#functions that begin with "fn." are custom r functions

#sql notes bulk insert to import a csv file https://codingsight.com/sql-server-bulk-insert-part-1/

#r-notes / "lessons"

#always examine how indices in functions such as ave, tapply, lapply, sapply, treat NA values ... "ave" treats an na value of an index sorta like a category in and of itself, ratehr than grouping NA valyes together

#"select" is used in more than 1 package

# fidn a way to identoify cran mirror associated with a certain "number", so we can confirm the cran mirrow we are selecting

#use select to form dataframe

#trim top 1 percent of each dvar and ivar, by year

#aggregate

#quantile

rm(list=ls()) ; gc(); gc() #removes all objects from memory

# Compustat Classic db has dvpsxm data for S&P 500 (gvkey 3) going back to 1979

#

#load and install packages

#Install RODBC

library( matrixStats )

library(RODBC)

library(zoo)

memory.limit(size = NA)/1000 #reports memory size

memory.size(max = F)/1000 #reports amount of memory currently in use

memory.size(max = T)/1000 #reports maximum amount of memory obtained from the OS

#

memory.size(999999999) #sets memory size to ...

memory.limit(size = NA)/1000 #reports memory size

#

#dir\_raw\_text <- 'c:/Users/TC/Documents/02)RI\_data\_text'

dir\_drive <- 'c:'

dir\_root <- 'r\_ri'

dir\_subs <- c( 'txt', 'ch', 'num', 'tr', 'chi' , 'numi' )

dir\_paths <- paste(dir\_drive, dir\_root, dir\_subs, sep="/")

dir\_names <- paste(dir\_root, dir\_subs, sep=".")

for(i in 1:length(dir\_names)) {assign (dir\_names[i],dir\_paths[i])} ; rm(i)

temp <- paste(dir\_drive, dir\_root, sep='/')

if( file.exists(temp)==F) dir.create(temp)

rm(temp)

for (i in 1:length(dir\_paths) ) {

assign( dir\_names[i], dir\_paths[i] )

if( file.exists(dir\_paths[i])==F) dir.create(dir\_paths[[i]])

} # next i

################################

#utils:::menuInstallPkgs()

#local({pkg <- select.list(sort(.packages(all.available = TRUE)))

#if(nchar(pkg)) library(pkg, character.only=TRUE)})

#need to install fUtilities package?

#library(utils)

#library(plyr) #rbind.fill

#######################

z <- function(x,quarterly=T){ #adds pastes "q" to x if quarterly=T

if(quarterly==T) paste(x,'q',sep='') else x

} # end z

fn.ofn <- function(x){ #obtain filename

#fucntion to obtain filename from text-input that might contain both location-path and filename

z=gregexpr('/',x)

m=z[[1]][length(z[[1]])]

#dir.temp = substr(x,1,m-1)

substr(x,m+1,nchar(x))

} # end fn.ofn

#these functions create a name and/or save an object

fn.r <- function(x,dir.fn=NULL) { if(is.null(dir.fn)) paste(x ,'.r', sep ='') else paste(dir.fn, '/', x ,'.r', sep ='') } #creates a name in form of "dir/x.r"

fs.r <- function(x) { if( exists (fn.ofn(x) ) == T) save( list=fn.ofn(x) , file = fn.r(x) ) } #saves an object to working directory

fl.r <- function(x) { load(file = fn.r(x) , envir = .GlobalEnv) } #loads an object from working directory

fl.r.if <- function(x) { if(exists(fn.ofn(x))==F) fl.r(x) } #if an object does NOT already exist in memory, loads an object from working directory

#

fn.as <- function(x=filename,value=get(fn.ofn(x))) {#assign,save #x is a text string, value is object to be saved

#value = if(is.null(value)==F) value else get(filenam

assign(fn.ofn(x),value,pos=.GlobalEnv )

fs.r(x)

} #end fn.as end

#

fn.asr <- function( x=filename,value=get(fn.ofn (x) ) ) {#assign,save,remove #x is a text string, value is object to be saved

#value = if(is.null(value)==F) value else get(filenam

fn.as(x=x, value=value)

rm(list=x,pos=1) } #end fn.asr end

#

fn.flg <- function(x){ #load,get,remove

y=fn.ofn (x)

test <- if (mode(x)=='list') 9 else if(x=='pi') F else if (exists(x)==F) F else if (mode(get(x)) =='function') F else if (exists(x)==T) T

#test <- if(x=='pi') F else if (mode(x)=='list') 9 else if (exists(x)==F) F else if (mode(get(x)) =='function') F else if (exists(x)==T) T

#test <- if (mode(x)=='list') 9 else if (exists(x)==F) F else ( (mode(get(x))!='function')==T & (mode(get(x))!='numeric')==T )==T

#test <- if (mode(x)=='list') 9 else if (exists(x)==F) F else if (exists(x)==T) T

#temp=get(load(file = fn.r(x)) )

temp <- if(test==9) x else if(test==F) get(load(file = fn.r(x)) ) else get(x)

#if(test==F) rm(list= y , pos=1 )

#need to better understand pos argument

#if ( is.data.frame ( temp ) == T | is.matrix ( temp ) == T ) temp [ , 1 ] <- as.numeric ( temp [ , 1 ] )

temp

}#end fn.flg end

#

#load and name

fn.flga <- function(x){ #load,get,remove

assign(fn.ofn(x),fn.flg(x),pos=.GlobalEnv )

}#end fn.flga end

#

fn.flgm <- function(x){

temp <- fn.flg(x)

if( is.data.frame ( temp ) == T ) as.matrix( temp) else temp }

#end fn.flgm end

#

fn.flgma <- function(x){ #load,get,remove

temp <- fn.flg(x)

temp <- if( is.data.frame ( temp ) == T ) as.matrix( temp) else temp

assign(fn.ofn(x), temp, pos=.GlobalEnv )

}#end fn.flgma end

#

fn.lag <- function(x, lag.i){

#mydata <- if (mode(x)=='list') x else fn.flg(x)

#mydata <- if (mode(x)=='list') x else fn.flg(x)

mydata <- x

#gvs <- mydata[,1]

#if(names(mydata)[1]=='gvkey') mydata[,1] <- NA

#colnamescol1 <- colnames(mydata)[1]

w <- ncol(mydata)

mydata2 <- data.frame( matrix(data=NA, nrow=nrow(mydata), ncol=w) , stringsAsFactors = F )

#cols2 <- if(lag.i>=0) (lag.i+2):w else 2:(w+lag.i)

#cols <- if(lag.i>=0) 2:(w-lag.i) else (-lag.i+2):w

cols2 <- if(lag.i>=0) (lag.i+1):w else 1:(w+lag.i)

cols <- if(lag.i>=0) 1:(w-lag.i) else (-lag.i+1):w

mydata2[,cols2] <- mydata[,cols]

#colnames(mydata2) <- 1:ncol(mydata2)

if( length (colnames( mydata2 ) [cols2] ) == length ( colnames( mydata) [cols] ) ) colnames( mydata2 ) [cols2] <- colnames( mydata) [cols]

#nchar\_colnames <- nchar( colnames(mydata) )

#dates\_colnames <- substr(colnames(mydata), nchar\_colnames - 5, nchar\_colnames )

#if( length(colnames(mydata)[cols]) == length(cols2) ) colnames(mydata2)[cols2] <- colnames(mydata)[cols]

#if( length( colnames(mydata2)[cols2] ) == length( colnames(mydata) [cols] ) colnames(mydata2) [cols2] <- colnames(mydata )[cols]

#colnames(mydata2) <- paste(dates\_colnames, colnames(mydata2), sep='.')

#if(colnamescol1 == 'gvkey') colnames(mydata2)[1] <- 'gvkey'

#if(colnamescol1 == 'gvkey') mydata2[,1] <- gvs

mydata2

}#end fn.lag

#

fn.chk.ot.tmp <- function(x) {

#fn.chk.ot.tmp: determines whether x is 'text' (i.e. type ='charcter' & length=1) or 'object';

#returns 't' if 'text', otherwise 'o'

if(typeof(x) == 'character' & length(x)==1) 't' else 'o'} #end fn.chk.ot.tmp end

#

fn.naz <- function(x) {

x [ is.na(x) ] <- 0

x

}# end fn.naz

#

#

#fn.naz.df <- function(x){

#isdformat <- is.data.frame(x)==T | is.matrix(x) == T

#if (isdformat == F) x <- matrix (x)

#xposixct <- sapply(sapply(x,class), function(x) x[1] )=="POSIXct"

#xtest <- is.na(x)

#for ( i in 1 :ncol(x) ) {

#if (xposixct [ i ] == T) next

#x[ unlist(xtest [i]) , i] <- 0

#} #end i

#if (isdformat == F) x[,1] else x

#}# end fn.naz

#x<- fn.flga ( 'mydata\_af.0' )

#i=10

#summary(x[i])

#str( xtest[i] )

fn.nar <- function(x, repl=0){

x2 <- fn.ld(x)

repl2 <- fn.ld(repl)

rm(x, repl)

test <- is.na(x2)

x\_ismatordf <- is.matrix(x2) == T | is.data.frame(x2)

x2\_n <- if ( x\_ismatordf == T ) nrow(x2) \* ncol(x2) else length( x2)

repl2\_n <- if ( is.matrix(repl2) == T | is.data.frame(repl2) == T) nrow(repl2) \* ncol(repl2) else length( repl2)

x3 <- if(x\_ismatordf == T & repl2\_n == 1 & repl2\_n >= -Inf) data.matrix(x2) else x2

rm(x2)

if( repl2\_n == 1) x3[test] <- repl2 else if( repl2\_n == x2\_n ) x3[test] <- repl2[test] else x3 <- NULL

if ( x\_ismatordf == T ) data.frame(x3) else x3

}# end fn.nar

#

fn.ld <- function(x){if(fn.chk.ot.tmp(x)=='o') x else fn.flg(x)}#end fn.ld

#end fn.ld

#

##

fn.status <- function(x) {

# -99 = no prccms, 0 = not started, 1 = first , 2 = active, 3 = last, 9 = dead … improve to indicate "gap" 4

#df\_of\_col\_nums <- x

#df\_of\_col\_nums[ , ] <- rep(1:ncol(x) , each=nrow(x))

df\_of\_col\_nums <- col ( x )

temp <- df\_of\_col\_nums

temp[ is.na(x) ] <- NA

rowmin <- apply( X = temp, MARGIN = 1, FUN = min, na.rm=T )

rowmax <- apply( X = temp, MARGIN = 1, FUN = max, na.rm=T )

rowsum <- apply( X=temp, MARGIN = 1, FUN = sum, na.rm=T )

#rowmin [ rowmin == Inf ] <- -99

#rowmax [ rowmax == -Inf ] <- -99

temp2 <- ifelse(df\_of\_col\_nums < rowmin , 0 , ifelse(df\_of\_col\_nums == rowmin , 1, ifelse(df\_of\_col\_nums == rowmax , 3, ifelse(df\_of\_col\_nums > rowmax , 9, 2 ) ) ) )

#temp2 [is.na(rowmin) ] <- NA

temp2 [ rowsum == 0 , ] <- -99

#test <- is.na(temp2) != is.na(x)

temp2 } # end fn.status

#

##

fn.next <- function(x, cnames = NULL , col = F) {

cnamestwo <- if ( is.null(cnames) == T ) NULL else if ( is.null ( dim(cnames) ) == F) cnames else matrix(data=colnames(x), nrow=nrow(x), ncol=ncol(x), byrow =T )

x <- if(is.matrix(x) == T) x else as.matrix(x)

w <- ncol(x)

y <- if( col == F) x else matrix(data=NA, nrow=nrow(x), ncol=ncol(x) )

y [,w] <- NA

#if( is.null(cnamestwo) != T ) mynames <- cnamestwo

if( col == T) y [ , w-1 ] <- ifelse ( is.na( x [ , w] ) == F , w , NA )

if( is.null(cnamestwo) == T & col == F) y [ , w-1 ] <- x [ , w ]

if( is.null(cnamestwo) == F & col == F ) y [ , w-1 ] <- ifelse( is.na( x [ , w ] ) == F, cnamestwo [, w ] , NA )

for ( i in (w-1) : 1 ) {

if(w < 1) break()

test <- is.na( x [,i+1] ) == T

if( col == T) y [ , i ] <- ifelse( test == F, i+1 , y [, i+1] )

if( col == F & is.null(cnamestwo) == T ) y [ , i ] <- ifelse ( test == F, x [, i+1] , y [ , i+1] )

if( col == F & is.null(cnamestwo) == F) y [ , i ] <- ifelse( test == F, cnamestwo [ , i+1 ] , y [, i+1] ) }

#if( col == T) y [ , 1 ] <- ifelse( is.na (y [ , 1 ] ) == T, w, y [ , 1 ] )

data.frame( y , stringsAsFactors = F )

} # end fn.next

#

fn.prior <- function(x, cnames = NULL , col = F ) {

cnamestwo <- if ( is.null(cnames) == T ) NULL else if ( is.null ( dim(cnames) ) == F) cnames else matrix(data=colnames(x), nrow=nrow(x), ncol=ncol(x), byrow =T )

x <- if(is.matrix(x) == T) x else as.matrix(x)

w <- ncol(x)

y <- matrix(data=NA, nrow=nrow(x), ncol=ncol(x) )

#y [ , 1] <- NA

for ( i in 2 : w ) {

if( ncol(y) == 1 ) break()

test <- is.na( x [,i-1] )

if( col == T) y [ , i ] <- ifelse( test == F, i-1 , y [, i-1] )

if( col == F & is.null(cnamestwo) == T ) y [, i] <- ifelse( test == F, x [, i -1] , y [, i-1] )

if( col == F & is.null(cnamestwo) == F ) y [, i] <- ifelse( test == F, cnamestwo [, i -1] , y [, i-1] )

} #next i

data.frame( y , stringsAsFactors = F )

} # end fn.prior

#

fn.now <- function(x, cnames = NULL , col = F ) {

cnamestwo <- if ( is.null(cnames) == T ) NULL else if ( is.null ( dim(cnames) ) == F) cnames else matrix(data=colnames(x), nrow=nrow(x), ncol=ncol(x), byrow =T )

x <- if(is.matrix(x) == T) x else as.matrix(x)

#if( is.null(cnamestwo) != T ) mynames <- cnamestwo

if( is.null(cnamestwo) == F ) x <- cnamestwo

temp <- matrix( data = 1 : ncol ( x ) , ncol = ncol ( x ) , nrow = nrow ( x ) , byrow=T )

if( col == T ) temp [ is.na(x) ] <- NA

if( col == T ) x <- temp

data.frame( x , stringsAsFactors = F )

} # end fn\_now

fn.nowpriorlb <- function( x, lb ) {

colnow <- col ( x )

colnow [ is.na (x ) == T ] <- NA

colprior <- fn.prior( x , cnames = NULL , col = T )

priorx <- fn.prior( x , cnames = NULL , col = F )

colnowprior <- fn.nar ( colnow , colprior )

xnowprior <- fn.nar ( x , priorx )

mycols <- col ( x )

mytest <- ( mycols - colnowprior ) >= lb

xnowprior [ mytest == T ] <- NA

xnowprior

} #end fn.nowpriorlb

#

fn.mm <- function(x, min.i=-Inf, max.i=Inf,s=F) {

x[ x>max.i] <- if(s==F) max.i else NA

x[ x<min.i] <- if(s==F) min.i else NA

x

} #end minmax

#

fn.rollapply.skip <- function(data.i, periods=36, skip=12, fun.i, na.rm.i=F , partial.i = F, align.i){

#need to install zoo package

#library(zoo)

w=ncol(data.i)

h=nrow(data.i)

temp <- matrix(data=NA, ncol=w, nrow=h)

for(i in 1:skip){

cols <- seq( from=i+1, by=periods, to=w)

mymat <- data.i[cols]

temp2 <- t( rollapply(data = t(mymat), width = periods/skip, FUN = fun.i, na.rm=na.rm.i , by.column=T, fill = NA, partial = partial.i, align = align.i ))

temp[cols] <- temp2

#names(temp)[cols] <- names(mymat)

}

temp} #end fn.rollapply.skip

#

tryCatch.W.E <- function(expr)

{

W <- NULL

w.handler <- function(w){ # warning handler

W <<- w

invokeRestart("muffleWarning")

}

list(value = withCallingHandlers(tryCatch(expr, error = function(e) e),

warning = w.handler),

warning = W)

} # end trycatch

#

#

#

#

#

#

#

#

#

#

#

#

fn.n <- function( x ) { if( is.numeric(x) == F) 0 else sum( x >= - Inf, na.rm=TRUE) }

#fn.counta <- function( x) { sum( is.na(x)==F, na.rm=TRUE) }

fn.countna <- function( x) { sum( is.na(x)==T | is.nan(x)==T, na.rm=TRUE) }

fn.ntile <- function( mydata, ntiles ) { cut( mydata, quantile( x= mydata, probs = ( 0:ntiles ) /ntiles , na.rm = T, type=4), include.lowest = T, labels = F) } # end fn

#fn.ntile <- function( mydata, ntiles ) { ceiling ( ( rank(mydata , na.last = 'keep' , ties.method= "min" ) / sum( mydata >= -Inf, na.rm = T ) ) / (1 / ntiles) )}

fn.agg <- function( x , by , FUN.i ) {

bys\_unique <- sort( unique( as.vector (by) ) , na.last = T )

by2 <- if( is.matrix(by) == T ) by else matrix( data = by , nrow = length(by) , ncol = ncol( x ) )

rm(by)

myres <- matrix( data=NA, nrow = length(bys\_unique) , ncol = ncol(x) )

for(i in 1:length(bys\_unique) ) {

temp <- x

test <- if( is.na( bys\_unique[ i ] ) == F) by2 == bys\_unique[ i ] else is.na(by2) == T

temp [ test == F | is.na(test) == T ] <- NA

myres [ i , ] <- if( class(FUN.i) != 'character') apply( X= temp , MARGIN=2 , FUN = FUN.i , na.rm=T) else if( FUN.i == 'count') apply( X= temp >= - Inf, MARGIN=2 , FUN = sum , na.rm=T) else NULL

} # end i loop

rownames(myres) <- bys\_unique

myres

} # end fn

###

fn.fnadj <- function( mydataitem ) {

#

temp <- getwd()

setwd(r\_ri.ch)

atf <- fn.flga ('atf[@yr(0m)]')

salef <- fn.flga ('salef[@yr(0m)]')

#

setwd(temp)

atf.test<- fn.nar( atf == 'AG' | atf == 'TL' | atf == 'GL' | atf == 'DB' )

salef.test<- fn.nar( salef == 'AG' | salef == 'TL' | salef == 'GL' | salef == 'DB' )

test <- atf.test == T | salef.test == T

setwd(r\_ri.num)

mydata <- fn.flg( mydataitem)

mydata [test==T] <- NA

fn.asr( paste (mydataitem, '\_fna' , sep="" ), mydata )

} # end fn.fnadj

#

fn.comb\_bm\_data <- function( mydataitem, bm.gv\_iid = "3 \_01" ) {

bm.row <- match( bm.gv\_iid , fn.flg('gvkey\_iid') )

mydata <- fn.flg(mydataitem)

fn.flga('bm\_Earnings\_E')

temp <- fn.nar( mydata[bm.row, 2:ncol(mydata) ] , bm\_Earnings\_E[1, 2:ncol(mydata) ] )

mydata [ bm.row , 2:ncol(mydata) ] <- unlist( temp , use.names=F )

fn.asr ( paste( mydataitem, "\_wBMrepl", sep="" ), mydata )

} # end fn.comb\_bm\_data

#

fn.inverse <- function(x) {

inverse2 <- fn.flg ( paste( '1\_db\_', x, sep="") )

inverse2[1] <- NA

raw <- fn.flg (x)

raw[1] <- NA

inverse\_inverse <- 1/inverse2

test <- fn.nar(raw < 1 & inverse\_inverse > raw, F)

raw [test] <- inverse\_inverse [test]

raw

}# end fn.inverse

#

#

fn.flgnu <-function(x) {

y <- fn.flg(x)

y [1] <- as.numeric(NA)

# y <- as.matrix(y, colnames = NULL )

y } # end fn.flgnu

#

#

fn.mad <- function( x, na.rm = T) { median( abs( x - median (x , na.rm = na.rm) ) , na.rm = na.rm) }

#

#

fn.winmad <- function( x, lim=5, reps = 3, na.rm = T) {

x <- unlist(x)

for( r in 1: reps) {

mymad\_lim <- fn.mad ( x, na.rm= na.rm ) \* lim

mytest <- fn.nar( abs( x ) > mymad\_lim , F )

x <- ifelse( mytest == T , sign(x) \* mymad\_lim , x)

} # end r

x

} # end fn.winmad

#

fn.winsd <- function( x, lim=3, reps = 3, na.rm = T) {

x <- unlist(x)

for( r in 1: reps) {

mysd\_lim <- sd ( x, na.rm = na.rm ) \* lim

mytest <- fn.nar( abs( x ) > mysd\_lim , F )

x <- ifelse( mytest == T , sign(x) \* mysd\_lim , x)

} # end r

x

} # end fn.winsd

#

fn.dots <- function(x, fun, na.rm=F, ...) {

#applies a function to differnt object types

#useful for: max, min, certain custom functions(fn.md,fn.slpal, fn.count?)

#work:take naming convention from function; add names to dfs and vectors; pass name as attribute to ther types

#work need to alter so i can pass multiple/flexible arguments to it...#perhaps use do.call?

#work: try mode to check type?

#work use class to check: matrix, df, list?

#x2=fn.fc(x)

x2= x

#gfun=get(fun)

gfun=fn.fc(fun)

if (is.matrix(x2)) apply(x2, 2, gfun, na.rm=na.rm, ...) else

if (is.data.frame(x2)==F && is.matrix(x2)==F && is.list(x2)==T) lapply(x2, gfun, na.rm=na.rm, ...) else

if (is.vector(x2)) gfun(x2, na.rm=na.rm, ...) else

if (is.data.frame(x2)) sapply(x2, gfun,na.rm=na.rm, ...) else

gfun(as.vector(x2), na.rm=na.rm)

###lapply(x2, gfun, na.rm=na.rm)

} #end fn.dots end

#

fn.fc <- function(x,cn=1) {

#work: if x is a vector,, shoudl i return a vector? preserve the typeof of x?

#takes a dataframe or object("x") and an optional column number or column name ("cn"),

#and returns a 1 column dataframe with the column name;

#x can be the name of the object, or the object itself (e.g. 'x' or x)

#if x is not a dataframe, the name will be 'noname'

#inputs: 'x'=a dataframe or object; 'cn'=an optional column number or column name

chk = fn.chk.ot.tmp(x)

x1 = if(chk=='t') fn.flg(x) else x

#ncol.x1 = ncol(x1)

#idea...embed cn within text string submitted to fn.fc...

#have fristcol get cn from within the etxt string

#cn.b = if(typeof(cn) == 'double'||typeof(cn) == 'integer') cn else which(names(x) == cn)

#temp0 = if(is.null(ncol.x1)==T && cn.b==1) data.frame (x1 , check.names=T) else data.frame (x1[,cn.b] , check.names=T )

#temp0 = if(is.null(ncol.x1)==T && cn.b==1) x1 else x1[cn.b]

#nm = names(x1)[cn.b]

#names(temp0) = if(chk=='t') ( if(is.null(nm)==F) nm else x) else (if(is.null(nm)==F) nm else 'noname')

#temp0

x1

} #end fn.fc end

#

#

fn.pctchg <- function(x , lag.i ) {

mydf <- fn.flga( x )

mydf\_lag <- fn.lag(mydf , lag.i)

mydf\_lag [ mydf\_lag <= 0 ] <- NA

(mydf / mydf\_lag ) -1

} #end fn.pctchg

#

#http://www.datasciencemadesimple.com/calculate-percentile-quantile-n-tile-of-dataframe-in-r-using-dplyr-create-column-with-percentile-rank/

#library(dplyr)

#mydata<-mtcars

#df1 = mutate(mydata, decile\_rank = ntile(mydata$mpg,10))

#df1

#

#library(dplyr)

#mydata<-mtcars

#df1 = mutate(mydata, percentile\_rank = ntile(mydata$mpg,100))

#df1

#percent\_rank(x)

#library(dplyr)

# dataf %>%

# group\_by(grpvar1, grpvar2) %>%

# mutate(percrank=rank(value)/length(value))

#df %>% group\_by(Year)

#df %>%

# group\_by(Year) %>%

# mutate(numbers\_per\_K\_percent\_rank = percent\_rank(numbers\_per\_K))

#fn.pr <- function( x , na.last = 'keep' ) { ( rank ( x , na.last = na.last , ties = 'average' ) - 1 ) / sum( is.na(x) == F ) }

fn.pr <- function( x, na.rm = "keep", repl = NULL ) {

temp <- ( (rank ( x, ties.method= "average" , na.last = na.rm ) - 0.5 ) / fn.n ( x ) )

if( is.null (repl) == T ) temp else ifelse ( is.na(x) == T, repl, temp) } #end fn.pr

#

#try using ave instead of tapply

#also learn about "Split-Apply-Combine" https://campus.datacamp.com/courses/introduction-to-r-for-finance/lists-5?ex=9

fn.pri.ave <- function( x, na.rm = 'keep', repl = NULL , index = NULL ) {

#percentile rank within an "index" (i.e. a group)

index <- if( is.null (index) == T ) rep( -99, length(x) ) else index

temp <- ave( x = x, index , FUN = fn.pr, na.rm = na.rm, repl = repl)

if( length( x ) == length( index) ) temp else 'length of index is not equal to length of x'

} #end fn.pri.ave end

#

fn.pri <- function( x, na.rm = 'keep', repl = NULL , index = NULL ) {

#percentile rank within an "index" (i.e. a group)

index <- if( is.null (index) == T ) rep( -99, length(x) ) else index

temp1 <- data.frame( x, index, rn = row( as.matrix(x) ), stringsAsFactors=F )

rownames ( temp1 ) = 1:nrow(temp1)

#temp2 <- temp1[order(temp1[,2]), ]

#temp3 <- unlist (tapply(X=temp2[,1] , FUN = fn.pr , INDEX = list(temp2[,2]) ) )

#temp5 <- temp4[order(temp4[,2]), ]

#temp6 <- data.frame(temp5[,1])

temp2 <- temp1[ order(temp1$index ), ]

temp3 <- unlist (tapply(X=temp2[,1] , FUN = fn.pr , INDEX = list(temp2$index) , na.rm = na.rm ) )

temp4 <- data.frame ( pr = temp3 , rn = temp2$rn , stringsAsFactors = F )

temp5 <- temp4[order(temp4$rn), ]

if( length( x ) == length( index) ) temp5$pr else 'length of index is not equal to length of x'

} #end fn.pri end

#

fn.tapplyi <- function( x, repl = NULL , index = NULL , fun.i ) {

#percentile rank within an "index" (i.e. a group)

index <- if( is.null (index) == T ) rep( -99, length(x) ) else index

temp1 <- data.frame( x, index, rn = row( as.matrix(x) ), stringsAsFactors=F )

rownames ( temp1 ) = 1:nrow(temp1)

#temp2 <- temp1[order(temp1[,2]), ]

#temp3 <- unlist (tapply(X=temp2[,1] , FUN = fn.pr , INDEX = list(temp2[,2]) ) )

#temp5 <- temp4[order(temp4[,2]), ]

#temp6 <- data.frame(temp5[,1])

temp2 <- temp1[ order(temp1$index ), ]

temp3 <- unlist (tapply(X=temp2[,1] , FUN = fun.i , INDEX = list(temp2$index) ) )

temp4 <- data.frame ( pr = temp3 , rn = temp2$rn , stringsAsFactors = F )

temp5 <- temp4[order(temp4$rn), ]

if( length( x ) == length( index) ) temp5$pr else 'length of index is not equal to length of x'

} #end fn.tapplyi end

#

fn.quantile <- function ( x, na.rm = 'keep', repl = NULL , index = NULL, nq = 5 ) { floor( ( fn.pri ( x = x, na.rm = na.rm, repl = repl , index = index ) \* nq ) ) + 1 }

fn.std <- function (x) { sd (x) / sqrt ( fn.n (x) ) }

#

#

fn.ranki <- function( x, na.last = 'keep', repl = NULL , index = NULL ) {

#percentile rank within an "index" (i.e. a group)

index <- if( is.null (index) == T ) rep( -99, length(x) ) else index

temp1 <- data.frame( x, index, rn = row( as.matrix(x) ), stringsAsFactors=F )

rownames ( temp1 ) = 1:nrow(temp1)

#temp2 <- temp1[order(temp1[,2]), ]

#temp3 <- unlist (tapply(X=temp2[,1] , FUN = fn.pr , INDEX = list(temp2[,2]) ) )

#temp5 <- temp4[order(temp4[,2]), ]

#temp6 <- data.frame(temp5[,1])

temp2 <- temp1[ order(temp1$index ), ]

temp3 <- unlist (tapply(X=temp2[,1] , FUN = rank , INDEX = list(temp2$index) , na.last = na.last ) )

temp4 <-data.frame ( pr = temp3 , rn = temp2$rn)

temp5 <- temp4[order(temp4$rn), ]

if( length( x ) == length( index) ) temp5$pr else 'length of index is not equal to length of x'

} #end fn.ranki end

#

fn.gm <- function(x, na.rm=TRUE, zero.propagate = T ){

if(any(x < 0, na.rm = TRUE)){

return(NaN)

}

if(zero.propagate){

if(any(x == 0, na.rm = TRUE)){

return(0)

}

exp(mean(log(x), na.rm = na.rm))

} else {

exp(sum(log(x[x > 0]), na.rm=na.rm) / length(x))

}

}

#

coalesce <- function(...) {

Reduce(function(x, y) {

i <- which(is.na(x))

y2 <- if( length(y) > 1 | length(y) == length(x) ) y else rep(y, length(x) )

x[i] <- y2[i]

#x[i] <- y[i]

x},

list(...))

}

elapsed\_months <- function(end\_date, start\_date) {

ed <- as.POSIXlt(end\_date)

sd <- as.POSIXlt(start\_date)

12 \* (ed$year - sd$year) + (ed$mon - sd$mon)

}

##########

setwd(r\_ri.num)

#fn.flga ( 'gvkey\_iid' )

#bm.row <- match( "3\\I01" , fn.flg( 'gvkey\_iid' ) ) #bm.row <- 20986

#bm.gvkeyiid <- "3\\I01"

#bm.row <- match( bm.gvkeyiid , gvkey\_iid ) #bm.row <- 20986

#

################

gc();gc()

memory.size(max = F)/1000 #reports amount of memory currently in use

#install.packages("zoo")

#list.start <- ls()

#rm(list=ls()[is.na(match(ls(), c(list.start, 'list.start')))]);gc();gc()

library(RODBC)

#install (tidyverse)

library(tidyverse)

#https://www.datanovia.com/en/lessons/select-data-frame-columns-in-r/

ch <- odbcConnect("xpressfeed", uid = "xpressfeed", pwd = "xpressfeed")

setwd(r\_ri.num)

fn.trimpct <- function(x, probs= 0.99, type = 7) {

trimlimits <- sapply(x, FUN = quantile, probs = probs, Type = type , na.rm = T)

trimlimits\_mat <- matrix(data = trimlimits, nrow = nrow(x), ncol = ncol(x), byrow = T )

x2 <- x

mytest <- fn.nar(x2 >= trimlimits\_mat, F )

x2 [ mytest == T ] <- trimlimits\_mat [ mytest == T ]

x2

}

x <- data.frame( 1:100, 1:1000 )

x2

summary(x)

summary(x2)

#summary(x)

#summary(trimlimits)

#summary(trimlimits\_df)

#summary(x2 >= trimlimits\_df )

#summary(trimlimits\_df [ x2 >= trimlimits\_df ])

#summary(x2)

#summary(trimlimits\_df)

#summary(trimlimits [ x2 >= trimlimits\_df ] )

#maybe calc a function to calc change between 2 items, that sets change to NA if denominator is <=0 ex. fn.dv(num, den)

fn.dv <- function (num, den) {

x <- (num/den)

x[den <=0] <- NA

x

} #end fn.dv

#

######################################

library(lubridate)

#notes/fixes

#fn.dfsum(myprefix = 'mydata\_af\_nums.' , mysuff = NULL , mystart = 5 , myend = 3 , npers = 3 , na.rm = T ) #~3 mins

#myprefix = 'mydata\_af\_nums.' ; mysuff = NULL ; mystart = 5 ; myend = 3 ; npers = 3 ; na.rm = T

#i <- mystart

#fix: change "npers" in fn.dfsum to "nper"

#create function to sum dfs

#calc tr3 year data for: co\_afnd, co\_afnd\_adj, co\_afnd\_psnums #calc cumulative forward data #re-write this as a function; experiment woth data.table

fn.dfsum <- function( myprefix, mysuff, mystart , myend , npers , na.rm = F ) {

#for ( i in mystart : (myend+npers-1) ) {

for ( i in mystart : myend ) {

endi <- i - npers + 1

for (p in i : endi ) {

myfilename <- paste(myprefix, if(p<0) paste('m', abs(p), sep='') else p, mysuff, sep = '')

mydata <- fn.flga(myfilename)

if ( p == i ) temp <- if(na.rm == T) fn.naz(mydata ) else mydata

if ( p != i ) temp <- if(na.rm == T) temp + fn.naz(mydata ) else temp + mydata

} #end p

temp <- data.frame (temp)

#add colnames

midname <- paste ( i, 't', endi, sep='')

#colnameadj <- ifelse(substr(colnames(temp), nchar(colnames(temp))-1, nchar(colnames(temp)) ) == '.j','.j','')

#substr(colnames(mydata),1,regexpr("\\.[^\\.]\*$", colnames(mydata)))

#colnames(temp) <- paste ( substr(colnames(mydata),1,regexpr("\\.[^\\.]\*$", colnames(mydata))), midname, colnameadj, sep='')

#colnames(temp) <- paste ( colnames(mydata), midname, sep='.')

#create name for the df

#from1 <- i

#from2 <- if(from1 < 0) paste('m', abs(from1), sep='') else from1

#to1 <- if(endi < 0) paste('m', abs(endi), sep='') else endi

#fn.asr( paste(myprefix,'.', mysuff,'.', from2, 't', to1, if(na.rm==T) '.nna', sep = '') , temp )

myfilename <- paste(myprefix, midname, mysuff, if(na.rm==T) '.nna', sep = '')

fn.asr( myfilename , temp)

} # end calc cumulative data

} #end fn

######################################

#create function to create entire per share split\_adjustred dataframe, use with co\_afnd, co\_afnd\_nums, co\_afndtr3, co\_afnd\_nums3

#create a df with flag indicating if each item should be split adjusted, this will be used when we create our custom ".ps" dataframe

fn.flga('my\_xfl\_column')

fn.flga('mycolnames\_af')

my\_xfl.adj\_tables.names <- paste ( my\_xfl\_column$tablename , my\_xfl\_column$columnname, sep = '.' )

my\_xfl\_column\_row <- match ( mycolnames\_af , my\_xfl.adj\_tables.names )

my.adj\_column <- my\_xfl\_column$adj\_column [ my\_xfl\_column\_row ]

my\_co\_afnd\_col\_noadj <- is.na ( my.adj\_column) == F & my.adj\_column == ''

#checks/review

#mycolnames\_af[is.na(my.adj\_column)==T]

#mycolnames\_af[my\_co\_afnd\_col\_noadj==F] #might be interesting to review items that in Compustat db are exprssed in a per share format

#mycolnames\_af[my\_co\_afnd\_col\_noadj==T]

fn.as('my\_co\_afnd\_col\_noadj')

###

#calc df with per share data for entire table, multiplying shares and numerator by 1 million

#note: SALE, IBCOM, and DVC are all displayed in millions

x = 'mydata\_af\_nums.m1' ; shsper = 'm1'

fn.dfpershare <- function( x , shsper) {

#myfilename <- paste(myprefix, if(p<0) paste('m', abs(p), sep='') else p, mysuff, sep = '')

mynumdf <- 1000000 \* fn.flga(x)

mydata\_ad <- fn.flga( paste('mydata\_ad.', shsper, sep = '') )

mydata\_shs <- fn.flga( paste('mydata\_af\_numsb.', shsper, '.adj', sep = '') )

myshs <- 1000000 \* coalesce(mydata\_shs$cshfd, mydata\_shs$cshpri)

summary(myshs < 1)

myshs [myshs < 1 ] <- NA #set share counts < 1 to NA, to ensure share count is >= 1

myshs\_df <- data.frame(matrix(data=myshs, nrow = nrow(mydata\_shs), ncol = ncol(mydata\_shs) , byrow = F ) )

rm(myshs)

#colnames(myshs\_df) <- colnames(mydata\_shs)

#tail(myshs\_df[1:10])

mypsdf <- mynumdf / myshs\_df

#split adjust entire mypsdf table

ajex\_df <- data.frame(matrix(data=mydata\_ad$ajex, nrow = nrow(mypsdf) , ncol = ncol(mypsdf), byrow = F))

#colnames(ajex\_df) <- colnames(mynumdf)

#tail(ajex\_df[1:10])

mypsdf\_adj <- mypsdf / ajex\_df

#colnames(mypsdf\_adj)

#colnames(mypsdf\_adj) <- paste(colnames(mypsdf\_adj),'.ps.j', sep='') #add line here to add.j to colnames

#colnames(mypsdf\_adj)

mycombodf\_adj <- mypsdf\_adj

#in mycombodf\_adj, for data that was already a per share item, replace with NA or with split-adjusted original per share item

#my\_co\_afnd\_col\_noadj

#mycolnames\_af[my\_co\_afnd\_col\_noadj==F] #check

fn.flga('my\_co\_afnd\_col\_noadj')

#mycombodf\_adj [ my\_co\_afnd\_col\_noadj == F ] <- mydata\_af.adj [ my\_co\_afnd\_col\_noadj == F]

mycombodf\_adj [ my\_co\_afnd\_col\_noadj == F ] <- NA

#check

#mycheck <- data.frame(( mycombodf\_adj != mydata\_af.adj & mynumdf != 0 & myshs\_df != 1)) [ , my\_co\_afnd\_col\_noadj == T] #checks dataitems not originally split-adjusted in Compustat; these "shoudl be" not equal

#fn.flga('mydata\_af.m5')

#temp <- data.frame(mydata\_af.m5 = subset(mydata\_af.m5$at.m5, mycheck$at.m5 == F) , mydata\_af = subset(mydata\_af$at.m5, mycheck$at.m5 == F), mydata\_af.adj = subset(mydata\_af.adj$at.m5, mycheck$at.m5 == F), ajex = subset(ajex\_df$at.m5, mycheck$at.m5 == F), num = subset(mynumdf$at.m5, mycheck$at.m5 == F), den = subset(myshs\_df$at.m5, mycheck$at.m5 == F) , combo = subset(mycombodf\_adj$at.m5.j, mycheck$at.m5 == F) )

#mycheck <- ( mycombodf\_adj == mydata\_af.adj ) [ , my\_co\_afnd\_col\_noadj == F] #checks dataitems that are originally split-adjusted in Compustat

#summary( as.vector(mycheck)) #check this should all Be Trues

#in mycombodf\_adj, for columns that were not already a per share item, set colname to colname of split-adjusted original per share item

#colnames (mycombodf\_adj) [ my\_co\_afnd\_col\_noadj == F ] <- colnames(mydata\_af.adj) [my\_co\_afnd\_col\_noadj == F]

#colnames (mycombodf\_adj) [ my\_co\_afnd\_col\_noadj == F ] <- NA

#colnames (mycombodf\_adj) [ my\_co\_afnd\_col\_noadj == T ] <- paste(colnames(mydata\_shs), 'ps.j', sep = '.') [ my\_co\_afnd\_col\_noadj == T ]

#save dataframe

#tempfilename <- paste( "mydata\_af", mysuff, sep='')

#fn.asr( tempfilename , mydata\_af )

fn.asr( paste ( x, '.ps', shsper, sep='') , mycombodf\_adj )

} #end fn

###

#library(lubridate) ; ceiling\_date(Sys.Date(), "month") - days(1)

#eom <- function(date) {

# # date character string containing POSIXct date

# date.lt <- as.POSIXlt(date) # add a month, then subtract a day:

# mon <- date.lt$mon + 2

# year <- date.lt$year

# year <- year + as.integer(mon==13) # if month was December add a year

# mon[mon==13] <- 1

# iso = ISOdate(1900+year, mon, 1, hour=0, tz=attr(date,"tz"))

# result = as.POSIXct(iso) - 86400 # subtract one day

# result + (as.POSIXlt(iso)$isdst - as.POSIXlt(result)$isdst)\*3600

#}

#create function to trim x several ways, based on an INDEX value that is a date

#as benchmark for trimming values, use all data that is within 12 months of INDEX,

#x = dilavx\_ibadj\_xsp\_fdpr..0; index = mydata\_use$dd\_af0; prmax = 0.95 ; prmin = 0.05 ; mult = 1.5 ; trim = F; na.rm=T; medout = T

#x = dilavx\_ibadj\_xsp\_fdpr..0; indexdate = mydata\_use$dd\_af0; index2 = sich2\_use; prmax = 0.95 ; prmin = 0.05 ; mult = 1.5 ; trim = F; na.rm=T; medout = T

#x = dilavx\_ibadj\_xsp\_fdpr..0; indexdate = mydata\_use$dd\_af0; index2 = NULL; prmax = 0.95 ; prmin = 0.05 ; mult = 1.5 ; trim = F; na.rm=T; medout = T

fn.trmqntl <- function (x , indexdate, index2 = NULL, prmax = 0.95 , prmin = 0.05 , mult = 1.5 , trim = T, na.rm = F, medout = F) {

indexdate <- as.Date(indexdate)

x2 <- if(medout==F) x else if(medout==T) rep(NA, length(x))

if( is.null(index2) == T) index2 <- rep(-99, length(x))

indexcombined <- data.frame(indexdate, index2)

#myindexdate\_unique <- unique(indexdate)

#myindex2\_unique <- unique(index2)

myindexcombined\_unique <- unique(indexcombined)

for (i in 1: nrow(myindexcombined\_unique) ) {

#myindexdate <- myindexcombined\_unique$indexdate[i]

#myindex2 <- myindexcombined\_unique$index2[i]

myindexcombined <- myindexcombined\_unique[i,]

#x\_ss <- subset(x, indexcombined$indexdate <= myindexcombined$indexdate & indexcombined$indexdate > eom((myindexcombined$indexdate) - (367\*60\*60\*24)) & myindexcombined$index2 == myindexcombined$index2 )

x\_ss <- subset(x, indexcombined$indexdate <= myindexcombined$indexdate & indexcombined$indexdate > ceiling\_date(myindexcombined$indexdate- 367, "month") -1 & myindexcombined$index2 == myindexcombined$index2 )

x\_med <- median(x\_ss, na.rm=T)

mytest <- indexdate == myindexcombined$indexdate & index2 == myindexcombined$index2

dothis <- if(medout==F) T else F

##############################

#start while

while (dothis == T){

x\_min <- ((quantile(x\_ss, probs=prmin, type=8, na.rm = na.rm) - x\_med) \* mult) + x\_med

x\_max <- ((quantile(x\_ss, probs=prmax, type=8, na.rm = na.rm) - x\_med) \* mult) + x\_med

x\_dd <- x2 [mytest]

if(trim == T) x\_dd <- fn.mm(x\_dd, x\_min, x\_max)

if(trim != T) x\_dd [ x\_dd < x\_min | x\_dd > x\_max] <- NA

if(medout==F) x2 [mytest] <- x\_dd

dothis <- F

} #end while

##############################

if(medout==T) x2 [mytest] <- x\_med

} #next i

x2

}#end fn

#

#x = mydvars\_use$dvar\_eps\_wrepl\_dvhcv\_roa , index = mydata\_use$dd\_af0, prmax = 0.95 , prmin = 0.05 , mult = 1.5

#xnew <- fn.trmqntl(x = mydvars\_use$dvar\_eps\_wrepl\_dvhcv\_roa , index = mydata\_use$dd\_af0, prmax = 0.95 , prmin = 0.05 , mult = 1.5 )

#END FUNCTIONS

use tempdb

go

dbcc shrinkfile (tempdev, 8)

go

-- this command shrinks the primary data file

dbcc shrinkfile (templog, 8)

go

-- this command shrinks the log file, examine the last paragraph.

dbcc shrinkfile (temp2, 8)

go

dbcc shrinkfile (temp3, 8)

go

dbcc shrinkfile (temp4, 8)

go

select (size\*8) as FileSizeKB from sys.database\_files

go

select \* from sys.database\_files

USE [xpressfeed]

GO

DBCC SHRINKDATABASE(N'xpressfeed' )

GO

USE [xpressfeed]

GO

DBCC SHRINKFILE (N'xpressfeed' , 0, TRUNCATEONLY)

GO

--ALTER INDEX ALL ON Table\_name

--REBUILD ;

--https://solutioncenter.apexsql.com/how-to-automate-sql-server-defragmentation-using-policies/

###

###

#end fns

#