Factor Model User Guide Version 1.0

Michael Di Pentima, Head of Factor Model Developement (Fall 19)

May 28, 2019

# Factor Model User Guide

## Version 1.0

## Table of Contents

* [Introduction](#intro)
* [Overview](#over)
* [main.R](#main)
* [packageLoad.R](#package)
* [New\_Factor\_Model\_Data.R](#dataGrab)
* [DataLoad.R](#dataLoad)
* [linearAnalysis.R](#linear)
* [arimaAnalysis.R](#arima)
* [randomForestAnalysis.R](#rf)
* [weight.R](#weight)
* [backtest.R](#backtest)
* [fullTest.R](#fulltest)

## Introduction

Welcome! If you’re reading this, that means that you’ve begun to work on the Stevens Student Managed Investment Fund’s Factor Modelling System. Congratulations, you’re in for a fun time. In the next couple dozen pages, we’ll go through the entire factor modelling system in depth to give you the best understanding we can of the entire system before you start working.

## Overview

The purpose of this short section is to tell you about the system itself and give a brief overview of its components.

This system is designed to download daily factor data and use it to train various models to predict market movements so that we can use that predicted data to find the optimal set of weights to help the fund deliver on its goals of beating the benchmark with less risk. To that that, the current system uses three methods: a linear regression, ARIMA-regression, and random forest regression to determine best fit models for each sector. Those models are combined and their results aggregated in order to predict how the market should move. Using that data, we generate sets of weights which satisfy a constrained optimization process. Each of those sets is tested again over the entire dataset and constrained again to find the best series of weights for our needs.

What makes this system so wonderful is that the entire process is relatively fast (the entire system can be run in less than an hour), is designed from a modular standpoint (each piece can be understood individually), and that the constraint systems are customizable to allow for a specified search far beyond the scope of classical models such as Mean-Variance Optimization.

## main.R

main.R is our command station. From here, we can run everything we need to throughout the system in one shot. We’re going to reference items in this file as we work on the others, so those few lines will be labeled when we get to them.

We’re going to make another note here. For progress’s sake, each file contains a print statement at the bottom alerting the user when each file has finished running. We’re going to skip over those because they’re non-essential and simple enough.

## packageLoad.R

packageLoad.R is exactly what it sounds like: A file which contains all of our package laoding routines. Because all of these subroutines are nearly identical (the only differences are which packages are loaded), we’re going to use Quantmod as the example:

if (require(quantmod) == FALSE) {

install.packages("quantmod")

library(quantmod)

} else {

library(quantmod)

}

Because require returns a boolean value depending on whether or not a package can be loaded, we use that to check for package existence. If the package doesn’t exist, we install and load it. If it exists, we simply load it.

This file usually takes a while on a computer that has never run the system before, but once the system has been run package loading becomes a lot faster.

## New\_Factor\_Model\_Data.R

From main:

YEARS\_IN\_TEST = 3

tryCatch(source("New\_Factor\_Model\_Data.R"),finally = source("DataLoad.R"))

YEARS\_IN\_TEST is a global variable used to establish the differences between the training and testing sets. With 14 years of total data, this number is set to test our predictions on the last three years in that set. Then we initiate our first two files: New\_Factor\_Model\_Data.R if you’re on a Bloomberg Terminal and DataLoad.R otherwise.

blpConnect()

data.startDate <- as.Date("9/30/2005", format = "%m/%d/%Y")

Time\_Interval = seq(data.startDate,Sys.Date(),1)

opt <- c("periodicitySelection" = "DAILY")

sectorNames <- c("IT","FIN","ENG","HLTH","CONS","COND","INDU","UTIL","TELS","MATR")

The first line here tries to connect you to the Bloomberg Terminal. If you are logged into Bloomberg, it’ll connect. If not, the connection fails. This is why we used tryCatch above; when that connection fails, R takes that error message, skips this file (since we can’t use it), and moves on to DataLoad.R.

The next four lines create variables we’re going to use throughout the program. data.startDate is the initial date for our data, September 30th, 2005. Time\_Interval is a Date vector containing all of the dates between our starting date and today. opt contains the option parameter that forces all of our data to be recorded daily. Finally, sectorNames contains the common names of each sector.

merge.all <- function(x, y) {

merge(x, y, all=TRUE, by="date")

}

spx.data <- function() {

spx <- bdh("SPX Index", c("PX\_LAST"), start.date = data.startDate,

options = opt)

write.csv(spx, file = "data/SPX.csv", row.names = FALSE)

}

Here are our first two functions. merge.all(x,y) takes two sets of timeseries data and combines them by date. It’s essentially an easy wrapper for the internal merge function, set up and locked the way we want it. spx.data() is a function to download and create a CSV out of the market’s daily prices between our start date to today (or Friday if it’s a weekend).

valuation.data <- function() {

sector.indices <<- c("S5INFT Index", "S5FINL Index", "S5ENRS Index",

"S5HLTH Index", "S5CONS Index", "S5COND Index",

"S5INDU Index", "S5UTIL Index", "S5TELS Index",

"S5MATR Index")

sector.names <<- sub(" Index", "", sector.indices)

valuation.indices <- c("PX\_LAST","PE\_RATIO", "PX\_TO\_SALES\_RATIO", "FREE\_CASH\_FLOW\_YIELD",

"EST\_LTG\_EPS\_AGGTE","TOT\_DEBT\_TO\_TOT\_ASSET","EARN\_YLD")

for (i in sector.indices) {

tes <- bdh(i, valuation.indices, start.date = data.startDate,

options = opt,include.non.trading.days = TRUE)

tes <- na.fill(tes,0)

for (j in 1:nrow(tes)) {

if (tes[j,2] == 0) {

tes[j,] = NA

}

}

tes = na.omit(tes)

tes <- na.fill(tes,0)

write.csv(tes[,-2],paste("data/valuation/",i," Value Data.csv",sep = ''),row.names = FALSE)

}

}

Here’s our first big function, so we’re gonna break it down. valuation.data() collects our value data, such as Price to Earnings. First we list our sector indices, which are created by Bloomberg to hold all manner of information about each market sector. We then make another vector of sector names without the word Index to use as shorthand names. We assign the values that we want to collect to valuation.indices, and then enter a loop for each sector. Within this loop, we collect the data we’re seeking and fill any NAs (which can break the system) with zeroes (which won’t). Then for each row in those dataframes, if the price is zero (meaning there was no trading that day), we replace that entire row with NAs and then remove all of those rows. We refill any remaining NAs with zeroes just to be safe, then write a CSV of our data named by sector and save it into the Valuation folder within the data folder so we can load them in later.

sentiment.data <- function() {

sector.indices <- c("VGT US Equity", "VFH US Equity", "VDE US Equity",

"VHT US Equity", "VDC US Equity", "VCR US Equity",

"VIS US Equity", "VPU US Equity", "VOX US Equity",

"VAW US Equity")

etfPullData <- c("PX\_LAST","PUT\_CALL\_OPEN\_INTEREST\_RATIO","EQY\_INST\_PCT\_SH\_OUT","PX\_VOLUME")

for (i in 1:length(sector.indices)) {

sector.etfs <- bdh(sector.indices[i], etfPullData, start.date=data.startDate,

options = opt, int.as.double = TRUE,include.non.trading.days = TRUE)

sector.etfs = na.fill(sector.etfs,0)

for (j in 1:nrow(sector.etfs)) {

if (sector.etfs[j,2] == 0) {

sector.etfs[j,] = NA

}

}

sector.etfs = na.omit(sector.etfs)

write.csv(sector.etfs,paste("data/sentiment/",sector.names[i]," Sentiment Data.csv",sep = ''), row.names = FALSE)

}

}

This is our other main data function, sentiment.data(). Much like valuation.data(), this function collects our sentiment data for each sector and downloads it into a CSV saved in the sentiment subfolder. Of note here is the fact that we are not using Bloomberg’s sectors for this section. In order to have sentiment we need something which is actually traded, so we are using the Vanguard Sector ETFs instead. This is also where our prices are coming from when we do our analyses.

sector.data <- function() {

sector.indices <- c("VGT US Equity", "VFH US Equity", "VDE US Equity",

"VHT US Equity", "VDC US Equity", "VCR US Equity",

"VIS US Equity", "VPU US Equity", "VOX US Equity",

"VAW US Equity")

sector.etfs <- bdh(sector.indices, "PX\_LAST", start.date=data.startDate,

options = opt, int.as.double = TRUE)

sector.names <- sector.names

sectors <- Reduce(merge.all,sector.etfs)

colnames(sectors)[-1] <- sector.names

write.csv(sectors,"data/sectors.csv", row.names = FALSE)

#head(sectors)

}

sector.data() is a miscelaneous function we use to store our sector prices. These all come from the Vanguard sectors, but we put the prices into a separate CSV that’s stored in the main data folder. It’s simply a cleaner, higher-level copy of the prices for us to use.

clean <- function(df, dataSet) {

df <- as.data.frame(df)

colnames(df) <- "PX\_LAST"

rownames(df) <- dataSet[, 1]

return(df)

}

numerify <- function(df) {

res <- sapply(df, function(x) as.numeric(as.character(x)))

rownames(res) <- rownames(df)

return(res)

}

spx.data()

valuation.data()

sentiment.data()

sector.data()

We’ve got two new functions here, clean(df,dataSet) and numerify(df). clean(df,dataSet) takes our dataframe and converts it into a smaller dataframe with one only one column. numerify(df) goes through and makes sure all of the terms are numeric and not characters. Then we run spx.data(), valuation.data(), sentiment.data() and sector.data() to update all of our data before moving on. This usually takes about a minute because of the amount of data we’re gathering.

getSectorPrice = function() {

sectors <- read.csv("data/sectors.csv", colClasses = "character")[-1,]

# log transform

res <- list(

"S5INFT" = list(  
 "price" = numerify(clean(sectors$S5INFT, sectors)),  
 "return" = log(numerify(clean(sectors$S5INFT, sectors))/numerify(clean(sectors$S5INFT, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,`  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5FINL" = list(  
 "price" = numerify(clean(sectors$S5FINL, sectors)),  
 "return" = log(numerify(clean(sectors$S5FINL, sectors))/numerify(clean(sectors$S5FINL, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5ENRS" = list(  
 "price" = numerify(clean(sectors$S5ENRS, sectors)),  
 "return" = log(numerify(clean(sectors$S5ENRS, sectors))/numerify(clean(sectors$S5ENRS, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5HLTH" = list(  
 "price" = numerify(clean(sectors$S5HLTH, sectors)),  
 "return" = log(numerify(clean(sectors$S5HLTH, sectors))/numerify(clean(sectors$S5HLTH, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5CONS" = list(  
 "price" = numerify(clean(sectors$S5CONS, sectors)),  
 "return" = log(numerify(clean(sectors$S5CONS, sectors))/numerify(clean(sectors$S5CONS, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5COND" = list(  
 "price" = numerify(clean(sectors$S5INDU, sectors)),  
 "return" = log(numerify(clean(sectors$S5COND, sectors))/numerify(clean(sectors$S5COND, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5INDU" = list(  
 "price" = numerify(clean(sectors$S5INDU, sectors)),  
 "return" = log(numerify(clean(sectors$S5INDU, sectors))/numerify(clean(sectors$S5INDU, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5UTIL" = list(  
 "price" = numerify(clean(sectors$S5UTIL, sectors)),  
 "return" = log(numerify(clean(sectors$S5UTIL, sectors))/numerify(clean(sectors$S5UTIL, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5TELS" = list(  
 "price" = numerify(clean(sectors$S5TELS, sectors)),   
 "return" = log(numerify(clean(sectors$S5TELS, sectors))/numerify(clean(sectors$S5TELS, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
),  
"S5MATR" = list(  
 "price" = numerify(clean(sectors$S5MATR, sectors)),  
 "return" = log(numerify(clean(sectors$S5MATR, sectors))/numerify(clean(sectors$S5MATR, sectors))[1,1]),  
 "P/E" = NULL,  
 "P/S" = NULL,  
 "FCF Yield" = NULL,  
 "PEG" = NULL,  
 "EPS Growth Rate" = NULL,  
 "Debt/Asset Percentage" = NULL,  
 "Earnings Yield" = NULL,  
 "Put/Call Open Interest" = NULL,  
 "Institution Ownership" = NULL,  
 "Volume" = NULL,  
   
 "Training Set" = NULL,  
 "Testing Set" = NULL  
)`

)

return(res)

}

sectorPrices <- getSectorPrice()