Trabalho - Econometria IV

Guilherme Luz, Guilherme Masuko, Caio Garzeri

August 2023

```r
library(lubridate)  # for handling dates
library(zoo)  # for time series
library(dynlm)  # for time series regressions
library(forecast)  # for the improved Pacf function
library(glmnet)  # for shrinkage methods
library(HDeconometrics)  # IC for glmnet

# Packages for parallel computation
library(future)
library(foreach)
library(doFuture)
library(doRNG)
```

Question 2

First of all, we must do some data wrangling.

```r
# Import the data
raw_data = read_csv("data/2021-12.csv")
# raw_data = read_csv('C:/Users/Caio Garzeri/OneDrive -
# puc-rio.br/Econometria
# IV/AssignmentEconometricsIV/data/2021-12.csv')

data0 = raw_data[-1, ] %>%
    select_if(~!any(is.na(.)))
transformation = raw_data[1, ]
```

The suggested transformations (in order to make the series stationary) are indicated according to the following numeration.

Transformation codes (from FRED):

1. no transformation
2. Δx_t
3. $\Delta^2 x_t$
4. $\log(x_t)$
5. $\Delta \log(x_t)$
6. $\Delta^2 \log(x_t)$
7. $\Delta(x_t/x_{t-1} - 1)$

For the CPI, we apply a specific transformation to turn it into an inflation series.

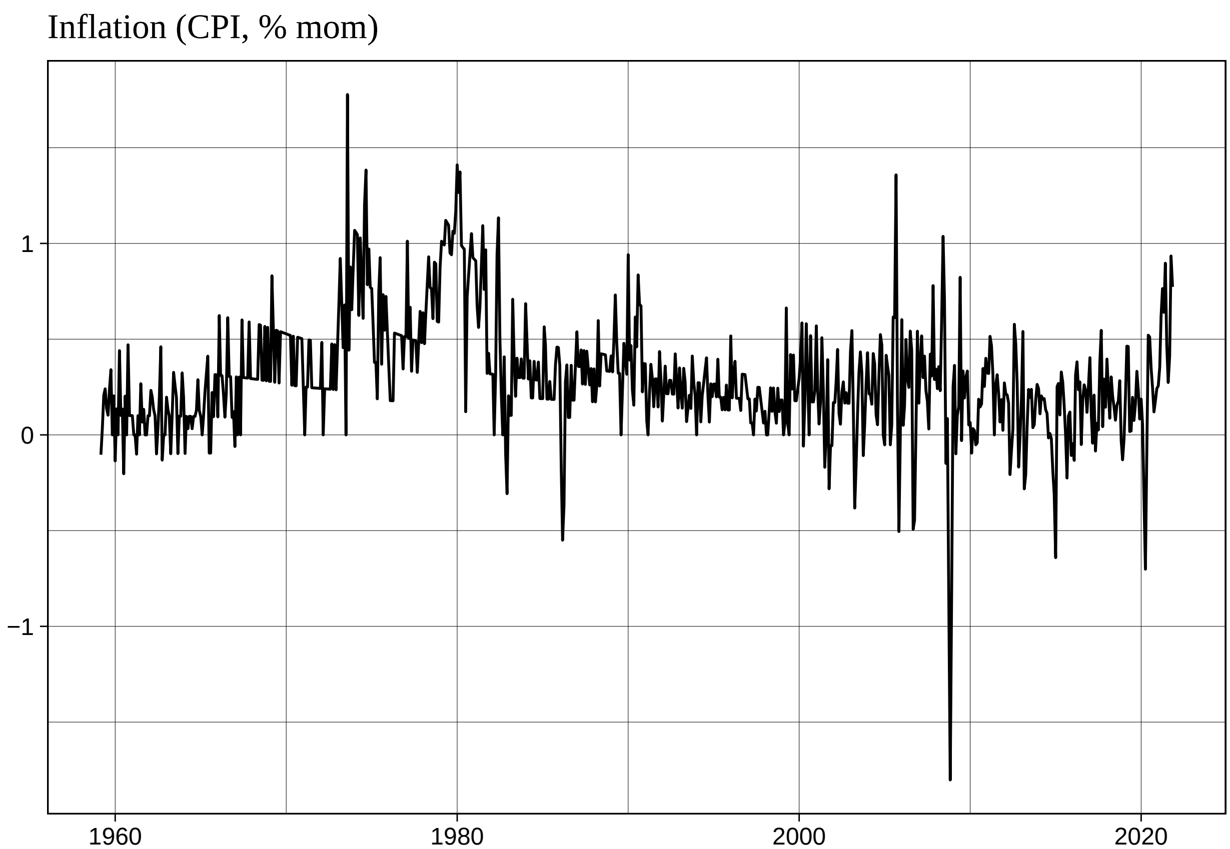
```r
# Data transformations based on the FRED transformation
# codes
```

```r
data = data0 %>%
    select(-sasdate) %>%
    rename(SP500 = "S&P 500", SPINDUST = "S&P: indust") %>%
    BVAR::fred_transform(type = "fred_md") %>%
    bind_cols(tibble(date = data0$sasdate[3:length(data0$sasdate)])) %>%
    mutate(date = as.Date(date, format = "%m/%d/%Y"))

# For the CPI, we transform into an inflation series
data = mutate(data, CPIAUCSL = 100 * (diff(data0$CPIAUCSL, differences = 1)/data0$CPIAUCSL[-1])[-1])
```

```r
# Inflation as time series
inflation = data$CPIAUCSL %>%
    ts(start = c(year(data$date[1]), month(data$date[1])), frequency = 12)
```

The resulting inflation series, which we want to forecast is shown below.

```r
# plot inflation

data %>%
    select(date, CPIAUCSL) %>%
    mutate(date = as.Date(date, format = "%m/%d/%Y")) %>%
    ggplot(aes(x = date, y = CPIAUCSL)) + geom_line() + labs(title = "Inflation (CPI, % mom)",
    x = NULL, y = NULL)
```

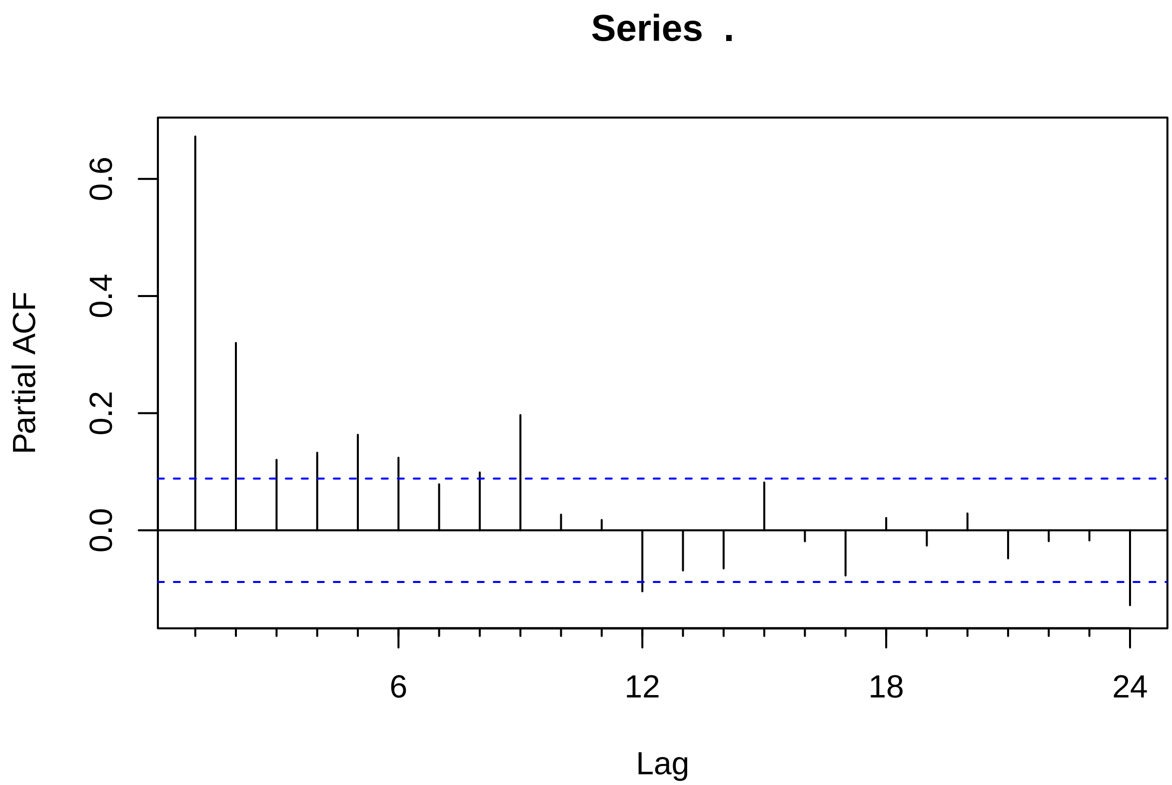


Now, we shall start the estimations.

AR

In order to get some idea of what the order of our AR(p) process is, we plot the partial autocorrelation of the inflation series for a particular window.

```
# Partial autocorrelation
inflation %>%
    window(start = start(inflation), end = start(inflation) +
        c(0, 492)) %>%
    Pacf(lag.max = 24, plot = T)
```



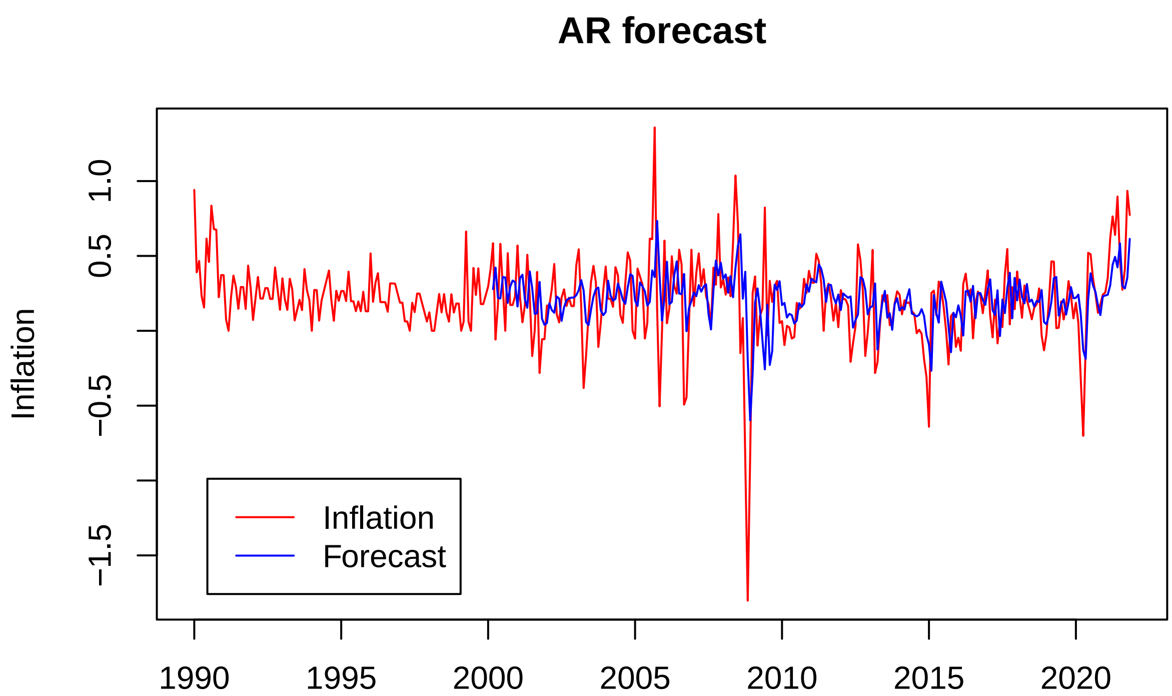
We believe that a maximum lag of 24 is more than reasonable. Then, the determine the actual order p based on the BIC.

```
# Function for calculating the BIC for AR models
BIC.ar <- function(model) {

    ssr <- sum(model$resid^2, na.rm = T)
    t <- sum(!is.na(model$resid))
    npar <- length(model$ar) + 1

    return(c(p = model$order, BIC = log(ssr/t) + npar * log(t)/t))
}
```

We proceed with a rolling window one-step-ahead forecast, in which we choose the optimal order of the AR in each window of estimation.

```
# Rolling window forecasting
rolling_window <- 492
p.max <- 24

forecast1 = list()
```

```r
popt_AR = data.frame(popt = numeric(261))

for (a in 0:(length(inflation) - rolling_window - 1)) {

    # get the window for training the model
    train = window(inflation, start = start(inflation) + c(0,
        a), end = start(inflation) + c(0, a + rolling_window -
        1))

    bic.table = c()

    for (p in 0:p.max) {
        # calculating the BIC for different orders of the
        # AR(p)
        AR = ar(train, order.max = p, method = "ols", aic = F)
        bic.line = BIC.ar(AR)
        bic.table = rbind(bic.table, bic.line)
    }
    bic.table = data.frame(bic.table)

    p.opt = bic.table$p[which.min(bic.table$BIC)]  # pick the optimal p

    popt_AR$popt[a + 1] <- p.opt

    AR = ar(train, order.max = p.opt, method = "ols", aic = F)  # run the AR model with the optimal p

    forecast1[[a + 1]] = predict(AR, n.ahead = 1)$pred  # one-step-ahead forecast
}

forecasts = forecast1 %>%
    unlist() %>%
    ts(start = start(forecast1[[1]]), frequency = frequency(forecast1[[1]]))
```



AR + PC

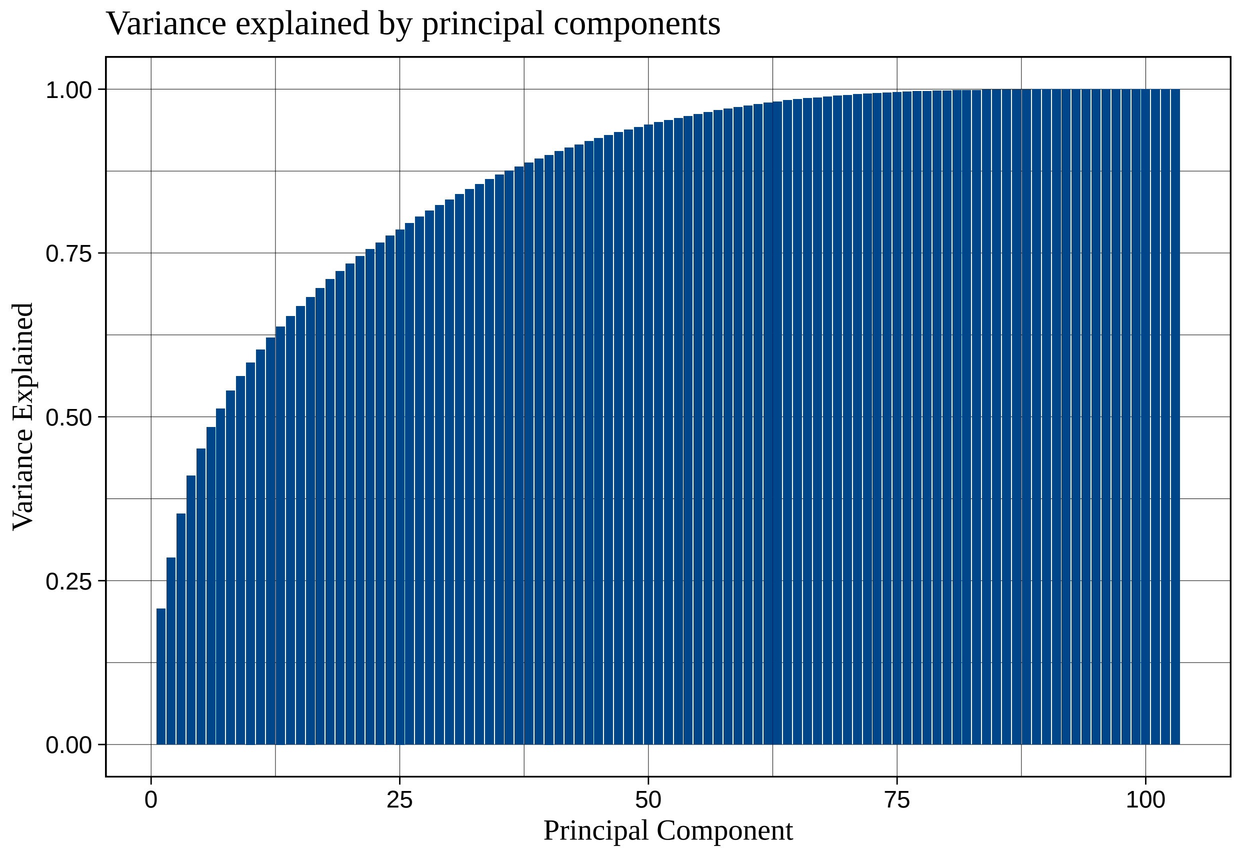
1. PCA

We do a Principal Component Analysis (PCA). Note that we must center and scale the data, since the series are in different scales.

```
# PCA
pca = data %>%
    select(-CPIAUCSL, -date) %>%
    prcomp(center = TRUE, scale = TRUE)
```

2. Select PCs

We can, then, choose the number of factors k and select the first k PCs. As seen in Question 1, there are different ways to choose the number of factors. We look at 3 common criterion (rule of thumb, informal way and biggest drop), but we opt for the rule of thumb as it seems to be the most parsimonious in this case.




```r
# Choosing the number of PCs

# Rule of thumb (3%)

pca.var.prop %>%
    filter(var.prop >= 0.03) %>%
    nrow() %>%
    paste("(rule of thumb)")
```

```
## [1] "6 (rule of thumb)"
```

```r
# Informal way (90%)

pca.var.prop %>%
    filter(var.prop.cum <= 0.9) %>%
    nrow() %>%
    paste("(informal way)")
```

```
## [1] "40 (informal way)"
```

```r
# Biggest drop

(lag(pca.var.prop$var.prop)/pca.var.prop$var.prop) %>%
    which.max() %>%
    -1 %>%
    paste("(biggest drop)")
```

```
## [1] "102 (biggest drop)"
```

```r
# Using the rule of thumb
n_pc = pca.var.prop %>%
    filter(var.prop >= 0.03) %>%
    nrow()
```

3. Regression

Given the number of factors, the order of the autoregressive component is determined by BIC in each rolling window.

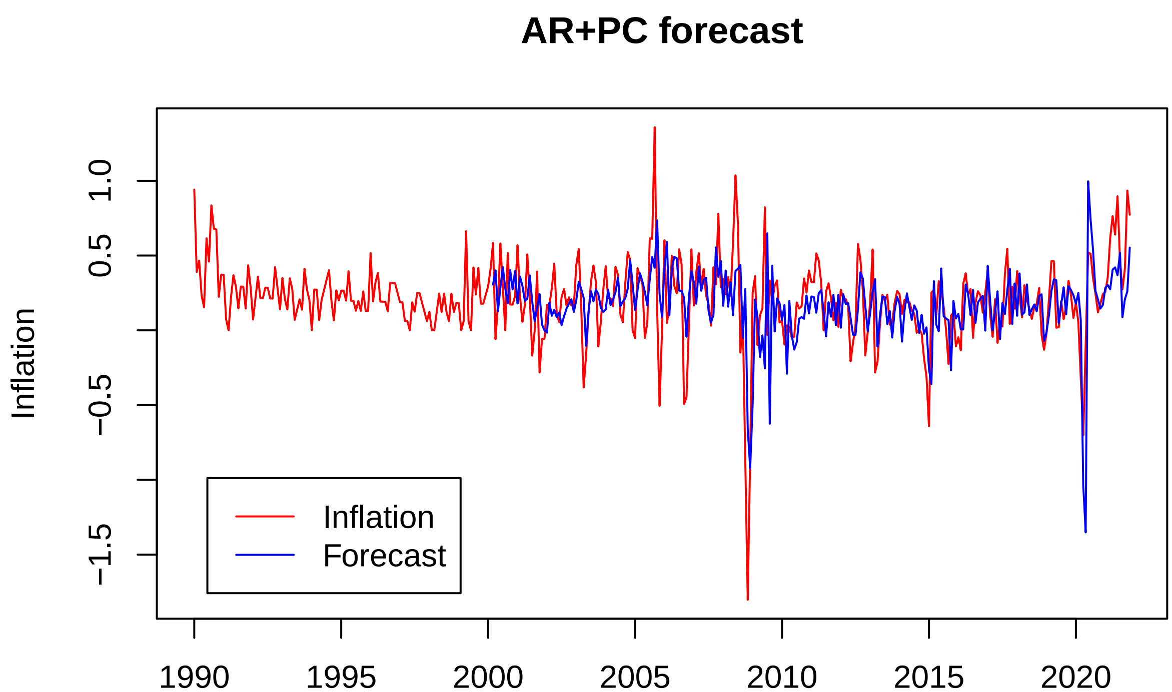
```r
# Get the factor from the PCA
Factors = pca$x[, 1:n_pc]

# Create the data matrix with the factors
variables = cbind(inflation, Factors)

variables_withdate = variables %>%
    bind_cols(date = as.Date.yearmon(time(inflation))) %>%
    setNames(c("inflation", colnames(Factors), "date"))
```

We proceed with the rolling window one-step-ahead forecast.

```r
# Function for creating the proper data matrix based on the
# regression formula

# Instead of manually creating data matrix, we use the
# dynlm() function and get only the $model component
create_datamatrix = function(train, p.opt) {
    new = dynlm(inflation ~ L(inflation, 1:p.opt) + L(train[,
        -1], 1), data = ts(rbind(train, 0), start = start(train),
        frequency = frequency(train)))
    new = new$model %>%
        tail(1) %>%
        select(-inflation) %>%
        as.matrix()
    return(new)
}
```

```r
# Rolling window forecasting
rolling_window <- 492
p.max <- 24

forecast1 = list()
coefficients_pc1 <- list()

# set up parallel computation
registerDoFuture()
plan("multisession", workers = 3)  # use 3 cores

# Loop
forecast1 = foreach(a = 0:(length(inflation) - rolling_window -
    1)) %dorng% {

    train = window(variables, start = start(inflation) + c(0,
        a), end = start(inflation) + c(0, a + rolling_window -
        1))
```

```
    bic.table = rep(NA, p.max)

    for (p in 1:p.max) {
        # calculating the BIC for different orders of the
        # AR(p)
        AR_PC = dynlm(inflation ~ L(inflation, 1:p) + L(train[,
            -1], 1), data = train)
        bic.table[p] = BIC(AR_PC)
    }

    p.opt = which.min(bic.table)  # pick the optimal p

    AR_PC = dynlm(inflation ~ L(inflation, 1:p.opt) + L(train[,
        -1], 1), data = train)  # run the AR-PC model with the optimal p

    new = create_datamatrix(train, p.opt)

    result = AR_PC$coefficients %*% c(1, new)  # one-step-ahead forecast
    result
}

forecast1 = forecast1 %>%
    unlist() %>%
    ts(start = start(inflation) + c(0, rolling_window), frequency = frequency(inflation))


# Loop to get coefficients
coefficients_pc1 = foreach(a = 0:(length(inflation) - rolling_window -
    1)) %dorng% {

    train = window(variables, start = start(inflation) + c(0,
        a), end = start(inflation) + c(0, a + rolling_window -
        1))

    bic.table = rep(NA, p.max)

    for (p in 1:p.max) {
        # calculating the BIC for different orders of the
        # AR(p)
        AR_PC = dynlm(inflation ~ L(inflation, 1:p) + L(train[,
            -1], 1), data = train)
        bic.table[p] = BIC(AR_PC)
    }

    p.opt = which.min(bic.table)  # pick the optimal p

    AR_PC = dynlm(inflation ~ L(inflation, 1:p.opt) + L(train[,
        -1], 1), data = train)  # run the AR-PC model with the optimal p

    result = AR_PC[[1]]  # one-step-ahead forecast
    result
}
```

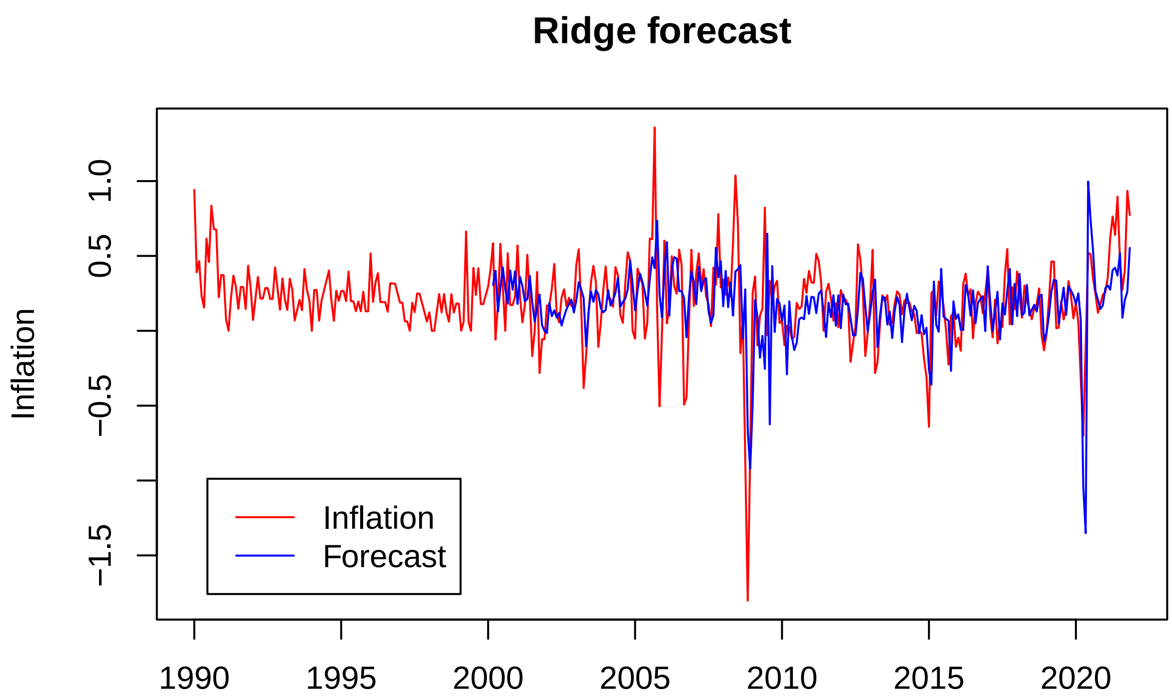



```
# Save forecasts
forecasts = cbind(AR = forecasts, AR_PC = forecast1) %>%
    as.ts()
```

Ridge Regression

We will choose penalty term according to the BIC. However, we must decide on the number of lags in the model and this criterion is obviously silent about this issue. Our strategy will be to run the models with 1, 2, 3 and 4 lags and choose the model with the smallest MSE.

```
# Embedding function that creates n_lags of all variables
# of a given data frame
my_embed = function(df, n_lags = 4) {
    Lags = list()
    Lags[[1]] = df %>%
        select(-contains("date"))
    if (n_lags >= 1) {
        for (i in 1:n_lags) {
            Lags[[i + 1]] = df %>%
                select(-contains("date")) %>%
                mutate_all(function(x) lag(x, n = i))
        }
    }
    lagged_data = reduce(Lags, function(x, y) {
        bind_cols(x, y, .name_repair = ~make.unique(.x))
    })

    return(lagged_data)
}
```



Using 4 lags of all variables

```
# Save forecasts
forecasts = cbind.zoo(forecasts, Ridge_4lags = forecast1) %>%
    as.ts()
```

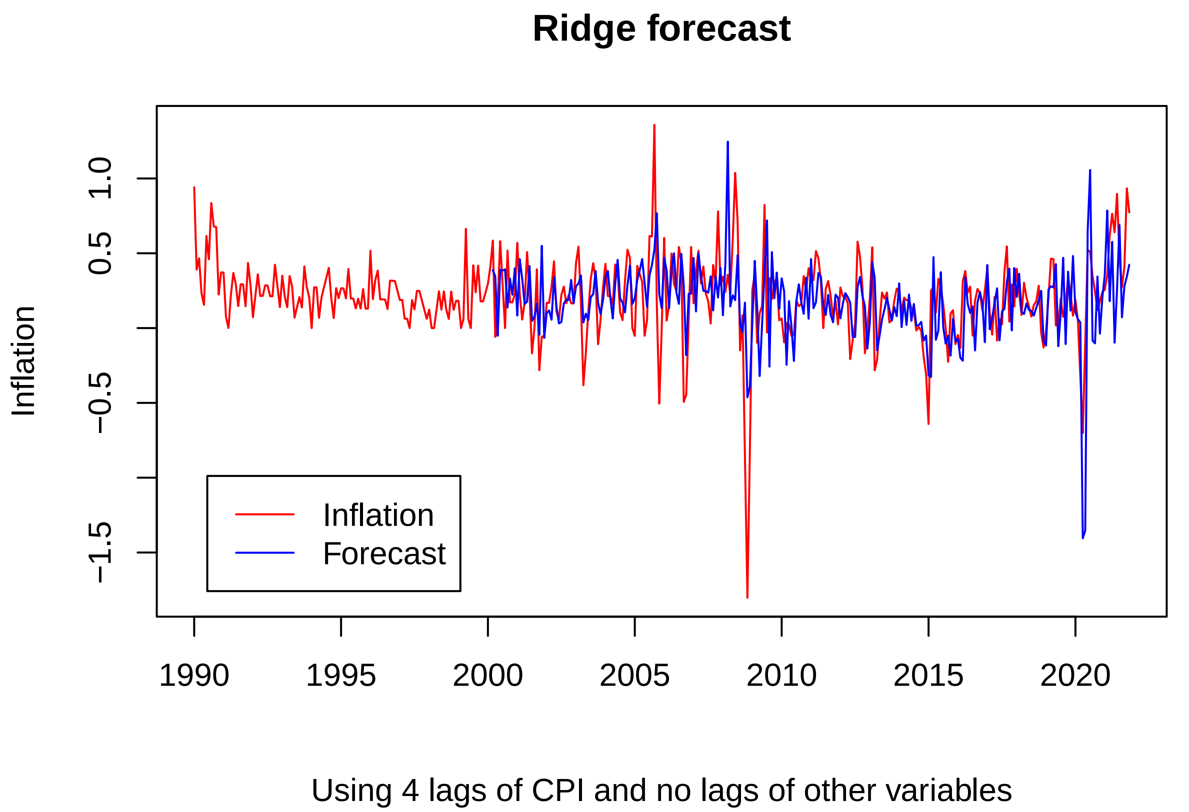
The forecast of the Ridge regression with 4 lags has a notably bad fit to the actual inflation series. We noticed that, since the ridge is not able to give a sparse solution, when there are too many variables, the estimated model becomes basically an intercept and almost all the other coefficients are very close to zero (but not zero). Hence, we tested other (more parsimonious) specifications. When we include the all the macroeconomics variables - without any lags - and lags of the CPI, we get a more reasonable result. The results are very robust to the number of CPI lags, so we keep 4 lags, as initially intended.

```
tic()
# Rolling window forecasting
rolling_window <- 492

# glmnet parameter
my_alpha = 0  # Ridge

forecast1 = list()

# set up parallel computation
registerDoFuture()
plan("multisession", workers = 3)  # use 3 cores

last_fcst = (length(inflation) - rolling_window)

output = foreach(a = 1:last_fcst) %dorng% {
    # get the window for training the model
    train = data[a:(a + rolling_window - 1), ] %>%
```

```r
        select(-CPIAUCSL)
    train_cpi = data[a:(a + rolling_window - 1), ] %>%
        select(CPIAUCSL)
    # embed
    reg_data = my_embed(train, n_lags = 0)
    cpi_lags = my_embed(train_cpi, n_lags = 4)
    # bind the embeded columns with the one-step-ahead
    # inflation
    reg_data = bind_cols(inflation.ahead = lead(inflation[a:(a +
        rolling_window - 1)]), cpi_lags, reg_data)

    # Ridge estimation
    ic_ridge <- ic.glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), crit = "bic", alpha = my_alpha)
    ridge <- glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), alpha = my_alpha, lambda = ic_ridge$lambda)

    # Prediction
    new = reg_data %>%
        select(-inflation.ahead) %>%
        tail(1)
    result1 = predict(ridge, newx = data.matrix(new), s = ic_ridge$lambda)

    # Coeficients
    result2 = coef(ridge, s = ic_ridge$lambda)

    result = list(forecast1 = result1, coef = result2)
    result
}

output = output %>%
    transpose()

forecast1 = output$forecast1 %>%
    unlist() %>%
    ts(start = start(inflation) + c(0, rolling_window), frequency = frequency(inflation))

ridge_coeficients = output$coef %>%
    reduce(cbind) %>%
    as.matrix()

toc()
```

```
## 115.872 sec elapsed
```

```r
beepr::beep()
```




```r
# Save forecasts
forecasts = cbind.zoo(forecasts, Ridge = forecast1) %>%
    as.ts()
```

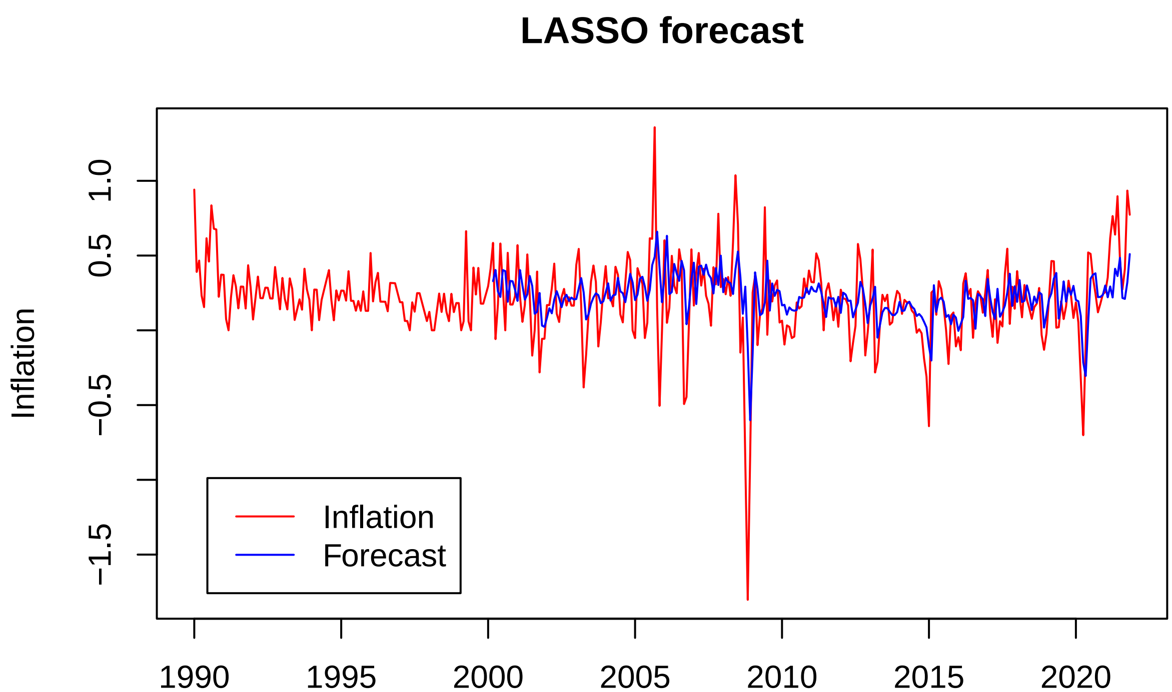
LASSO Regression

```r
tic()
# Rolling window forecasting
rolling_window <- 492

# glmnet parameter
my_alpha = 1  # LASSO

forecast1 = list()
coefficients_lasso = list()


for (a in 1:(length(inflation) - rolling_window)) {
    # get the window for training the model
    train = data[a:(a + rolling_window - 1), ]
    # embed
    reg_data = my_embed(train)
    # bind the embeded columns with the one-step-ahead
    # inflation
    reg_data = bind_cols(inflation.ahead = lead(inflation[a:(a +
        rolling_window - 1)]), reg_data)
```

```
    # Ridge estimation
    ic_lasso <- ic.glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), crit = "bic", alpha = my_alpha)
    lasso <- glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), alpha = my_alpha, lambda = ic_lasso$lambda)

    # Prediction
    new = reg_data %>%
        select(-inflation.ahead) %>%
        tail(1)
    forecast1[a] = predict(lasso, newx = data.matrix(new), s = ic_lasso$lambda)

    # Coefficients
    coefficients_lasso[a] = coef(lasso)

}

forecast1 = forecast1 %>%
    unlist() %>%
    ts(start = start(inflation) + c(0, rolling_window), frequency = frequency(inflation))

toc()
```

```
## 37.407 sec elapsed
```

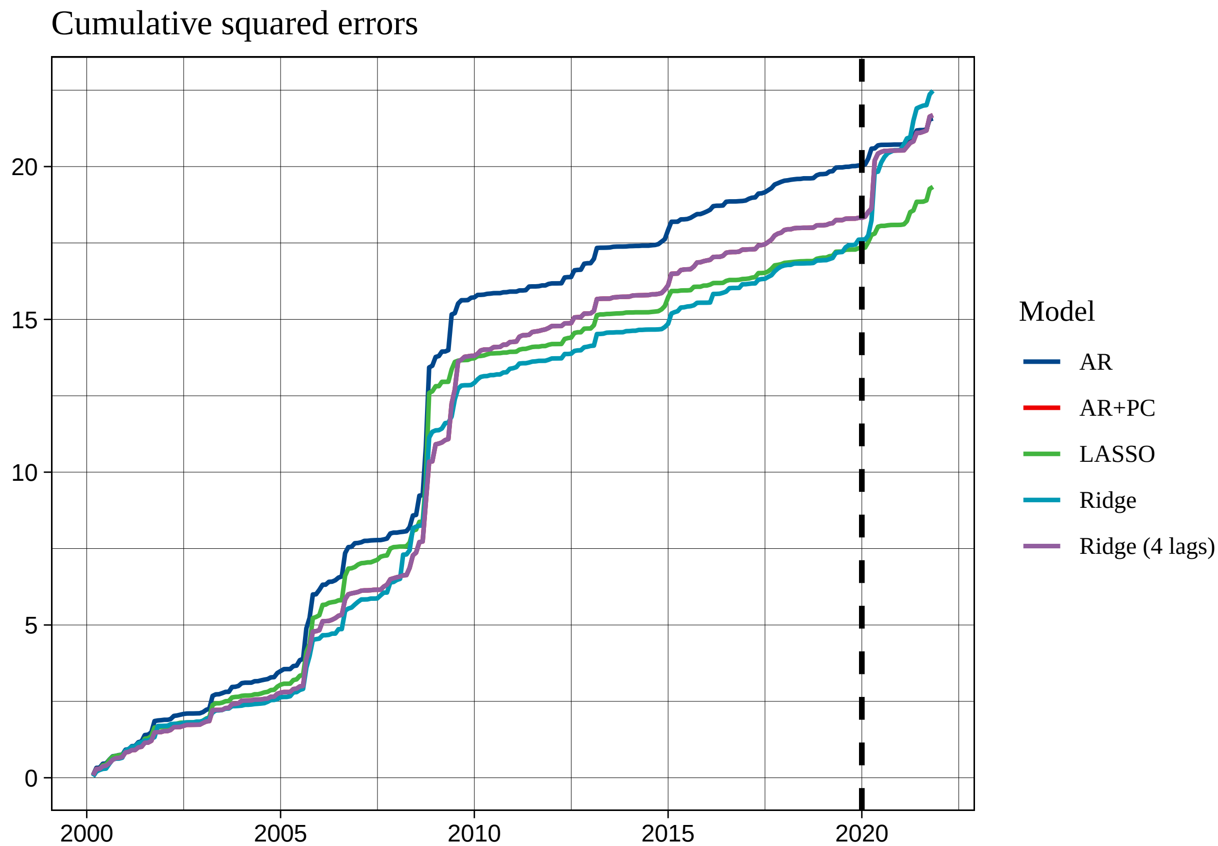
```
beepr::beep()
```




```
# Save forecasts
forecasts = cbind.zoo(forecasts, LASSO = forecast1) %>%
    as.ts()
```

Item A

```
# Forecasting error
error = inflation - forecasts
cum_error = sapply(error, function(x) {
    x^2 %>%
        cumsum()
}) %>%
    bind_cols(date = as.Date.yearmon(time(error))) %>%
    setNames(c("AR", "AR+PC", "Ridge (4 lags)", "Ridge", "LASSO",
        "date"))


# cum_error = sapply(error, function(x){x^2 %>% cumsum()})
# %>% bind_cols(date = as.Date.yearmon(time(error))) %>%
# setNames( c('AR', 'AR+PC', 'Ridge', 'LASSO', 'date' ) )
```



Item B

We will follow the FRED-MD classification of variables into 8 groups: (i) output and income; (ii) labor market; (iii) housing; (iv) consumption,orders and inventories; (v) money and credit; (vi) interest and exchange rates; (vii) prices; and (viii) stock market. We are adding a ninth group called (ix) lags, with the lagged inflation series.

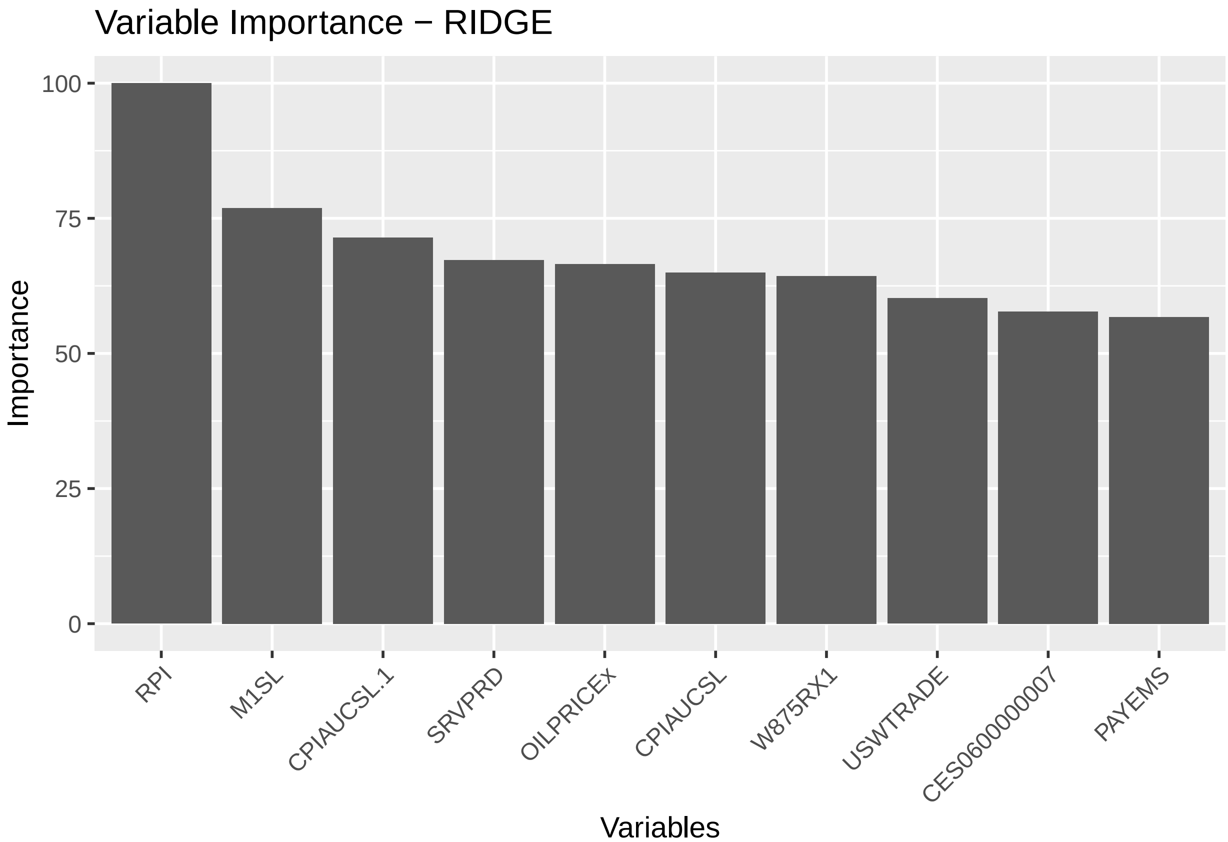
```
# Get FRED groups groups = read_xlsx('C:\\Users\\Caio
# Garzeri\\OneDrive - puc-rio.br\\Econometria
# IV\\AssignmentEconometricsIV\\data\\FRED-MD_updated_appendix.xlsx')
groups = read_xlsx("data/FRED-MD_updated_appendix.xlsx")
groups <- groups %>%
    select(fred, group)


# Change some names manually because they have minor
# differences with the variable names in existing dataframe
groups$fred[groups$fred == "S&P 500"] <- "SP500"
groups$fred[groups$fred == "IPB51222s"] <- "IPB51222S"
groups$fred[groups$fred == "S&P: indust"] <- "SPINDUST"

names <- function(base_name, n) {
    new_name = paste0(base_name, ".", n)
    return(new_name)
}

# Expand group_df with new variable names
```

```
expandgroup <- groups %>%
    rowwise() %>%
    mutate(NewVariables = list(names(fred, 1:4)), NewGroups = list(rep(group,
        length(NewVariables)))) %>%
    unnest(c(NewVariables, NewGroups)) %>%
    select(c(NewVariables, NewGroups)) %>%
    rename(fred = "NewVariables", group = "NewGroups")

# Merge with original group_df
endgroups <- bind_rows(groups, expandgroup)

# Sort by variable name
endgroups <- endgroups %>%
    arrange(fred)

# Change CPI lags to group 'lags' (9)
endgroups$group[endgroups$fred == "CPIAUCSL"] <- 9
endgroups$group[endgroups$fred == "CPIAUCSL.1"] <- 9
endgroups$group[endgroups$fred == "CPIAUCSL.2"] <- 9
endgroups$group[endgroups$fred == "CPIAUCSL.3"] <- 9
endgroups$group[endgroups$fred == "CPIAUCSL.4"] <- 9

groups <- endgroups

rm(endgroups, expandgroup, names)
```

We compute variable importance for Ridge and pick the top 10 most important overtime.

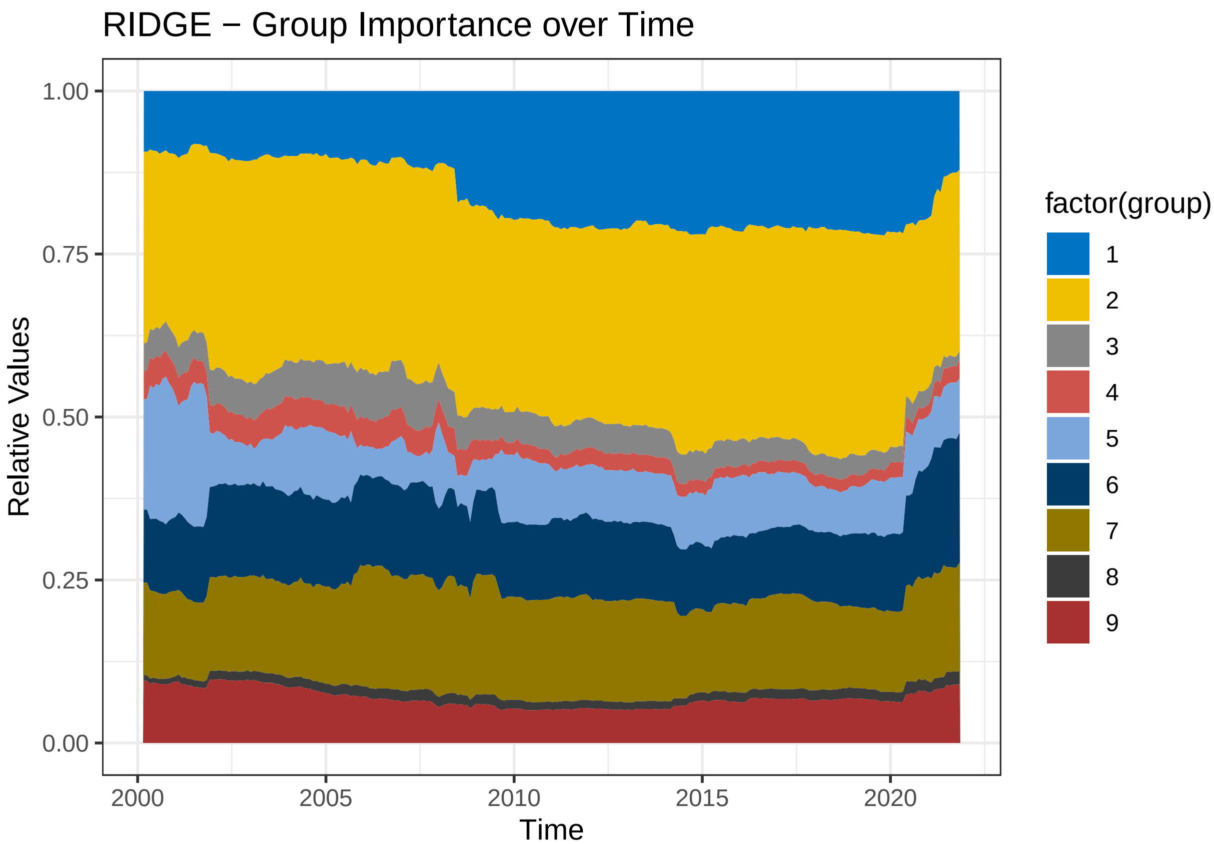
```
# Computing variable importance for RIDGE

ridge_coeff <- as.data.frame(ridge_coeficients)
colnames(ridge_coeff) <- NULL
ridge_coeff <- ridge_coeff[2:109, ]

ridge_names <- ridge_coeff %>%
    row.names(.)
names <- as.data.frame(ridge_names)
ridge_coeff <- cbind(names, ridge_coeff)
reg_data2 <- reg_data %>%
    select(-inflation.ahead)
std_deviations <- apply(reg_data2, 2, sd)
std_dev_df <- data.frame(Column_Names = colnames(reg_data2),
    Standard_Deviation = std_deviations)
std_dev_df <- std_dev_df %>%
    rename(ridge_names = "Column_Names")

ridge_coeff <- merge(ridge_coeff, std_dev_df, by = "ridge_names",
    all.x = TRUE)

ridge_coeff$Standard_Deviation[ridge_coeff$ridge_names == "CPIAUCSL.1"] <- ridge_coeff$Standard_Deviati
    "CPIAUCSL"]
ridge_coeff$Standard_Deviation[ridge_coeff$ridge_names == "CPIAUCSL.2"] <- ridge_coeff$Standard_Deviati
    "CPIAUCSL"]
ridge_coeff$Standard_Deviation[ridge_coeff$ridge_names == "CPIAUCSL.3"] <- ridge_coeff$Standard_Deviati
```

```
    "CPIAUCSL"]
ridge_coeff$Standard_Deviation[ridge_coeff$ridge_names == "CPIAUCSL.4"] <- ridge_coeff$Standard_Deviati
    "CPIAUCSL"]

ridge_coeff_std <- ridge_coeff

for (col in 2:262) {
    ridge_coeff_std[[col]] <- ridge_coeff_std[[col]] * ridge_coeff_std$Standard_Deviation
}

top10_ridge <- ridge_coeff_std %>%
    mutate(Mean_Value = rowMeans(across(2:262, ~abs(.)))) %>%
    select(ridge_names, Mean_Value) %>%
    arrange(desc(Mean_Value)) %>%
    head(10)

top10_ridge <- top10_ridge %>%
    mutate(importance = 100 * Mean_Value/Mean_Value[1]) %>%
    arrange(desc(importance))

ggplot(top10_ridge, aes(x = reorder(ridge_names, -importance),
    y = importance)) + geom_bar(stat = "identity") + labs(title = "Variable Importance - RIDGE",
    x = "Variables", y = "Importance") + theme(axis.text.x = element_text(angle = 45,
    hjust = 1))
```



We now compute importance by group in Ridge.

```r
# Get sum over groups Sum over cells based on groups

ridge_coeff_std <- ridge_coeff_std %>%
    rename(fred = "ridge_names")

ridge_coeff_std <- merge(ridge_coeff_std, groups, by = "fred",
    all.x = TRUE)

ridge_group <- ridge_coeff_std

for (i in 2:262) {
    for (j in 1:108) {
        ridge_group[j, i] <- abs(ridge_coeff_std[j, i])/sum(abs((ridge_coeff_std[,
            i])))
    }
}

group_sums <- ridge_group %>%
    group_by(group) %>%
    summarize(across(2:262, ~sum(.)))

colnames(group_sums)[2:262] <- as.Date(time(forecast1))

group_sums_long <- pivot_longer(group_sums, cols = -group, names_to = "Time",
    values_to = "Value")
group_sums_long$Time <- as.integer(group_sums_long$Time)
group_sums_long$date <- as.Date(group_sums_long$Time)

ggplot(group_sums_long, aes(x = date, y = Value, fill = factor(group))) +
    geom_area() + labs(title = "RIDGE - Group Importance over Time",
    x = "Time", y = "Relative Values") + scale_fill_discrete(name = "Groups") +
    theme_bw() + scale_fill_jco()
```



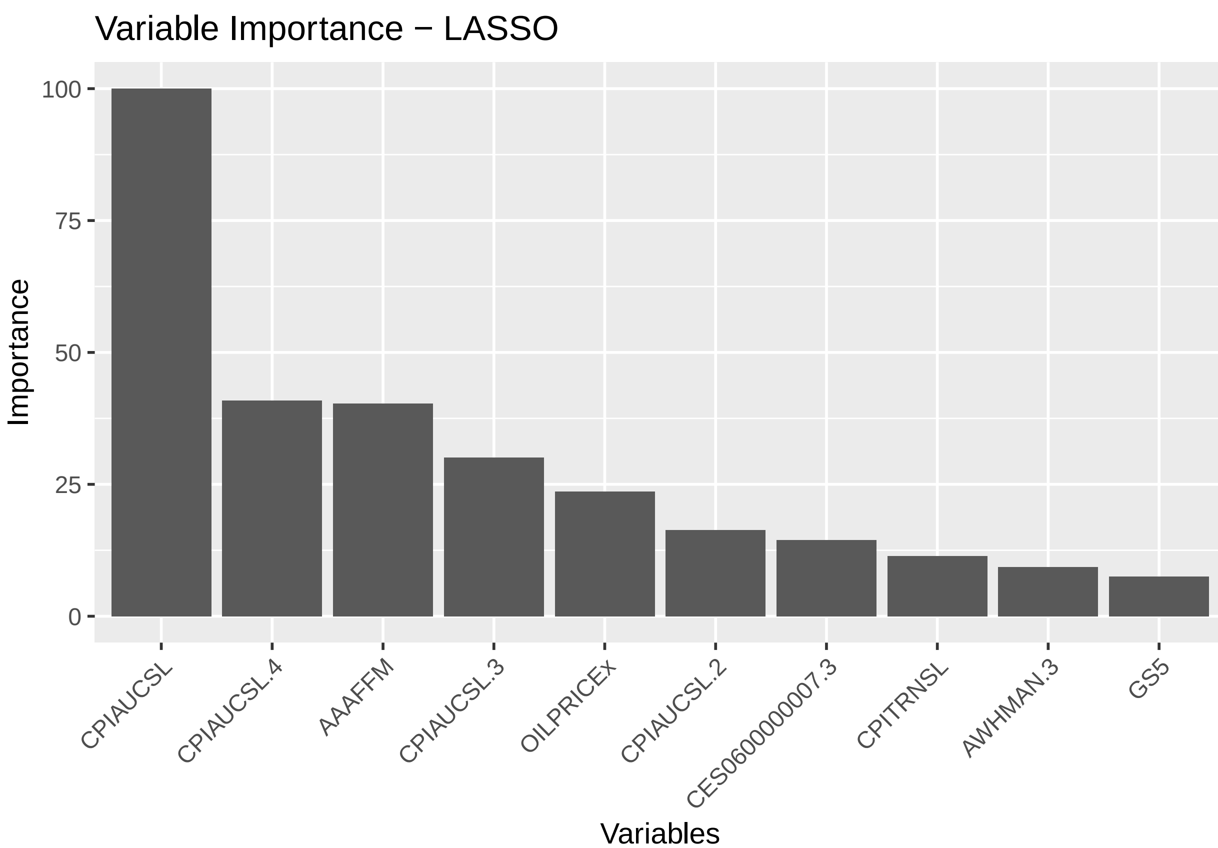
We repeat the exercise for LASSO. First selecting the top 10 most important variables.

```
# Computing variable importance for LASSO

# Create a matrix to store coefficients

coeff_lasso <- data.frame(matrix(ncol = ncol(reg_data2), nrow = length(forecast1)))
colnames(coeff_lasso) <- colnames(reg_data2)

# Retrieve coefficients and variable identifiers from lists
var_lasso = modify_depth(coefficients_lasso, 1, "i")
co_lasso = modify_depth(coefficients_lasso, 1, "x")

for (i in 1:length(forecast1)) {
    a = var_lasso[[i]] %>%
        unlist()
    b = co_lasso[[i]] %>%
        unlist()
    for (c in 2:length(a)) {
        coeff_lasso[i, a[c]] <- b[c]
    }
}

rm(var_lasso, co_lasso)

# Multiply for sd
for (i in 1:length(forecast1)) {
    coeff_lasso[, i] = coeff_lasso[, i] * sd(reg_data2[, i])
```

```
}
```

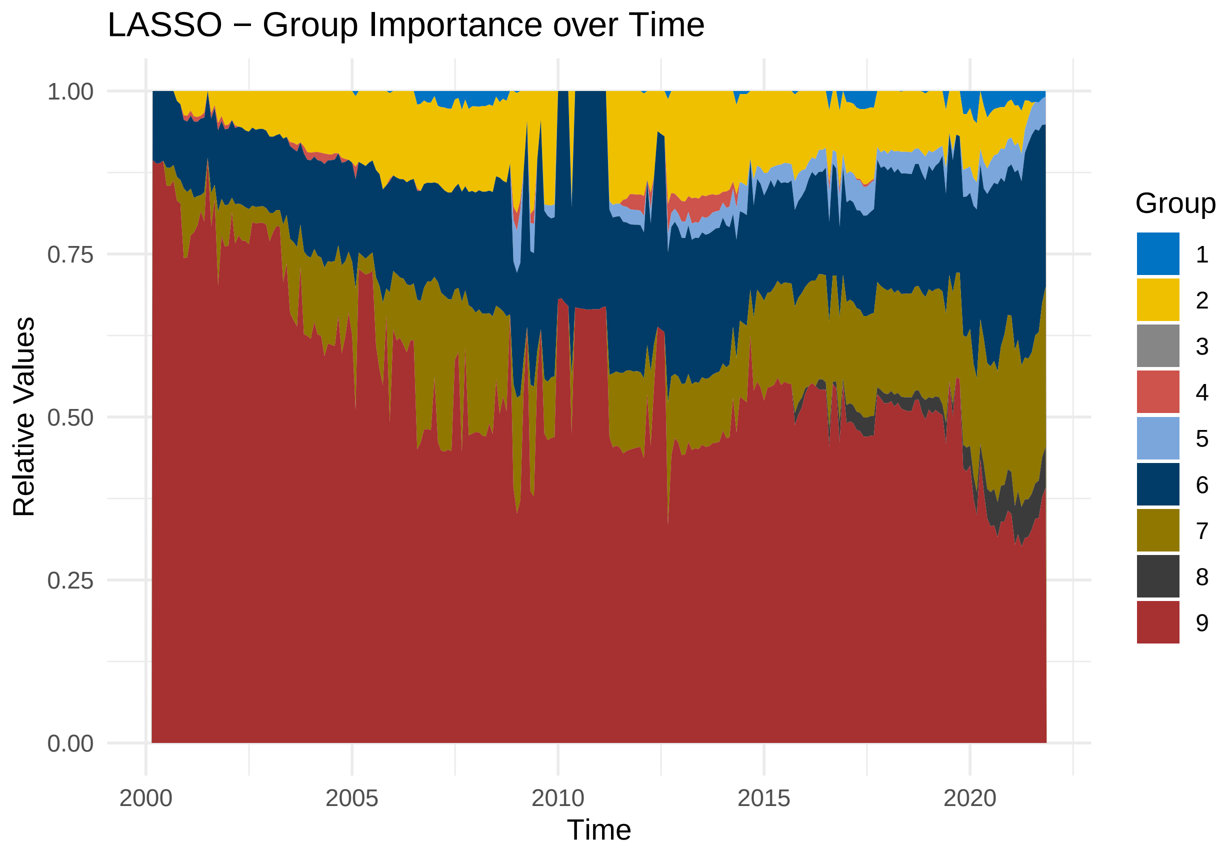
We compute the 10 most relevant predictors considering the mean absolute value of the coefficients over all estimation windows.

```
coeff_lasso <- coeff_lasso %>%
    mutate_all(~replace_na(., 0))

top10_lasso <- colMeans(abs(coeff_lasso))

top_10_lasso <- coeff_lasso %>%
    summarise_all(~mean(abs(.))) %>%
    pivot_longer(everything()) %>%
    arrange(desc(value)) %>%
    head(10)

top_10_lasso <- top_10_lasso %>%
    mutate(importance = 100 * value/value[1]) %>%
    arrange(desc(importance))

ggplot(top_10_lasso, aes(x = reorder(name, -importance), y = importance)) +
    geom_bar(stat = "identity") + labs(title = "Variable Importance - LASSO",
    x = "Variables", y = "Importance") + theme(axis.text.x = element_text(angle = 45,
    hjust = 1))
```



We now present results for groups. Group 9 (previous inflation) is consistently the most important though less important throught the sample. Groups 6 (bond and exchange rates) and 7 (prices) are important. Group (2) labor market used to be relevant. For the most recent windows, not so much.

```r
# Get sum over groups Sum over cells based on groups
coeff_long <- data.frame(variable = rep(colnames(coeff_lasso),
    each = nrow(coeff_lasso)), row_index = rep(1:nrow(coeff_lasso),
    times = ncol(coeff_lasso)), value = as.vector(as.matrix(coeff_lasso)))

coeff_long <- coeff_long %>%
    arrange(row_index)
groups <- groups %>%
    rename(variable = "fred")
merged_data <- merge(coeff_long, groups, by = "variable", all.x = TRUE)
merged_data <- merged_data %>%
    arrange(row_index)

groupfinal_lasso <- merged_data %>%
    group_by(row_index, group) %>%
    summarise(total = sum(abs(value)))

wide_group_lasso <- groupfinal_lasso %>%
    pivot_wider(names_from = group, values_from = total) %>%
    ungroup()

wide_group_lasso_rel <- wide_group_lasso %>%
    mutate(across(-1, ~./rowSums(across(-1))))

wide_group_lasso_rel$dates <- as.Date(time(forecast1))

# Melt the dataframe to long format for plotting
melted_df <- melt(wide_group_lasso_rel, id.vars = "dates", variable.name = "Column")
melted_df <- melted_df %>%
    filter(Column != "row_index")
melted_df$Group <- as.integer(melted_df$Column) - 1
melted_df$Group <- as.character(melted_df$Group)


# Create a stacked column plot
ggplot(data = melted_df, aes(x = dates, y = value, fill = Group)) +
    geom_area() + labs(title = "LASSO - Group Importance over Time",
    x = "Time", y = "Relative Values") + scale_fill_discrete(name = "Groups") +
    theme_minimal() + scale_fill_jco()
```



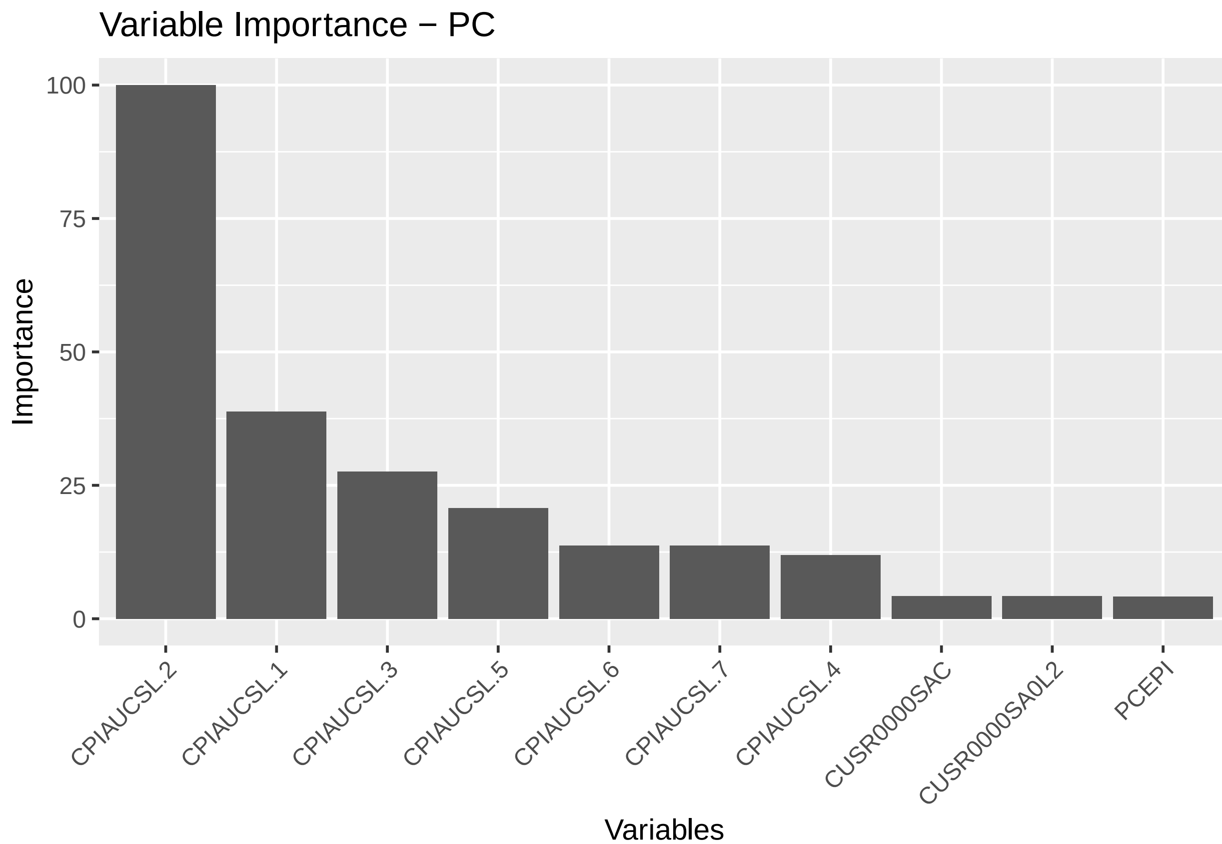
Finally, we do that for the AR+PC model, computing variable importance of the PC block of the model. This is slightly more complicated than LASSO and Ridge. We retrieve the alphas from the Factor on Variables and then multiply them be the coefficients in the model regression.

```
# Computing variable importance for PC

# Get the alphas
alpha = as.matrix(pca$rotation[, 1:n_pc])

# Get the lambdas (coefficients of the Factors) and the
# phis
lambdas = matrix(NA, 261, 6)
phis = matrix(NA, 261, 24)

for (i in 1:261) {
    size = length(coefficients_pc1[[i]])
    lags = size - 1 - 6  # intercept and 6 factors
    for (j in 1:6) {
        lambdas[i, j] <- coefficients_pc1[[i]][size - 6 + j]
    }
    for (l in 1:24) {
        phis[i, l] <- coefficients_pc1[[i]][l + 1]
    }
}

# Multiply alpha by lambdas to get 'coefficient' of each
# variable in each window
importpc = as.data.frame(alpha %*% t(lambdas))
```

```
phist = as.data.frame(t(phis))
row_names <- paste("CPIAUCSL", seq(1, 24), sep = ".")
importpc$fred = rownames(importpc)
phist$fred = row_names
importpc <- rbind(importpc, phist)

groups <- groups %>%
    rename(fred = "variable")
importpc = merge(importpc, groups, by = "fred", all.x = TRUE)
importpc$group <- ifelse(is.na(importpc$group), 9, importpc$group)  # giving all lags of inflation grou


# Get the number of lags - we use this in item A
lags_PC_AR <- table(colSums(!is.na(phist)))

lags_PC_AR <- data.frame(lags = as.numeric(names(lags_PC_AR)),
    count = as.numeric(lags_PC_AR))

lags_PC_AR = lags_PC_AR[1:13, ]
lags_PC_AR <- lags_PC_AR %>%
    arrange(desc(count))
```

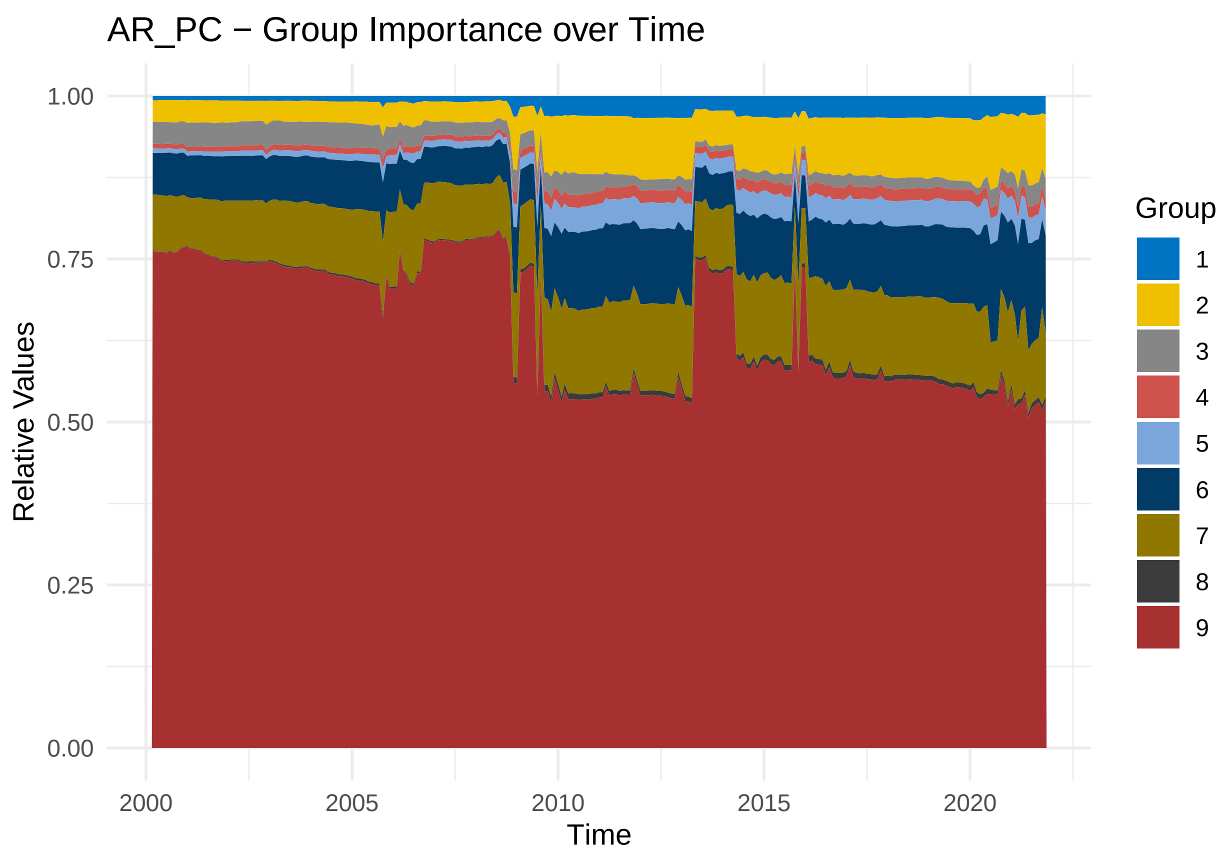
Top 10 most relevant variables

```
top_10_pc <- importpc %>%
    rowwise() %>%
    mutate(mean_abs = mean(abs(c_across(-c(fred, group))))) %>%
    ungroup() %>%
    select(fred, mean_abs) %>%
    arrange(desc(mean_abs)) %>%
    head(10)

top_10_pc <- top_10_pc %>%
    mutate(importance = 100 * mean_abs/mean_abs[1]) %>%
    arrange(desc(importance))

ggplot(top_10_pc, aes(x = reorder(fred, -importance), y = importance)) +
    geom_bar(stat = "identity") + labs(title = "Variable Importance - PC",
    x = "Variables", y = "Importance") + theme(axis.text.x = element_text(angle = 45,
    hjust = 1))
```



We again compute importance by group. Pattern is very close to that of LASSO: Groups 9, 7, 6, 2

```
result <- importpc %>%
    mutate(across(starts_with("V"), ~abs(.), .names = "abs_{.col}")) %>%
    group_by(group) %>%
    summarise(across(starts_with("abs_V"), ~sum(., na.rm = TRUE)))

pc_rel <- result %>%
    mutate(across(starts_with("abs_V"), ~./sum(., na.rm = TRUE),
        .names = "rel_{.col}")) %>%
    select(starts_with("rel_"))

pc_rel_transposed <- as.data.frame((t(pc_rel)))

pc_rel_transposed <- pc_rel_transposed %>%
    mutate(date = as.Date(time(forecast1)))

importpc_long <- pc_rel_transposed %>%
    pivot_longer(cols = starts_with("V"), names_to = "variable",
        values_to = "value")

importpc_long$Group <- as.character(gsub("\\D", "", importpc_long$variable))

ggplot(importpc_long, aes(x = date, y = value, fill = Group)) +
    geom_area() + labs(title = "AR_PC - Group Importance over Time",
    x = "Time", y = "Relative Values") + scale_fill_discrete(name = "Groups") +
    theme_minimal() + scale_fill_jco()
```


AR_PC − Group Importance over Time
Relative Values
1.00
0.75
0.50
0.25
0.00
2000
2005
2010
2015
2020
Time
Group
1
2
3
4
5
6
7
8
9