	6.6473469952822	25949.7118056025	1119.87022468742	154.793296861581	4826.92242730041	494.903094649183	72.530787010352	7607.83405438193
cal_Estimate	NA	NA	NA	0.0238724384575419	2.71948246216953	-0.168054407187466	NA	NA	NA	NA	NA	NA	NA	NA	NA
cal_Pr...z..	NA	NA	NA	1.44956616452661e-33	9.21076021290446e-10	5.67614501764414e-39	NA	NA	NA	NA	NA	NA	NA	NA	NA
cal_Std.Error	NA	NA	NA	0.00197718112735887	0.444177077282291	0.0128692506794877	NA	NA	NA	NA	NA	NA	NA	NA	NA
cal_zvalue	NA	NA	NA	12.0739764947235	6.1225187008946	-13.0586007975839	NA	NA	NA	NA	NA	NA	NA	NA	NA
wealthIdx_Estimate	NA	NA	NA	NA	NA	NA	0.144503490136948	69.1816142883022	-1.91414470908345	NA	NA	NA	NA	NA	NA
wealthIdx_Pr...z..	NA	NA	NA	NA	NA	NA	3.72983264926432e-06	2.23442991281176e-07	0	NA	NA	NA	NA	NA	NA
wealthIdx_Std.Error	NA	NA	NA	NA	NA	NA	0.0312379492766376	13.358888551386	0.0371054140359243	NA	NA	NA	NA	NA	NA
wealthIdx_zvalue	NA	NA	NA	NA	NA	NA	4.62589553677969	5.17869536991717	-51.5866689219593	NA	NA	NA	NA	NA	NA
p.A.prot_Estimate	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.00148073028434642	0.221916473012486	-0.00520794333267238	NA	NA	NA
p.A.prot_Pr...z..	NA	NA	NA	NA	NA	NA	NA	NA	NA	2.50759287066563e-156	8.30126393398654e-33	3.00201194005694e-197	NA	NA	NA
p.A.prot_Std.Error	NA	NA	NA	NA	NA	NA	NA	NA	NA	5.55884799941827e-05	0.0186022369560791	0.000173813943639721	NA	NA	NA
p.A.prot_zvalue	NA	NA	NA	NA	NA	NA	NA	NA	NA	26.6373587567312	11.9295584469998	-29.9627476577329	NA	NA	NA
p.A.nProt_Estimate	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.0141317656200726	2.11856940494335	-0.0494468877742109
p.A.nProt_Pr...z..	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	2.61782083774363e-226	4.81511329043196e-35	0
p.A.nProt_Std.Error	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.000440019589949091	0.17153115470458	0.00128926108222202
p.A.nProt_zvalue	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	32.1162192385744	12.3509307017263	-38.3528894620707

5.1.2.3.6 Test Program IV Return All

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=vars.z, df=df)))
```

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=TRUE) +
                                       mean(df[[y]], na.rm=TRUE)))))
# lapplytwice
```

```
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=vars.z, df=df))
         )))
```

```
# df.reg.out.all %>%
#   kable() %>%
#   kable_styling_fc_wide()
```

5.1.2.4.3 Nested Lapply All

```
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=vars.z, df=df)) %>%
              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 %>%
  kable() %>%
  kable_styling_fc_wide()
```

vars_var.y	prot_tvalue	cal_tvalue	wealthIdx_tvalue	p.A.prot_tvalue	p.A.nProt_tvalue
hgt	18.8756010031786	23.4421863484661	13.508899618216	3.83682180045518	32.5448257554855
wgt	16.3591125056062	17.3686031309332	14.1390521528113	1.36958319982295	12.0961557911467
vil.id	-14.9385580468907	-19.6150110809452	34.0972558327347	8.45943342783186	17.7801422421419

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)
head(list.out$dfmain,10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

S.country	vil.id	indi.id	svymthRound	prot	male	wgt0	hgt0	variable	value	prot_mean	male_mean	wgt0_mean	hgt0_mean	svymthRound_mean	value_mean	value_A	value_B	value_C
Guatemala	3	1352	18	13.3	1	2545.2	47.4	hgt	70.2	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	71.37891	71.70649	71.99313
Guatemala	3	1352	24	46.3	1	2545.2	47.4	hgt	75.8	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	76.97891	75.83533	72.07980
Guatemala	3	1354	12	1.0	1	3634.3	51.2	hgt	66.3	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	65.24416	66.12009	70.44889
Guatemala	3	1354	18	9.8	1	3634.3	51.2	hgt	69.2	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	68.14416	68.62778	68.91442
Guatemala	3	1354	24	15.4	1	3634.3	51.2	hgt	75.3	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	74.24416	74.47813	70.72260
Guatemala	3	1356	12	8.6	1	3911.8	51.9	hgt	68.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	66.63250	67.16961	71.49842
Guatemala	3	1356	18	17.8	1	3911.8	51.9	hgt	74.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	72.63250	72.75947	73.04611
Guatemala	3	1356	24	30.5	1	3911.8	51.9	hgt	77.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	75.63250	75.19330	71.43777
Guatemala	3	1357	12	1.0	1	3791.4	52.6	hgt	71.5	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	69.62083	70.49676	74.82557
Guatemala	3	1357	18	12.7	1	3791.4	52.6	hgt	77.8	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	75.92083	76.27517	76.56181

```
options(repr.plot.width = 10, repr.plot.height = 4)
list.out$dfsumm %>%
  kable() %>%
  kable_styling_fc_wide()
```

variable	value_var	value_mean_var	value_A_var	value_B_var	value_C_var	value_var_frac	value_mean_var_frac	value_A_var_frac	value_B_var_frac	value_C_var_frac
hgt	21.864	NA	20.264	18.384	8.395	1	NA	0.927	0.841	0.384
wgt	2965693.245	NA	2863501.267	2659434.374	2346296.982	1	NA	0.966	0.897	0.791

5.2.1.4 Example Guatemala IV = vil.id

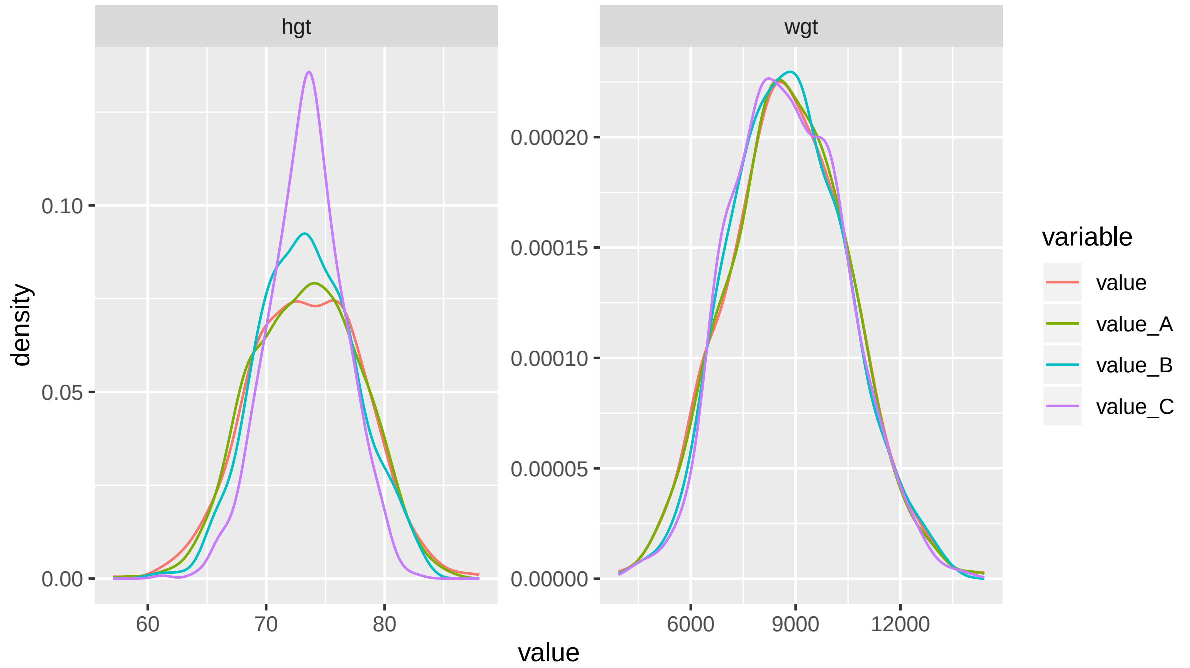
```
df.use <- df %>% filter(S.country == 'Guatemala') %>%
  filter(svymthRound %in% c(12, 18, 24))
vars.z <- c('vil.id')
list.out <- ff_lr_decompose(
  df=df.use, vars.y, vars.x, vars.c, vars.z, vars.other.keep,
  list.vars.tomean, list.vars.tomean.name.suffix,
  graph=TRUE, graph.nrow=1)
```

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

variable	value_var	value_mean_var	value_A_var	value_B_var	value_C_var	value_var_frac	value_mean_var_frac	value_A_var_frac	value_B_var_frac	value_C_var_frac
hgt	21.864	NA	20.235	16.323	10.03	1	NA	0.926	0.747	0.459
wgt	2965693.245	NA	2876682.895	2676220.156	2583301.29	1	NA	0.970	0.902	0.871

```
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)
head(list.out$dfmain, 10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

S.country	vil.id	indi.id	svymthRound	prot	male	wgt0	hgt0	variable	value	prot_mean	male_mean	wgt0_mean	hgt0_mean	svymthRound_mean	value_mean	value_A	value_B	value_C
Cebu	1	1	12	11.3	1	2043.8	44.2	hgt	70.8	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	72.12171	72.33074	76.35522
Cebu	1	2	12	5.9	0	2839.9	49.7	hgt	72.2	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	72.64812	73.05659	77.08108
Cebu	1	2	18	0.5	0	2839.9	49.7	hgt	76.5	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	76.94812	77.55604	77.47000
Cebu	1	2	24	14.1	0	2839.9	49.7	hgt	79.2	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	79.64812	79.75374	75.55718
Cebu	1	3	12	21.4	0	3445.6	51.7	hgt	68.0	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	67.70200	67.53800	71.56248
Cebu	1	3	18	23.6	0	3445.6	51.7	hgt	71.6	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	71.30200	71.05674	70.97071
Cebu	1	3	24	20.6	0	3445.6	51.7	hgt	76.7	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	76.40200	76.26754	72.07099
Cebu	1	4	12	0.7	0	3090.9	50.2	hgt	69.1	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	69.36159	69.96212	73.98660
Cebu	1	4	18	7.2	0	3090.9	50.2	hgt	74.3	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	74.56159	74.92205	74.83601
Cebu	1	4	24	10.3	0	3090.9	50.2	hgt	78.1	16.95957	0.5263013	2988.773	49.23897	17.87441	74.99584	78.36159	78.60755	74.41100

```
options(repr.plot.width = 10, repr.plot.height = 4)
list.out$dfsumm %>%
  kable() %>%
  kable_styling_fc_wide()
```

variable	value_var	value_mean_var	value_A_var	value_B_var	value_C_var	value_var_frac	value_mean_var_frac	value_A_var_frac	value_B_var_frac	value_C_var_frac
hgt	24.375	NA	22.561	21.309	10.001	1	NA	0.926	0.874	0.410
wgt	3337460.957	NA	3218987.397	3039513.634	2558514.368	1	NA	0.965	0.911	0.767

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(
```

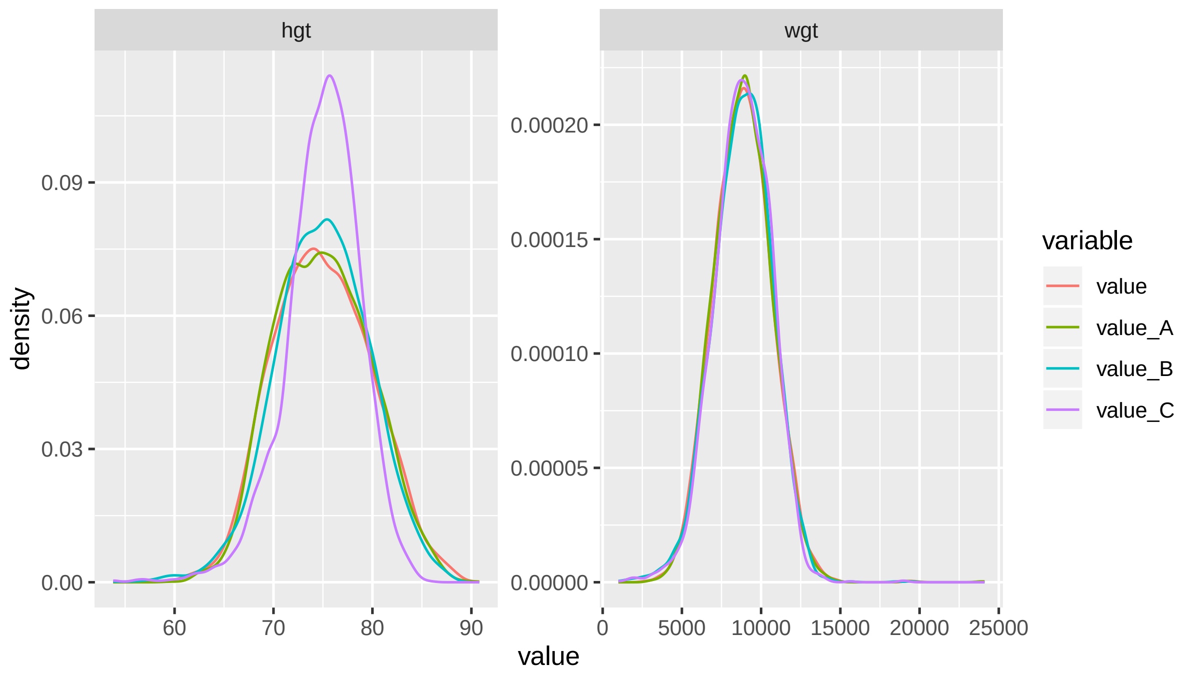
```
  df=df.use, vars.y, vars.x, vars.c, vars.z, vars.other.keep,
  list.vars.tomean, list.vars.tomean.name.suffix,
  graph=TRUE, graph.nrow=1)
```

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

variable	value_var	value_mean_var	value_A_var	value_B_var	value_C_var	value_var_frac	value_mean_var_frac	value_A_var_frac	value_B_var_frac	value_C_var_frac
hgt	24.375	NA	22.625	22.194	14.392	1	NA	0.928	0.911	0.590
wgt	3337460.957	NA	3237415.252	3385814.742	3158659.340	1	NA	0.970	1.014	0.946

```
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))
```

```
# df.reg.out
```

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

	prot_Estimate	male_Estimate	wgt0_Estimate	hgt0_Estimate	svymthRound_Estimate
hgt	-0.2714772	1.244735	0.0004430	0.6834853	1.133919
wgt	-59.0727542	489.852902	0.7696158	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
```

```
head(df.decompose_step1, 10) %>%
  kable() %>%
  kable_styling_fc()
```

S.country	vil.id	indi.id	svymthRound	prot	male	wgt0	hgt0	variable	value
Guatemala	3	1352	18	13.3	1	2545.2	47.4	hgt	70.2
Guatemala	3	1352	24	46.3	1	2545.2	47.4	hgt	75.8
Guatemala	3	1354	12	1.0	1	3634.3	51.2	hgt	66.3
Guatemala	3	1354	18	9.8	1	3634.3	51.2	hgt	69.2
Guatemala	3	1354	24	15.4	1	3634.3	51.2	hgt	75.3
Guatemala	3	1356	12	8.6	1	3911.8	51.9	hgt	68.1
Guatemala	3	1356	18	17.8	1	3911.8	51.9	hgt	74.1
Guatemala	3	1356	24	30.5	1	3911.8	51.9	hgt	77.1
Guatemala	3	1357	12	1.0	1	3791.4	52.6	hgt	71.5
Guatemala	3	1357	18	12.7	1	3791.4	52.6	hgt	77.8

```
# 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
```

```
head(df.decompose_step2,10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

S.country	vil.id	indi.id	svymthRound	prot	male	wgt0	hgt0	variable	value	prot_mean	male_mean	wgt0_mean	hgt0_mean	svymthRound_mean	value_mean
Guatemala	3	1352	18	13.3	1	2545.2	47.4	hgt	70.2	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1352	24	46.3	1	2545.2	47.4	hgt	75.8	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1354	12	1.0	1	3634.3	51.2	hgt	66.3	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1354	18	9.8	1	3634.3	51.2	hgt	69.2	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1354	24	15.4	1	3634.3	51.2	hgt	75.3	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1356	12	8.6	1	3911.8	51.9	hgt	68.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1356	18	17.8	1	3911.8	51.9	hgt	74.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1356	24	30.5	1	3911.8	51.9	hgt	77.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1357	12	1.0	1	3791.4	52.6	hgt	71.5	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216
Guatemala	3	1357	18	12.7	1	3791.4	52.6	hgt	77.8	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216

```

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
```

```
head(df.decompose_step3, 10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

S.country	vil.id	indi.id	svymthRound	prot	male	wgt0	hgt0	variable	value	prot_mean	male_mean	wgt0_mean	hgt0_mean	svymthRound_mean	value_mean	value_mh0me2m	value_mh0mep2m	value_mh0mepm2m
Guatemala	3	1352	18	13.3	1	2545.2	47.4	hgt	70.2	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	73.19390	71.19903	71.68148
Guatemala	3	1352	24	46.3	1	2545.2	47.4	hgt	75.8	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	78.79390	85.75778	79.43671
Guatemala	3	1354	12	1.0	1	3634.3	51.2	hgt	66.3	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	63.61689	58.28285	65.56882
Guatemala	3	1354	18	9.8	1	3634.3	51.2	hgt	69.2	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	66.51689	63.57185	64.05430
Guatemala	3	1354	24	15.4	1	3634.3	51.2	hgt	75.3	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	72.61689	71.19213	64.87106
Guatemala	3	1356	12	8.6	1	3911.8	51.9	hgt	68.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	64.33707	61.06626	68.35222
Guatemala	3	1356	18	17.8	1	3911.8	51.9	hgt	74.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	70.33707	69.56385	70.04630
Guatemala	3	1356	24	30.5	1	3911.8	51.9	hgt	77.1	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	73.33707	76.01161	69.69055
Guatemala	3	1357	12	1.0	1	3791.4	52.6	hgt	71.5	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	66.83353	61.49949	68.78545
Guatemala	3	1357	18	12.7	1	3791.4	52.6	hgt	77.8	20.64819	0.5499276	3312.297	49.75137	18.42547	73.41216	73.13353	70.97578	71.45823

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

value_var	value_mean_var	value_mh0me2m_var	value_mh0mep2m_var	value_mh0mepm2m_var	value
21.864	NA	25.35	49.047	23.06	
2965693.245	NA	2949187.64	4192769.518	3147506.60	

5.2.1.7.6 Decomposition Step 4 Variance

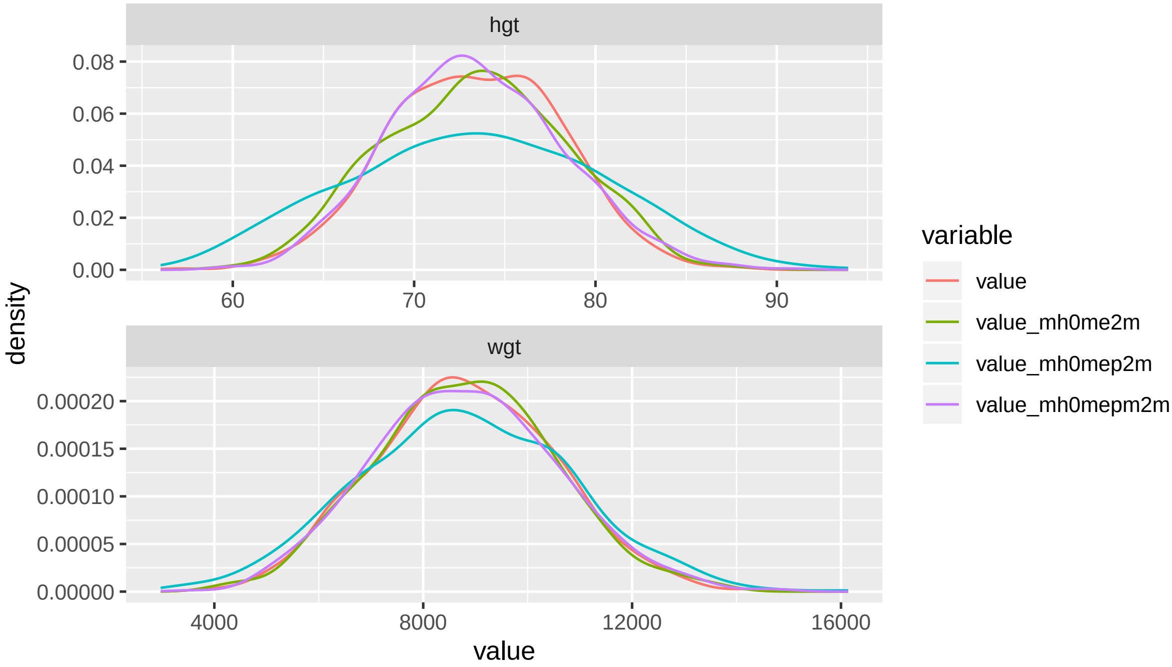
5.2.1.7.7 Graphical Results Graphically, difficult to pick up exact differences in variance, a 50 percent reduction in variance visually does not look like 50 percent. Intuitively, we are kind of seeing standard deviation, not variance on the graph if we think abou the x-scale.

```
head(df.decompose_step3 %>%
    select(variable, contains('value'), -value_mean), 10) %>%
  kable() %>%
  kable_styling_fc_wide()
```

variable	value	value_mh0me2m	value_mh0mep2m	value_mh0mepm2m
hgt	70.2	73.19390	71.19903	71.68148
hgt	75.8	78.79390	85.75778	79.43671
hgt	66.3	63.61689	58.28285	65.56882
hgt	69.2	66.51689	63.57185	64.05430
hgt	75.3	72.61689	71.19213	64.87106
hgt	68.1	64.33707	61.06626	68.35222
hgt	74.1	70.33707	69.56385	70.04630
hgt	77.1	73.33707	76.01161	69.69055
hgt	71.5	66.83353	61.49949	68.78545
hgt	77.8	73.13353	70.97578	71.45823

```
options(repr.plot.width = 10, repr.plot.height = 4)
df.decompose_step3 %>%
    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=2)
```



5.2.1.8 Additional Decomposition Testings

```
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
```

```
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
```

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

variable	value_mean	pred_new_mean	value_sd	pred_new_sd
hgt	73.41216	73.41216	4.675867	4.534947
wgt	8807.87656	8807.87656	1722.118824	1695.221845

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

variable	value_mean	pred_new_mean	value_var	pred_new_var	ratio
hgt	73.41216	73.41216	2.186374e+01	25.3504	1.1594724
wgt	8807.87656	8807.87656	2.965693e+06	2949187.6357	0.9944345

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")
```

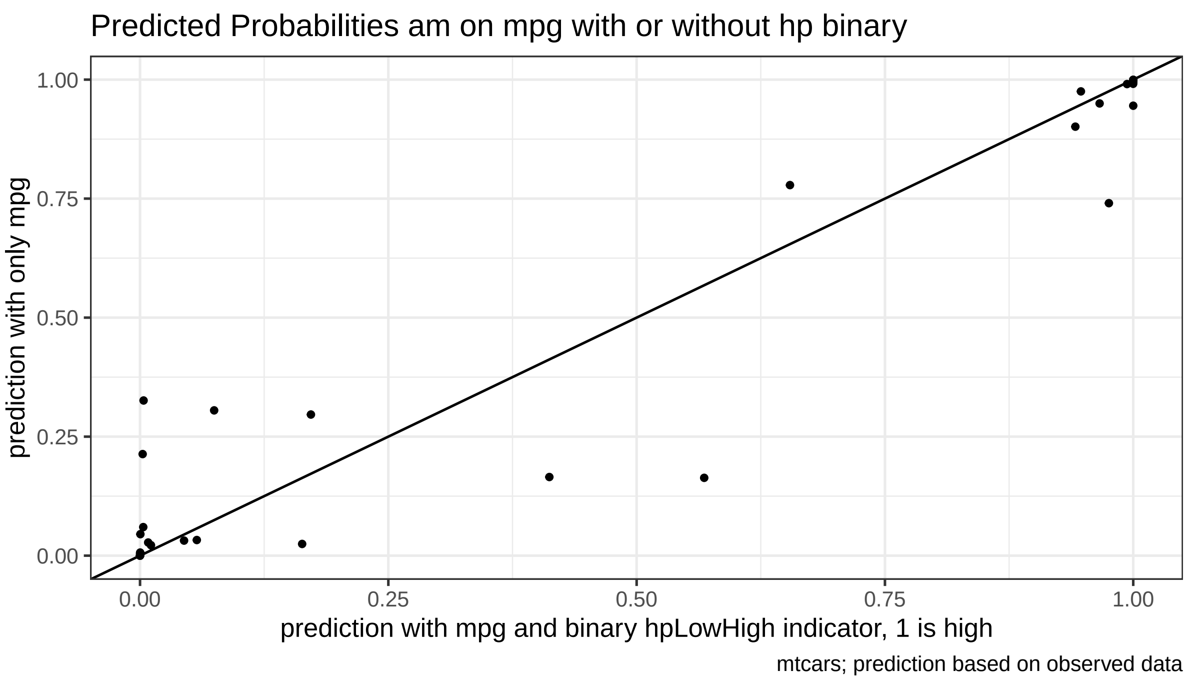
6.1.1.1.1 Prediction with Observed Binary Input Logit regression with a continuous variable and a binary variable. Predict outcome with observed continuous variable as well as observed binary input variable.

```
# 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.

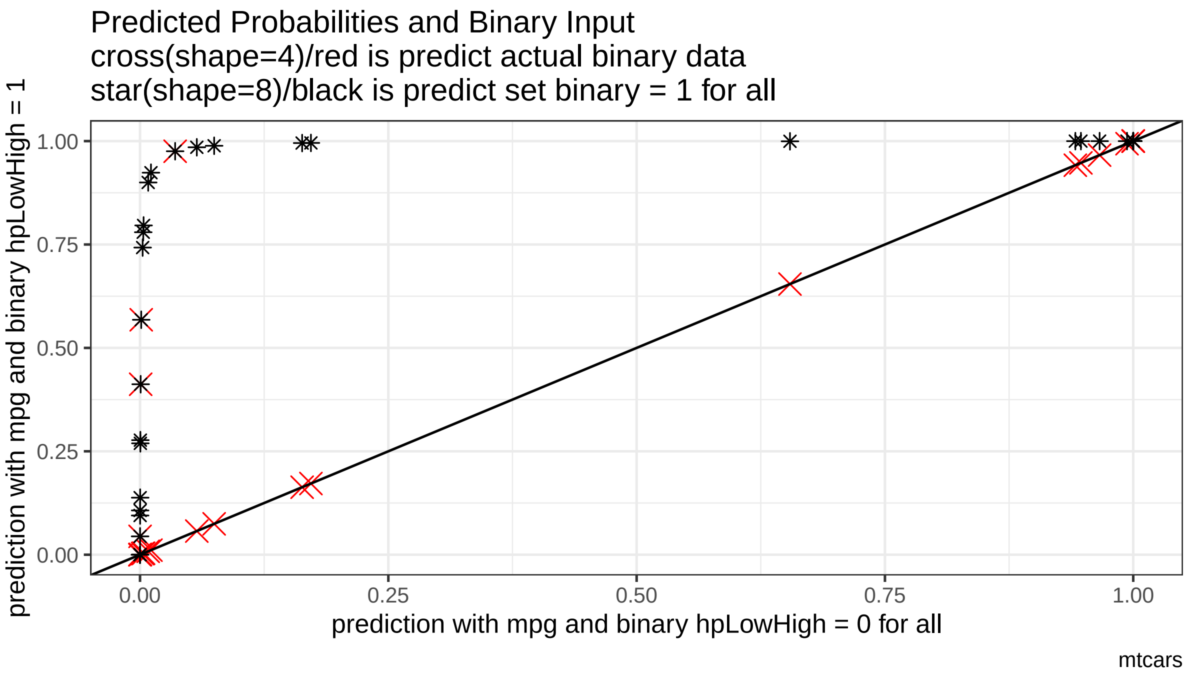
Note that given the example here, the probability changes a lot when we

```
# 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
```

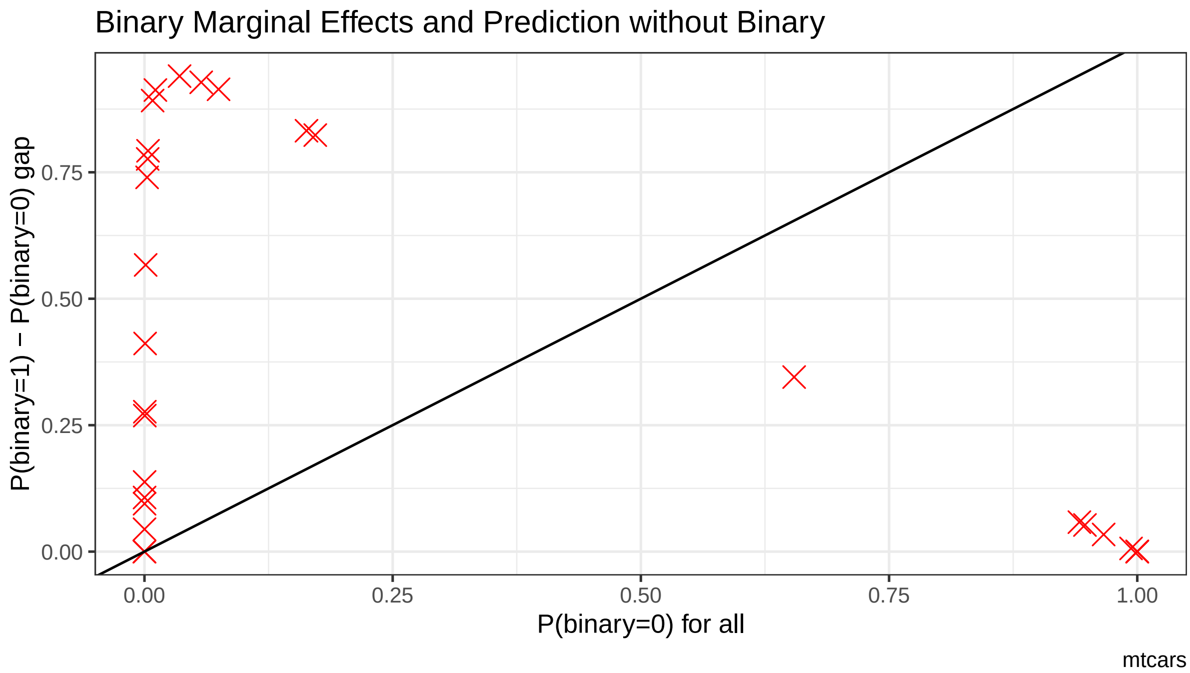
```
# Two different dataframes, mutate the binary regressor
df_mtcars_bi0 <- df_mtcars %>% mutate(!!sym(svr_binary) := svr_binary_lb0)
df_mtcars_bi1 <- df_mtcars %>% mutate(!!sym(svr_binary) := svr_binary_lb1)

# 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.

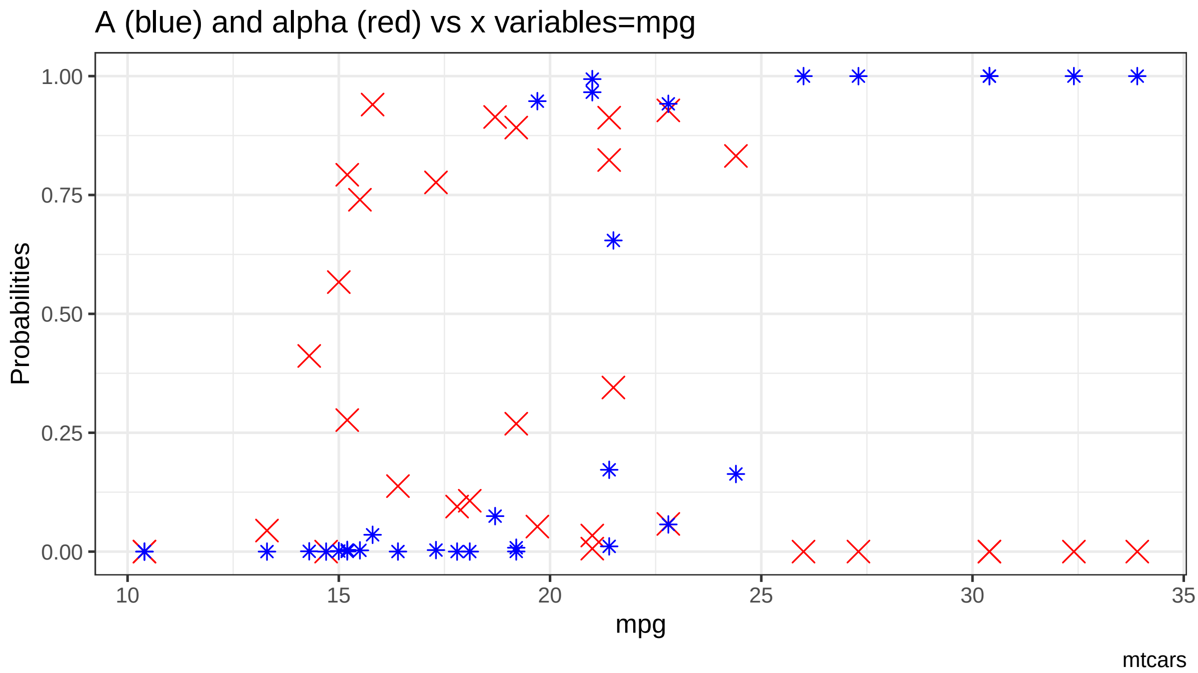
```
# Generate Gap Variable
df_mtcars <- df_mtcars %>% mutate(alpha_i = p_mpg_hp_bi1 - p_mpg_hp_bi0) %>%
                mutate(A_i = p_mpg_hp_bi0)

# 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)
```



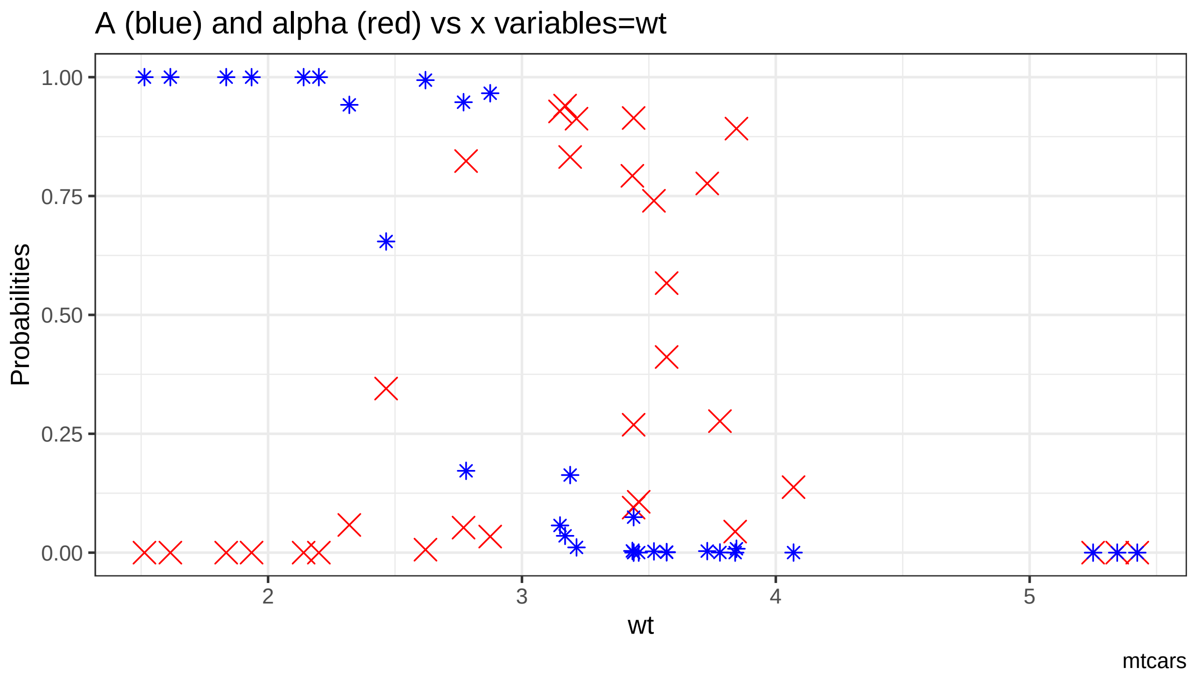
6.1.1.1.4 X variables and A and alpha Given the x-variables included in the logit regression, how do they relate to A_i and alpha_i

```
# 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)
```

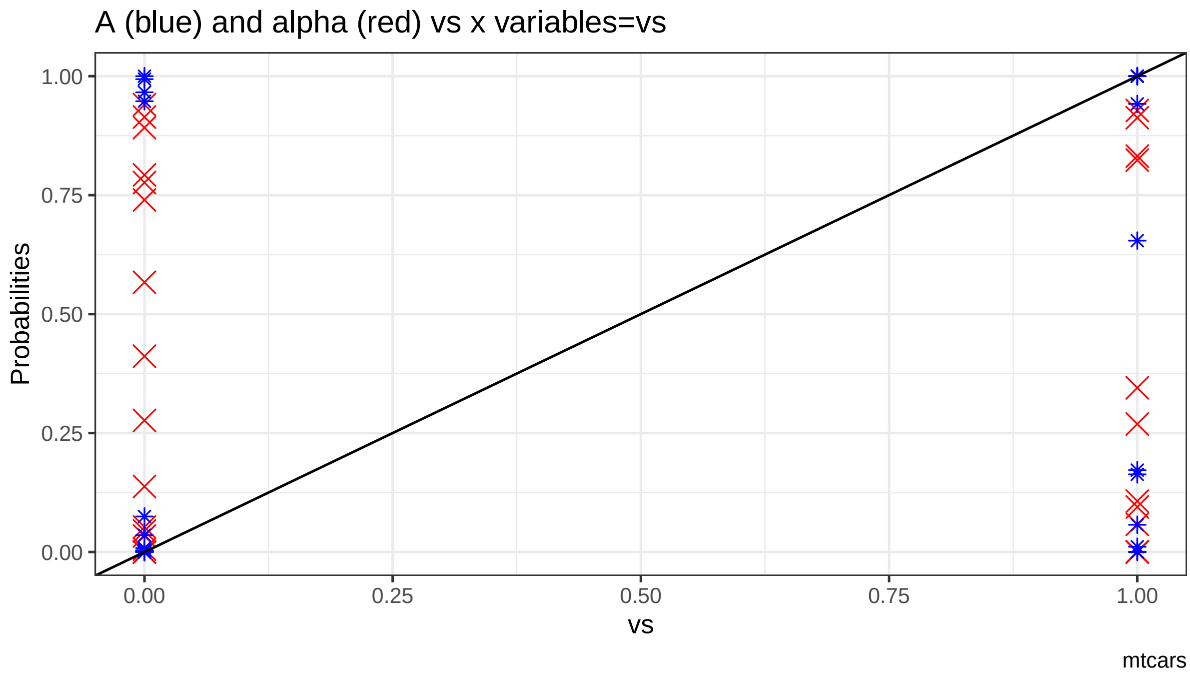
```
## [[1]]
```


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)
```

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)
# str(tb_states_choices_bisec)
```

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)
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 uniquly 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)
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')))
```

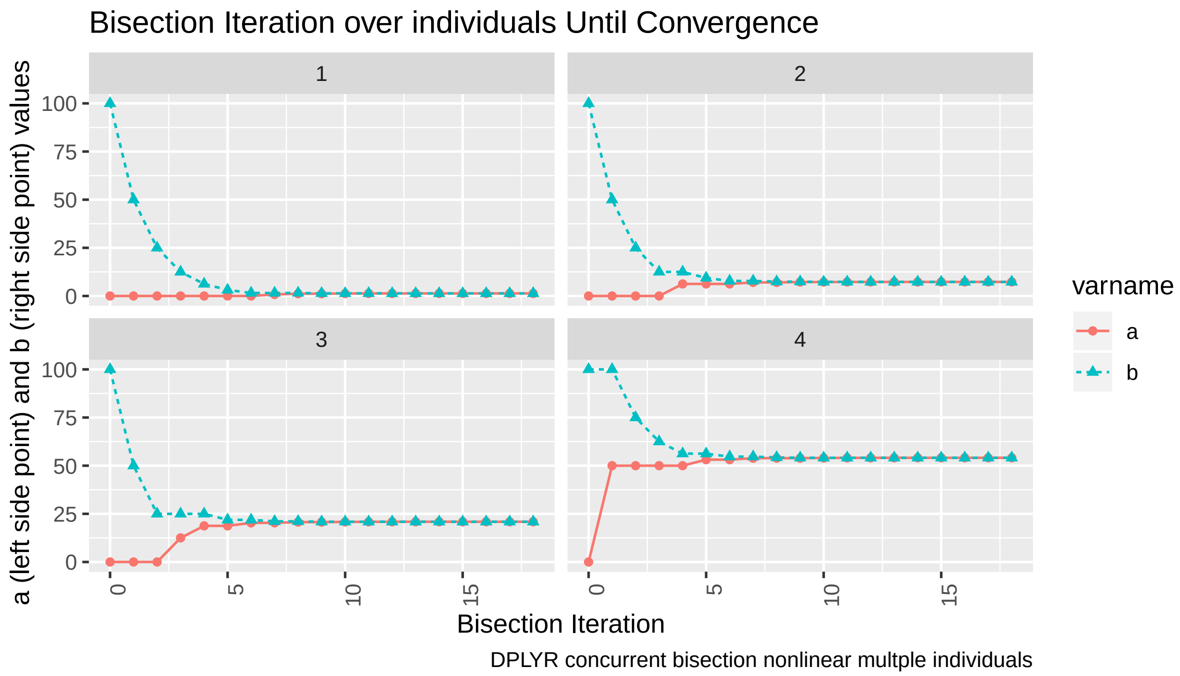

```
## # A tibble: 76 x 8
##    INDI_ID  fl_A fl_alpha biseciter     a      b    fa        fb
##      <int> <dbl>    <dbl> <chr>     <dbl>  <dbl> <dbl>     <dbl>
##  1       1    -2      0.1 0         0      100     100  -15058.
##  2       1    -2      0.1 1         0       50     100   -6660.
##  3       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)
```



Chapter 8

Mathmatics and Statistics

8.1 Distributions

8.1.1 Integrate Over Normal Guassian Process Shock

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

Some Common parameters

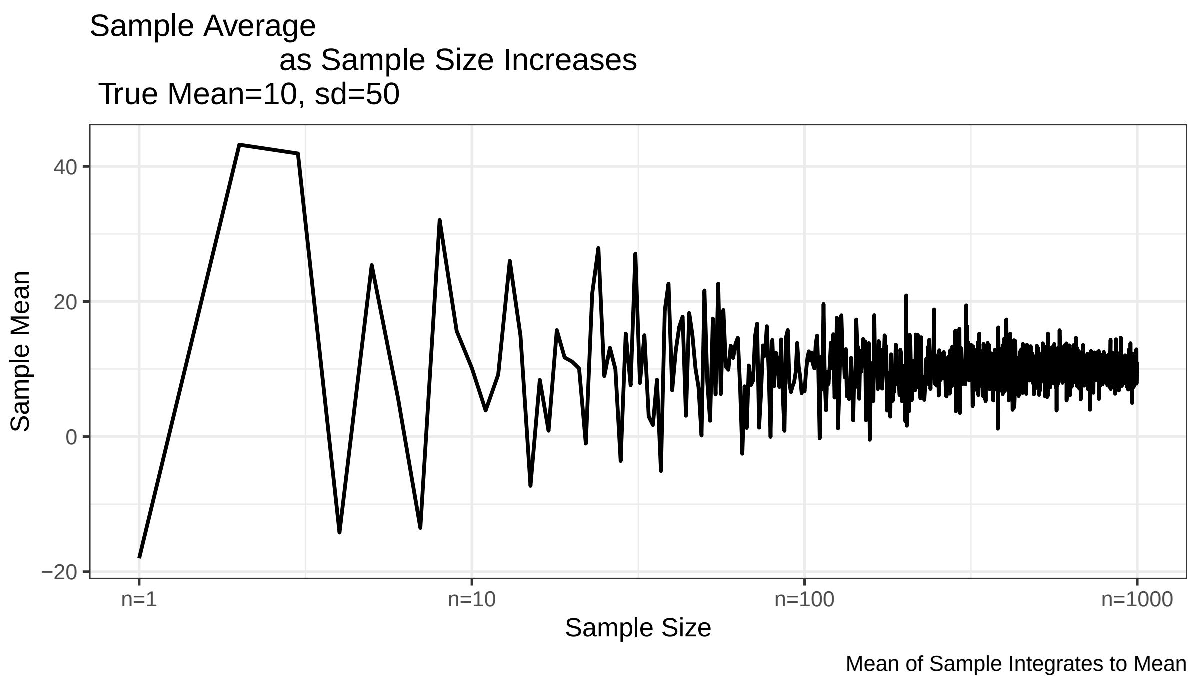
```
fl_eps_mean = 10
fl_eps_sd = 50
fl_cdf_min = 0.000001
fl_cdf_max = 0.999999
ar_it_draws <- seq(1, 1000)
```

8.1.1.1 Randomly Sample and Integrate (Monte Carlo integration)

Compare randomly drawn normal shock mean and known mean. How does simulated mean change with draws. Actual integral equals to 10, as sample size increases, the sample mean approaches the integration results, but this is expensive, even with ten thousand draws, not very exact.

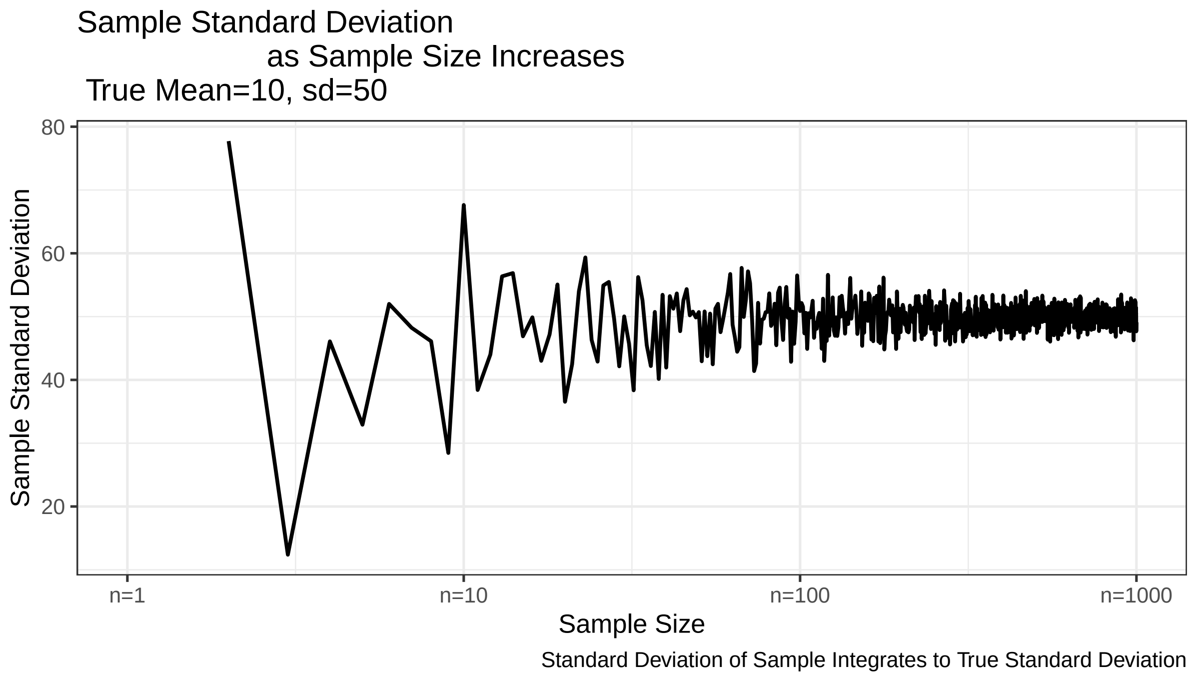
```
# Simulate Draws
set.seed(123)
ar_fl_means <-
  sapply(ar_it_draws, function(x)
    return(mean(rnorm(x[1], mean=fl_eps_mean, sd=fl_eps_sd))))
ar_fl_sd <-
  sapply(ar_it_draws, function(x)
    return(sd(rnorm(x[1], mean=fl_eps_mean, sd=fl_eps_sd))))

mt_sample_means <- cbind(ar_it_draws, ar_fl_means, ar_fl_sd)
colnames(mt_sample_means) <- c('draw_count', 'mean', 'sd')
tb_sample_means <- as_tibble(mt_sample_means)

# Graph
# x-labels
x.labels <- c('n=1', 'n=10', 'n=100', 'n=1000')
x.breaks <- c(1, 10, 100, 1000)

# Graph Results--Draw
plt_mean <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=mean)) +
  geom_line(size=0.75) +
```

```
  labs(title = paste0('Sample Average
                      as Sample Size Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Sample Size',
       y = 'Sample Mean',
       caption = 'Mean of Sample Integrates to Mean') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_mean)
```




```
plt_sd <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=sd)) +
  geom_line(size=0.75) +
  labs(title = paste0('Sample Standard Deviation
                      as Sample Size Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Sample Size',
       y = 'Sample Standard Deviation',
       caption = 'Standard Deviation of Sample Integrates to True Standard Deviation') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_sd)
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```



8.1.1.2 Integration By Symmetric Uneven Rectangle

Draw on grid from probability space, and then find use norm inverse to find corresponding x point. Under this approach, each rectangle is suppose to approximate the same same. So even area, but uneven width.

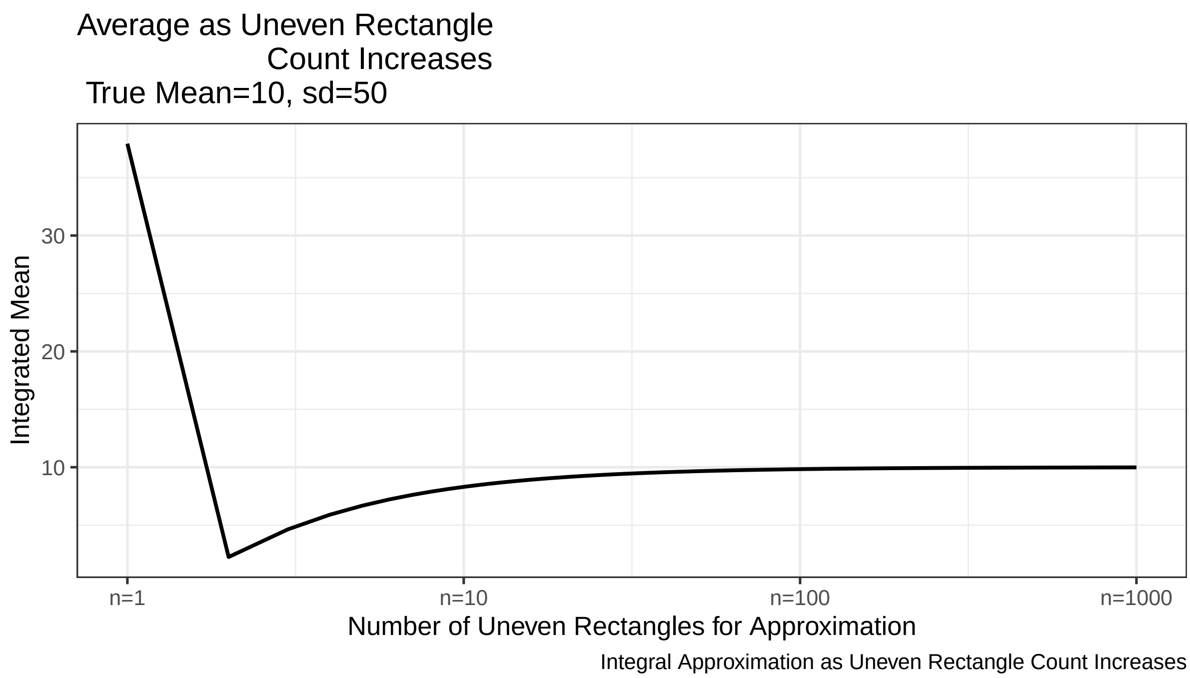
Resulting integration is rectangle based, but rectangle width differ. Th rectangle have wider width as they move away from the mean, and thin width close to the mean. This is much more stable than the random draw method, and approximates the true answer more accurately.

```
mt_fl_means <-
  sapply(ar_it_draws, function(x) {

    fl_prob_break = (fl_cdf_max - fl_cdf_min)/(x[1])
    ar_eps_bounds <- qnorm(seq(fl_cdf_min, fl_cdf_max,
                               by=(fl_cdf_max - fl_cdf_min)/(x[1])),
                           mean = fl_eps_mean, sd = fl_eps_sd)
    ar_eps_val <- (tail(ar_eps_bounds, -1) + head(ar_eps_bounds, -1))/2
    ar_eps_prb <- rep(fl_prob_break/(fl_cdf_max - fl_cdf_min), x[1])
    ar_eps_fev <- dnorm(ar_eps_val,
                        mean = fl_eps_mean, sd = fl_eps_sd)

    fl_cdf_total_approx <- sum(ar_eps_fev*diff(ar_eps_bounds))
    fl_mean_approx <- sum(ar_eps_val*(ar_eps_fev*diff(ar_eps_bounds)))
    fl_sd_approx <- sqrt(sum((ar_eps_val-fl_mean_approx)^2*(ar_eps_fev*diff(ar_eps_bounds))))

    return(list(cdf=fl_cdf_total_approx, mean=fl_mean_approx, sd=fl_sd_approx))
  })

mt_sample_means <- cbind(ar_it_draws, as_tibble(t(mt_fl_means)) %>% unnest())
```

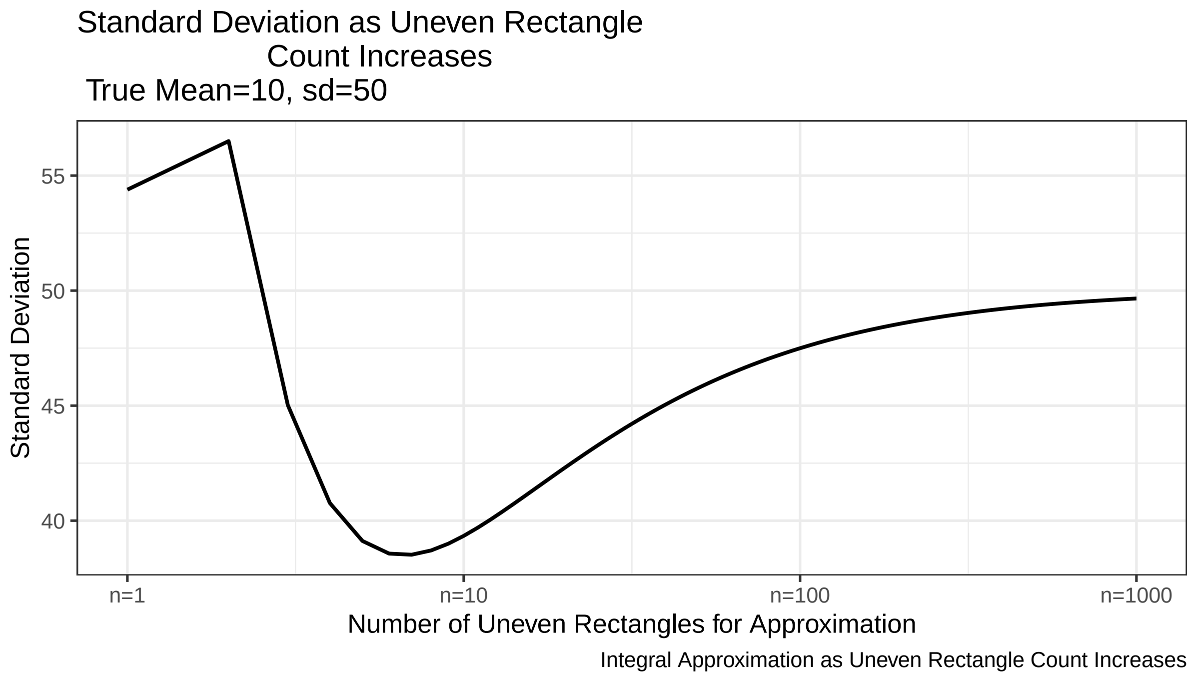
```
## Warning: `cols` is now required.
## Please use `cols = c(cdf, mean, sd)`
```

```
colnames(mt_sample_means) <- c('draw_count', 'cdf', 'mean', 'sd')
tb_sample_means <- as_tibble(mt_sample_means)
```

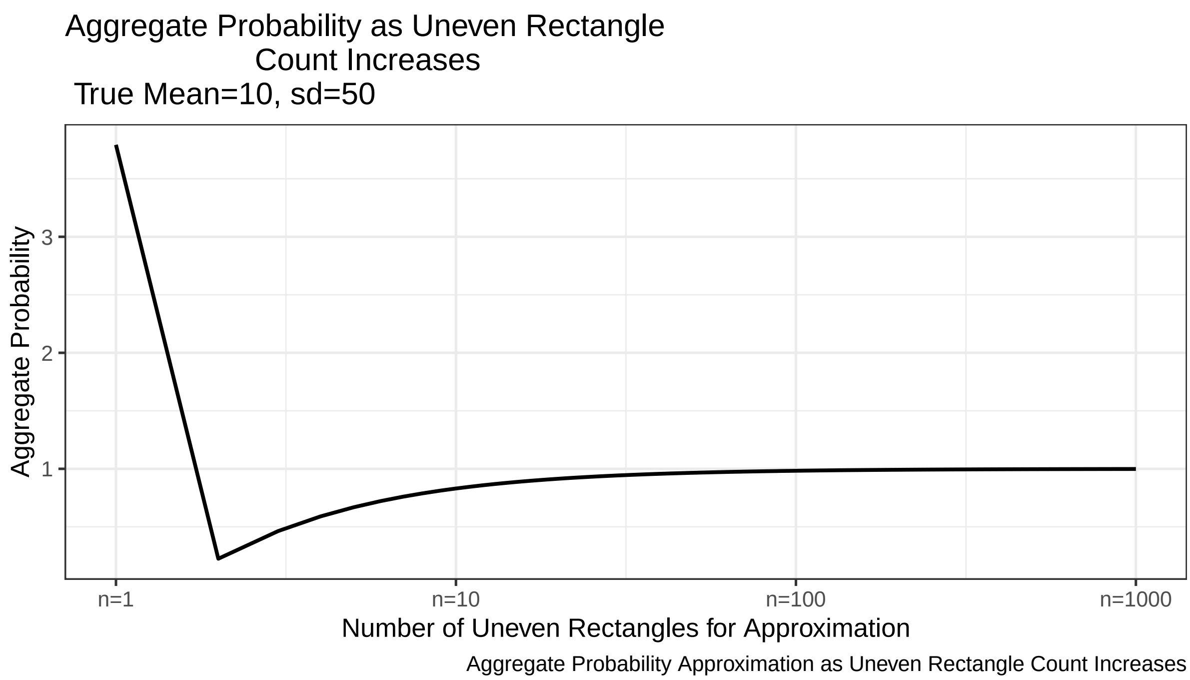
```r
# Graph
# x-labels
x.labels <- c('n=1', 'n=10', 'n=100', 'n=1000')
x.breaks <- c(1, 10, 100, 1000)

# Graph Results--Draw
plt_mean <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=mean)) +
  geom_line(size=0.75) +
  labs(title = paste0('Average as Uneven Rectangle
                      Count Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Number of Uneven Rectangles for Approximation',
       y = 'Integrated Mean',
       caption = 'Integral Approximation as Uneven Rectangle Count Increases') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_mean)
```




```r
plt_sd <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=sd)) +
  geom_line(size=0.75) +
  labs(title = paste0('Standard Deviation as Uneven Rectangle
                      Count Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Number of Uneven Rectangles for Approximation',
       y = 'Standard Deviation',
       caption = 'Integral Approximation as Uneven Rectangle Count Increases') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_sd)
```




```
plt_cdf <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=cdf)) +
  geom_line(size=0.75) +
  labs(title = paste0('Aggregate Probability as Uneven Rectangle
                      Count Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Number of Uneven Rectangles for Approximation',
       y = 'Aggregate Probability',
       caption = 'Aggregate Probability Approximation as Uneven Rectangle Count Increases') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_cdf)
```



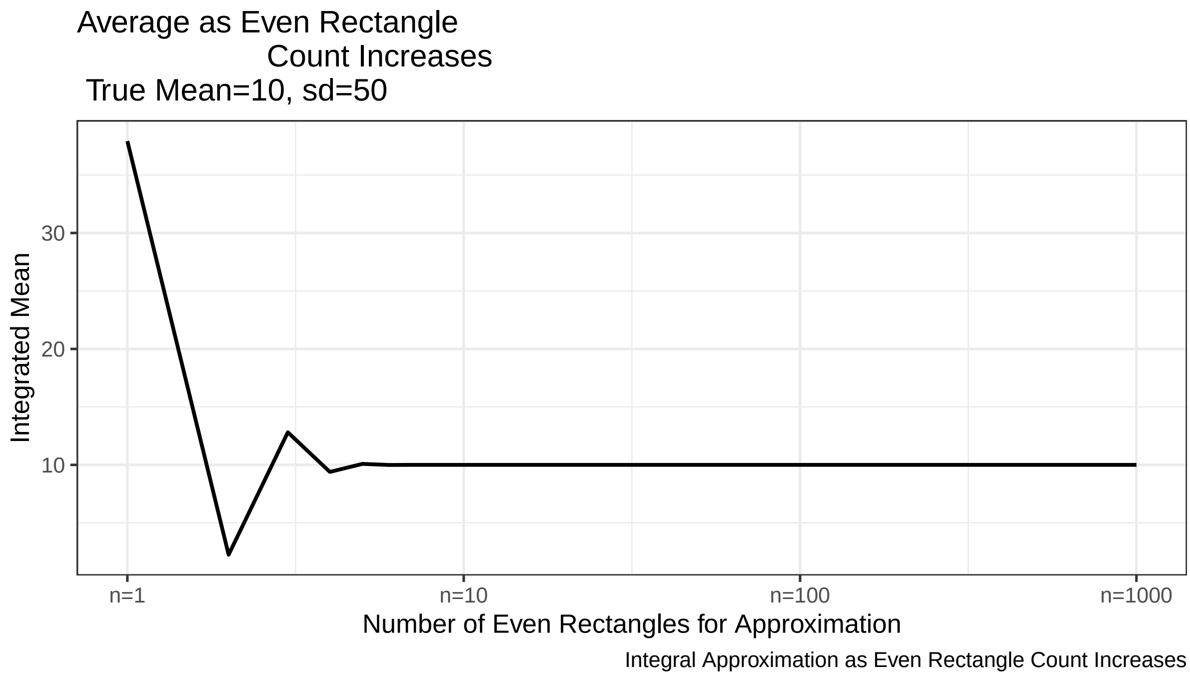
8.1.1.3 Integration By Constant Width Rectangle (Trapezoidal rule)

This is implementing even width recentagle, even along x-axis take points, and measure $f(x)$. Rectangle width are the same. This is even width, but uneven area. Note that this method approximates the true answer much better and more quickly.

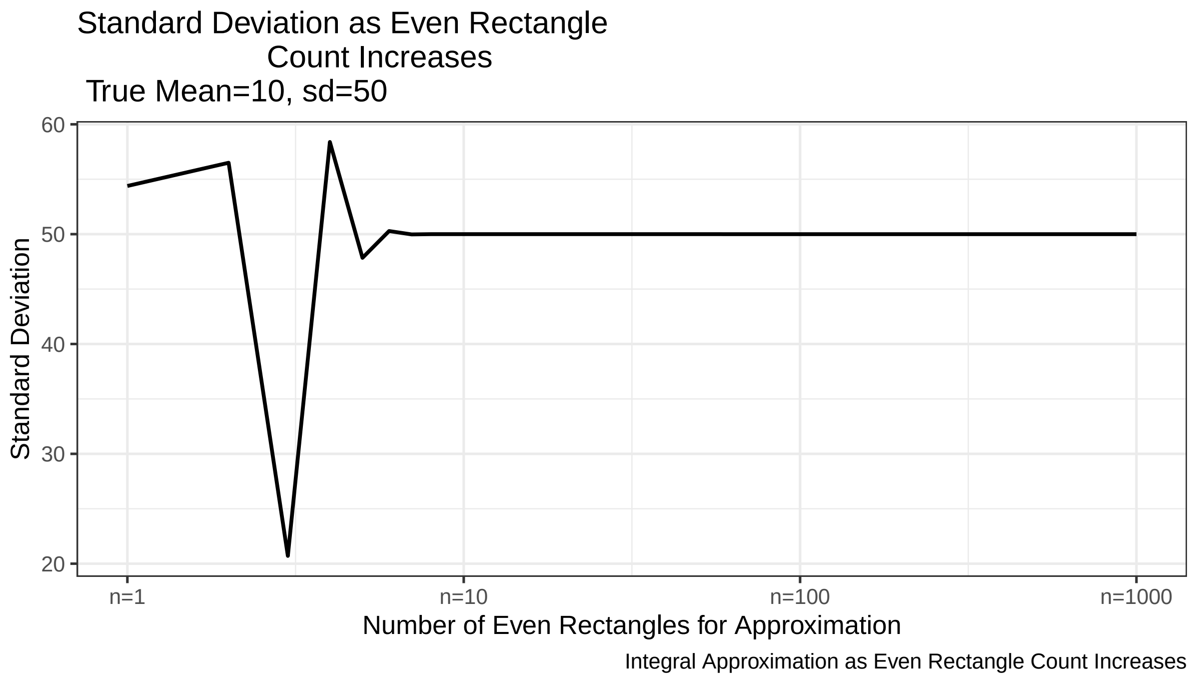
```
mt_fl_means <-
  sapply(ar_it_draws, function(x) {

    fl_eps_min <- qnorm(fl_cdf_min, mean = fl_eps_mean, sd = fl_eps_sd)
    fl_eps_max <- qnorm(fl_cdf_max, mean = fl_eps_mean, sd = fl_eps_sd)
    fl_gap <- (fl_eps_max-fl_eps_min)/(x[1])
    ar_eps_bounds <- seq(fl_eps_min, fl_eps_max, by=fl_gap)
    ar_eps_val <- (tail(ar_eps_bounds, -1) + head(ar_eps_bounds, -1))/2
    ar_eps_prb <- dnorm(ar_eps_val, mean = fl_eps_mean, sd = fl_eps_sd)*fl_gap

    fl_cdf_total_approx <- sum(ar_eps_prb)
    fl_mean_approx <- sum(ar_eps_val*ar_eps_prb)
    fl_sd_approx <- sqrt(sum((ar_eps_val-fl_mean_approx)^2*ar_eps_prb))

    return(list(cdf=fl_cdf_total_approx, mean=fl_mean_approx, sd=fl_sd_approx))
  })

mt_sample_means <- cbind(ar_it_draws, as_tibble(t(mt_fl_means)) %>% unnest())
```

```
## Warning: `cols` is now required.
## Please use `cols = c(cdf, mean, sd)`
```

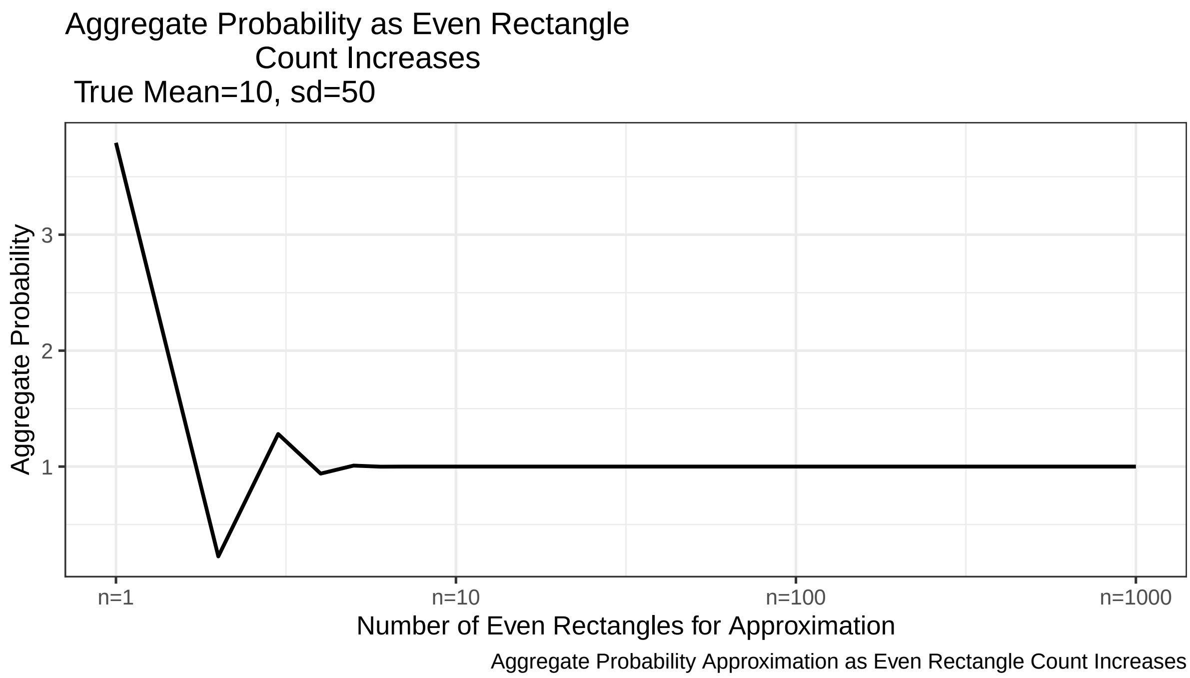
```
colnames(mt_sample_means) <- c('draw_count', 'cdf', 'mean', 'sd')
tb_sample_means <- as_tibble(mt_sample_means)

# Graph
# x-labels
x.labels <- c('n=1', 'n=10', 'n=100', 'n=1000')
x.breaks <- c(1, 10, 100, 1000)

# Graph Results--Draw
plt_mean <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=mean)) +
  geom_line(size=0.75) +
  labs(title = paste0('Average as Even Rectangle
                      Count Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Number of Even Rectangles for Approximation',
       y = 'Integrated Mean',
       caption = 'Integral Approximation as Even Rectangle Count Increases') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_mean)
```




```
plt_sd <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=sd)) +
  geom_line(size=0.75) +
  labs(title = paste0('Standard Deviation as Even Rectangle
                      Count Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Number of Even Rectangles for Approximation',
       y = 'Standard Deviation',
       caption = 'Integral Approximation as Even Rectangle Count Increases') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_sd)
```



```r
plt_cdf <- tb_sample_means %>%
  ggplot(aes(x=draw_count, y=cdf)) +
  geom_line(size=0.75) +
  labs(title = paste0('Aggregate Probability as Even Rectangle
                      Count Increases\n True Mean=',
                      fl_eps_mean,', sd=',fl_eps_sd),
       x = 'Number of Even Rectangles for Approximation',
       y = 'Aggregate Probability',
       caption = 'Aggregate Probability Approximation as Even Rectangle Count Increases') +
  scale_x_continuous(trans='log10', labels = x.labels, breaks = x.breaks) +
  theme_bw()
print(plt_cdf)
```



8.2 Analytical Solutions

8.2.1 Linear Scalar f(x)=0 Solutions

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

8.2.1.1 Ratio

Here are some common ratios.

8.2.1.1.1 Unif Draw Min and Max Ratio We want to draw numbers such that we have some mean b, and that the possible maximum and minimum value drawn are at most a times apart. Given b and a, solve for x.

$$f(x) = \frac{b+x}{b-x} - a = 0$$

$$b \cdot a - x \cdot a = b + xb \cdot a - b = x + x \cdot ab\left(a-1\right) = x\left(a+1\right)x = \frac{b\left(a-1\right)}{a+1}$$

Uniformly draw

```
b <- 100
a <- 2
x <- (b*(a-1))/(a+1)
ar_unif_draws <- runif(100, min=b-x, max=b+x)
fl_max_min_ratio <- max(ar_unif_draws)/min(ar_unif_draws)
cat('fl_max_min_ratio =', fl_max_min_ratio, 'is close to a =', a, '\n')
```

```
## fl_max_min_ratio = 1.965882 is close to a = 2
```

8.3 Inequality Models

8.3.1 Gini Discrete Sample

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

This works out how the ff_dist_gini_vector_pos function works from Fan's *REconTools* Package.

8.3.1.1 Gini Formula for Discrete Sample

There is an vector values (all positive). This could be height information for N individuals. It could also be income information for N individuals. Calculate the GINI coefficient treating the given vector as population. This is not an estimation exercise where we want to estimate population gini based on a sample. The given array is the population. The population is discrete, and only has these N individuals in the length n vector.

Note that when the sample size is small, there is a limit to inequality using the formula defined below given each N. So for small N, can not really compare inequality across arrays with different N, can only compare arrays with the same N. In another word, if 1 of N individual holds all resource, as N increases, GINI will asymptote to 1, but it is very far away from 1 for low N.

The GINI formula used here is:

$$GINI = 1 - \frac{2}{N+1} \cdot \left(\sum_{i=1}^{N} \sum_{j=1}^{i} x_j \right) \cdot \left(\sum_{i=1}^{N} x_i \right)^{-1}$$

Derive the formula in the steps below.

Step 1 Area Formula

$$\Gamma = \sum_{i=1}^{N} \frac{1}{N} \cdot \left(\sum_{j=1}^{i} \left(\frac{x_j}{\sum_{\hat{j}=1}^{N} x_{\hat{j}}} \right) \right)$$

Step 2 Total Area Given Perfect equality

With perfect equality $x_i = a$ for all i, so need to divide by that.

$$\Gamma^{\text{equal}} = \sum_{i=1}^{N} \frac{1}{N} \cdot \left(\sum_{j=1}^{i} \left(\frac{a}{\sum_{\hat{j}=1}^{N} a} \right) \right) = \frac{N+1}{N} \cdot \frac{1}{2}$$

As the number of elements of the vecotr increases:

$$\lim_{N \to \infty} \Gamma^{\text{equal}} = \lim_{N \to \infty} \frac{N+1}{N} \cdot \frac{1}{2} = \frac{1}{2}$$

Step 3 Arriving at Finite Vector Gini Formula

Given what we have from above, we obtain the gini formula, divide by total area below 45 degree line.

$$GINI = 1 - \left(\sum_{i=1}^{N}\sum_{j=1}^{i} x_j\right) \cdot \left(N \cdot \sum_{i=1}^{N} x_i\right)^{-1} \cdot \left(\frac{N+1}{N} \cdot \frac{1}{2}\right)^{-1} = 1 - \frac{2}{N+1} \cdot \left(\sum_{i=1}^{N}\sum_{j=1}^{i} x_j\right) \cdot \left(\sum_{i=1}^{N} x_i\right)^{-1}$$

Step 4 Maximum Inequality given N

Suppose $x_i = 0$ for all $i < N$, then:

$$GINI^{x_i=0 \text{ except } i=N} = 1 - \frac{2}{N+1} \cdot X_N \cdot \left(X_N\right)^{-1} = 1 - \frac{2}{N+1}$$

$$\lim_{N\rightarrow\infty} GINI^{x_i=0 \text{ except } i=N} = 1 - \lim_{N\rightarrow\infty}\frac{2}{N+1} = 1$$

Note that for small N, for example if $N = 10$, even when one person holds all income, all others have 0 income, the formula will not produce gini is zero, but that gini is equal to $\frac{2}{11} \approx 0.1818$. If $N = 2$, inequality is at most, $\frac{2}{3} \approx 0.667$.

$$MostUnequalGINI\left(N\right) = 1 - \frac{2}{N+1} = \frac{N-1}{N+1}$$

8.3.1.2 Implement GINI Formula

The **GINI** formula just derived is trivial to compute.

1. scalar: $\frac{2}{N+1}$
2. cumsum: $\sum_{j=1}^{i} x_j$
3. sum of cumsum: $\left(\sum_{i=1}^{N}\sum_{j=1}^{i} x_j\right)$
4. sum: $\sum_{i=1}^{N} X_i$

There are no package dependencies. Define the formula here:

```
# Formula, directly implement the GINI formula Following Step 4 above
fv_dist_gini_vector_pos_test <- function(ar_pos) {
  # Check length and given warning
  it_n <- length(ar_pos)
  if (it_n <= 100)  warning('Data vector has n=',it_n,', max-inequality/max-gini=',(it_n-1)/(it_n +
  # Sort
  ar_pos <- sort(ar_pos)
  # formula implement
  fl_gini <- 1 - ((2/(it_n+1)) * sum(cumsum(ar_pos))*(sum(ar_pos))^(-1))
  return(fl_gini)
}
```

Generate a number of examples Arrays for testing

```
# Example Arrays of data
ar_equal_n1 = c(1)
ar_ineql_n1 = c(100)

ar_equal_n2 = c(1,1)
ar_ineql_alittle_n2 = c(1,2)
ar_ineql_somewht_n2 = c(1,2^3)
ar_ineql_alotine_n2 = c(1,2^5)
ar_ineql_veryvry_n2 = c(1,2^8)
ar_ineql_mostmst_n2 = c(1,2^13)

ar_equal_n10 = c(2,2,2,2,2,2, 2, 2, 2, 2)
```

```
ar_ineql_some_n10 = c(1,2,3,5,8,13,21,34,55,89)
ar_ineql_very_n10 = c(1,2^2,3^2,5^2,8^2,13^2,21^2,34^2,55^2,89^2)
ar_ineql_extr_n10 = c(1,2^2,3^3,5^4,8^5,13^6,21^7,34^8,55^9,89^10)
```

Now test the example arrays above using the function based no our formula:

```
##
## Small N=1 Hard-Code

## Warning in fv_dist_gini_vector_pos_test(ar_equal_n1): Data vector has n=1, max-inequality/max-gin

## ar_equal_n1: 0

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_n1): Data vector has n=1, max-inequality/max-gin

## ar_ineql_n1: 0

##
## Small N=2 Hard-Code, converge to 1/3, see formula above

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_alittle_n2): Data vector has n=2, max-inequality

## ar_ineql_alittle_n2: 0.1111111

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_somewht_n2): Data vector has n=2, max-inequality

## ar_ineql_somewht_n2: 0.2592593

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_alotine_n2): Data vector has n=2, max-inequality

## ar_ineql_alotine_n2: 0.3131313

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_veryvry_n2): Data vector has n=2, max-inequality

## ar_ineql_veryvry_n2: 0.3307393

##
## Small N=10 Hard-Code, convege to 9/11=0.8181, see formula above

## Warning in fv_dist_gini_vector_pos_test(ar_equal_n10): Data vector has n=10, max-inequality/max-g

## ar_equal_n10: 0

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_some_n10): Data vector has n=10, max-inequality/

## ar_ineql_some_n10: 0.5395514

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_very_n10): Data vector has n=10, max-inequality/

## ar_ineql_very_n10: 0.7059554

## Warning in fv_dist_gini_vector_pos_test(ar_ineql_extr_n10): Data vector has n=10, max-inequality/

## ar_ineql_extr_n10: 0.8181549
```

8.3.2 Atkinson Family Utility

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

8.3.2.1 Individual Outcomes and Preference

How does the Aktinson Family utility function work? THe Atkinson Family Utility has the following functional form.

$$V^{\text{social}} = \left(\alpha \cdot A^{\lambda} + \beta \cdot B^{\lambda}\right)^{\frac{1}{\lambda}}$$

Several key issues here:

1. V^{social} is the utility of some social planner

2. A and B are allocations for Alex and Ben.
3. α and β are biases that a social planner has for Alex and Ben: $\alpha+\beta=1$, $\alpha>0$, and $\beta>0$
4. $-\infty<\lambda\leq 1$ is a measure of inequality aversion
   - $\lambda=1$ is when the planner cares about weighted total allocations (efficient, Utilitarian)
   - $\lambda=-\infty$ is when the planner cares about only the minimum between A and B allocations (equality, Rawlsian)

What if only care about Alex? Clearly, if the planner only cares about Ben, $\beta=1$, then:

$$V^{\text{social}}=\left(B^{\lambda}\right)^{\frac{1}{\lambda}}=B$$

Clearly, regardless of the value of λ, as B increases V increases. What Happens to V when A or B increases? What is the derivative of V with respect to A or B?

$$\frac{\partial V}{\partial A}=\frac{1}{\lambda}\left(\alpha A^{\lambda}+\beta B^{\lambda}\right)^{\frac{1}{\lambda}-1}\cdot\lambda\alpha A^{\lambda-1}$$

$$\frac{\partial V}{\partial A}=\left(\alpha A^{\lambda}+\beta B^{\lambda}\right)^{\frac{1-\lambda}{\lambda}}\cdot\alpha A^{\lambda-1}>0$$

Note that $\frac{\partial V}{\partial A}>0$. When $\lambda<0$, $Z^{\lambda}>0$. For example $10^{-2}=\frac{1}{100}$. And For example $0.1^{\frac{3}{-2}}=\frac{1}{0.1^{1.5}}$. Still Positive.

While the overall V increases with increasing A, but if we did not have the outter power term, the situation is different. In particular, when $\lambda<0$:

$$\text{if }\lambda<0\ \text{ then }\ \frac{d\left(\alpha A^{\lambda}+\beta B^{\lambda}\right)}{dA}=\alpha\lambda A^{\lambda-1}<0$$

Without the outter $\frac{1}{\lambda}$ power, negative λ would lead to decreasing weighted sum. But:

$$\text{if }\lambda<0\ \text{ then }\ \frac{dG^{\frac{1}{\lambda}}}{dG}=\frac{1}{\lambda}\cdot G^{\frac{1-\lambda}{\lambda}}<0$$

so when G is increasing and $\lambda<0$, V would decrease. But when $G(A,B)$ is decreasing, as is the case with increasing A when $\lambda<0$, V will actually increase. This confirms that $\frac{\partial V}{\partial A}>0$ for $\lambda<0$. The result is symmetric for $\lambda>0$.

8.3.2.2 Indifference Curve Graph

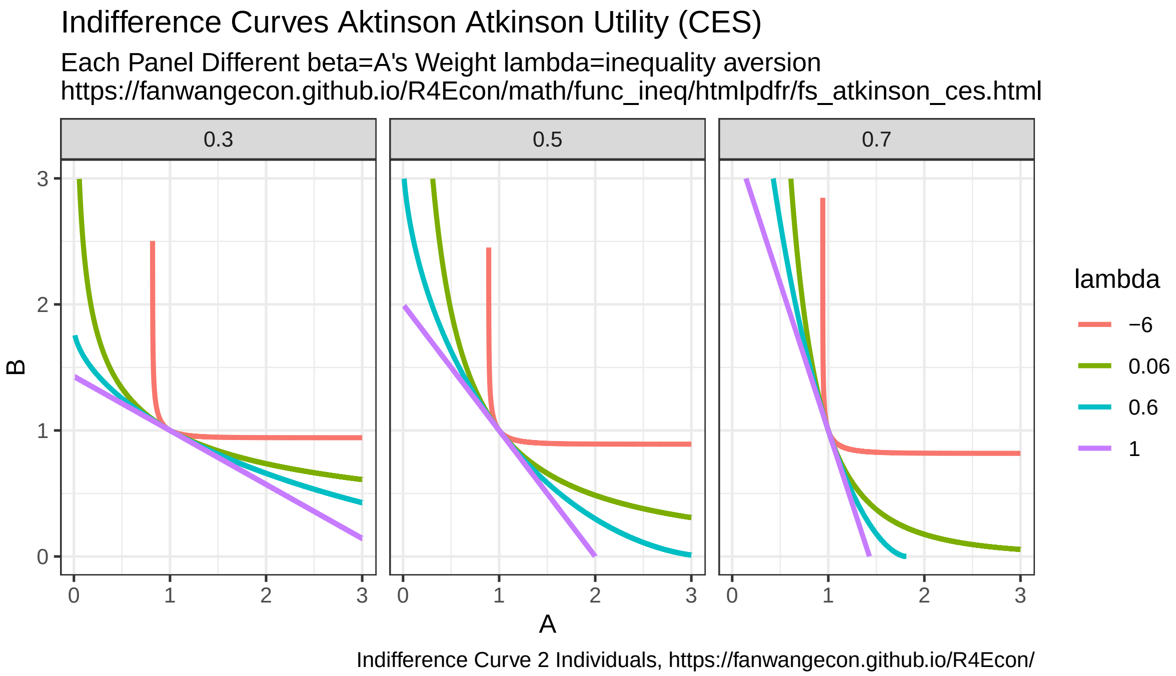
Given V^{*}, we can show the combinations of A and B points that provide the same utility. We want to be able to potentially draw multiple indifference curves at the same time. Note that indifference curves are defined by α, λ only. Each indifference curve is a set of A and B coordinates. So to generate multiple indifference curves means to generate many sets of A, B associated with different planner preferences, and then these could be graphed out.

```
# A as x-axis, need bounds on A
fl_A_min = 0.01
fl_A_max = 3
it_A_grid = 10000

# Define parameters
# ar_lambda <- 1 - (10^(c(seq(-2,2, length.out=3))))
ar_lambda <- c(1, 0.6, 0.06, -6)
ar_beta <- seq(0.25, 0.75, length.out = 3)
ar_beta <- c(0.3, 0.5, 0.7)
ar_v_star <- seq(1, 2, length.out = 1)
tb_pref <- as_tibble(cbind(ar_lambda)) %>%
  expand_grid(ar_beta) %>% expand_grid(ar_v_star) %>%
  rename_all(~c('lambda', 'beta', 'vstar')) %>%
```

```
  rowid_to_column(var = "indiff_id")

# Generate indifference points with apply and anonymous function
# tb_pref, whatever is selected from it, must be all numeric
# if there are strings, would cause conversion error.
ls_df_indiff <- apply(tb_pref, 1, function(x){
  indiff_id <- x[1]
  lambda <- x[2]
  beta <- x[3]
  vstar <- x[4]
  ar_fl_A_indiff <- seq(fl_A_min, fl_A_max, length.out=it_A_grid)
  ar_fl_B_indiff <- (((vstar^lambda) -
                      (beta*ar_fl_A_indiff^(lambda)))/(1-beta))^(1/lambda)
  mt_A_B_indiff <- cbind(indiff_id, lambda, beta, vstar,
                         ar_fl_A_indiff, ar_fl_B_indiff)
  colnames(mt_A_B_indiff) <- c('indiff_id', 'lambda', 'beta', 'vstar',
                               'indiff_A', 'indiff_B')
  tb_A_B_indiff <- as_tibble(mt_A_B_indiff) %>%
    rowid_to_column(var = "A_grid_id") %>%
    filter(indiff_B >= 0 & indiff_B <= max(ar_fl_A_indiff))
  return(tb_A_B_indiff)
})
df_indiff <- do.call(rbind, ls_df_indiff) %>% drop_na()
```

Note that many more A grid points are needed to fully plot out the leontief line.

```
# Labeling
st_title <- paste0('Indifference Curves Aktinson Atkinson Utility (CES)')
st_subtitle <- paste0('Each Panel Different beta=A\'s Weight lambda=inequality aversion\n',
                      'https://fanwangecon.github.io/',
                      'R4Econ/math/func_ineq/htmlpdfr/fs_atkinson_ces.html')
st_caption <- paste0('Indifference Curve 2 Individuals, ',
                     'https://fanwangecon.github.io/R4Econ/')
st_x_label <- 'A'
st_y_label <- 'B'

# Graphing
plt_indiff <-
  df_indiff %>% mutate(lambda = as_factor(lambda),
                       beta = as_factor(beta),
                       vstar = as_factor(vstar)) %>%
  ggplot(aes(x=indiff_A, y=indiff_B,
             colour=lambda)) +
  facet_wrap( ~ beta) +
  geom_line(size=1) +
  labs(title = st_title, subtitle = st_subtitle,
       x = st_x_label, y = st_y_label, caption = st_caption) +
  theme_bw()

# show
print(plt_indiff)
```


Indifference Curves Aktinson Atkinson Utility (CES)
Each Panel Different beta=A's Weight lambda=inequality aversion
https://fanwangecon.github.io/R4Econ/math/func_ineq/htmlpdfr/fs_atkinson_ces.html
0.3
0.5
0.7
B
A
lambda
−6
0.06
0.6
1
Indifference Curve 2 Individuals, https://fanwangecon.github.io/R4Econ/

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.