# FIN580: Midterm

Chris, Utsav, Srishti

Spring 2020

---

## Objective

To forecast daily volatilities of major stock indexes after the COVID-19 outbreak and provide risk measures than can help portfolio managers to make investment decisions during this turbulent period.

---

## Data Preparation

We use the following libraries.

```
> library('fasttime')
> library('data.table')
> library('dplyr')
> library('matrixStats')
> library('bit64')
> library('ggplot2')
> library('gdata')
> library('naniar')
> library('mice')
> library('forecast')
> library('imputeTS')
> library('zoo')
> library('png')
> library('grid')
> library('HDeconometrics')
> library('tictoc')
> library('cluster')
> library('factoextra')
> library('tseries')
> library('segMGarch')
```

## Data Cleaning

Our dataset has values for daily open, close and RV for 31 indices - except IBEX and SPX (for which we have RR values).

```
> data <- fread('data/data.csv')
> summary(data)
```

We need to clean the data. Here we check for the differences between the two time formats, and see which symbols take either format.

```
> times <- substr(data$Date, 12, nchar(data$Date))
> unique_times <- unique(times)
> dates1 <- unique(substr(data[substr(Date, 12, 26) == unique_times[1]]$Date, 1, 10))
> dates2 <- unique(substr(data[substr(Date, 12, 26) == unique_times[2]]$Date, 1, 10))
> intersect(dates1, dates2) # returns character(0), so can drop times
```

```
> symbols <- unlist(unique(data[, "Symbol"]))
> for (sym in symbols) {
+   subset_dates <- data["Symbol" == sym, "Date"]
+   if (length(subset_dates) != length(unique(subset_dates))) {
+     print(sym)
+   }
+ }
> print("Done")
```

We first transform our data to show these values as columns (three for each index) such that each row corresponds to a unique day.

```
> grid <- as.data.frame(as.character(unique(data$Date)))
> names(grid) <- "Date"
> symbols <- as.character(unique(data$Symbol))
> for (i in 1:length(symbols)){
+   mergewith <- data[as.character(data$Symbol)==symbols[i],-c(2)]
+   names(mergewith) <- c("Date", paste0("open",symbols[i]),
+                         paste0("close",symbols[i]), paste0("rv",symbols[i]))
+   grid <- merge(grid, mergewith, by=c("Date"), all = TRUE )
+ }
>
> index_names <- substr(symbols, 2, nchar(symbols))
> index_countries <- c("Amsterdam", "Australia", "Belgium", "India", "Portugal",
+                      "Brazil", "USA", "France", "Italy", "UK", "Germany",
+                      "Canada", "Hong Kong", "Spain", "USA", "Korea", "Pakistan",
+                      "Mexico", "Japan", "India2", "Denmark", "Finland",
+                      "Sweden", "Norway", "UK2", "Spain2", "USA", "China",
+                      "Switzerland", "Singapore", "EU")
```

Checking how much data is missing

```
> count_col_na <- function(data) {
+   sapply(data, function(x) sum(is.na(x)))
```

```
+ }
```

```
> col <- seq(from=2, to=92, by = 3)
> sample2 <- grid[, col]
> colnames(sample2) <- index_names
> rownames(sample2) <- substr(grid$Date, 1, 10)
```

```
> index <- seq(2, nrow(grid), 500)
> date_labels <- substr(grid$Date[index], 1, 10)
> vis_miss(sample2, show_perc_col=FALSE) +
+   theme(axis.text.x = element_text(angle=90)) +
+   scale_y_continuous(breaks = index, labels=date_labels) +
+   theme(axis.text.y = element_text(angle=45)) +
+   ylab("Date")
```

```
> col <- c(1, seq(from=2, to=92, by = 3))
> vis_miss(sample[,col]) # total 6.7% missing data, most for BVLG and STI
> gg_miss_var(sample[,col])
```

Also, we check what are the sizes of continuous gaps of data. The max continuous gaps as we already know are for BVLG and STI in the begining of the data. Apart from that the gap range is at max 5.

Dates where no data is missing - useful to create a partition for imputation.

```
> ind <- which(rowSums(is.na(grid)) == 0)
> grid$Date[ind]
```

## Data Imputation

### Linear interpolation

```
> linear_imputed_data <- na_interpolation(sample, option = "linear")
```

### Spline interpolation

```
> spline_imputed_data <- na_interpolation(sample, option = "spline")
```

### Constant interpolation

```
> constant_imputed_data <- na.locf(sample, na.rm=FALSE)
```

Visualizing the imputations for:

1) AORD (not much missing data)

```
> plotNA.imputations(sample$open.AORD[1400:1800],
+                    linear_imputed_data$open.AORD[1400:1800],
+                    ylab='', xlab='Date', main='', xaxt='n')
> axis(1, at=c(0, 1/5, 2/5, 3/5, 4/5, 1),
+      labels=substr(sample$Date[c(1400, 1480, 1560, 1640, 1720, 1800)], 1, 7), las=0)
```

```
> plotNA.imputations(sample$open.AORD[1400:1800],
+                    spline_imputed_data$open.AORD[1400:1800],
+                    ylab='', xlab='Date', main='', xaxt='n')
> axis(1, at=c(0, 1/5, 2/5, 3/5, 4/5, 1),
+      labels=substr(sample$Date[c(1400, 1480, 1560, 1640, 1720, 1800)], 1, 7), las=0)
```

```
> plotNA.imputations(sample$open.AORD[1400:1800],
+                    constant_imputed_data$open.AORD[1400:1800],
+                    ylab='', xlab='Date', main='', xaxt='n')
> axis(1, at=c(0, 1/5, 2/5, 3/5, 4/5, 1),
+      labels=substr(sample$Date[c(1400, 1480, 1560, 1640, 1720, 1800)], 1, 7), las=0)
```

2) STI (lot of missing data - especially at the start)

(The linear method basically fills one value for them all whereas spline does not, constant/forward fill doesn't fill in any data.)

```
> plotNA.imputations(sample$open.STI[1400:1800],
+                    linear_imputed_data$open.STI[1400:1800],
+                    ylab='', xlab='Date', main='', xaxt='n')
> axis(1, at=c(0, 1/5, 2/5, 3/5, 4/5, 1),
+      labels=substr(sample$Date[c(1400, 1480, 1560, 1640, 1720, 1800)], 1, 7), las=0)
```

```
> plotNA.imputations(sample$open.STI[1400:1800],
+                    spline_imputed_data$open.STI[1400:1800],
+                    ylab='', xlab='Date', main='', xaxt='n')
> axis(1, at=c(0, 1/5, 2/5, 3/5, 4/5, 1),
+      labels=substr(sample$Date[c(1400, 1480, 1560, 1640, 1720, 1800)], 1, 7), las=0)
```

```
> plotNA.imputations(sample$open.STI[1400:1800],
+                    constant_imputed_data$open.STI[1400:1800],
+                    ylab='', xlab='Date', main='', xaxt='n')
> axis(1, at=c(0, 1/5, 2/5, 3/5, 4/5, 1),
+      labels=substr(sample$Date[c(1400, 1480, 1560, 1640, 1720, 1800)], 1, 7), las=0)
```

## Checking for normality of $\epsilon_t$

We calculate daily standardized returns and check for their normality using Shapiro-Wilk and Jarque-Bera Normality Tests. If p-value given by the test is greater than 0.05 then we can not reject the normality assumption.

```
> new_data <- fread('data/final_data.csv')
> new_data <- new_data[, -c(1, 96)]
```

```
> col <- seq(from=2, to=92, by = 3)
> pvalsw <- c()
> pvaljb <- c()
> for (x in col){
+   sub <- new_data[,c(..x, ..x+1, ..x+2)]
+   eps <- log(sub[,2]/sub[,1])/sqrt(sub[,3])
```

```
+   eps <- unlist(eps)
+   print(paste(x, which(is.na(eps))))
+   eps <- eps[!is.na(eps)]
+   a <- shapiro.test(eps)
+   b <- jarque.bera.test(eps)
+   pvalsw <- c(pvalsw, a$p.value)
+   pvaljb <- c(pvaljb, b$p.value)
+ }
```

Plooting the barplots

```
> # Change pvalsw to pvaljb
> names(pvalsw) <- index_names
> barplot(pvalsw, ylim=c(0, 1), ylab='p-values', xlab='', las=2, cex.names=0.9)
> abline(h=0.05, lty=2, col='red')
```

Plotting both p-values together

```
> plot(pvalsw, pvaljb, ylim=c(0, 1), xlim=c(0,1),
+      xlab="p-values (Shapiro-Wilk)", ylab="p-values (Jarque-Bera)")
> text(pvalsw, pvaljb, index_names, cex=0.75)
> abline(h=0.05, lty=2, col='red')
> abline(v=0.05, lty=2, col='red')
```

Comparing the rejections from each test

```
> sw_reject <- which(pvalsw < 0.05)
> jb_reject <- which(pvaljb < 0.05)
> print(sw_reject)
> print(jb_reject)
> print(intersect(sw_reject, jb_reject))
> print(index_names[intersect(sw_reject, jb_reject)])
```

## Working on what variables to use for predicting RVs

As RVs are generally consistent over time, using lagged values as a variable would be a good idea. We plot the autocorrelation charts for each of the RV series.

```
> par(mfrow=c(4,4))
> col <- seq(from=52, to=94, by = 3)
> for (j in col){
+    acf(linear_imputed_data[,j], main=paste0(names(linear_imputed_data)[j]))
+ }
```

We would also be interested in seeing how the cross rv terms relate. As the interactions would be large (31C2 i.e. 465 terms), one of the ways that we decided to look at is average correlation over time using 500 values in each window. (Let us remove BVLG, FTMIB, STI and remove 1152 intial rows (now data starts from roughly Oct 2005) as their std would be zero when a constant value is imputed)

```
> sample <- grid[year(grid$Date)>=2019,]
> col <- seq(from=4, to=94, by = 3)
> rvs <- na.locf(sample[,col], na.rm=FALSE)
> window <- 500
> covars <- c()
> for (i in 1:(nrow(rvs)-(window-1))){
+   sub <- rvs[i:(window-1+i),]
+   mean_cor <- mean(cor(sub))
+   covars <- c(covars, mean_cor)
+ }
> plot(tail(grid$Date, length(covars)), covars, type='l', xlab="Years",
+      ylab="Average Correlation", main = "Rolling 500 days Correlation", col='blue' )
```

Performing the same over a 90 day rolling window to identify local trends

```
> col <- seq(from=4, to=94, by = 3)
> rvs <- na_interpolation(grid[1512:5303,col], option = "linear")
> rvs <- rvs[,-c(5,9,30)]
> covars <- c()
> window <- 90
> for (i in 1:(nrow(rvs)-(window-1))){
+   sub <- rvs[i:(window-1+i),]
+   mean_cor <- mean(cor(sub))
+   covars <- c(covars, mean_cor)
+ }
> plot(as.Date(grid$Date[1601:5303]), covars, type='l', xlab="Years",
+      ylab="Average Correlation", main = "Rolling 90 days Correlation")
```

Writing the correlation column (500 days window) to the final data.

```
> correl <- as.data.frame(cbind(tail(grid$Date, length(covars)), covars))
> correl$V1 <- as.Date(correl$V1)
> names(correl)[1] <- "Date"
> correl <- na.omit(correl)
> sample <- grid[((year(grid$Date)>=2019 & month(grid$Date)>= 7)) | (year(grid$Date)>=2020),]
> correl <- correl[((year(correl$Date)>=2019 & month(correl$Date)>= 7)) |
+                    (year(correl$Date)>=2020),]
> finaldata <- merge(sample, correl, by=c("Date"), all = TRUE )
> finaldata <- na.locf(na.locf(finaldata, na.rm=FALSE) , na.rm = TRUE)
> write.csv(finaldata,"final_data.csv")
```

This variable could be used for switching between models as when market is in distress, correlations tend to peak and maybe a different model from usual could be suggested.

## Addtional datasets that we can use

Maybe like NVIX or Economic Uncertainity Data - it is however only mothly and last data point available is Jan 2020 so we decided against using them for predictions in May 2020.

Another dataset that we looked at is Google trends.

```
> words <- c("coronavirus")
> countries <- c("BE","IN","PT","FR","IT","DK","FI","SE","NO","ES","NL","AU",
+                "DE","BR","US","GB","CA","CH","MX","CN","HK","KR","PK","JP","SG")
> start_date <- "2020-01-01"
> end_date <- "2020-04-16"
> toReplace <- 0.5 # numeric chosen to represent "<1" in data
>
> datalist = list()
> i <- 1
> for(w_idx in seq(length(words))) {
+   for(c_idx in seq(length(countries))) {
+     google.trends <- gtrends(words[w_idx], geo = countries[c_idx],
+                              gprop = "web", time = paste(start_date, end_date))[[1]]
+     google.trends <- dcast(google.trends, date ~ keyword + geo, value.var = "hits")
+     rownames(google.trends) <- google.trends$date
+     google.trends$date <- NULL
+     datalist[[i]] <- google.trends
+     i <- i + 1
+   }
+ }
>
> big_data <- do.call(cbind, datalist)
> big_data[big_data == "<1"] <- toReplace
```

For plotting correlation heat map using google trend series

```
> d <- cbind(rownames(big_data), data.frame(big_data, row.names=NULL))
> names(d)[1] <- "Date"
> d <- d[,-c(1)]
> t <- sapply(d, as.numeric)
>
> get_upper_tri <- function(cormat){
+   cormat[upper.tri(cormat)] <- NA
+   return(cormat)
+ }
>
> melted_cormat <- melt(get_upper_tri(cor(diff(t, lag = 1))), na.rm = TRUE)
>
> ggplot(data = melted_cormat, aes(Var2, Var1, fill = value))+
+  geom_tile(color = "white")+
+  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
+    midpoint = 0, limit = c(-1,1), space = "Lab",
+    name="Pearson\nCorrelation") +
+   theme_minimal()+
+  theme(axis.text.x = element_text(angle = 45, vjust = 1,
+     size = 12, hjust = 1))+
+  coord_fixed()
```

## Using Adaptive LASSO for variable selection

We find the average proportion of beta assigned to each covariate for every model that we run

```
> make_lagged_dataset <- function(lag) {
+   n_rows <- nrow(realized_covariances) - lag
+
+   x <- c()
+   y <- realized_covariances[(lag+1):nrow(realized_covariances), ]
+
+   for (i in 1:lag) {
+     x <- cbind(x, realized_covariances[(lag+1-i):(nrow(realized_covariances)-i), ])
+   }
+
+   return (list(x, y))
+ }
>
> lag <- 1
> lagged_dataset <- make_lagged_dataset(lag)
> x <- lagged_dataset[[1]]
> y <- lagged_dataset[[2]]
>
> window_length <- 90
> n_windows <- nrow(x) - window_length + 1
> coef_arr <- c()
> # For each realized covariance, train a LASSO model
> for (i in 1:31) {
+   coefs <- c()
+   for (j in 1:n_windows) {
+     x_data <- x[j:(j+window_length-1),]
+     y_data <- unlist(y[j:(j+window_length-1), ..i])
+
+     lasso <- ic.glmnet(x_data, y_data, crit="aic", intercept = FALSE)
+     # size of coefs: no. of covariates * no. of windows
+     # Stores the lasso coefficients for every windows
+     coefs <- cbind(coefs, coef(lasso))
+   }
+   # coef_arr stores coefs for each of the 31 dependent variables
+   coef_arr[[i]] <- coefs
+ }
> # Calculate the proportion of beta assigned to each predictor
> coef_arr_norm <- c()
> for (i in 1:31) {
+   coefs <- coef_arr[[i]]
+   # normalized_coefs is also of size no. of covariates * no. of windows
+   # except that every column has been normalized to add up to 1
+   normalized_coefs <- abs(coefs) %*% diag(1/colSums(abs(coefs)))
+   # Take average across all windows to get vector of length no. of covariates
```

```
+   # representing average proportion of beta on each covariate.
+   coef_arr_norm[[i]] <- rowMeans(normalized_coefs)
+ }
```

Then, we run k-means clustering to find the common dependencies.

```
> dd  <-  as.data.frame(matrix(unlist(coef_arr_norm), nrow=length(unlist(coef_arr_norm[1]))))
> # first row all zeros bc no intercept
> dd <- dd[-1,]
>
> #intuitive starts (India1, USA2, China, UK1)
> starts <- c(4, 15, 28, 10)
> start_mat <- c()
> for(s in starts) {
+   start_mat <- cbind(start_mat, dd[,s])
+ }
>
> colnames(dd) <- index_countries
> rownames(dd) <- paste(index_countries, "_rv", sep="")
>
> tmp <- Kmeans(t(dd), centers = t(start_mat), method = "maximum")
> sort(tmp$cluster)
```

Finding the optimal number of clusters using elbow plots

```
> get_inter_cluster_dists <- function(centers) {
+   toRet <- 0
+   for(i in 1:nrow(centers)) {
+     for(j in i:nrow(centers)) {
+       toRet <- toRet + (dist(rbind(centers[i,], centers[j,]))^2)
+     }
+   }
+   return(mean(toRet) / length(toRet))
+ }
>
> elbow_scores <- c()
> for(i in 1:10) {
+   tmp <- Kmeans(t(dd), centers = i, method = "maximum", iter.max = 1000)
+   inter_cluster_dists <- get_inter_cluster_dists(tmp$centers)
+   elbow_scores <- c(elbow_scores, sum(tmp$withinss) / inter_cluster_dists )
+ }
> plot(1:10, elbow_scores, type='l', col='blue',
+      xlab = 'Number of centers', ylab = 'Score',
+      main = 'Score with increased # centers in kmeans')
>
```

We get the following clusters:

```
> group1 <- c(3, 4, 5, 8, 9, 20, 21, 22, 23, 24, 26)
> group2 <- c(1, 2, 6, 7, 11, 12, 15, 25, 27, 29, 31)
```

```
> group3 <- c(18, 28)
> group4 <- c(10, 13, 14, 16, 17, 19, 30)
```

Now, we try to find the best dataset for each cluster. We check for different lags, sqquared terms, cross terms, and Google Trends data.

```
> realized_covariances <- fread('data/final_data.csv')
> dates <- realized_covariances[, c("Date")]
> realized_covariances <- realized_covariances[, -c("V1", "Date", "covars")]
> realized_covariances <- realized_covariances[, c(seq(3, 93, 3))]
```

Running Adaptive LASSO to find the relevant variables and MSEs for each new dataset.

```
> # x: x from make_lagged_dataset
> # indices: indices for which you want all combinations of cross terms
> # itself: whether to multiply a vector by itself; default is to not include these
> add_cross_terms <- function(x, indices, itself = FALSE) {
+   toRet <- x
+   combos <- combinations(length(indices), 2, indices)
+   for(r_idx in seq(1, dim(combos)[1])) {
+     c_idx1 <- combos[r_idx,][1]
+     c_idx2 <- combos[r_idx,][2]
+     toAdd <- x[,..c_idx1] * x[,..c_idx2]
+     toRet <- cbind(toRet, toAdd)
+   }
+   if(itself) {
+     for(c_idx in indices) {
+       toAdd <- x[,..c_idx] * x[,..c_idx]
+       toRet <- cbind(toRet, toAdd)
+     }
+   }
+   return(toRet)
+ }
>
> get_mses <- function(group_idx, lags_to_try = c(1), sq = FALSE, cross = FALSE,
+                      cross_itself = FALSE, gtrends = NULL, debug = FALSE) {
+
+   mse_mat <- matrix(0, nrow = length(lags_to_try), ncol = length(group_idx))
+   mse_rw_mat <- matrix(0, nrow = length(lags_to_try), ncol = length(group_idx))
+
+   for(l_idx in 1:length(lags_to_try)) {
+     lag <- lags_to_try[l_idx]
+     lagged_dataset <- make_lagged_dataset(lag)
+     x <- lagged_dataset[[1]]
+     if(sq) { x <- cbind(x, x*x)}
+     if(cross) { x <- add_cross_terms(x, group_idx, itself = cross_itself) }
+     if(!is.null(gtrends)) {
+       gtrends <- gtrends[1:(dim(gtrends)[1] - lag),]
+       x <- cbind(x, gtrends)
```

```
+     }
+     x <- data.matrix(x)
+     y <- lagged_dataset[[2]]
+
+     window_length <- 90
+     n_windows <- nrow(x) - window_length
+     coef_arr <- c()
+     mse_grp <- 0
+     mse_rw <- 0
+
+     tmp_cnt <- 1
+
+     for (i in group_idx) {
+       coefs <- c()
+       err <- c()
+       err_rw <- c()
+
+       for (j in 1:n_windows) {
+         x_data <- x[j:(j+window_length-1),]
+         y_data <- unlist(y[j:(j+window_length-1), ..i])
+
+         lasso <- ic.glmnet(x_data, y_data, crit="bic", intercept = FALSE, maxit = 1e+07)
+         first.step.coef <- coef(lasso)[-1]
+         pf <- (abs(first.step.coef)+1/sqrt(abs(dim(x_data)[1])))^(-tau)
+         adalasso <- ic.glmnet(x_data,y_data,crit="bic",penalty.factor=pf, intercept = FALSE)
+         rv_pred <- predict(adalasso, x[(j+window_length),])[1][1]
+         rv_actual_val <-unlist(y[(j+window_length),..i])
+
+         # check error only when the actual value is present
+         if (is.na(rv_actual_val) == FALSE){
+           err <- c(err, (rv_pred - rv_actual_val)^2)
+           err_rw <- c(err_rw, (unlist(y[(j+window_length - 1),..i]) - rv_actual_val)^2)
+         }
+
+         # size of coefs: no. of covariates * no. of windows
+         # Stores the lasso coefficients for every windows
+         coefs <- cbind(coefs, coef(adalasso))
+       }
+       # coef_arr stores coefs for each of the 31 dependent variables
+       coef_arr[[tmp_cnt]] <- coefs
+       tmp_cnt <- tmp_cnt + 1
+       mse_grp <- c(mse_grp, mean(err))
+       mse_rw <- c(mse_rw, mean(err_rw))
+     }
+
+     mse_mat[l_idx,] <- mse_grp[2:length(mse_grp)]
+     mse_rw_mat[l_idx,] <- mse_rw[2:length(mse_rw)]
```

```
+   }
+   if(debug) {
+     toRet <- list()
+     toRet[[1]] <- mse_mat
+     toRet[[2]] <- mse_rw_mat
+     return(toRet)
+   }
+   return(mse_mat)
+ }
>
> countries <- c("US", "IT", "CA", "IN", "CN",
+                "MX", "SG", "CH", "ES", "GB", "SE", "DK",
+                "JP", "PK", "KR", "HK", "DE", "AU", "BE",
+                "NL", "PT", "BR", "FR", "FI")
> # US, Italy, Canada, India, China,
> # Mexico, Singapore, Switzerland, Spain, United Kingdom, Stockholm, Copenhagen,
> # Japan, Pakistan, South Korea, Hong Kong, Germany, Australia, Belgium,
> # Netherlands, Portugal, Brazil, France, Finland
>
> group1_idx <- c(3, 4, 5, 8, 9, 20, 21, 22, 23, 24, 26)
> group3_idx <- c(18, 28)
> group2_idx <- c(1, 2, 6, 7, 11, 12, 15, 25, 27, 29, 31)
> group4_idx <- c(10, 13, 14, 16, 17, 19, 30)
> lags_to_try <- c(1, 2, 3, 4, 5)
>
> words <- c("coronavirus", "COVID-19")
> start <- as.Date(all_dates[1])
> end <- as.Date(Sys.Date())
>
> countries1 <- c("BE", "IN", "PT", "FR", "IT", "FI", "DK", "ES")
> gtrends1 <- get_word_data(words, countries1, start, end)
> gtrends1 <- gtrends1[which(rownames(gtrends1) %in% all_dates),]
> gtrends1[is.na(gtrends1)] <- 0
>
> countries2 <- c("AU", "BR", "US", "DE", "CA", "GB", "CH")
> gtrends2 <- get_word_data(words, countries2, start, end)
> gtrends2 <- gtrends2[which(rownames(gtrends2) %in% all_dates),]
> gtrends2[is.na(gtrends2)] <- 0
>
> countries3 <- c("MX", "CN")
> gtrends3 <- get_word_data(words, countries3, start, end)
> gtrends3 <- gtrends3[which(rownames(gtrends3) %in% all_dates),]
> gtrends3[is.na(gtrends3)] <- 0
>
> countries4 <- c("GB", "HK", "ES", "KR", "PK", "JP", "SG")
> gtrends4 <- get_word_data(words, countries4, start, end)
> gtrends4 <- gtrends4[which(rownames(gtrends4) %in% all_dates),]
```

```
> gtrends4[is.na(gtrends4)] <- 0
>
>
> labels <- c('Lag 1', 'Lag 2', 'Lag 3', 'Lag 4', 'Lag 5',
+             'Squared', 'Cross', 'Google trends')
>
> grp1_res_lag <- get_mses(group1_idx, lags_to_try)
> grp1_cluster_mses_lag <- rowMeans(grp1_res_lag)
> grp1_res_sq <- get_mses(group1_idx, sq = TRUE)
> grp1_cluster_mses_sq <- mean(grp1_res_sq)
> grp1_res_cross <- get_mses(group1_idx, cross = TRUE, cross_itself = TRUE)
> grp1_cluster_mses_cross <- mean(grp1_res_cross)
> grp1_res_gtrends <- get_mses(group1_idx, gtrends = gtrends1)
> grp1_cluster_mses_gtrends <- mean(grp1_res_gtrends)
>
> grp2_res_lag <- get_mses(group2_idx, lags_to_try)
> grp2_cluster_mses_lag <- rowMeans(grp2_res_lag)
> grp2_res_sq <- get_mses(group2_idx, sq = TRUE)
> grp2_cluster_mses_sq <- mean(grp2_res_sq)
> grp2_res_cross <- get_mses(group2_idx, cross = TRUE, cross_itself = TRUE)
> grp2_cluster_mses_cross <- mean(grp2_res_cross)
> grp2_res_gtrends <- get_mses(group2_idx, gtrends = gtrends2)
> grp2_cluster_mses_gtrends <- mean(grp2_res_gtrends)
>
> grp3_res_lag <- get_mses(group3_idx, lags_to_try = lags_to_try)
> grp3_cluster_mses_lag <- rowMeans(grp3_res_lag)
> grp3_res_sq <- get_mses(group3_idx, sq = TRUE)
> grp3_cluster_mses_sq <- mean(grp3_res_sq)
> grp3_res_cross <- get_mses(group3_idx, cross = TRUE, cross_itself = TRUE)
> grp3_cluster_mses_cross <- mean(grp3_res_cross)
> grp3_res_gtrends <- get_mses(group3_idx, gtrends = gtrends3)
> grp3_cluster_mses_gtrends <- mean(grp3_res_gtrends)
>
> grp4_res_lag <- get_mses(group4_idx, lags_to_try)
> grp4_cluster_mses_lag <- rowMeans(grp4_res_lag)
> grp4_res_sq <- get_mses(group4_idx, sq = TRUE)
> grp4_cluster_mses_sq <- mean(grp4_res_sq)
> grp4_res_cross <- get_mses(group4_idx, cross = TRUE, cross_itself = TRUE)
> grp4_cluster_mses_cross <- mean(grp4_res_cross)
> grp4_res_gtrends <- get_mses(group4_idx, gtrends = gtrends4)
> grp4_cluster_mses_gtrends <- mean(grp4_res_gtrends)
>
> grp1_all_mses <- c(grp1_cluster_mses_lag, grp1_cluster_mses_sq,
+                    grp1_cluster_mses_cross, grp1_cluster_mses_gtrends)
> grp2_all_mses <- c(grp2_cluster_mses_lag, grp2_cluster_mses_sq,
+                    grp2_cluster_mses_cross, grp2_cluster_mses_gtrends)
> grp3_all_mses <- c(grp3_cluster_mses_lag, grp3_cluster_mses_sq,
```

```
+                    grp3_cluster_mses_cross, grp3_cluster_mses_gtrends)
> grp4_all_mses <- c(grp4_cluster_mses_lag, grp4_cluster_mses_sq,
+                    grp4_cluster_mses_cross, grp4_cluster_mses_gtrends)
> res <- cbind(grp1_all_mses, grp2_all_mses, grp3_all_mses, grp4_all_mses)
>
> rownames(res) <- labels
```

Graph the change in number of relevant variables over time.

```
> n_relevant_vars <- fread("data/prop_sig_coef_per_window.csv")
> plot(unlist(n_relevant_vars[, 1]), type='s', col='orange', ylim=c(0, 1),
+      xlab='', ylab='Percentage of Relevant Variables', xaxt='n')
> axis(1, at=c(seq(1, 120, 20), 119), labels=substr(y_date[c(seq(1, 120, 20), 119)], 1, 7),
+      las=1, cex.axis=0.95)
> lines(unlist(n_relevant_vars[, 2]), type='s', col='red')
> lines(unlist(n_relevant_vars[, 3]), type='s', col='green')
> lines(unlist(n_relevant_vars[, 4]), type='s', col='blueviolet')
> lines(rowMeans(n_relevant_vars), type='s', col='black')
> legend("top", c(index_names[1], index_names[2], index_names[3], index_names[4], "Average"),
+        col=c('orange', 'red', 'green', 'blueviolet', 'black'),
+        lwd=3, cex=0.7, horiz=TRUE)
```

After running these tests, we get that the best datasets to consider are: Group1, Group2, Group4: Lag 1 all, and group indices up to Lag 3 Group3: Lag 1 all, and cross terms of group indices

---

# Models

## Building Baseline models

We want to start the forecasts from February 1st, 2020.

a) Random Walk: For predicting value at time t, use the value at time t-1.

```
> finaldata <- fread("data/final_data.csv")
> col <- c(2,seq(from=5, to=95, by = 3))
> finaldata <- finaldata[, ..col]
> index_names <- substr(colnames(finaldata)[-1], 4, nchar(colnames(finaldata)[-1]))
> feb_start <- which(finaldata$Date == '2020-02-01')
> sq_errs <- diff(as.matrix(finaldata[, c(-1)]))^2
> sq_errs_pre <- sq_errs[93:(feb_start-2),]
> sq_errs_post <- sq_errs[(feb_start-1):nrow(sq_errs),]
> mse_pre <- colMeans(sq_errs_pre)
> mse_post <- colMeans(sq_errs_post)
> mse_overall <- colMeans(sq_errs[93:nrow(sq_errs),])
> mse_csv <- cbind(index_names, mse_pre, mse_post, mse_overall)
```

b) Heterogeneous Autoregressive (HAR) model: We can use 200 days (to account for

recency) rolling window for our predictions?

```
> col <- c(1,seq(from=4, to=94, by = 3))
> traindata <- finaldata[,col]
> mean5 <- as.data.frame(rollmean(zoo(traindata[, -c(1)]), 5))
> mean22 <- as.data.frame(rollmean(zoo(traindata[, -c(1)]), 22))
> testdata <- finaldata[year(finaldata$Date)==2020,col]
> rv_actual <- testdata[month(testdata$Date)>=2, ]
> n <- nrow(rv_actual)
> # for 90 day rolling windows
> window <- 90
> rv <- tail(traindata, (window+n+1))
> rv5 <- tail(mean5, (window+n+1))
> rv22 <- tail(mean22, (window+n+1))
> #par(mfrow=c(3,3))
> mse_har <- c()
> for (s in 2:32){
+   err <- c()
+   for (x in 1:n){
+     tab <- as.data.frame(cbind(rv[x:(window - 1 +x),s], rv5[x:(window - 1+x),(s-1)],
+                                rv22[x:(window - 1 +x),(s-1)], rv[(x+1):(window+x),s]))
+     linearMod <- lm(V4 ~ ., data=tab)
+     x_new <- as.data.frame(cbind(rv[(window+x),s], rv5[(window+x),(s-1)],
+                                  rv22[(window+x),(s-1)]))
+     rv_pred <- predict(linearMod, x_new)
+     rv_actual_val <- rv_actual[x,s]
+     err <- c(err, (rv_pred - rv_actual_val)^2)
+   }
+   #plot(err)
+   mse_har <- c(mse_har, mean(err))
+ }
> plot(mse_har, main="Mse for each index with HAR Predictions")
```

c) LASSO model: We can use 90 days (to account for recency) rolling window for our predictions.

```
> col <- c(1,seq(from=4, to=94, by = 3)) # add 95 if want to use covars
> traindata <- finaldata[,col]
>
> testdata <- finaldata[year(finaldata$Date)==2020,col]
> rv_actual <- testdata[month(testdata$Date)>=2, ]
> n <- nrow(rv_actual)
> lag <- 5
>
> test <- traindata
> for (q in 2:lag){
+   shifted <- test
+   shifted$Date <- shift(shifted$Date , (-q +1))
+   traindata <- merge(traindata, shifted, by= c("Date"))
```

```
+ }
>
> window <- 90
> traindata <- tail(traindata, window+n+1)
>
> #par(mfrow=c(3,3))
> mse_lasso <- c()
> for (j in 2:32){
+   err <- c()
+   for (i in 1:n){
+     y <- traindata[(i+1):(window +i),j]
+     y <- as.numeric(unlist(y))
+     x <- traindata[i:(window - 1+i),-c(1)]
+     LASSO <-  HDeconometrics::ic.glmnet(x,y,crit = "bic", alpha = 0)
+     x_new <- traindata[(window +i),-c(1)]
+     rv_pred <- predict(LASSO, x_new)
+     rv_actual_val <- rv_actual[i,j]
+     err <- c(err, (rv_pred - rv_actual_val)^2)
+   }
+   #plot(err)
+   mse_lasso <- c(mse_lasso, mean(err))
+ }
```

LASSO definitely seems better than Ridge. Still RW is better than LASSO.

## Simple Moving Average and Hybrid Model

```
> col <- c(1,seq(from=4, to=94, by = 3))
> traindata <- finaldata[,col]
> mean5 <- as.data.frame(rollmean(zoo(traindata[, -c(1)]), 5))
> mean22 <- as.data.frame(rollmean(zoo(traindata[, -c(1)]), 22))
>
> n <- nrow(mean22)
> traindata <- tail(traindata, n)
>
> rv <- tail(traindata, n)
> rv5 <- tail(mean5, n)
> rv22 <- tail(mean22, n)
>
> err_rw <- rep(list(c()), 31)
> err_sma <- rep(list(c()), 31)
> pred_sma <- rep(list(c()), 31)
> pred_hyb <- rep(list(c()), 31)
> pred_rw <- rep(list(c()), 31)
>
> mse_avg <- c()
> mse_rv  <- c()
```

```
> mse_hyb <- c()
> for (s in 2:32){
+   err <- c()
+   err_rv <- c()
+   err_hyb <- c()
+   smapred <- c()
+   hybpred <- c()
+   rwpred <- c()
+   for (x in 1:(n-1)){
+     pred <- (0.1*rv[x,s] + 0.8*rv5[x,(s-1)] + 0.1*rv22[x,(s-1)])
+     actual <- rv[(x+1),s]
+     smapred <- c(smapred, pred)
+     err <- c(err, (pred - actual)^2)
+     err_rv <- c(err_rv, (rv[x,s] - actual)^2)
+     rwpred <- c(rwpred, rv[x,s])
+     if (x <= 150){
+       err_hyb <- c(err_hyb, (pred - actual)^2)
+       hybpred <- c(hybpred, pred)
+     }
+     else{
+       err_hyb <- c(err_hyb, (rv[x,s] - actual)^2)
+       hybpred <- c(hybpred, rv[x,s])
+     }
+   }
+   pred_rw[[(s-1)]] <- c(pred_rw[[(s-1)]], rwpred)
+   pred_sma[[(s-1)]] <- c(pred_sma[[(s-1)]], smapred)
+   pred_hyb[[(s-1)]] <- c(pred_hyb[[(s-1)]], hybpred)
+   err_sma[[(s-1)]] <- c(err_sma[[(s-1)]], err)
+   err_rw[[(s-1)]] <- c(err_rw[[(s-1)]], err_rv)
+   mse_avg <- c(mse_avg, mean(tail(err, 58)))
+   mse_rv <- c(mse_rv, mean(tail(err_rv, 58)))
+   mse_hyb <- c(mse_hyb, mean(tail(err_hyb, 58)))
+ }
> plot(mse_avg)
> points(mse_rv, col='red')
>
> mses <- rbind(mse_avg, mse_rv)
> barplot(mses, beside=TRUE, xaxt = "n",xlab='',
+         main="MSE for each index with SMA and RW Predcitions", ylab= 'MSE')
> axis(1, at= seq(2,94,3), labels = index_names, las = 2)
```

Checking for better model (Hybrid)

```
> count <- c()
> for (i in 1:31){
+   count <- c(count, sum(head(err_sma[[i]], 150) < head(err_rw[[i]], 150)))
+ }
```

```
> count <- c()
> for (i in 1:31){
+   count <- c(count, sum(tail(err_sma[[i]], 41) > tail(err_rw[[i]], 41)))
+ }
```

### Modified HAR (varying time and log scale)

```
> get_mses_har <- function(group_idx, har_days = c(5, 22)) {
+   lag <- 1
+   lagged_dataset <- make_lagged_dataset(lag)
+   x <- lagged_dataset[[1]]
+   idx_names <- colnames(x)
+   for(har_day in har_days) {
+     toAdd <- as.data.frame(rollmean(zoo(x[, 1:31]), har_day))
+     colnames(toAdd) <- paste0(idx_names, "_rm", har_day)
+     pad <- as.data.frame(matrix(NA, nrow = (har_day - 1), ncol = ncol(toAdd)))
+     colnames(pad) <- paste0(idx_names, "_rm", har_day)
+     toAdd <- rbind(pad, toAdd)
+     x <- cbind(x, toAdd)
+   }
+   x <- x[(max(har_days):nrow(x)),]
+   x <- data.matrix(x)
+   y <- lagged_dataset[[2]]
+   y <- tail(y, nrow(x))
+
+   window_length <- 90
+   n_windows <- nrow(x) - window_length
+   coef_arr <- c()
+   mse_grp <- 0
+
+   tmp_cnt <- 1
+
+   for (i in group_idx) {
+     coefs <- c()
+     err <- c()
+
+     for (j in 1:n_windows) {
+       x_data <- x[j:(j+window_length),]
+       rel_cols <- seq(0, length(har_days)) * 31 + i
+       x_data <- data.frame(x_data) %>% select(rel_cols)
+       y_data <- unlist(y[j:(j+window_length-1), ..i])
+
+       train_x <- x_data[1:(nrow(x_data) - 1),]
+       test_x <- x_data[nrow(x_data),]
+
+       linear_mod <- lm(y_data ~ ., data = data.frame(train_x))
+       rv_pred <- sum(linear_mod$coefficients * c(1, as.numeric(test_x)), na.rm = TRUE)
```

```
+
+       rv_actual_val <-unlist(y[(j+window_length),..i])
+
+       # check error only when the actual value is present
+       if (is.na(rv_actual_val) == FALSE){
+         err <- c(err, (rv_pred - rv_actual_val)^2)
+       }
+
+       # size of coefs: no. of covariates * no. of windows
+       # Stores the lasso coefficients for every windows
+       coefs <- cbind(coefs, linear_mod$coefficients)
+     }
+     # coef_arr stores coefs for each of the 31 dependent variables
+     coef_arr[[tmp_cnt]] <- coefs
+     tmp_cnt <- tmp_cnt + 1
+     mse_grp <- cbind(mse_grp, err)
+   }
+   return(mse_grp)[, 2:(ncol(mse_grp))]
+ }
>
> get_mses_har_ln <- function(group_idx, har_days = c(5, 22)) {
+   lag <- 1
+   lagged_dataset <- make_lagged_dataset(lag)
+   x <- lagged_dataset[[1]]
+   x <- data.frame(x)
+   x[x == 0] <- 1e-15
+   x <- log(x)
+   idx_names <- colnames(x)
+   for(har_day in har_days) {
+     toAdd <- as.data.frame(rollmean(zoo(x[, 1:31]), har_day))
+     colnames(toAdd) <- paste0(idx_names, "_rm", har_day)
+     pad <- as.data.frame(matrix(NA, nrow = (har_day - 1), ncol = ncol(toAdd)))
+     colnames(pad) <- paste0(idx_names, "_rm", har_day)
+     toAdd <- rbind(pad, toAdd)
+     x <- cbind(x, toAdd)
+   }
+   x <- x[(max(har_days):nrow(x)),]
+   y <- lagged_dataset[[2]]
+   y <- tail(y, nrow(x))
+
+   window_length <- 90
+   n_windows <- nrow(x) - window_length
+   coef_arr <- c()
+   mse_grp <- 0
+
+   tmp_cnt <- 1
+
```

```
+   for (i in group_idx) {
+     coefs <- c()
+     err <- c()
+
+     for (j in 1:n_windows) {
+       x_data <- x[j:(j+window_length),]
+       rel_cols <- seq(0, length(har_days)) * 31 + i
+       x_data <- data.frame(x_data) %>% select(rel_cols)
+       y_data <- unlist(y[j:(j+window_length-1), ..i])
+       y_data[y_data == 0] <- 1e-15
+       y_data <- log(y_data)
+
+       train_x <- x_data[1:(nrow(x_data) - 1),]
+       test_x <- x_data[nrow(x_data),]
+
+       linear_mod <- lm(y_data ~ ., data = data.frame(train_x))
+       rv_pred <- exp(sum(linear_mod$coefficients * c(1, as.numeric(test_x)), na.rm = TRUE))
+
+       rv_actual_val <-unlist(y[(j+window_length),..i])
+
+       # check error only when the actual value is present
+       if (is.na(rv_actual_val) == FALSE){
+         err <- c(err, (rv_pred - rv_actual_val)^2)
+       }
+
+       # size of coefs: no. of covariates * no. of windows
+       # Stores the lasso coefficients for every windows
+       coefs <- cbind(coefs, linear_mod$coefficients)
+     }
+     # coef_arr stores coefs for each of the 31 dependent variables
+     coef_arr[[tmp_cnt]] <- coefs
+     tmp_cnt <- tmp_cnt + 1
+     mse_grp <- cbind(mse_grp, err)
+   }
+   return(mse_grp)[, 2:(ncol(mse_grp))]
+ }
>
> ################################################################################
>
> split_mses <- function(mses, max_har_days, start_idx = 2) {
+   split_idx <- which(all_dates == '2020-02-01') - 90 - max_har_days
+   post_feb <- mses[split_idx:(nrow(mses)),]
+   toRet <- cbind(colMeans(post_feb), colMeans(mses))
+   return(toRet[start_idx:nrow(toRet),])
+ }
>
> ################################################################################
```

```
>
> h_days_1 <- c(5, 22)
> grp1_res_har_5_22 <- get_mses_har(group_idx = group1_idx, har_days = h_days_1)
> grp1_res_har_ln_5_22 <- get_mses_har_ln(group_idx = group1_idx, har_days = h_days_1)
> grp1_5_22 <- cbind(split_mses(grp1_res_har_5_22, 22), split_mses(grp1_res_har_ln_5_22, 22))
> grp2_res_har_5_22 <- get_mses_har(group_idx = group2_idx, har_days = h_days_1)
> grp2_res_har_ln_5_22 <- get_mses_har_ln(group_idx = group2_idx, har_days = h_days_1)
> grp2_5_22 <- cbind(split_mses(grp2_res_har_5_22, 22), split_mses(grp2_res_har_ln_5_22, 22))
> grp3_res_har_5_22 <- get_mses_har(group_idx = group3_idx, har_days = h_days_1)
> grp3_res_har_ln_5_22 <- get_mses_har_ln(group_idx = group3_idx, har_days = h_days_1)
> grp3_5_22 <- cbind(split_mses(grp3_res_har_5_22, 22), split_mses(grp3_res_har_ln_5_22, 22))
> grp4_res_har_5_22 <- get_mses_har(group_idx = group4_idx, har_days = h_days_1)
> grp4_res_har_ln_5_22 <- get_mses_har_ln(group_idx = group4_idx, har_days = h_days_1)
> grp4_5_22 <- cbind(split_mses(grp4_res_har_5_22, 22), split_mses(grp4_res_har_ln_5_22, 22))
>
> h_days_2 <- c(3, 5)
> grp1_res_har_3_5 <- get_mses_har(group_idx = group1_idx, har_days = h_days_2)
> grp1_res_har_ln_3_5 <- get_mses_har_ln(group_idx = group1_idx, har_days = h_days_2)
> grp1_3_5 <- cbind(split_mses(grp1_res_har_3_5, 5), split_mses(grp1_res_har_ln_3_5, 5))
> grp2_res_har_3_5 <- get_mses_har(group_idx = group2_idx, har_days = h_days_2)
> grp2_res_har_ln_3_5 <- get_mses_har_ln(group_idx = group2_idx, har_days = h_days_2)
> grp2_3_5 <- cbind(split_mses(grp2_res_har_3_5, 5), split_mses(grp2_res_har_ln_3_5, 5))
> grp3_res_har_3_5 <- get_mses_har(group_idx = group3_idx, har_days = h_days_2)
> grp3_res_har_ln_3_5 <- get_mses_har_ln(group_idx = group3_idx, har_days = h_days_2)
> grp3_3_5 <- cbind(split_mses(grp3_res_har_3_5, 5), split_mses(grp3_res_har_ln_3_5, 5))
> grp4_res_har_3_5 <- get_mses_har(group_idx = group4_idx, har_days = h_days_2)
> grp4_res_har_ln_3_5 <- get_mses_har_ln(group_idx = group4_idx, har_days = h_days_2)
> grp4_3_5 <- cbind(split_mses(grp4_res_har_3_5, 5), split_mses(grp4_res_har_ln_3_5, 5))
>
> h_days_3 <- c(5, 10, 22)
> grp1_res_har_5_10_22 <- get_mses_har(group_idx = group1_idx, har_days = h_days_3)
> grp1_res_har_ln_5_10_22 <- get_mses_har_ln(group_idx = group1_idx, har_days = h_days_3)
> grp1_5_10_22 <- cbind(split_mses(grp1_res_har_5_10_22, 22),
+                       split_mses(grp1_res_har_ln_5_10_22, 22))
> grp2_res_har_5_10_22 <- get_mses_har(group_idx = group2_idx, har_days = h_days_3)
> grp2_res_har_ln_5_10_22 <- get_mses_har_ln(group_idx = group2_idx, har_days = h_days_3)
> grp2_5_10_22 <- cbind(split_mses(grp2_res_har_5_10_22, 22),
+                       split_mses(grp2_res_har_ln_5_10_22, 22))
> grp3_res_har_5_10_22 <- get_mses_har(group_idx = group3_idx, har_days = h_days_3)
> grp3_res_har_ln_5_10_22 <- get_mses_har_ln(group_idx = group3_idx, har_days = h_days_3)
> grp3_5_10_22 <- cbind(split_mses(grp3_res_har_5_10_22, 22),
+                       split_mses(grp3_res_har_ln_5_10_22, 22))
> grp4_res_har_ln_5_10_22 <- get_mses_har_ln(group_idx = group4_idx, har_days = h_days_3)
> grp4_res_har_5_10_22 <- get_mses_har(group_idx = group4_idx, har_days = h_days_3)
> grp4_5_10_22 <- cbind(split_mses(grp4_res_har_5_10_22, 22),
+                       split_mses(grp4_res_har_ln_5_10_22, 22))
>
```

```
> h_days_4 <- c(3, 5, 10, 22)
> grp1_res_har_3_5_10_22 <- get_mses_har(group_idx = group1_idx, har_days = h_days_4)
> grp1_res_har_ln_3_5_10_22 <- get_mses_har_ln(group_idx = group1_idx, har_days = h_days_4)
> grp1_3_5_10_22 <- cbind(split_mses(grp1_res_har_3_5_10_22, 22),
+                         split_mses(grp1_res_har_ln_3_5_10_22, 22))
> grp2_res_har_3_5_10_22 <- get_mses_har(group_idx = group2_idx, har_days = h_days_4)
> grp2_res_har_ln_3_5_10_22 <- get_mses_har_ln(group_idx = group2_idx, har_days = h_days_4)
> grp2_3_5_10_22 <- cbind(split_mses(grp2_res_har_3_5_10_22, 22),
+                         split_mses(grp2_res_har_ln_3_5_10_22, 22))
> grp3_res_har_3_5_10_22 <- get_mses_har(group_idx = group3_idx, har_days = h_days_4)
> grp3_res_har_ln_3_5_10_22 <- get_mses_har_ln(group_idx = group3_idx, har_days = h_days_4)
> grp3_3_5_10_22 <- cbind(split_mses(grp3_res_har_3_5_10_22, 22),
+                         split_mses(grp3_res_har_ln_3_5_10_22, 22))
> grp4_res_har_3_5_10_22 <- get_mses_har(group_idx = group4_idx, har_days = h_days_4)
> grp4_res_har_ln_3_5_10_22 <- get_mses_har_ln(group_idx = group4_idx, har_days = h_days_4)
> grp4_3_5_10_22 <- cbind(split_mses(grp4_res_har_3_5_10_22, 22),
+                         split_mses(grp4_res_har_ln_3_5_10_22, 22))
>
> allgrp1_har <- cbind(grp1_5_22, grp1_3_5, grp1_5_10_22, grp1_3_5_10_22)
> allgrp2_har <- cbind(grp2_5_22, grp2_3_5, grp2_5_10_22, grp2_3_5_10_22)
> allgrp3_har <- cbind(grp3_5_22, grp3_3_5, grp3_5_10_22, grp3_3_5_10_22)
> allgrp4_har <- cbind(grp4_5_22, grp4_3_5, grp4_5_10_22, grp4_3_5_10_22)
```

## Modified HAR (residuals)

```
> get_mses_har_ln_resid <- function(group_idx, har_days = c(3, 5)) {
+   lag <- 1
+   lagged_dataset <- make_lagged_dataset(lag)
+   x <- lagged_dataset[[1]]
+   x <- data.frame(x)
+   x[x == 0] <- 1e-15
+   x <- log(x)
+   idx_names <- colnames(x)
+   for(har_day in har_days) {
+     toAdd <- as.data.frame(rollmean(zoo(x[, 1:31]), har_day))
+     colnames(toAdd) <- paste0(idx_names, "_rm", har_day)
+     pad <- as.data.frame(matrix(NA, nrow = (har_day - 1), ncol = ncol(toAdd)))
+     colnames(pad) <- paste0(idx_names, "_rm", har_day)
+     toAdd <- rbind(pad, toAdd)
+     x <- cbind(x, toAdd)
+   }
+   x <- x[(max(har_days):nrow(x)),]
+   y <- lagged_dataset[[2]]
+   y <- tail(y, nrow(x))
+
+   window_length <- 90
```

```
+   n_windows <- nrow(x) - window_length
+   coef_arr <- c()
+   mse_grp <- 0
+
+   tmp_cnt <- 1
+
+   for (i in group_idx) {
+     coefs <- c()
+     err <- c()
+     err_rw <- c()
+
+     for (j in 1:n_windows) {
+       x_data <- x[j:(j+window_length),]
+       rel_cols <- seq(0, length(har_days)) * 31 + i
+       x_data1 <- data.frame(x_data) %>% select(rel_cols)
+       x_data2 <- select(data.frame(x_data), -rel_cols)
+       y_data <- unlist(y[j:(j+window_length-1), ..i])
+       y_data1 <- y_data
+       y_data1[y_data1 == 0] <- 1e-15
+       y_data1 <- log(y_data1)
+
+       train_x1 <- x_data1[1:(nrow(x_data1) - 1),]
+       test_x1 <- x_data1[nrow(x_data1),]
+       linear_mod <- lm(y_data1 ~ ., data = data.frame(train_x1))
+       resid <- y_data - exp(linear_mod$fitted.values)
+       train_x2 <- x_data2[1:(nrow(x_data2) - 1),]
+       test_x2 <- x_data2[nrow(x_data2),]
+       linear_mod_resid <- lm(resid ~ ., data = data.frame(train_x2))
+
+       rv_pred <- exp(sum(linear_mod$coefficients * c(1, as.numeric(test_x1)), na.rm = TRUE))
+         + sum(linear_mod_resid$coefficients * c(1, as.numeric(test_x2)), na.rm = TRUE)
+
+       rv_actual_val <-unlist(y[(j+window_length),..i])
+
+       # check error only when the actual value is present
+       if (is.na(rv_actual_val) == FALSE){
+         err <- c(err, (rv_pred - rv_actual_val)^2)
+       }
+
+       # size of coefs: no. of covariates * no. of windows
+       # Stores the lasso coefficients for every windows
+       coefs <- cbind(coefs, linear_mod$coefficients)
+     }
+     # coef_arr stores coefs for each of the 31 dependent variables
+     coef_arr[[tmp_cnt]] <- coefs
+     tmp_cnt <- tmp_cnt + 1
+     mse_grp <- cbind(mse_grp, err)
```

```
+   }
+   return(mse_grp)[, 2:(ncol(mse_grp))]
+ }
>
> get_mses_har_ln_resid2 <- function(group_idx, har_days = c(3, 5)) {
+   lag <- 1
+   lagged_dataset <- make_lagged_dataset(lag)
+   x <- lagged_dataset[[1]]
+   x <- data.frame(x)
+
+   x[x == 0] <- 1e-15
+   x <- log(x)
+   idx_names <- colnames(x)
+   for(har_day in har_days) {
+     toAdd <- as.data.frame(rollmean(zoo(x[, 1:31]), har_day))
+     colnames(toAdd) <- paste0(idx_names, "_rm", har_day)
+     pad <- as.data.frame(matrix(NA, nrow = (har_day - 1), ncol = ncol(toAdd)))
+     colnames(pad) <- paste0(idx_names, "_rm", har_day)
+     toAdd <- rbind(pad, toAdd)
+     x <- cbind(x, toAdd)
+   }
+   x <- x[(max(har_days):nrow(x)),]
+
+   y <- lagged_dataset[[2]]
+   y <- tail(y, nrow(x))
+
+   window_length <- 90
+   n_windows <- nrow(x) - window_length
+   coef_arr <- c()
+   mse_grp <- 0
+   tmp_cnt <- 1
+
+   for (i in group_idx) {
+     coefs <- c()
+     err <- c()
+
+     for (j in 1:n_windows) {
+       x_data <- x[j:(j+window_length),]
+       rel_cols <- seq(0, length(har_days)) * 31 + i
+       x_data1 <- data.frame(x_data) %>% select(rel_cols)
+       y_data <- unlist(y[j:(j+window_length-1), ..i])
+       y_data1 <- y_data
+       y_data1[y_data1 == 0] <- 1e-15
+       y_data1 <- log(y_data1)
+
+       train_x1 <- x_data1[1:(nrow(x_data1) - 1),]
+       test_x1 <- x_data1[nrow(x_data1),]
```

```
+       linear_mod <- lm(y_data1 ~ ., data = data.frame(train_x1))
+
+       resid <- y_data - exp(linear_mod$fitted.values)
+       train_x2 <- train_x1^2
+       test_x2 <- test_x1^2
+       linear_mod_resid <- lm(resid ~ ., data = data.frame(train_x2))
+
+       rv_pred <- exp(sum(linear_mod$coefficients * c(1, as.numeric(test_x1)), na.rm = TRUE))
+         + sum(linear_mod_resid$coefficients * c(1, as.numeric(test_x2)), na.rm = TRUE)
+
+       rv_actual_val <-unlist(y[(j+window_length),..i])
+
+       # check error only when the actual value is present
+       if (is.na(rv_actual_val) == FALSE){
+         err <- c(err, (rv_pred - rv_actual_val)^2)
+       }
+
+       # size of coefs: no. of covariates * no. of windows
+       # Stores the lasso coefficients for every windows
+       coefs <- cbind(coefs, linear_mod$coefficients)
+     }
+     # coef_arr stores coefs for each of the 31 dependent variables
+     coef_arr[[tmp_cnt]] <- coefs
+     tmp_cnt <- tmp_cnt + 1
+     mse_grp <- cbind(mse_grp, err)
+   }
+   return(mse_grp)[, 2:(ncol(mse_grp))]
+ }
>
> ################################################################################
>
> grp1_res_har_ln_resid <- get_mses_har_ln_resid(group1_idx)
> grp1_res_har_ln_resid2 <- get_mses_har_ln_resid2(group1_idx)
> grp1_resid <- cbind(split_mses(grp1_res_har_ln_resid, 5),
+                     split_mses(grp1_res_har_ln_resid2, 5))
> grp2_res_har_ln_resid <- get_mses_har_ln_resid(group2_idx)
> grp2_res_har_ln_resid2 <- get_mses_har_ln_resid2(group2_idx)
> grp2_resid <- cbind(split_mses(grp2_res_har_ln_resid, 5),
+                     split_mses(grp2_res_har_ln_resid2, 5))
> grp3_res_har_ln_resid <- get_mses_har_ln_resid(group3_idx)
> grp3_res_har_ln_resid2 <- get_mses_har_ln_resid2(group3_idx)
> grp3_resid <- cbind(split_mses(grp3_res_har_ln_resid, 5),
+                     split_mses(grp3_res_har_ln_resid2, 5))
> grp4_res_har_ln_resid <- get_mses_har_ln_resid(group4_idx)
> grp4_res_har_ln_resid2 <- get_mses_har_ln_resid2(group4_idx)
> grp4_resid <- cbind(split_mses(grp4_res_har_ln_resid, 5),
+                     split_mses(grp4_res_har_ln_resid2, 5))
```

# Tree Based Methods

April 30, 2020

[1]:
```
import numpy as np
import pandas as pd
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

import time
```

We first import the constructed files of predictors and response variables from R.

[2]:
```
group1_x = pd.read_csv("data/group1_lag3_x.csv")
group1_y = pd.read_csv("data/group1_lag3_y.csv")
```

[3]:
```
group2_x = pd.read_csv("data/group2_lag3_x.csv")
group2_y = pd.read_csv("data/group2_lag3_y.csv")
```

[4]:
```
group3_x = pd.read_csv("data/group3_cross_x.csv")
group3_y = pd.read_csv("data/group3_cross_y.csv")
```

[5]:
```
group4_x = pd.read_csv("data/group4_lag3_x.csv")
group4_y = pd.read_csv("data/group4_lag3_y.csv")
```

As both tree based methods are not affected by the scale of the variables, we do not need to standardize the data for either of the two methods shown below.

### 0.0.1 Gradient Boosted Trees

We use the sklearn function `GradientBoostingRegressor` along with `Multi Output Regressor` to adapt gradient boosting trees to a multi-target regression setting. We first need to tune the hyperparameters for the model: learning rate, number of iterations (known as `n_estimators` in the sklearn model), maximum depth of each tree, and whether we perform early stopping of the training or not. If we decide to allow for early stopping, we need to decide the number of iterations that will be used to decide. The default loss function is the squared loss.

Note that because of the lack of interpretability of Gradient Boosting Trees (and Random Forests later), it is very difficult to estimate why some combination of parameters performed better than

the others during the tuning process. However, what we can strive to achieve is check the relative importance of features for the final models that we arrive upon.

[45]:
```
def train_and_predict_boosting_tree(model, x, y, window_length):
    n_windows = x.shape[0] - window_length
    predictions = pd.DataFrame(0, index=range(n_windows), columns=y.columns[1:])
    y_actual = y.iloc[window_length:, 1:].reset_index()

    for i in range(n_windows):
        x_data = x.iloc[i:(i+window_length), 1:]
        y_data = y.iloc[i:(i+window_length), 1:]

        g_boost_multi = MultiOutputRegressor(model)
        g_boost_multi.fit(x_data, y_data)

        pred_x = x.iloc[(i+window_length):(i+window_length+1), 1:]
        pred_y = g_boost_multi.predict(pred_x)

        predictions.loc[i] = pred_y[0]

    sq_err = (y_actual.iloc[:, 1:]-predictions)**2
    feb_start = y.index[y["Date"] == '2020-02-01'][0]
    sq_err_feb_start = y_actual.index[y_actual["index"] == 152][0]

    mse_pre = sq_err.iloc[0:sq_err_feb_start, :].mean(axis=0)
    mse_post = sq_err.iloc[sq_err_feb_start:, :].mean(axis=0)
    mse_overall = sq_err.mean(axis=0)

    mse_df = pd.concat([mse_pre, mse_post, mse_overall], axis=1)

    return (predictions, mse_df)
```

We will first tune `learning_rate=0.1, 0.2, 0.5` and `n_estimators=50, 100, 200` in conjuction, as there is a trade-off between the two parameters.

[38]:
```
models_gbt = [
    GradientBoostingRegressor(loss='ls', learning_rate=0.01, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=100,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=200,␣
↪max_depth=3, n_iter_no_change=None),

    GradientBoostingRegressor(loss='ls', learning_rate=0.2, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.2, n_estimators=100,␣
↪max_depth=3, n_iter_no_change=None),
```

```
    GradientBoostingRegressor(loss='ls', learning_rate=0.2, n_estimators=200,␣
↪max_depth=3, n_iter_no_change=None),

    GradientBoostingRegressor(loss='ls', learning_rate=0.5, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.5, n_estimators=100,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.5, n_estimators=200,␣
↪max_depth=3, n_iter_no_change=None)
]
```

[43]:
```
mse_store = pd.DataFrame()
for model in models_gbt:
    predictions = train_and_predict_boosting_tree(model, group4_x, group4_y, 90)
    mse_store = pd.concat([mse_store, predictions], axis=1)
    print(predictions[1].iloc[:, 2].mean())
```

Next we try `max_depth=3, 5, 10` for the best model chosen above.

[ ]:
```
models_gbt = [
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=50,␣
↪max_depth=5, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=50,␣
↪max_depth=10, n_iter_no_change=None)
]
```

[ ]:
```
mse_store = pd.DataFrame()
for model in models_gbt:
    predictions = train_and_predict_boosting_tree(model, group3_x, group3_y, 90)
    mse_store = pd.concat([mse_store, predictions], axis=1)
    print(predictions.iloc[:, 2].mean())
```

### 0.0.2 Random Forest

Now, we look at training a Random Forest over the given data. We have two hyperparamters to tune - maximum depth (`max_depth`) of each tree, and the number of trees (`n_estimators`). Note that the default criterion is to measure the quality of the split is the mean squared error, and the default setting is to take a bootstrap sample.

[66]:
```
def train_and_predict_random_forest(model, x, y, window_length):
    n_windows = x.shape[0] - window_length
    predictions = pd.DataFrame(0, index=range(n_windows), columns=y.columns[1:])
    y_actual = y.iloc[window_length:, 1:].reset_index()

    for i in range(n_windows):
```

```
        x_data = x.iloc[i:(i+window_length), 1:]
        y_data = y.iloc[i:(i+window_length), 1:]

        random_forest = model
        random_forest.fit(x_data, y_data)

        pred_x = x.iloc[(i+window_length):(i+window_length+1), 1:]
        pred_y = random_forest.predict(pred_x)

        predictions.loc[i] = pred_y[0]

    sq_err = (y_actual.iloc[:, 1:]-predictions)**2
    feb_start = y.index[y["Date"] == '2020-02-01'][0]
    sq_err_feb_start = y_actual.index[y_actual["index"] == feb_start][0]

    mse_pre = sq_err.iloc[0:sq_err_feb_start, :].mean(axis=0)
    mse_post = sq_err.iloc[sq_err_feb_start:, :].mean(axis=0)
    mse_overall = sq_err.mean(axis=0)

    mse_df = pd.concat([mse_pre, mse_post, mse_overall], axis=1)

    return predictions
```

We consider the following options for each parameter - `n_estimators=10, 100 (default), 200` and `max_depth=None (default), 3, 5, 10`.

[86]:
```
models_rf = [
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=None),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=10),

    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=None),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=10),

    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=None),
```

```
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=10),

    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=None),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=10)
           ]
```

[91]:
```
mse_store = pd.DataFrame()
for model in models_rf:
    predictions = train_and_predict_random_forest(model, group4_x, group4_y, 90)
    mse_store = pd.concat([mse_store, predictions], axis=1)
    print(predictions.iloc[:, 2].mean())
```

### 0.0.3 Feature Importances

We calculate average feature importances to see what predictors are preferred. We look at pre- and post-Feb, as well as overall, to get an idea on how the structural break affects values. We plot barplots for all three cases as well.

**Gradient Boosted Trees**

[ ]:
```
final_models_gbt = [GradientBoostingRegressor(loss='ls', learning_rate=0.2,␣
↪n_estimators=100),
                    GradientBoostingRegressor(loss='ls', learning_rate=0.2,␣
↪n_estimators=50),
                    GradientBoostingRegressor(loss='ls', learning_rate=0.1,␣
↪n_estimators=100),
                    GradientBoostingRegressor(loss='ls', learning_rate=0.1,␣
↪n_estimators=50, max_depth=5)]
x_list = [group1_x, group2_x, group3_x, group4_x]
y_list = [group1_y, group2_y, group3_y, group4_y]
```

[ ]:
```
window_length = 90
k = 0

final_model_gbt_multi = MultiOutputRegressor(final_models_gbt[k])
x = x_list[k]
```

```
y = y_list[k]
n_windows = x.shape[0] - window_length
n_predictors = y.shape[1]-1
n_covariates = x.shape[1]-1

feature_imp_gbt = [[] for j in range(n_predictors)]
for i in range(n_windows):
    x_data = x.iloc[i:(i+window_length), 1:]
    y_data = y.iloc[i:(i+window_length), 1:]

    final_model_gbt_multi.fit(x_data, y_data)
    for j in range(n_predictors):
        feature_imp_gbt[j].append(final_model_gbt_multi.estimators_[j].
 ↪feature_importances_)
```

[ ]:
```
avg_feature_imp_gbt = np.mean(np.mean(feature_imp_gbt, axis=1), axis=0)
# Normalising them to sum up to 1
avg_feature_imp_gbt = avg_feature_imp_gbt/sum(avg_feature_imp_gbt)

avg_feature_imp_gbt_pre = np.mean([np.mean(i[0:62], axis=0) for i in␣
 ↪feature_imp_gbt], axis=0)
if sum(avg_feature_imp_gbt_pre) != 0:
    avg_feature_imp_gbt_pre = avg_feature_imp_gbt_pre/
 ↪sum(avg_feature_imp_gbt_pre)

avg_feature_imp_gbt_post = np.mean([np.mean(i[62:], axis=0) for i in␣
 ↪feature_imp_gbt], axis=0)
avg_feature_imp_gbt_post = avg_feature_imp_gbt_post/
 ↪sum(avg_feature_imp_gbt_post)
```

[ ]:
```
# sort labels in the same order as avg_feature_imp_gbt for graph labels
sorted_barplot_labels_gbt = [x for _, x in sorted(zip(avg_feature_imp_gbt,␣
 ↪range(n_covariates)))]
sorted_barplot_labels_gbt_pre = [x for _, x in␣
 ↪sorted(zip(avg_feature_imp_gbt_pre, range(n_covariates)))]
sorted_barplot_labels_gbt_post = [x for _, x in␣
 ↪sorted(zip(avg_feature_imp_gbt_post, range(n_covariates)))]
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_gbt)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_gbt,␣
 ↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Gradient Boosting Trees: Overall")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_gbt_pre)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_gbt_pre,␣
 ↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Gradient Boosting Trees: Pre Feb")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_gbt_post)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_gbt_post,␣
 ↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Gradient Boosting Trees: Post Feb")
plt.show()
```

### Random Forests

[ ]:
```
final_models_rf = [RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=10, max_depth=10),
                   RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=200, max_depth=5),
                   RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=100, max_depth=3),
                   RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=100, max_depth=5)]
x_list = [group1_x, group2_x, group3_x, group4_x]
y_list = [group1_y, group2_y, group3_y, group4_y]
```

[ ]:
```
window_length = 90
k = 2

final_model_rf = final_models_rf[k]
x = x_list[k]
y = y_list[k]
n_windows = x.shape[0] - window_length
n_predictors = y.shape[1]-1
n_covariates = x.shape[1]-1

feature_imp_rf = []
for i in range(n_windows):
    x_data = x.iloc[i:(i+window_length), 1:]
    y_data = y.iloc[i:(i+window_length), 1:]

    final_model_rf.fit(x_data, y_data)
    feature_imp_rf.append(final_model_rf.feature_importances_)
```

[ ]:
```
avg_feature_imp_rf = np.mean(feature_imp_rf, axis=0)
avg_feature_imp_rf = avg_feature_imp_rf/sum(avg_feature_imp_rf)
```

```
avg_feature_imp_rf_pre = np.mean(feature_imp_rf[0:62], axis=0)
if sum(avg_feature_imp_rf_pre) != 0:
    avg_feature_imp_rf_pre = avg_feature_imp_rf_pre/sum(avg_feature_imp_rf_pre)

avg_feature_imp_rf_post = np.mean(feature_imp_rf[62:], axis=0)
avg_feature_imp_rf_post = avg_feature_imp_rf_post/sum(avg_feature_imp_rf_post)
```

[ ]:
```
sorted_barplot_labels_rf = [x for _, x in sorted(zip(avg_feature_imp_rf,␣
↪range(n_covariates)))]
sorted_barplot_labels_rf_pre = [x for _, x in␣
↪sorted(zip(avg_feature_imp_rf_pre, range(n_covariates)))]
sorted_barplot_labels_rf_post = [x for _, x in␣
↪sorted(zip(avg_feature_imp_rf_post, range(n_covariates)))]
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_rf)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_rf,␣
↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Random Forest: Overall")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_rf_pre)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_rf_pre,␣
↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Random Forest: Pre Feb")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_rf_post)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_rf_post,␣
↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Random Forest: Post Feb")
plt.show()
```

## Coverage Tests

```
> final_data <- fread("data/final_data.csv")
> dates <- final_data[, 2]
> final_data  <- final_data[, -c(1, 2, 96)]
> returns <- c()
> for (k in 0:30) {
+   opening_col <- 3*k + 1
+   closing_col <- 3*k + 2
+   opening <- final_data[, ..opening_col]
+   closing <- final_data[, ..closing_col]
+   index_returns <- log(closing/opening) # (closing-opening)/opening
+   returns <- cbind(returns, index_returns)
+ }
> colnames(returns) <- index_names
> write.csv(returns, "log_returns.csv", row.names = FALSE)
```

```
> log_returns <- fread("data/log_returns.csv")
> log_returns_feb_start <- which(log_returns$Date == '2/1/2020')
> post_feb_log_returns <- log_returns[log_returns_feb_start:nrow(log_returns), c(-1)]
> index_names <- c("AEX", "AORD", "BFX", "BSESN", "BVLG", "BVSP", "DJI", "FCHI",
+                  "FTMIB", "FTSE", "GDAXI", "GSPTSE", "HSI", "IBEX", "IXIC",
+                  "KS11", "KSE", "MXX", "N225", "NSEI", "OMXC20", "OMXHPI",
+                  "OMXSPI", "OSEAX", "RUT", "SMSI", "SPX", "SSEC", "SSMI", "STI", "STOXX50E")
```

### Kupiec Tests

```
> kupiec_test <- function(returns, preds) {
+   vars <- qnorm(0.05)*sqrt(preds)
+
+   for (i in 1:31) {
+     print(index_names[i])
+     z <- kupiec(unlist(returns[, ..i]), unlist(vars[, ..i]), 0.95, verbose=TRUE, test="PoF")
+     print(z)
+   }
+ }
```

### Traffic Light Test

```
> traffic_light_test <- function(returns, preds) {
+   vars <- qnorm(0.05)*sqrt(preds)
+
+   red <- c()
+   yellow <- c()
+   green <- c()
+
+   for (i in 1:31) {
```

```
+     res <- TL(y=unlist(returns[, ..i]), VaR=unlist(vars[, ..i]), VaR_level=0.95)
+     if (res$color == "red") {
+       red <- c(red, index_names[i])
+     }
+
+     if (res$color == "yellow") {
+       yellow <- c(yellow, index_names[i])
+     }
+
+     if (res$color == "green") {
+       green <- c(green, index_names[i])
+     }
+   }
+
+   print("Red")
+   print(red)
+   print("Yellow")
+   print(yellow)
+   print("Green")
+   print(green)
+ }
```

### Creating VaR Graphs

```
> returns <- fread('data/log_returns_closing.csv')
> # only 58
> rw_pred <- fread('data/RW_predictions.csv')
> sma_pred <- fread('data/SMA_predictions.csv')
> hyb_pred <- fread('data/HYB_predictions.csv')
> # need tail for the rest
> har_pred <- fread('data/har_preds.csv')
> modhar_pred <- fread('data/modhar_preds.csv')
> gbt_pred <- fread('data/boosted_tree_predictions.csv')
> rf_pred <- fread('data/random_forest_predictions.csv')
> preds <- list(rw_pred, sma_pred, hyb_pred, har_pred, modhar_pred, gbt_pred, rf_pred)
> tailed_preds <- lapply(preds, function(x) tail(x, 58))
> dates <- unlist(returns[, 1])
> tailed_dates <- tail(dates, 58)
```

```
> # index between 2 and 32 (column 1 is dates for all)
> # Choose from 3 5 7 10 13 14 15 18 19 20 22 23 28
> index <- 3
> date_indices <- c(1, 12, 23, 34, 45, 56)
> c_alph <- qnorm(0.05)
> plot(tail(unlist(returns[, ..index]), 58), type='l', xaxt='n', xlab='',
+      ylab="Standardized Returns", ylim=c(-0.18, 0.05), lty='dotted')
> axis(1, at=date_indices,
```

```
+      labels=c("Feb 01", "Feb 17", "March 03", "March 18", "Apr 02", "Apr 17"), cex.axis=0.8)
> colors <- c('red', 'orange', 'yellow', 'green', 'blue', 'purple', 'violet')
> for (i in 1:7) {
+   curr_preds <- tailed_preds[[i]]
+   pred_vars <- c_alph * sqrt(unlist(curr_preds[, ..index]))
+   lines(pred_vars, col=colors[i])
+ }
> legend("bottomleft", c("Ret", "RW", "SWA", "Hybrid", "HAR", "Mod. HAR", "GBT", "RF"),
+        col=c('black', colors),
+        lwd=2, cex=0.5, horiz=TRUE, x.intersp=0.6, text.width=4.14, lty=c(3, rep(1, 7)))
```