

Econometrics IV - Final Assignment - Code

Caio Garzeri, Guilherme Luz, Guilherme Masuko

August 15, 2023

```
In [ ]: import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns

from functions.linear_models import PCA_function, OLS_regression
from functions.number_PC import rule_thumb, informal_way, biggest_drop
```

```
In [ ]: # hide warning messages
import warnings
warnings.filterwarnings("ignore")
```

```
In [ ]: # better plots
sns.set(rc={'figure.figsize':(12,8)});
```

```
In [ ]: directory = os.path.dirname(os.getcwd())
directory
```

```
Out[ ]: 'd:\\github\\AssignmentEconometricsIV'
```

Question

The first question consists of a factor analysis of a large dataset. We consider monthly close-to-close excess returns from a cross-section of 9,456 firms traded in the New York Stock Exchange. The data starts on November 1991 and runs until December 2018. There are 326 monthly observations in total.

In addition to the returns we also consider 16 monthly factors:

- Market (MKT)
- Small-minus-Big (SMB)
- High-minus-Low (HML)
- Conservative-minus-Aggressive (CMA)
- Robust-minus-Weak (RMW)
- earning/price ratio (EP)
- cash-flow/price ratio (CFP)
- dividend/price ratio
- accruals (ACC)
- market beta (BETA)
- net share issues
- daily variance (RETVOL)
- daily idiosyncratic variance (IDIOVOL)
- 1-month momentum (MOM1)
- 36-month momentum (MOM36)

The dataset is organized as an excel file named `returns.xlsx`.

```
In [ ]: input_path = f'{directory}\\data\\returns.xlsx'
df = pd.read_excel(input_path, index_col=0)
```

```
In [ ]: df.head()
```

```
Out[ ]:
```

	MKT	HML	SMB	MOM1	MOM36	ACC	BETA	CFP	CHCSH
dates									
1991-11-29	-0.041264	-0.028083	0.004779	-0.007336	-0.025496	-0.013692	0.035433	-0.015116	-0.00677
1991-12-31	0.107984	-0.022529	-0.027366	0.010963	-0.021188	-0.027887	-0.082499	-0.032122	-0.00589
1992-01-31	-0.007668	0.051012	0.085547	0.050916	0.108588	0.021978	-0.072801	0.028117	-0.00819
1992-02-28	0.010796	0.070501	0.002794	-0.027398	0.079286	0.003860	-0.024906	0.037363	0.01562
1992-03-31	-0.025367	0.039029	-0.015135	-0.009367	0.024631	0.004612	0.041266	0.037916	0.01711

5 rows × 9472 columns

```
In [ ]: factors_name = df.columns[:16]
factors_name
```

```
Out[ ]: Index(['MKT', 'HML', 'SMB', 'MOM1', 'MOM36', 'ACC', 'BETA', 'CFP', 'CHCSH',
            'DY', 'EP', 'IDIOVOL', 'CMA', 'UMD', 'RMW', 'RETVOL'],
            dtype='object')
```

```
In [ ]: factors = df[factors_name]
factors.head()
```

```
Out[ ]:
```

	MKT	HML	SMB	MOM1	MOM36	ACC	BETA	CFP	CHCSH
dates									
1991-11-29	-0.041264	-0.028083	0.004779	-0.007336	-0.025496	-0.013692	0.035433	-0.015116	-0.00677
1991-12-31	0.107984	-0.022529	-0.027366	0.010963	-0.021188	-0.027887	-0.082499	-0.032122	-0.00589
1992-01-31	-0.007668	0.051012	0.085547	0.050916	0.108588	0.021978	-0.072801	0.028117	-0.00819
1992-02-28	0.010796	0.070501	0.002794	-0.027398	0.079286	0.003860	-0.024906	0.037363	0.01562
1992-03-31	-0.025367	0.039029	-0.015135	-0.009367	0.024631	0.004612	0.041266	0.037916	0.01711

```
In [ ]: returns_name = df.columns[16:]
returns_name
```

```
Out [ ]: Index(['r_ 1', 'r_ 2', 'r_ 3', 'r_ 4', 'r_ 5', 'r_ 6', 'r_ 7',
        'r_ 8', 'r_ 9', 'r_ 10',
        ...
        'r_9447', 'r_9448', 'r_9449', 'r_9450', 'r_9451', 'r_9452', 'r_9453',
        'r_9454', 'r_9455', 'r_9456'],
        dtype='object', length=9456)
```

```
In [ ]: returns = df[returns_name]
        returns.head()
```

```
Out [ ]:
```

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
dates									
1991-11-29	0.130715	-0.053900	-0.110696	-0.043900	0.218322	0.050645	0.575047	0.008921	-0.105349
1991-12-31	-0.010580	-0.056432	0.213591	0.183700	1.299230	0.306545	0.040644	-0.003800	0.318787
1992-01-31	-0.055124	-0.003400	-0.164114	-0.205154	-0.082347	0.075547	-0.024677	-0.066691	0.155137
1992-02-28	-0.202800	5.219422	0.039753	-0.057745	-0.117086	0.204517	-0.089757	-0.083881	0.018251
1992-03-31	0.078418	-0.181971	-0.023808	0.043112	0.415955	0.064782	-0.146257	-0.179871	-0.209580

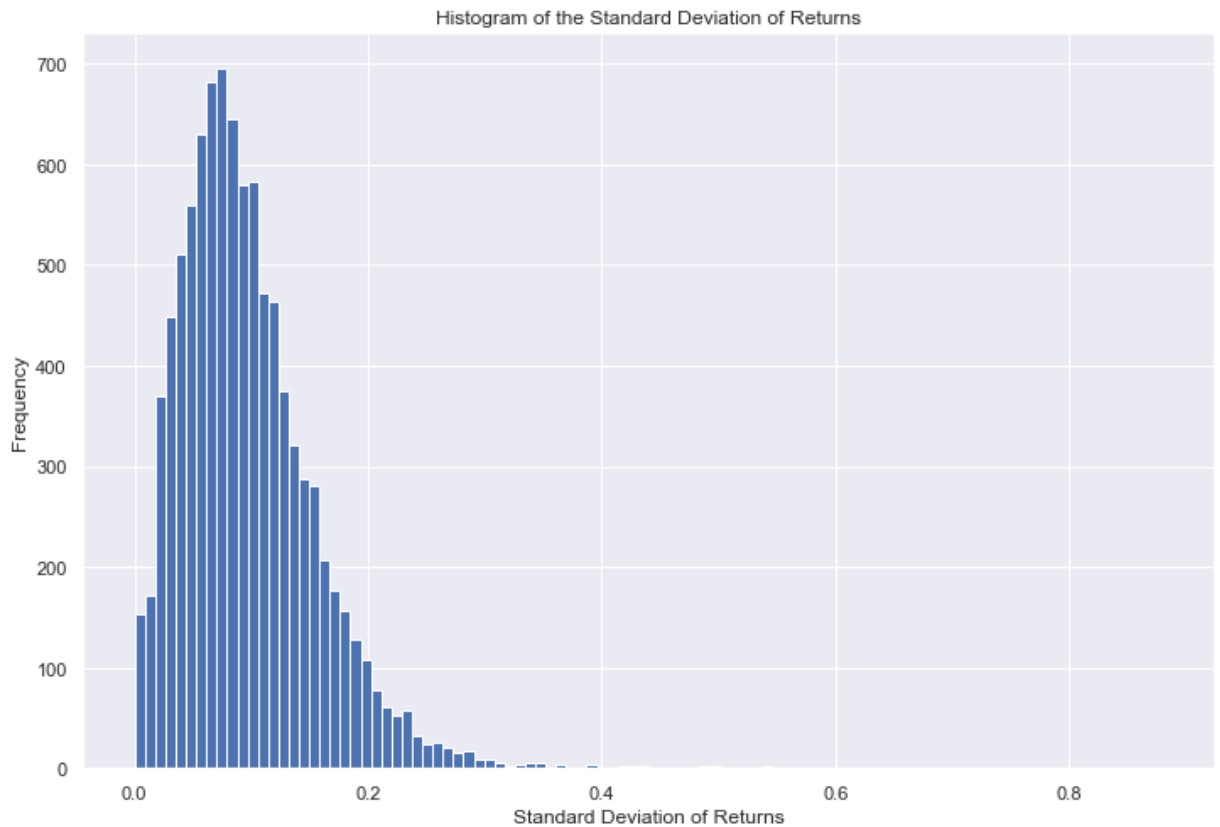
5 rows × 9456 columns



```
In [ ]: daterange = returns.index
        daterange
```

```
Out [ ]: DatetimeIndex(['1991-11-29', '1991-12-31', '1992-01-31', '1992-02-28',
        '1992-03-31', '1992-04-30', '1992-05-29', '1992-06-30',
        '1992-07-31', '1992-08-31',
        ...
        '2018-03-29', '2018-04-30', '2018-05-31', '2018-06-29',
        '2018-07-31', '2018-08-31', '2018-09-28', '2018-10-31',
        '2018-11-30', '2018-12-31'],
        dtype='datetime64[ns]', name='dates', length=326, freq=None)
```

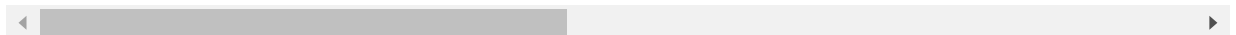
```
In [ ]: plt.hist(returns.std(axis=0), bins=100)
        plt.xlabel('Standard Deviation of Returns')
        plt.ylabel('Frequency')
        plt.title('Histogram of the Standard Deviation of Returns');
```



```
In [ ]: standardized_returns = (returns - returns.mean(axis=0))/returns.std(axis=0)
        standardized_returns.head()
```

```
Out[ ]:      r_1      r_2      r_3      r_4      r_5      r_6      r_7      r_8      r_
dates
1991-11-29  1.483764 -0.219996 -0.914187 -0.665776  1.031984  0.207163  3.474317 -0.028135 -0.77523
1991-12-31 -0.223389 -0.226904  1.435410  2.066812  6.526489  1.869218  0.235135 -0.125593  2.07608
1992-01-31 -0.761584 -0.082197 -1.301224 -2.601812 -0.496389  0.368900 -0.160796 -0.607431  0.97594
1992-02-28 -2.545832 14.169328  0.175881 -0.832001 -0.672972  1.206552 -0.555266 -0.739131  0.05571
1992-03-31  0.851900 -0.569465 -0.284648  0.378895  2.036596  0.298978 -0.897734 -1.474551 -1.47598
```

5 rows × 9456 columns



```
In [ ]: standardized_factors = (factors - factors.mean(axis=0))/factors.std(axis=0)
        standardized_factors.head()
```

```
Out[ ]:      MKT      HML      SMB      MOM1      MOM36      ACC      BETA      CFP      CHCSH
dates
1991-11-29 -1.146021 -0.942950  0.110054 -0.298999 -0.981611 -0.855086  0.608329 -0.530965 -0.39099
```

	MKT	HML	SMB	MOM1	MOM36	ACC	BETA	CFP	CHCSH
dates									
1991-12-31	2.438375	-0.768570	-0.927363	0.228826	-0.830584	-1.573833	-1.458119	-1.015147	-0.35606
1992-01-31	-0.339177	1.540227	2.716591	1.381237	3.719257	0.951073	-1.288179	0.699949	-0.44761
1992-02-28	0.104274	2.152085	0.045976	-0.877648	2.691977	0.033673	-0.448956	0.963170	0.50052
1992-03-31	-0.764238	1.164027	-0.532641	-0.357583	0.775803	0.071737	0.710532	0.978913	0.55994

The model

$$\begin{aligned}
 Y_t &= \beta_0 + \beta_1 X_{1t} + \cdots + \beta_p X_{pt} + U_t, \quad t = 1, \dots, T \\
 &= \boldsymbol{\beta}' \mathbf{X}_t + U_t \\
 \mathbf{Y} &= \mathbf{X}\boldsymbol{\beta} + \mathbf{U} \quad (\text{matrix notation}).
 \end{aligned}$$

However, $n > T$ (more columns than rows in \mathbf{X}).

The usual ordinary least squares (OLS) solution

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

is not valid anymore.

Solution: reduce the dimension of \mathbf{X} by postulating that:

$$\underset{(n \times 1)}{\mathbf{X}_t} = \underset{(n \times k)}{\boldsymbol{\Lambda}} \underset{(k \times 1)}{\mathbf{F}_t} + \underset{(n \times 1)}{\mathbf{V}_t},$$

where:

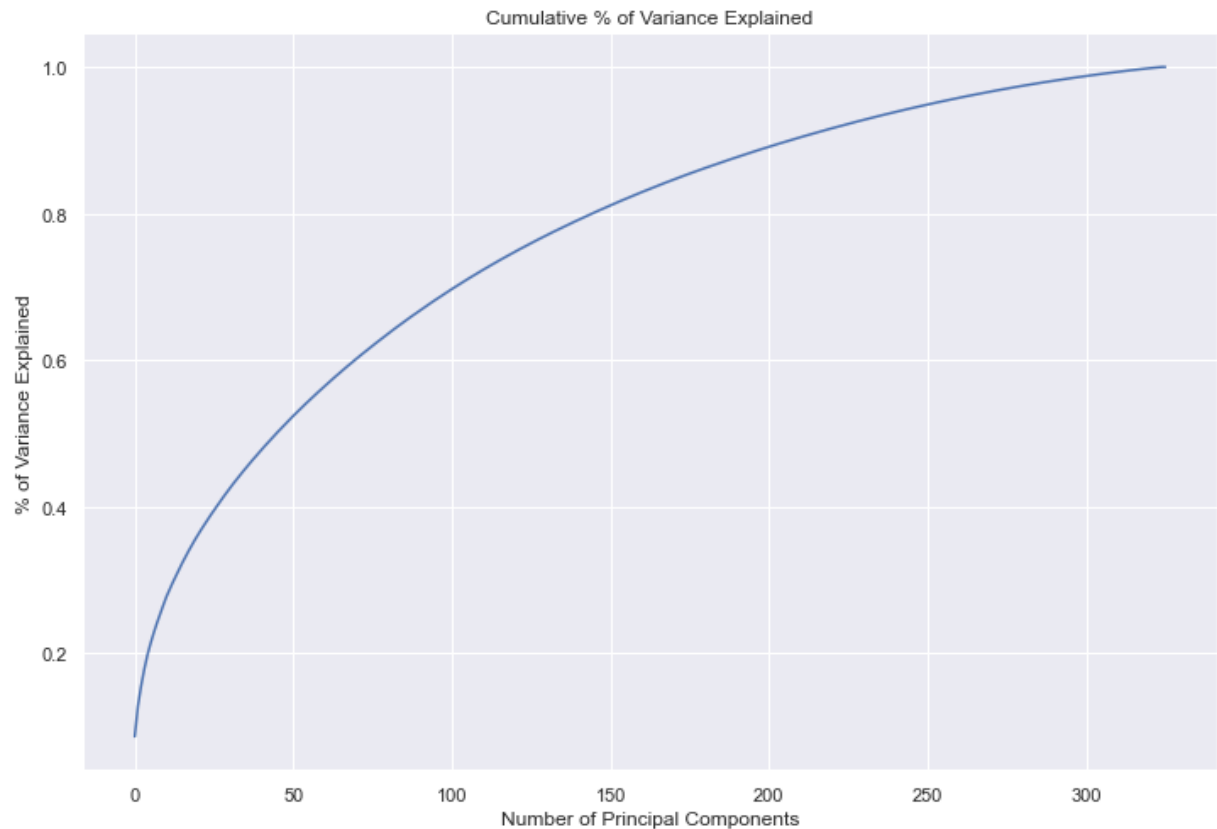
- \mathbf{F}_t is a set of $k \ll n$ unobserved factors;
- \mathbf{V}_t is the vector of idiosyncratic errors;
- $\boldsymbol{\Lambda}$ is the matrix of unobserved factor loadings.

(a) (30 points)

Compute the principal components of the returns and determine the optimal number of principal factors by one the methods described in Lecture 2. How much of the variance will the factors be able to explain?

In []:

```
pcs, gammas, lambdas, alphas = PCA_function(returns)
```



The transformed dataset containing only the first k PCs is the $(T \times k)$ matrix is given by

$$\begin{aligned} Z_{(k)} &:= \mathbf{X}\mathbf{\Gamma}_k \\ &:= (Z_1, \dots, Z_k). \end{aligned}$$

In []:

pcs

Out[]:

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC
dates									
1991-11-29	-0.932311	0.655585	0.023695	-0.023417	-0.126589	0.216290	-0.287319	0.317324	0.36757
1991-12-31	0.684426	-0.766075	0.098573	0.101246	0.403129	-0.197598	0.258110	-0.600949	-0.53229
1992-01-31	2.223682	-0.458117	-0.226293	-0.012401	0.090334	-0.388412	0.722587	-1.958945	-1.08676
1992-02-28	0.562873	-0.140579	0.159400	0.006865	0.158465	-0.255173	0.436401	-0.784899	-0.00900
1992-03-31	-0.743650	0.306019	0.224420	-0.089483	-0.141921	0.050060	-0.242078	0.170550	0.23308
...
2018-08-31	0.161871	-0.265360	-0.020674	-0.097998	0.026575	-0.056679	-0.109673	-0.051071	0.16519
2018-09-28	-0.942054	0.462936	0.124402	-0.082200	-0.051525	0.076335	-0.044931	0.278179	0.38150
2018-10-31	-2.535630	2.642636	-0.063295	0.056235	0.753975	0.545761	-0.142676	-0.170747	0.01582

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC
dates									
2018-11-30	-0.418945	0.200803	0.180173	0.041064	0.087297	0.077100	-0.050033	0.015725	0.007100
2018-12-31	-3.014504	3.133093	-0.011031	0.038419	0.759613	0.698890	-0.215036	-0.120849	0.340360

326 rows × 326 columns

Given the desired number of PCs, say $1 \leq k \leq n$, we collect all the vectors $\gamma_1, \dots, \gamma_k$ in a $(n \times k)$ matrix

$$\Gamma_k := (\gamma_1, \dots, \gamma_k)$$

In []:

```
gammas
```

Out[]:

	gamma 1	gamma 2	gamma 3	gamma 4	gamma 5	gamma 6	gamma 7	gamma 8	gamma 9
r_1	0.002032	-0.002777	-0.000804	0.000050	0.002960	0.001060	-0.002195	0.001189	0.003200
r_2	0.027143	0.015565	-0.009166	-0.002057	0.016365	0.000631	0.050304	-0.038719	0.031000
r_3	0.005195	-0.013071	-0.003946	-0.000183	-0.011596	-0.007778	0.006669	-0.008175	-0.001600
r_4	0.005707	-0.010162	-0.002122	-0.000947	0.001569	-0.003475	0.010309	-0.008786	-0.013100
r_5	0.012439	-0.006495	-0.005756	0.003450	0.002966	-0.002718	0.000583	0.010150	-0.025100
...
r_9452	0.000369	-0.000801	0.000187	0.000163	-0.000448	-0.000325	-0.000036	0.000097	-0.000000
r_9453	-0.000875	-0.000733	0.001381	-0.001983	-0.001478	-0.002800	-0.001009	0.006402	0.006900
r_9454	0.000287	-0.000691	0.000124	0.000281	-0.000370	-0.000841	0.000233	-0.000081	-0.000000
r_9455	0.002324	-0.005739	0.001035	-0.000288	-0.003415	-0.003190	0.000175	0.002162	0.000100
r_9456	0.001238	-0.003543	0.000601	-0.000469	-0.002063	-0.002022	-0.000150	0.002713	0.001400

9456 rows × 326 columns

In []:

```
# PCA function centered the data before to compute the PCs
centered_returns = (returns - returns.mean(axis=0))
centered_factors = (factors - factors.mean(axis=0))
```

In []:

```
# just checking
sum(round(centered_returns.dot(np.array(gammas["gamma 1"])), 10) == round(pcs["PC 1"]
```

Out[]: 326

Also, by construction, the columns of $Z_{(k)}$ (the PCs) are orthogonal random variables, and with sample variances $\lambda_1, \dots, \lambda_k$.

Thus, the sample covariance matrix of $Z_{(k)}$ is given by the $(k \times k)$ diagonal matrix

$$\mathbf{\Lambda}_k := \text{diag}(\lambda_1, \dots, \lambda_k)$$

```
In [ ]: lambdas[:10]
```

```
Out[ ]: array([10.59291481,  5.04991983,  3.3383555,  2.76500035,  2.40685564,
          1.94349613,  1.7572486 ,  1.61955915,  1.51600711,  1.45686289])
```

It is a good idea to start by running a full PCA ($k = n$) and plotting the quantity

$$\alpha_j = \frac{\lambda_j}{\sum_{j=1}^n \lambda_j}$$

for $j \in \{1, \dots, n\}$.

```
In [ ]: alphas[:10]
```

```
Out[ ]: array([0.0868341 , 0.04139609, 0.0273287 , 0.02266575, 0.0197299 ,
          0.01593157, 0.01440483, 0.01327614, 0.01242728, 0.01194245])
```

```
In [ ]: Lambda = round(pcs.cov(), 2)
```

Also, by construction, the columns of $Z_{(k)}$ (the PCs) are orthogonal random variables, and with sample variances $\lambda_1, \dots, \lambda_k$. Thus, the sample covariance matrix of $Z_{(k)}$ is given by the $(k \times k)$ diagonal matrix

$$\mathbf{\Lambda}_k := \text{diag}(\lambda_1, \dots, \lambda_k).$$

```
In [ ]: Lambda.head(10)[['PC 1', 'PC 2', 'PC 3', 'PC 4', 'PC 5', 'PC 6', 'PC 7', 'PC 8', 'PC 9', 'PC 10']
```

```
Out[ ]:
```

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC 10
PC 1	10.59	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00
PC 2	0.00	5.05	-0.00	-0.00	0.00	-0.00	0.00	-0.00	-0.00	0.00
PC 3	0.00	-0.00	3.33	0.00	-0.00	0.00	0.00	-0.00	-0.00	0.00
PC 4	0.00	-0.00	0.00	2.77	0.00	0.00	-0.00	0.00	-0.00	0.00
PC 5	0.00	0.00	-0.00	0.00	2.41	-0.00	0.00	0.00	0.00	0.00
PC 6	0.00	-0.00	0.00	0.00	-0.00	1.94	0.00	0.00	-0.00	-0.00
PC 7	-0.00	0.00	0.00	-0.00	0.00	0.00	1.76	0.00	0.00	0.00
PC 8	-0.00	-0.00	-0.00	0.00	0.00	0.00	0.00	1.62	-0.00	-0.00
PC 9	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	0.00	-0.00	1.52	-0.00
PC 10	0.00	0.00	0.00	0.00	0.00	-0.00	0.00	-0.00	-0.00	1.46

```
In [ ]: lambdas[:10]
```

```
Out[ ]: array([10.59291481,  5.04991983,  3.3383555,  2.76500035,  2.40685564,
          1.94349613,  1.7572486 ,  1.61955915,  1.51600711,  1.45686289])
```

Rule of Thumb

Stop at a k such that the $(k + 1)$ -th PC does not add much to the already explained variance (say $< 3\%$).

```
In [ ]: n_pc_rt = rule_thumb(alphas)
```

The first 2 PCs explain 12.82% of the returns variance.

Informal Way

Choose the number of components such that a large portion (say 90\%) of the variance is explained.

```
In [ ]: n_pc_iw = informal_way(alphas)
```

The first 207 PCs explain 89.91% of the returns variance.

Biggest Drop

Onatski (2010) suggests looking for the biggest drop computing

$$r := \arg \max_{1 \leq j < n} \frac{\lambda_j}{\lambda_{j+1}}.$$

```
In [ ]: n_pc_bd = biggest_drop(lambdas)
```

The first 1 PCs explain 8.68% of the returns variance.

(b) (30 points)

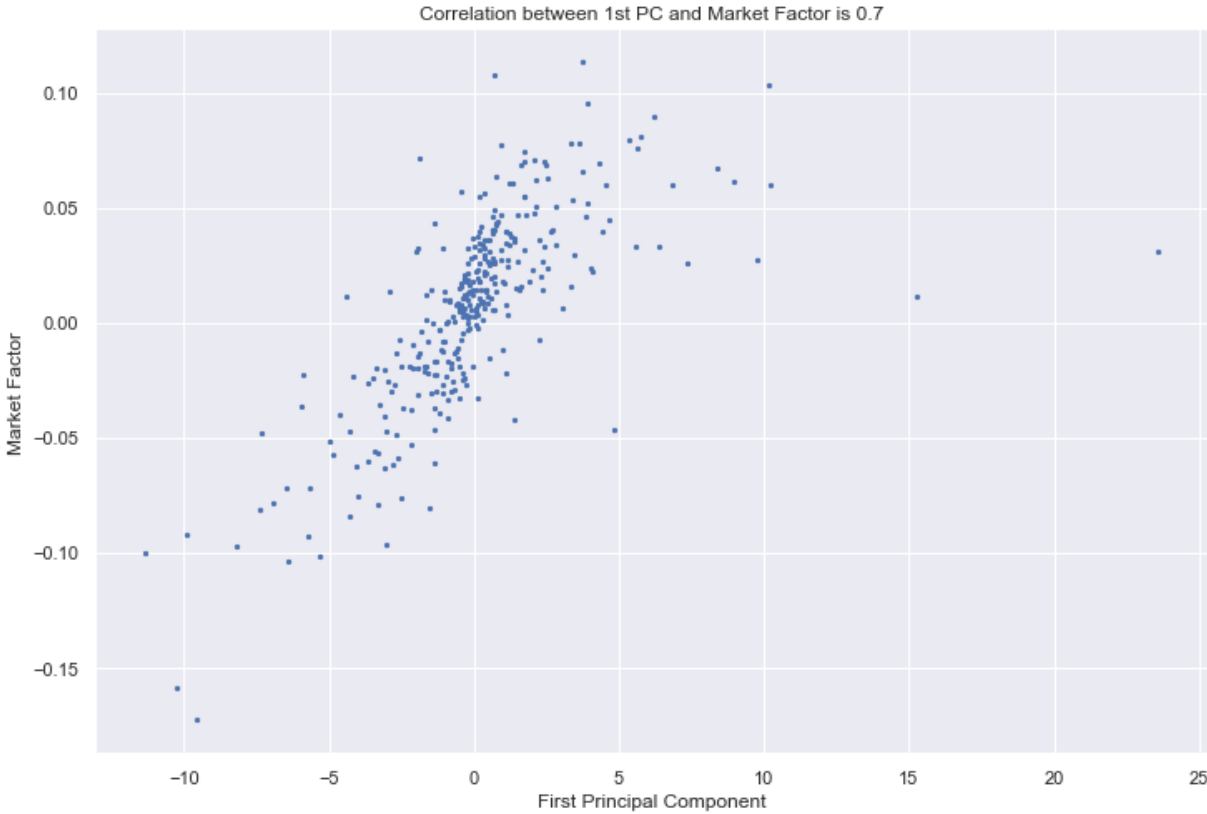
Regress the selected factors on the 16 observed "anomaly" factors described above. How do the "principal component factors" relate to the "anomaly factors"?

```
In [ ]: # rule of thumb
pc_ret_rt = pcs.iloc[:, :n_pc_rt]
pc_ret_rt.head()
```

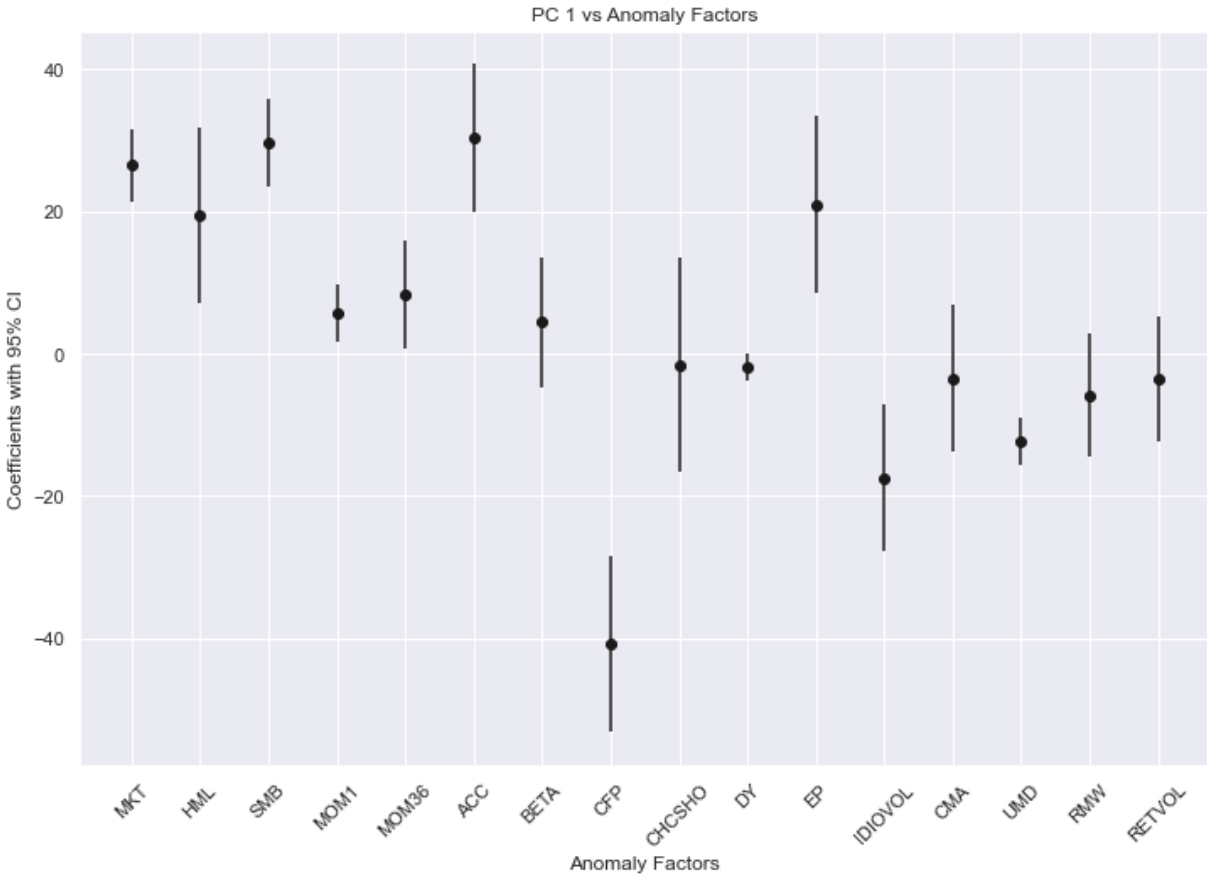
```
Out[ ]:          PC 1      PC 2
```

dates		
1991-11-29	-0.932311	0.655585
1991-12-31	0.684426	-0.766075
1992-01-31	2.223682	-0.458117
1992-02-28	0.562873	-0.140579
1992-03-31	-0.743650	0.306019

```
In [ ]: corr = round(factors['MKT'].corr(pc_ret_rt['PC 1']), 2)
plt.scatter(pc_ret_rt['PC 1'], factors['MKT'], s=5)
plt.xlabel('First Principal Component')
plt.ylabel('Market Factor')
plt.title(f'Correlation between 1st PC and Market Factor is {corr}');
```



```
In [ ]: pc1 = OLS_regression(y = pc_ret_rt, X = factors, y_column = "PC 1", is_pc = False)
```



```
In [ ]: print(pc1.summary())
```

OLS Regression Results

=====

```

Dep. Variable:          PC 1    R-squared:          0.878
Model:                OLS      Adj. R-squared:       0.872
Method:              Least Squares  F-statistic:        139.3
Date:                Sat, 15 Jul 2023  Prob (F-statistic):    1.04e-130
Time:                16:35:49    Log-Likelihood:     -503.55
No. Observations:    326      AIC:                1041.
Df Residuals:        309      BIC:                1105.
Df Model:            16
Covariance Type:      nonrobust

```

```

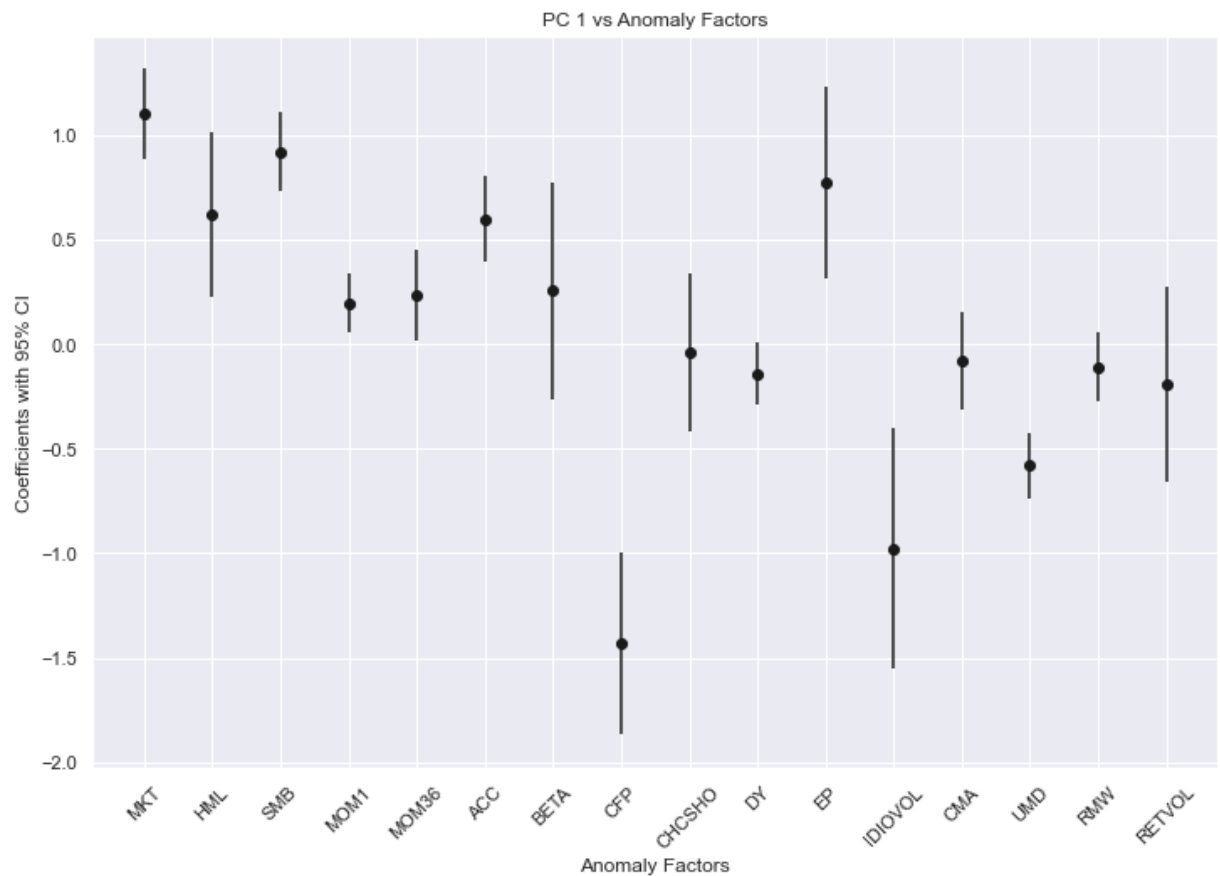
=====
              coef      std err          t      P>|t|      [0.025      0.975]
-----
const         -0.1940        0.074      -2.629      0.009      -0.339      -0.049
MKT           26.4644        2.614     10.123      0.000      21.321      31.608
HML           19.4347        6.242       3.114      0.002       7.153      31.717
SMB           29.7067        3.142       9.455      0.000      23.524      35.889
MOM1           5.6690        2.073       2.734      0.007       1.589       9.749
MOM36          8.2647        3.877       2.131      0.034       0.635      15.894
ACC           30.3074        5.272       5.748      0.000      19.933      40.682
BETA           4.4525        4.636       0.960      0.338      -4.670      13.575
CFP          -40.7554        6.249      -6.522      0.000     -53.051     -28.460
CHCSHO        -1.5690        7.657      -0.205      0.838     -16.636      13.498
DY            -1.9132        1.005      -1.903      0.058      -3.891       0.065
EP            20.9376        6.344       3.301      0.001       8.455      33.420
IDIOVOL       -17.4350        5.192      -3.358      0.001     -27.652     -7.218
CMA           -3.4515        5.189       -0.665      0.506     -13.662       6.759
UMD          -12.2663        1.692      -7.250      0.000     -15.595     -8.937
RMW           -5.8582        4.399       -1.332      0.184     -14.514       2.798
RETVOL        -3.6166        4.429       -0.817      0.415     -12.331       5.097
=====
Omnibus:                186.517    Durbin-Watson:           1.791
Prob(Omnibus):           0.000    Jarque-Bera (JB):        3153.860
Skew:                    1.974    Prob(JB):                0.00
Kurtosis:                17.717    Cond. No.                147.
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [ ]: pc1std = OLS_regression(y = pc_ret_rt, X = standardized_factors, y_column = "PC 1",
```



```
In [ ]: print(pc1std.summary())
```

OLS Regression Results

```
=====
Dep. Variable:          PC 1    R-squared:                0.878
Model:                  OLS     Adj. R-squared:            0.872
Method:                 Least Squares    F-statistic:           139.3
Date:                   Sat, 15 Jul 2023    Prob (F-statistic):    1.04e-130
Time:                   16:35:50    Log-Likelihood:       -503.55
No. Observations:       326    AIC:                  1041.
Df Residuals:           309    BIC:                  1105.
Df Model:               16
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	-1.162e-16	0.065	-1.8e-15	1.000	-0.127	0.127
MKT	1.1019	0.109	10.123	0.000	0.888	1.316
HML	0.6190	0.199	3.114	0.002	0.228	1.010
SMB	0.9205	0.097	9.455	0.000	0.729	1.112
MOM1	0.1965	0.072	2.734	0.007	0.055	0.338
MOM36	0.2357	0.111	2.131	0.034	0.018	0.453
ACC	0.5986	0.104	5.748	0.000	0.394	0.803
BETA	0.2541	0.265	0.960	0.338	-0.266	0.775
CFP	-1.4315	0.219	-6.522	0.000	-1.863	-1.000
CHCSHO	-0.0394	0.192	-0.205	0.838	-0.418	0.339
DY	-0.1430	0.075	-1.903	0.058	-0.291	0.005
EP	0.7719	0.234	3.301	0.001	0.312	1.232
IDIOVOL	-0.9774	0.291	-3.358	0.001	-1.550	-0.405
CMA	-0.0798	0.120	-0.665	0.506	-0.316	0.156
UMD	-0.5809	0.080	-7.250	0.000	-0.739	-0.423
RMW	-0.1117	0.084	-1.332	0.184	-0.277	0.053
RETVOL	-0.1943	0.238	-0.817	0.415	-0.663	0.274

```
=====
Omnibus:                186.517    Durbin-Watson:          1.791
```

```

Prob(Omnibus):          0.000   Jarque-Bera (JB):          3153.860
Skew:                  1.974   Prob(JB):              0.00
Kurtosis:              17.717   Cond. No.              14.8
=====

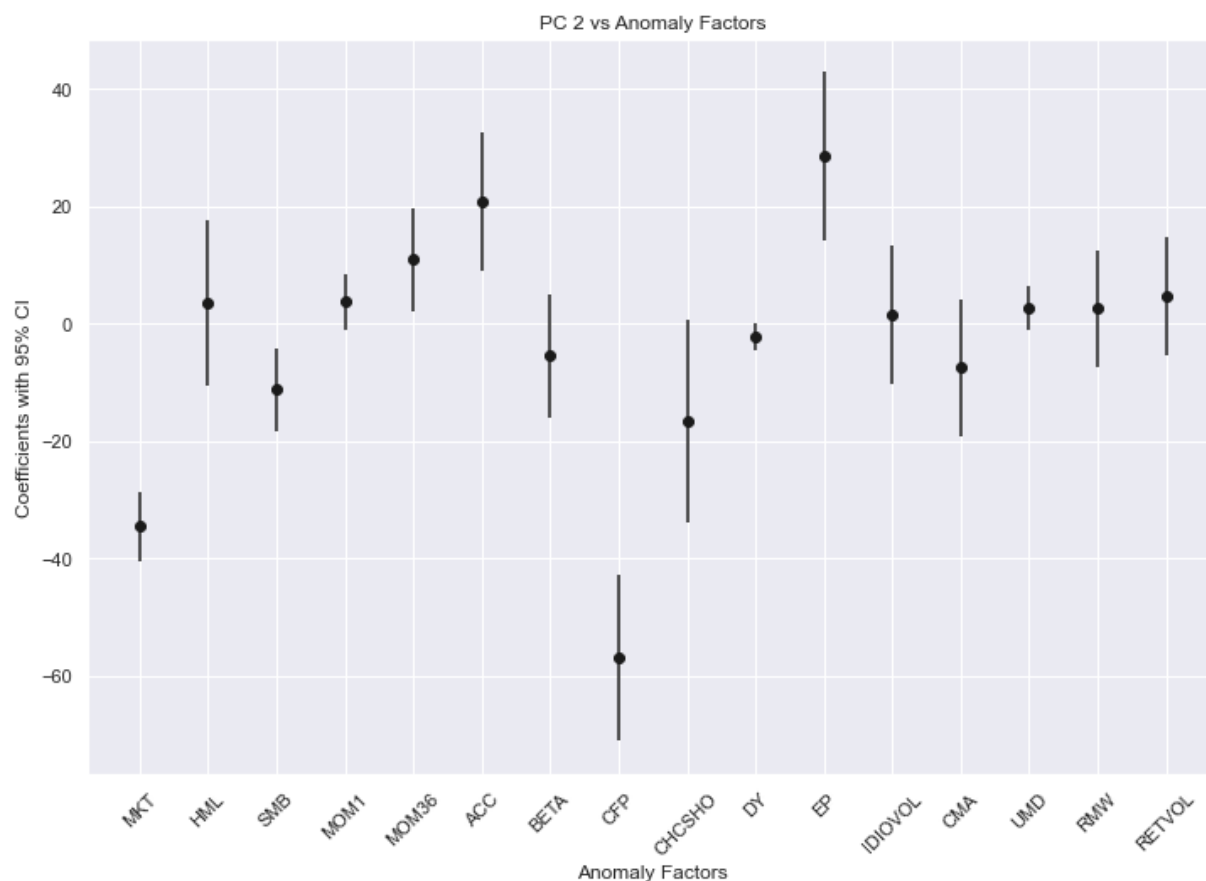
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In []:

```
pc2 = OLS_regression(y = pc_ret_rt, X = factors, y_column = "PC 2", is_pc = False)
```



In []:

```
print(pc2.summary())
```

OLS Regression Results

```

=====
Dep. Variable:          PC 2   R-squared:              0.662
Model:                  OLS   Adj. R-squared:         0.645
Method:                 Least Squares   F-statistic:           37.87
Date:                  Sat, 15 Jul 2023   Prob (F-statistic):    4.19e-63
Time:                  16:35:50   Log-Likelihood:        -549.10
No. Observations:      326   AIC:                   1132.
Df Residuals:          309   BIC:                   1197.
Df Model:              16
Covariance Type:       nonrobust
=====

```

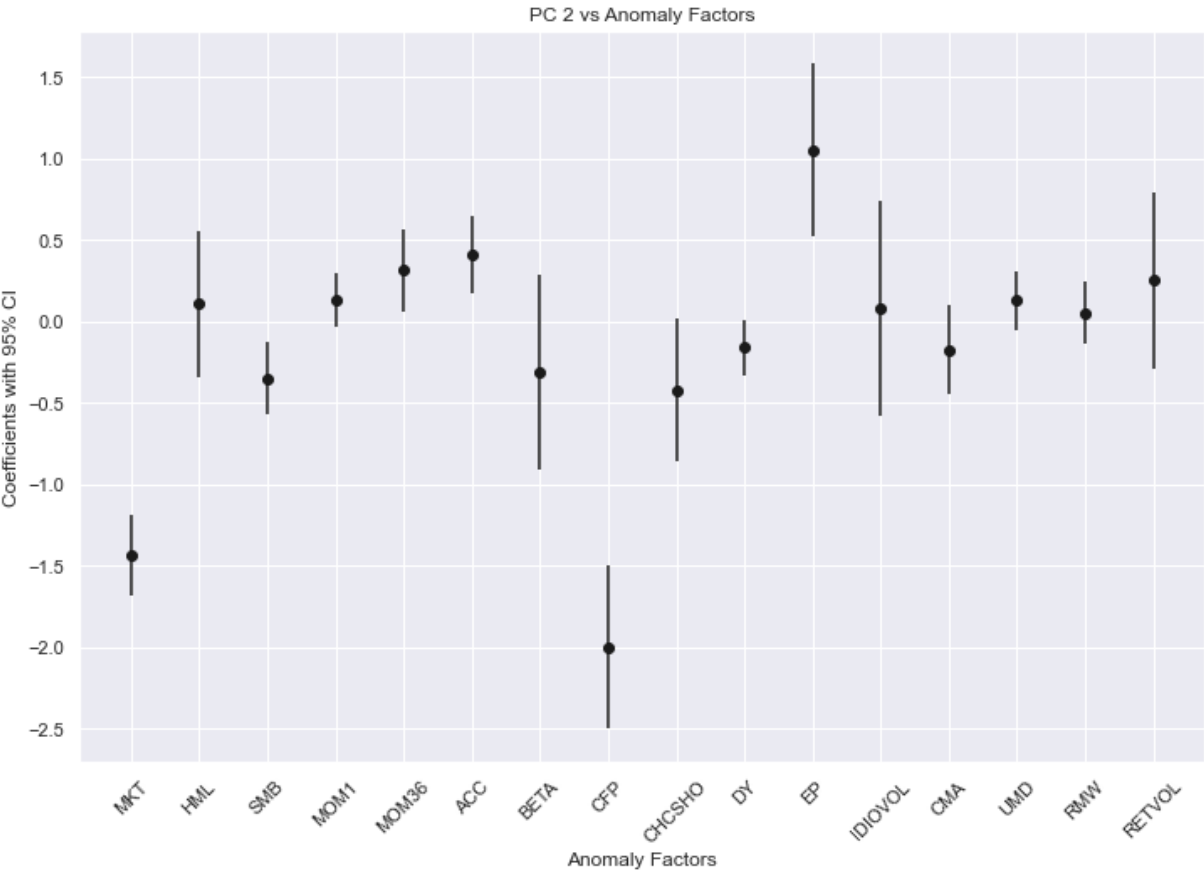
	coef	std err	t	P> t	[0.025	0.975]
const	0.3155	0.085	3.718	0.000	0.149	0.482
MKT	-34.5536	3.006	-11.494	0.000	-40.469	-28.638
HML	3.3906	7.178	0.472	0.637	-10.733	17.514
SMB	-11.3074	3.613	-3.130	0.002	-18.417	-4.198
MOM1	3.7126	2.384	1.557	0.120	-0.979	8.404
MOM36	10.9439	4.459	2.454	0.015	2.170	19.718
ACC	20.8124	6.063	3.433	0.001	8.883	32.742

BETA	-5.5253	5.331	-1.036	0.301	-16.015	4.965
CFP	-56.8628	7.186	-7.913	0.000	-71.002	-42.724
CHCSHO	-16.7244	8.805	-1.899	0.058	-34.050	0.601
DY	-2.1662	1.156	-1.874	0.062	-4.440	0.108
EP	28.5162	7.295	3.909	0.000	14.162	42.870
IDIOVOL	1.4526	5.971	0.243	0.808	-10.296	13.201
CMA	-7.5439	5.967	-1.264	0.207	-19.286	4.198
UMD	2.6894	1.946	1.382	0.168	-1.139	6.518
RMW	2.6344	5.059	0.521	0.603	-7.320	12.589
RETVOL	4.6764	5.093	0.918	0.359	-5.344	14.697
=====						
Omnibus:	257.183		Durbin-Watson:		1.685	
Prob(Omnibus):	0.000		Jarque-Bera (JB):		11439.574	
Skew:	2.753		Prob(JB):		0.00	
Kurtosis:	31.493		Cond. No.		147.	
=====						

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In []:

pc2std = OLS_regression(y = pc_ret_rt, X = standardized_factors, y_column = "PC 2",



In []:

print(pc2std.summary())

OLS Regression Results			
=====			
Dep. Variable:	PC 2	R-squared:	0.662
Model:	OLS	Adj. R-squared:	0.645
Method:	Least Squares	F-statistic:	37.87
Date:	Sat, 15 Jul 2023	Prob (F-statistic):	4.19e-63
Time:	16:35:50	Log-Likelihood:	-549.10
No. Observations:	326	AIC:	1132.
Df Residuals:	309	BIC:	1197.

Df Model: 16
Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	4.684e-17	0.074	6.31e-16	1.000	-0.146	0.146
MKT	-1.4387	0.125	-11.494	0.000	-1.685	-1.192
HML	0.1080	0.229	0.472	0.637	-0.342	0.558
SMB	-0.3504	0.112	-3.130	0.002	-0.571	-0.130
MOM1	0.1287	0.083	1.557	0.120	-0.034	0.291
MOM36	0.3122	0.127	2.454	0.015	0.062	0.562
ACC	0.4110	0.120	3.433	0.001	0.175	0.647
BETA	-0.3153	0.304	-1.036	0.301	-0.914	0.283
CFP	-1.9972	0.252	-7.913	0.000	-2.494	-1.501
CHCSHO	-0.4201	0.221	-1.899	0.058	-0.855	0.015
DY	-0.1619	0.086	-1.874	0.062	-0.332	0.008
EP	1.0513	0.269	3.909	0.000	0.522	1.581
IDIOVOL	0.0814	0.335	0.243	0.808	-0.577	0.740
CMA	-0.1744	0.138	-1.264	0.207	-0.446	0.097
UMD	0.1274	0.092	1.382	0.168	-0.054	0.309
RMW	0.0502	0.096	0.521	0.603	-0.140	0.240
RETVOL	0.2513	0.274	0.918	0.359	-0.287	0.790
=====						
Omnibus:		257.183	Durbin-Watson:		1.685	
Prob(Omnibus):		0.000	Jarque-Bera (JB):		11439.574	
Skew:		2.753	Prob(JB):		0.00	
Kurtosis:		31.493	Cond. No.		14.8	

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In []:

```
corr_matrix = round(pd.concat([pc_ret_rt, factors], axis=1).corr(), 2)
corr_matrix[factors.columns][:2]
```

Out[]:

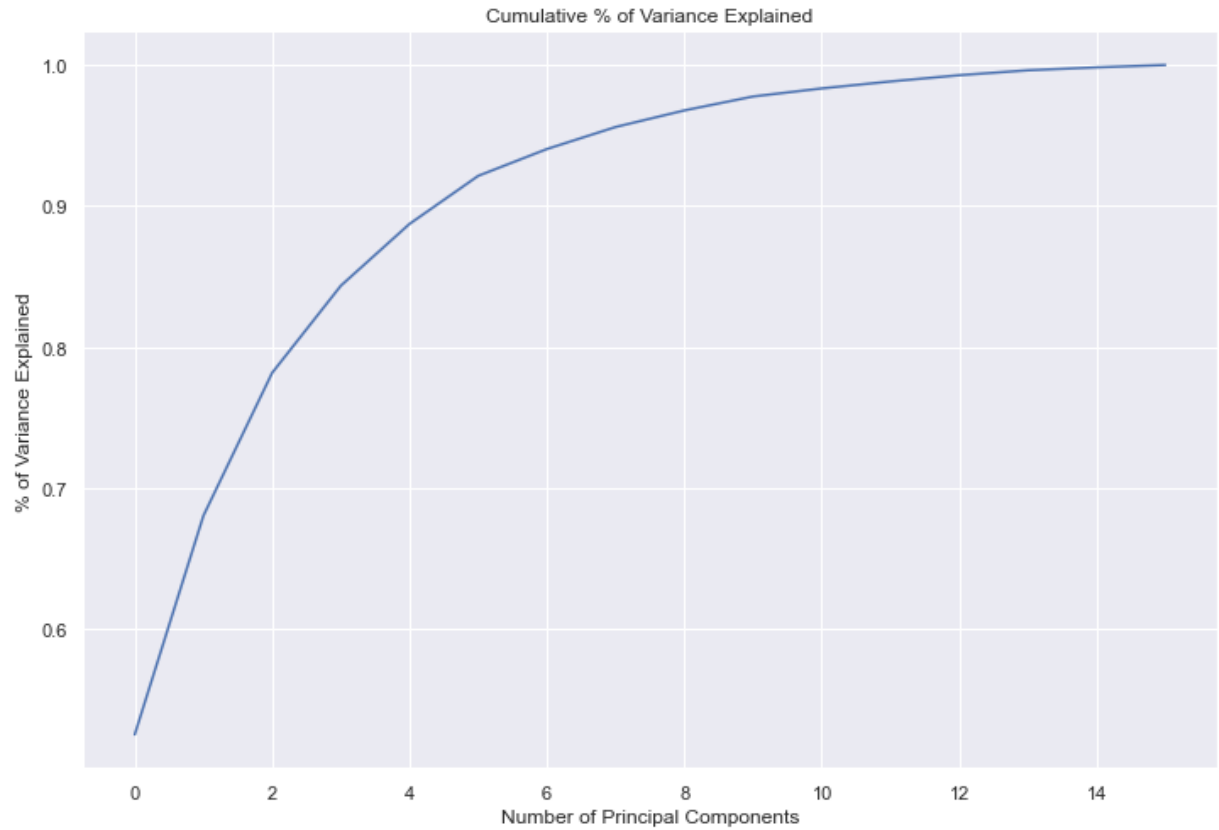
	MKT	HML	SMB	MOM1	MOM36	ACC	BETA	CFP	CHCSHO	DY	EP	IDIOVOL	CMA
PC													
1	0.70	-0.38	0.70	0.31	0.16	0.40	-0.87	-0.54	-0.73	-0.39	-0.69	-0.85	-0.36
PC													
2	-0.47	-0.46	-0.03	-0.11	-0.19	0.21	0.11	-0.47	-0.15	-0.12	-0.24	0.00	-0.24

(c) (30 points)

Now, run a principal component analysis on the 16 "anomaly factors" and select the optimal number of principal components using the same criterion adopted in the first item of the exercise. By inspecting the principal eigenvectors can you identify a dominating "anomaly"?

In []:

```
pcs, gammas, lambdas, alphas = PCA_function(factors)
```

The transformed dataset containing only the first k PCs is the $(T \times k)$ matrix is given by

$$\begin{aligned} Z_{(k)} &:= \mathbf{X}\mathbf{\Gamma}_k \\ &:= (Z_1, \dots, Z_k). \end{aligned}$$

In []:

pcs

Out []:

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC
dates									
1991-11-29	-0.040773	-0.040907	-0.034789	-0.042608	0.021649	0.016281	-0.002945	0.013785	-0.017115
1991-12-31	0.112512	-0.012316	-0.038009	0.025986	-0.097625	-0.067016	0.003296	0.040766	-0.00733
1992-01-31	0.121870	0.061630	0.066577	0.094953	0.039049	0.094894	-0.019855	-0.018094	0.02758
1992-02-28	-0.005106	0.043395	0.047053	0.084682	0.035254	-0.009129	0.001453	-0.009655	0.01877
1992-03-31	-0.108007	0.012929	0.026053	0.021665	0.019578	0.005955	0.008637	-0.012087	-0.00738
...
2018-08-31	0.070953	-0.020014	-0.064739	-0.014552	-0.031384	0.001708	0.000216	0.006447	0.03309
2018-09-28	0.006513	-0.057196	0.004248	-0.013213	0.005573	-0.003490	0.028139	-0.015625	0.00580
2018-10-31	-0.167718	-0.033319	0.045695	-0.010240	0.027620	0.030816	0.020208	-0.008359	-0.03652

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC
dates									
2018-11-30	-0.058530	0.037814	-0.010477	-0.049877	-0.016665	-0.006877	0.009479	0.020282	0.00273
2018-12-31	-0.101082	0.010640	-0.083628	-0.054505	0.049028	0.031435	0.002180	-0.014540	-0.03352

326 rows × 16 columns

Given the desired number of PCs, say $1 \leq k \leq n$, we collect all the vectors $\gamma_1, \dots, \gamma_k$ in a $(n \times k)$ matrix

$$\Gamma_k := (\gamma_1, \dots, \gamma_k)$$

In []:

gammas

Out[]:

	gamma 1	gamma 2	gamma 3	gamma 4	gamma 5	gamma 6	gamma 7	gamma 8	gamma 9
MKT	0.228449	0.191721	0.189615	0.091736	-0.553227	-0.518328	0.175670	0.206177	0.36
HML	-0.154749	0.062009	0.342674	0.383140	-0.006026	0.005076	-0.142897	-0.089115	-0.18
SMB	0.168694	0.058650	0.028389	0.137712	0.194181	0.452578	-0.387099	0.476201	0.35
MOM1	0.066700	0.065963	0.211633	-0.201040	-0.644130	0.663915	0.070393	-0.177102	-0.05
MOM36	0.003330	0.127738	0.096021	0.497916	0.209302	0.227409	0.441680	-0.126307	0.40
ACC	0.081187	0.003743	-0.132634	-0.012960	0.050161	0.014296	-0.020514	-0.583009	0.32
BETA	-0.447878	-0.218960	-0.081117	-0.134744	0.020983	0.067217	0.129617	0.021959	0.16
CFP	-0.204639	0.012831	0.344791	0.252459	-0.054861	-0.105223	-0.433638	-0.350681	0.13
CHCSHO	-0.182823	-0.029610	0.115116	0.107098	0.027947	0.004601	0.053857	0.082061	0.13
DY	-0.388018	0.862257	-0.301306	-0.070640	-0.025850	0.012490	-0.094100	0.008833	-0.00
EP	-0.256139	-0.056180	0.290519	0.065395	-0.105811	-0.090116	-0.339623	0.191388	-0.12
IDIOVOL	-0.447106	-0.141603	0.120198	-0.128713	-0.067057	0.024197	0.211033	0.162011	0.03
CMA	-0.105896	0.053198	0.110473	0.264680	-0.007434	0.081863	0.393200	0.296365	-0.20
UMD	-0.081444	-0.254533	-0.653139	0.526627	-0.407090	0.031903	-0.176837	0.052899	-0.06
RMW	0.011606	-0.055604	-0.054078	-0.243840	-0.045945	-0.031349	-0.182397	0.216336	0.46
RETVOL	-0.417622	-0.221104	-0.064418	-0.106175	-0.080566	-0.024605	0.052871	-0.070598	0.28

Also, by construction, the columns of $Z_{(k)}$ (the PCs) are orthogonal random variables, and with sample variances $\lambda_1, \dots, \lambda_k$.

Thus, the sample covariance matrix of $Z_{(k)}$ is given by the $(k \times k)$ diagonal matrix

$$\Lambda_k := \text{diag}(\lambda_1, \dots, \lambda_k)$$

```
In [ ]: lambdas[:10]
```

```
Out[ ]: array([0.0143643 , 0.00424359, 0.00277031, 0.00168956, 0.00119888,
              0.00093209, 0.00052007, 0.00042722, 0.00032125, 0.00027015])
```

It is a good idea to start by running a full PCA ($k = n$) and plotting the quantity

$$\alpha_j = \frac{\lambda_j}{\sum_{j=1}^n \lambda_j}$$

for $j \in \{1, \dots, n\}$.

```
In [ ]: alphas[:10]
```

```
Out[ ]: array([0.52522946, 0.15516658, 0.10129623, 0.06177849, 0.04383691,
              0.03408185, 0.01901649, 0.01562144, 0.01174643, 0.00987803])
```

```
In [ ]: Lambda = round(pcs.cov(), 4)
```

Also, by construction, the columns of $Z_{(k)}$ (the PCs) are orthogonal random variables, and with sample variances $\lambda_1, \dots, \lambda_k$. Thus, the sample covariance matrix of $Z_{(k)}$ is given by the $(k \times k)$ diagonal matrix

$$\Lambda_k := \text{diag}(\lambda_1, \dots, \lambda_k).$$

```
In [ ]: Lambda.head(10)[['PC 1', 'PC 2', 'PC 3', 'PC 4', 'PC 5', 'PC 6', 'PC 7', 'PC 8', 'PC 9', 'PC 10']
```

```
Out[ ]:
```

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC 10
PC 1	0.0144	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000	0.0000	0.0000
PC 2	0.0000	0.0042	0.0000	0.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000	-0.0000
PC 3	0.0000	0.0000	0.0028	-0.0000	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	-0.0000
PC 4	0.0000	0.0000	-0.0000	0.0017	0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000
PC 5	-0.0000	-0.0000	0.0000	0.0000	0.0012	-0.0000	0.0000	-0.0000	-0.0000	-0.0000
PC 6	-0.0000	0.0000	-0.0000	0.0000	-0.0000	0.0009	0.0000	0.0000	0.0000	0.0000
PC 7	0.0000	-0.0000	-0.0000	-0.0000	0.0000	0.0000	0.0005	0.0000	-0.0000	-0.0000
PC 8	0.0000	0.0000	-0.0000	-0.0000	-0.0000	0.0000	0.0000	0.0004	0.0000	0.0000
PC 9	0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000	0.0000	0.0003	0.0000
PC 10	0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000	0.0000	0.0000	0.0003

```
In [ ]: lambdas[:10]
```

```
Out[ ]: array([0.0143643 , 0.00424359, 0.00277031, 0.00168956, 0.00119888,
              0.00093209, 0.00052007, 0.00042722, 0.00032125, 0.00027015])
```

Rule of Thumb

Stop at a k such that the $(k + 1)$ -th PC does not add much to the already explained variance (say $< 3\%$).

```
In [ ]: n_pc_rt = rule_thumb(alphas)
```

The first 6 PCs explain 92.14% of the returns variance.

Informal Way

Choose the number of components such that a large portion (say 90\%) of the variance is explained.

```
In [ ]: n_pc_iw = informal_way(alphas)
```

The first 5 PCs explain 88.73% of the returns variance.

Biggest Drop

Onatski (2010) suggests looking for the biggest drop computing

$$r := \arg \max_{1 \leq j < n} \frac{\lambda_j}{\lambda_{j+1}}.$$

```
In [ ]: n_pc_bd = biggest_drop(lambdas)
```

The first 1 PCs explain 52.52% of the returns variance.

```
In [ ]: # rule of thumb
pc_fac_rt = pcs.iloc[:, :n_pc_rt]
pc_fac_rt.head()
```

```
Out[ ]:
```

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6
dates						
1991-11-29	-0.040773	-0.040907	-0.034789	-0.042608	0.021649	0.016281
1991-12-31	0.112512	-0.012316	-0.038009	0.025986	-0.097625	-0.067016
1992-01-31	0.121870	0.061630	0.066577	0.094953	0.039049	0.094894
1992-02-28	-0.005106	0.043395	0.047053	0.084682	0.035254	-0.009129
1992-03-31	-0.108007	0.012929	0.026053	0.021665	0.019578	0.005955

By inspecting the principal eigenvectors can you identify a dominating anomaly?

```
In [ ]: gammas
```

```
Out[ ]:
```

	gamma 1	gamma 2	gamma 3	gamma 4	gamma 5	gamma 6	gamma 7	gamma 8	gamma 9
MKT	0.228449	0.191721	0.189615	0.091736	-0.553227	-0.518328	0.175670	0.206177	0.36
HML	-0.154749	0.062009	0.342674	0.383140	-0.006026	0.005076	-0.142897	-0.089115	-0.18
SMB	0.168694	0.058650	0.028389	0.137712	0.194181	0.452578	-0.387099	0.476201	0.35
MOM1	0.066700	0.065963	0.211633	-0.201040	-0.644130	0.663915	0.070393	-0.177102	-0.05
MOM36	0.003330	0.127738	0.096021	0.497916	0.209302	0.227409	0.441680	-0.126307	0.40

	gamma 1	gamma 2	gamma 3	gamma 4	gamma 5	gamma 6	gamma 7	gamma 8	gamma 9
ACC	0.081187	0.003743	-0.132634	-0.012960	0.050161	0.014296	-0.020514	-0.583009	0.32
BETA	-0.447878	-0.218960	-0.081117	-0.134744	0.020983	0.067217	0.129617	0.021959	0.16
CFP	-0.204639	0.012831	0.344791	0.252459	-0.054861	-0.105223	-0.433638	-0.350681	0.13
CHCSHO	-0.182823	-0.029610	0.115116	0.107098	0.027947	0.004601	0.053857	0.082061	0.13
DY	-0.388018	0.862257	-0.301306	-0.070640	-0.025850	0.012490	-0.094100	0.008833	-0.00
EP	-0.256139	-0.056180	0.290519	0.065395	-0.105811	-0.090116	-0.339623	0.191388	-0.12
IDIOVOL	-0.447106	-0.141603	0.120198	-0.128713	-0.067057	0.024197	0.211033	0.162011	0.03
CMA	-0.105896	0.053198	0.110473	0.264680	-0.007434	0.081863	0.393200	0.296365	-0.20
UMD	-0.081444	-0.254533	-0.653139	0.526627	-0.407090	0.031903	-0.176837	0.052899	-0.06
RMW	0.011606	-0.055604	-0.054078	-0.243840	-0.045945	-0.031349	-0.182397	0.216336	0.46
RETVOL	-0.417622	-0.221104	-0.064418	-0.106175	-0.080566	-0.024605	0.052871	-0.070598	0.28

```
In [ ]: sum(round(centered_factors.dot(np.array(gammas["gamma 1"])), 10) == round(pcs["PC 1"]
```

```
Out[ ]: 326
```

```
In [ ]: fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(20, 26))

aux1 = gammas["gamma 1"].abs().sort_values(ascending=False)
ax[0,0].bar(aux1.index, aux1.values, color='red')
ax[0,0].tick_params(axis='x', rotation=45)
ax[0,0].set_xlabel('Anomaly Factors')
ax[0,0].set_ylabel('Loadings ($\gamma$)')
ax[0,0].set_title(f'Loadings ($\gamma$) of the 1st PC');

aux2 = gammas["gamma 2"].abs().sort_values(ascending=False)
ax[0,1].bar(aux2.index, aux2.values, color='orange')
ax[0,1].tick_params(axis='x', rotation=45)
ax[0,1].set_xlabel('Anomaly Factors')
ax[0,1].set_ylabel('Loadings ($\gamma$)')
ax[0,1].set_title(f'Loadings ($\gamma$) of the 2nd PC');

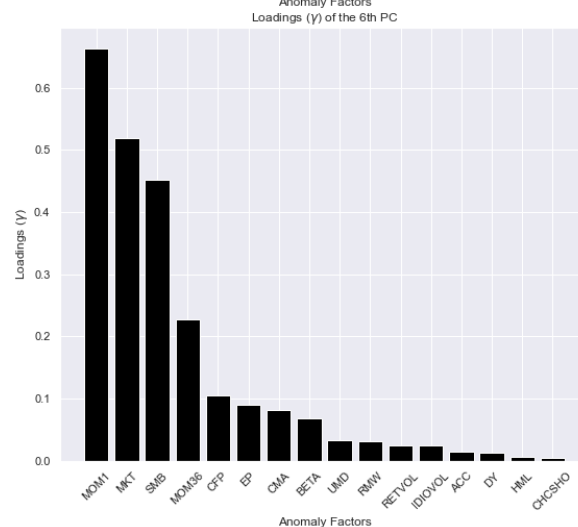
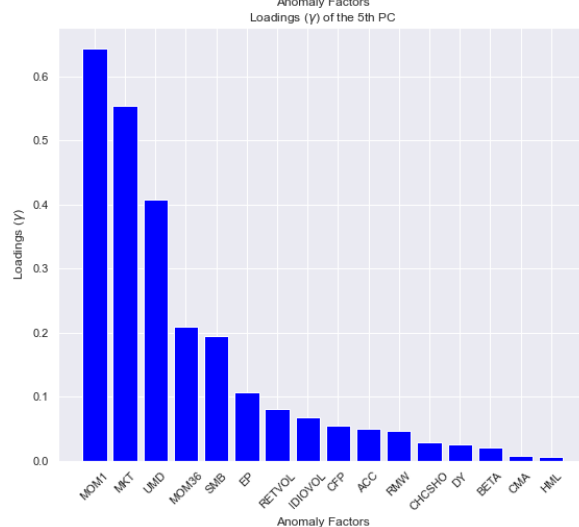
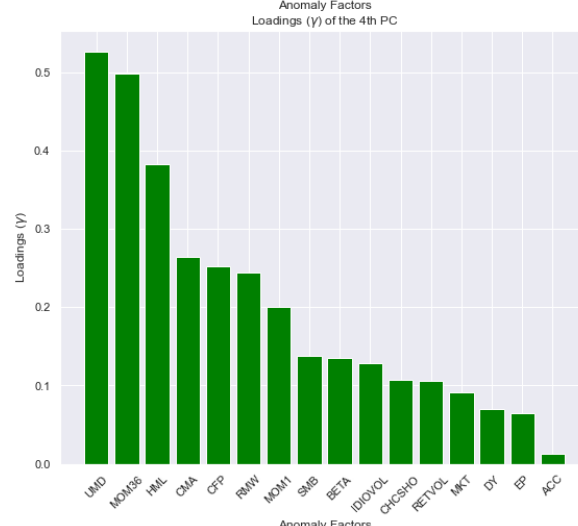
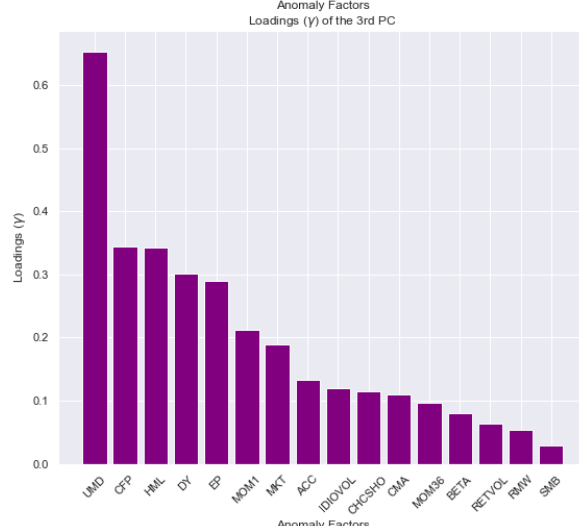
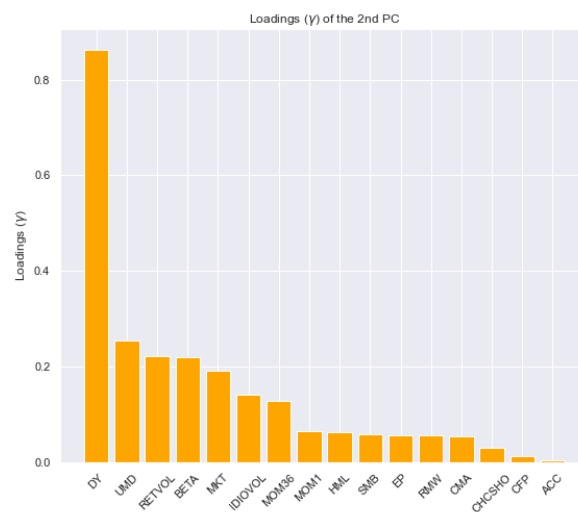
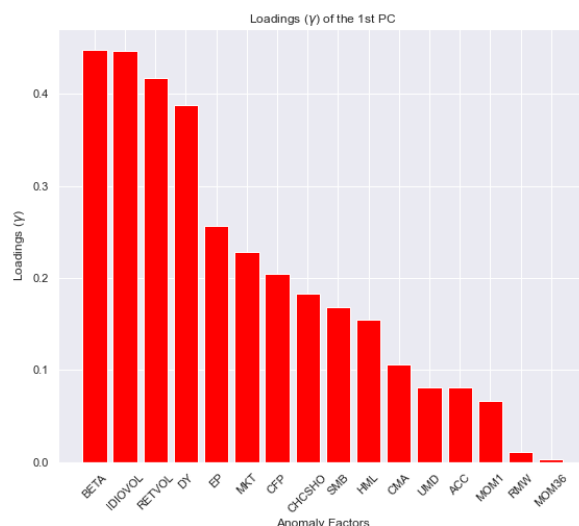
aux3 = gammas["gamma 3"].abs().sort_values(ascending=False)
ax[1,0].bar(aux3.index, aux3.values, color='purple')
ax[1,0].tick_params(axis='x', rotation=45)
ax[1,0].set_xlabel('Anomaly Factors')
ax[1,0].set_ylabel('Loadings ($\gamma$)')
ax[1,0].set_title(f'Loadings ($\gamma$) of the 3rd PC');

aux4 = gammas["gamma 4"].abs().sort_values(ascending=False)
ax[1,1].bar(aux4.index, aux4.values, color='green')
ax[1,1].tick_params(axis='x', rotation=45)
ax[1,1].set_xlabel('Anomaly Factors')
ax[1,1].set_ylabel('Loadings ($\gamma$)')
ax[1,1].set_title(f'Loadings ($\gamma$) of the 4th PC');

aux5 = gammas["gamma 5"].abs().sort_values(ascending=False)
ax[2,0].bar(aux5.index, aux5.values, color='blue')
ax[2,0].tick_params(axis='x', rotation=45)
ax[2,0].set_xlabel('Anomaly Factors')
```

```
ax[2,0].set_ylabel('Loadings ($\gamma$)')
ax[2,0].set_title(f'Loadings ($\gamma$) of the 5th PC');

aux6 = gammas["gamma 6"].abs().sort_values(ascending=False)
ax[2,1].bar(aux6.index, aux6.values, color='black')
ax[2,1].tick_params(axis='x', rotation=45)
ax[2,1].set_xlabel('Anomaly Factors')
ax[2,1].set_ylabel('Loadings ($\gamma$)')
ax[2,1].set_title(f'Loadings ($\gamma$) of the 6th PC');
```



```
In [ ]: gammas["gamma 1"]
```

```
Out[ ]: MKT      0.228449
        HML     -0.154749
```

```

SMB      0.168694
MOM1     0.066700
MOM36    0.003330
ACC       0.081187
BETA     -0.447878
CFP      -0.204639
CHCSHO   -0.182823
DY       -0.388018
EP       -0.256139
IDIOVOL  -0.447106
CMA      -0.105896
UMD      -0.081444
RMW      0.011606
RETVOL   -0.417622
Name: gamma 1, dtype: float64

```

```
In [ ]: (gammas["gamma 1"].abs()).nlargest(2)
```

```

Out[ ]: BETA      0.447878
        IDIOVOL   0.447106
        Name: gamma 1, dtype: float64

```

```
In [ ]: (gammas["gamma 1"].abs()).nsmallest(2)
```

```

Out[ ]: MOM36    0.003330
        RMW      0.011606
        Name: gamma 1, dtype: float64

```

```

In [ ]: fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(18, 12))

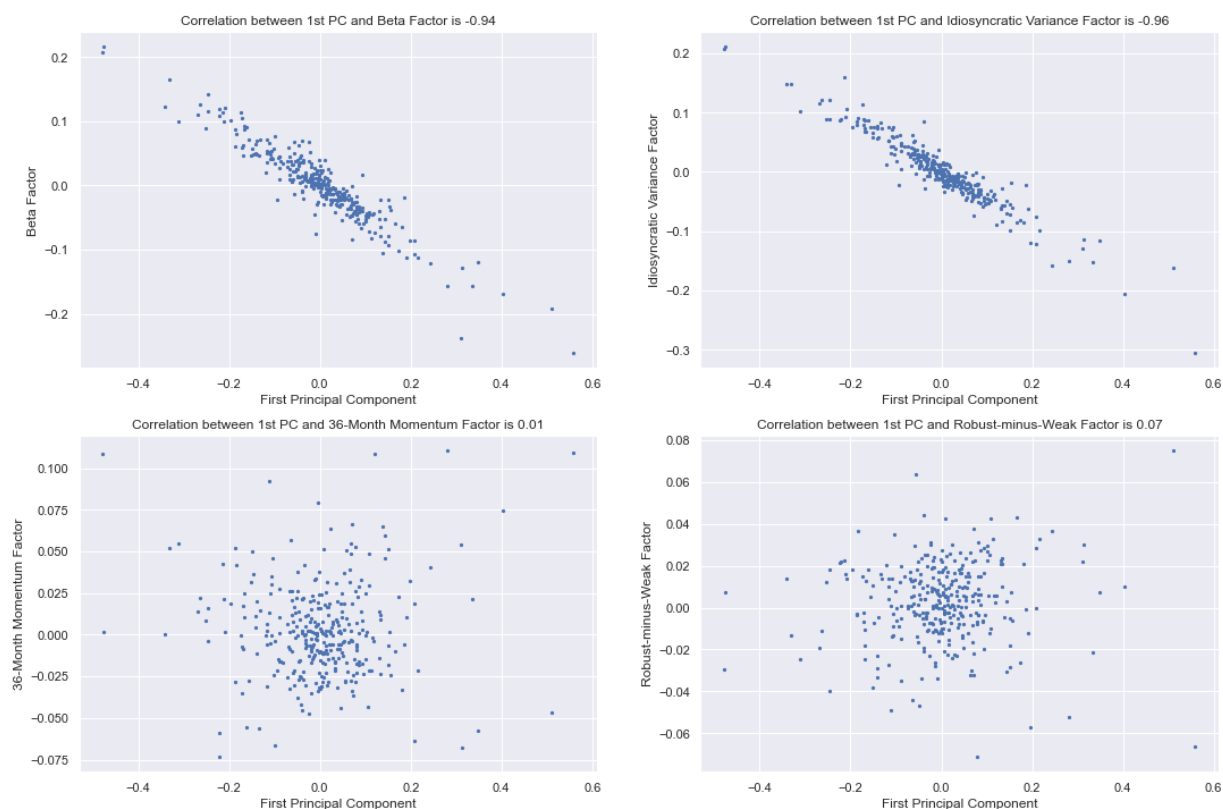
corr_1 = round(factors['BETA'].corr(pc_fac_rt['PC 1']), 2)
ax[0,0].scatter(pc_fac_rt['PC 1'], factors['BETA'], s=5)
ax[0,0].set_xlabel('First Principal Component')
ax[0,0].set_ylabel('Beta Factor')
ax[0,0].set_title(f'Correlation between 1st PC and Beta Factor is {corr_1}');

corr_2 = round(factors['IDIOVOL'].corr(pc_fac_rt['PC 1']), 2)
ax[0,1].scatter(pc_fac_rt['PC 1'], factors['IDIOVOL'], s=5)
ax[0,1].set_xlabel('First Principal Component')
ax[0,1].set_ylabel('Idiosyncratic Variance Factor')
ax[0,1].set_title(f'Correlation between 1st PC and Idiosyncratic Variance Factor is {corr_2}');

corr_3 = round(factors['MOM36'].corr(pc_fac_rt['PC 1']), 2)
ax[1,0].scatter(pc_fac_rt['PC 1'], factors['MOM36'], s=5)
ax[1,0].set_xlabel('First Principal Component')
ax[1,0].set_ylabel('36-Month Momentum Factor')
ax[1,0].set_title(f'Correlation between 1st PC and 36-Month Momentum Factor is {corr_3}');

corr_4 = round(factors['RMW'].corr(pc_fac_rt['PC 1']), 2)
ax[1,1].scatter(pc_fac_rt['PC 1'], factors['RMW'], s=5)
ax[1,1].set_xlabel('First Principal Component')
ax[1,1].set_ylabel('Robust-minus-Weak Factor')
ax[1,1].set_title(f'Correlation between 1st PC and Robust-minus-Weak Factor is {corr_4}');

```



(d) (30 points)

How do the "anomaly-based principal factors" related to the "return-based principal factors"?

```
In [ ]: pc_ret_rt.head()
```

```
Out[ ]:
```

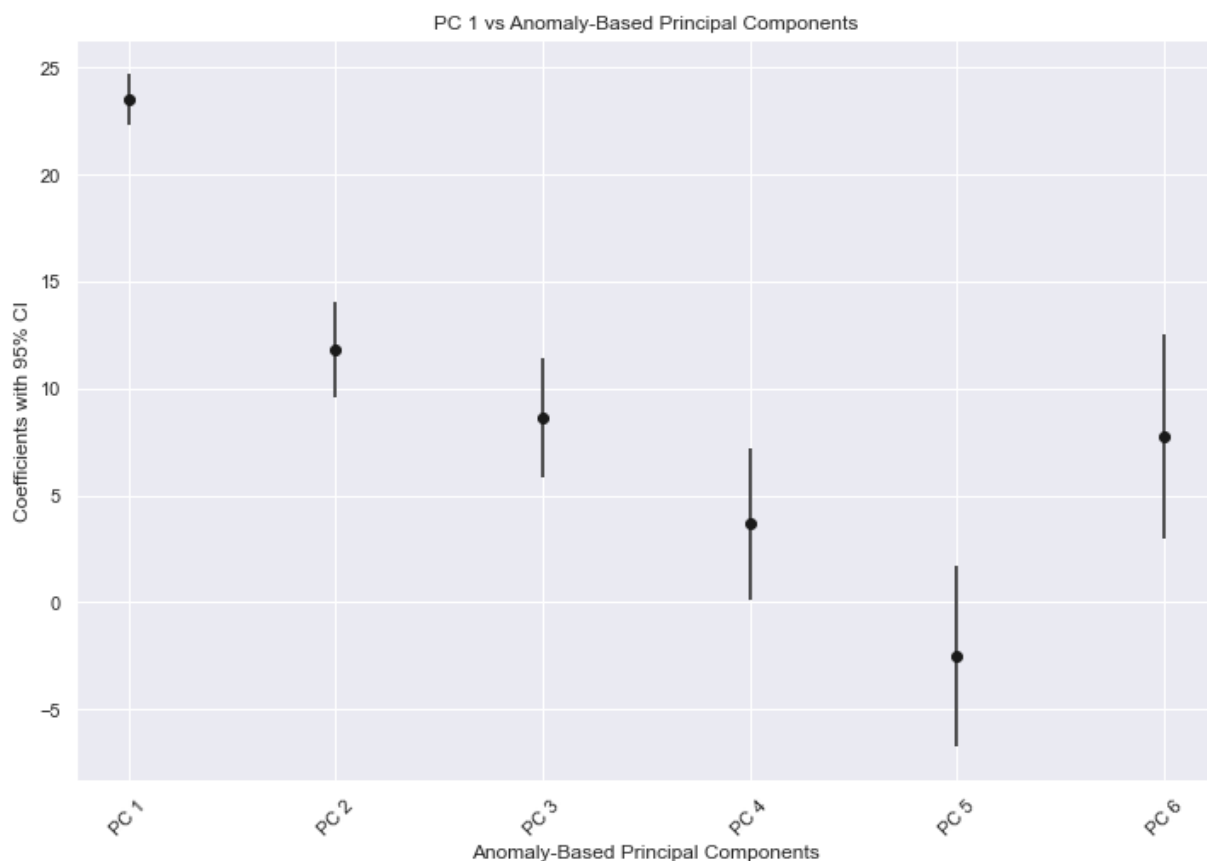
	PC 1	PC 2
dates		
1991-11-29	-0.932311	0.655585
1991-12-31	0.684426	-0.766075
1992-01-31	2.223682	-0.458117
1992-02-28	0.562873	-0.140579
1992-03-31	-0.743650	0.306019

```
In [ ]: pc_fac_rt.head()
```

```
Out[ ]:
```

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6
dates						
1991-11-29	-0.040773	-0.040907	-0.034789	-0.042608	0.021649	0.016281
1991-12-31	0.112512	-0.012316	-0.038009	0.025986	-0.097625	-0.067016
1992-01-31	0.121870	0.061630	0.066577	0.094953	0.039049	0.094894
1992-02-28	-0.005106	0.043395	0.047053	0.084682	0.035254	-0.009129
1992-03-31	-0.108007	0.012929	0.026053	0.021665	0.019578	0.005955


```
In [ ]: pc1 = OLS_regression(y = pc_ret_rt, X = pc_fac_rt, y_column = "PC 1", is_pc = True)
```



```
In [ ]: print(pc1.summary())
```

OLS Regression Results

```
=====
Dep. Variable:          PC 1    R-squared:                0.834
Model:                  OLS     Adj. R-squared:           0.831
Method:                 Least Squares    F-statistic:            268.0
Date:                   Sat, 15 Jul 2023  Prob (F-statistic):    2.33e-121
Time:                   17:02:03    Log-Likelihood:         -553.61
No. Observations:       326    AIC:                    1121.
Df Residuals:           319    BIC:                    1148.
Df Model:                6
Covariance Type:        nonrobust
=====
```

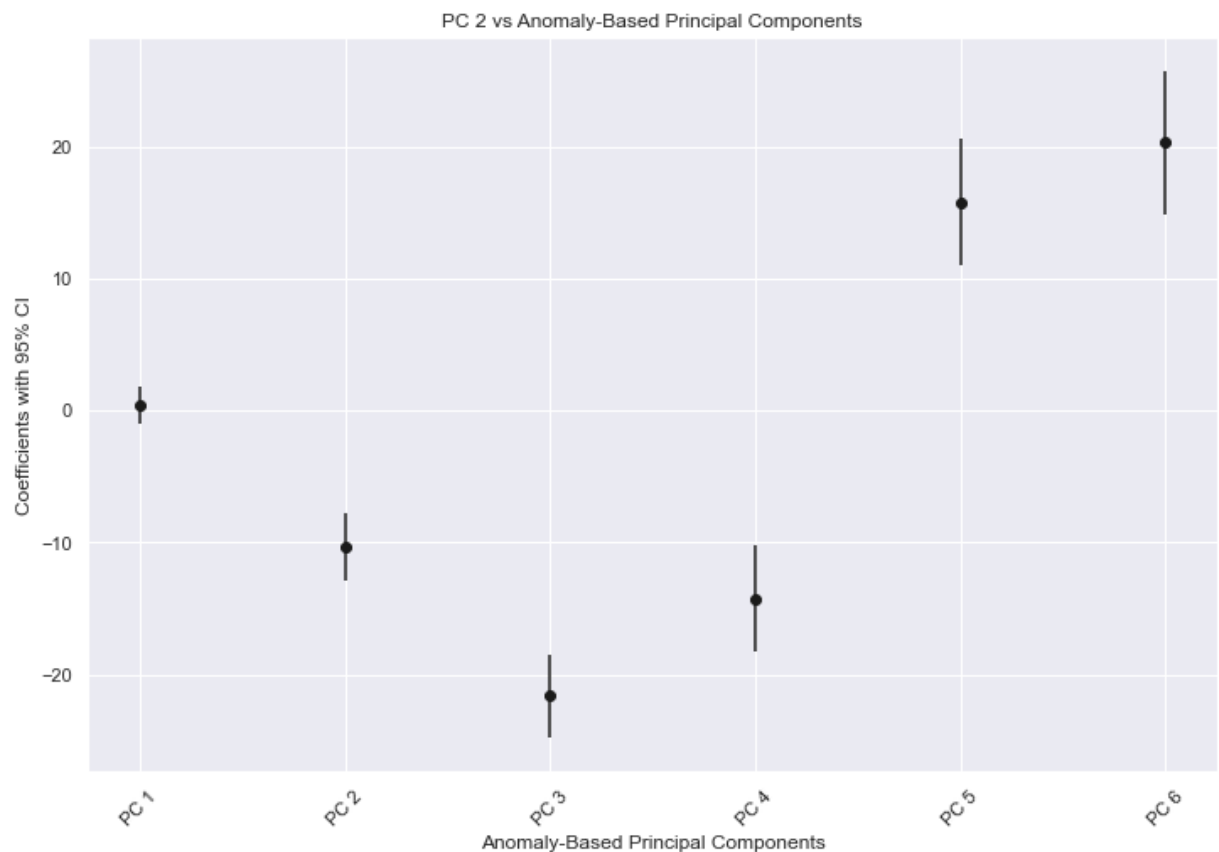
	coef	std err	t	P> t	[0.025	0.975]
const	-1.162e-16	0.074	-1.57e-15	1.000	-0.146	0.146
PC 1	23.5303	0.619	38.038	0.000	22.313	24.747
PC 2	11.8198	1.138	10.385	0.000	9.581	14.059
PC 3	8.6444	1.409	6.137	0.000	5.873	11.416
PC 4	3.6781	1.804	2.039	0.042	0.129	7.227
PC 5	-2.5226	2.141	-1.178	0.240	-6.735	1.690
PC 6	7.7491	2.428	3.191	0.002	2.971	12.527

```
=====
Omnibus:                171.684    Durbin-Watson:           1.764
Prob(Omnibus):           0.000    Jarque-Bera (JB):        2422.522
Skew:                    1.820    Prob(JB):                 0.00
Kurtosis:                15.849    Cond. No.                 32.8
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [ ]: pc2 = OLS_regression(y = pc_ret_rt, X = pc_fac_rt, y_column = "PC 2", is_pc = True)
```



```
In [ ]: print(pc2.summary())
```

```

=====
                        OLS Regression Results
=====
Dep. Variable:          PC 2      R-squared:                0.550
Model:                  OLS      Adj. R-squared:           0.542
Method:                 Least Squares  F-statistic:             65.07
Date:                  Sat, 15 Jul 2023  Prob (F-statistic):       1.72e-52
Time:                  17:02:05    Log-Likelihood:           -595.75
No. Observations:      326        AIC:                     1206.
Df Residuals:          319        BIC:                     1232.
Df Model:              6
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	4.684e-17	0.084	5.56e-16	1.000	-0.166	0.166
PC 1	0.3608	0.704	0.513	0.609	-1.024	1.746
PC 2	-10.3232	1.295	-7.971	0.000	-12.871	-7.775
PC 3	-21.6396	1.603	-13.500	0.000	-24.793	-18.486
PC 4	-14.2854	2.053	-6.960	0.000	-18.324	-10.247
PC 5	15.7840	2.437	6.478	0.000	10.990	20.578
PC 6	20.3048	2.764	7.347	0.000	14.868	25.742

```

=====
Omnibus:                233.976    Durbin-Watson:           1.704
Prob(Omnibus):           0.000    Jarque-Bera (JB):         7600.490
Skew:                    2.480    Prob(JB):                 0.00
Kurtosis:                26.129    Cond. No.                 32.8
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [ ]: pc_ret_rt.rename(columns={'PC 1': 'PC 1 (returns)', 'PC 2': 'PC 2 (returns)'}, inplace=True)
pc_fac_rt.rename(columns={'PC 1': 'PC 1 (factors)', 'PC 2': 'PC 2 (factors)', 'PC 3': 'PC 3 (factors)'}, inplace=True)
```

```
In [ ]: corr_matrix2 = round(pd.concat([pc_ret_rt, pc_fac_rt], axis=1).corr(), 2)
corr_matrix2[pc_fac_rt.columns[:2]]
```

```
Out[ ]:
```

	PC 1 (factors)	PC 2 (factors)	PC 3 (factors)	PC 4 (factors)	PC 5 (factors)	PC 6 (factors)
PC 1 (returns)	0.87	0.24	0.14	0.05	-0.03	0.07
PC 2 (returns)	0.02	-0.30	-0.51	-0.26	0.24	0.28

Trabalho - Econometria IV

Guilherme Luz, Guilherme Masuko, Caio Garzeri

August 2023

```
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.

```
# 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.

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

```

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])

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

```

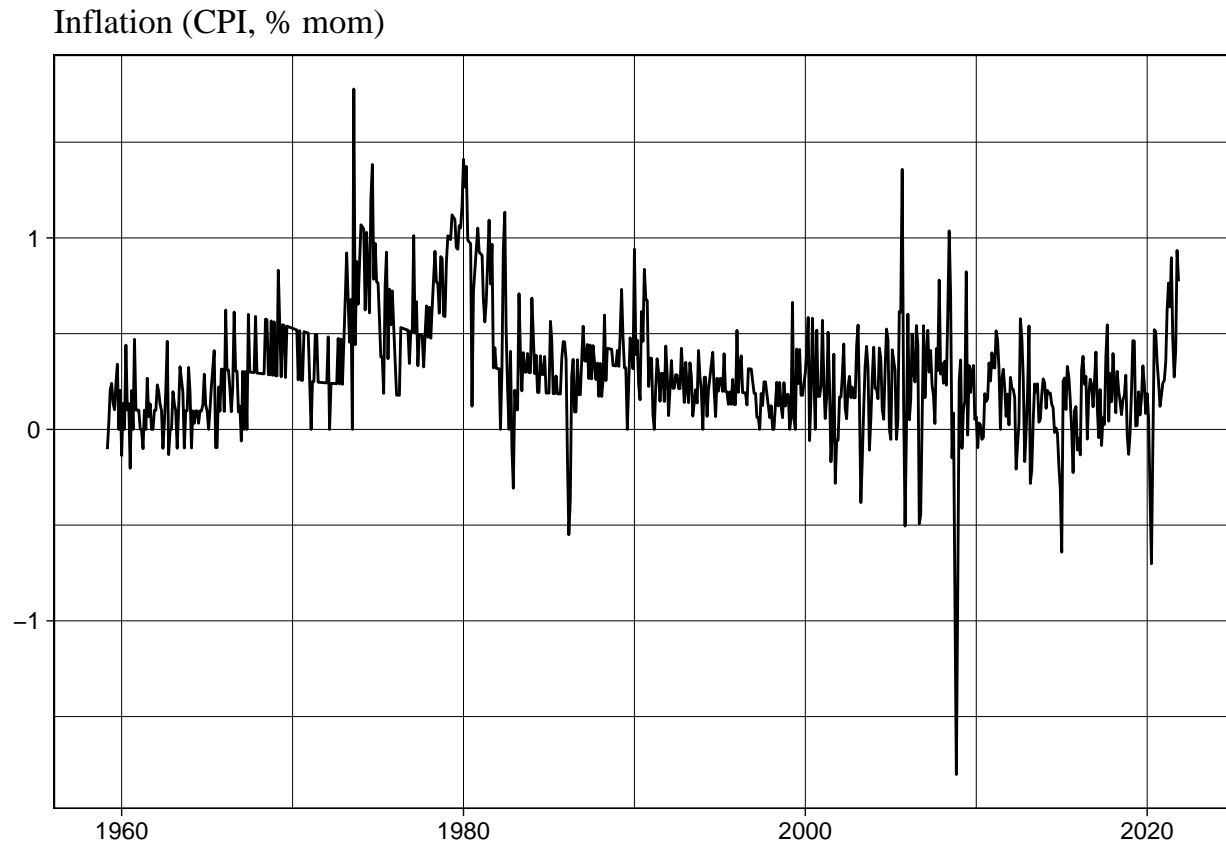
The resulting inflation series, which we want to forecast is shown below.

```

# 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)

```



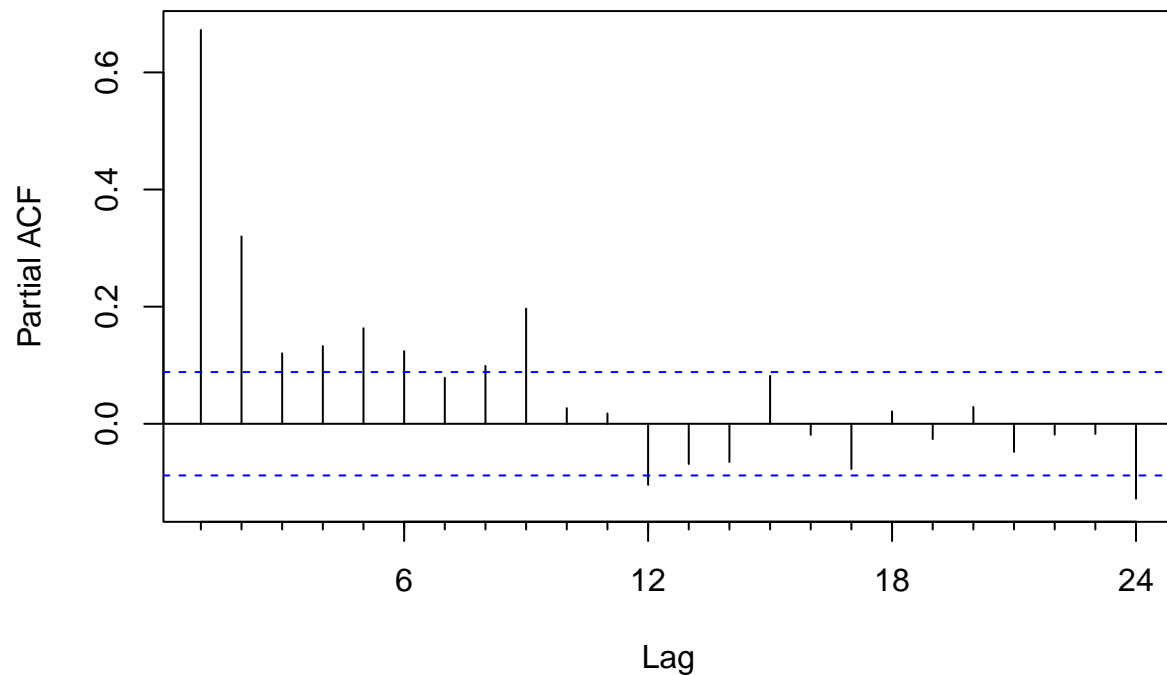
Now, we shall start the estimations.

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)
```

Series .



We believe that a maximum lag of 24 is more than reasonable. Then, we 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()
```

```

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

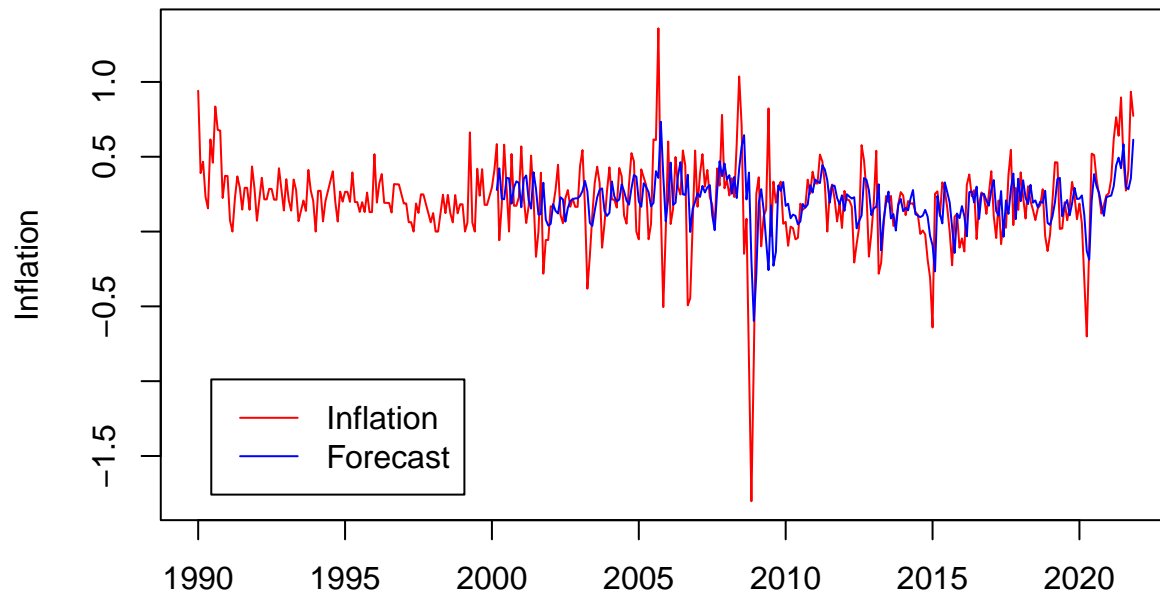
  AR = ar(train, order.max = p.opt, method = "ols", aic = F) # run the AR model with the optimal p

  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 forecast



AR + PC

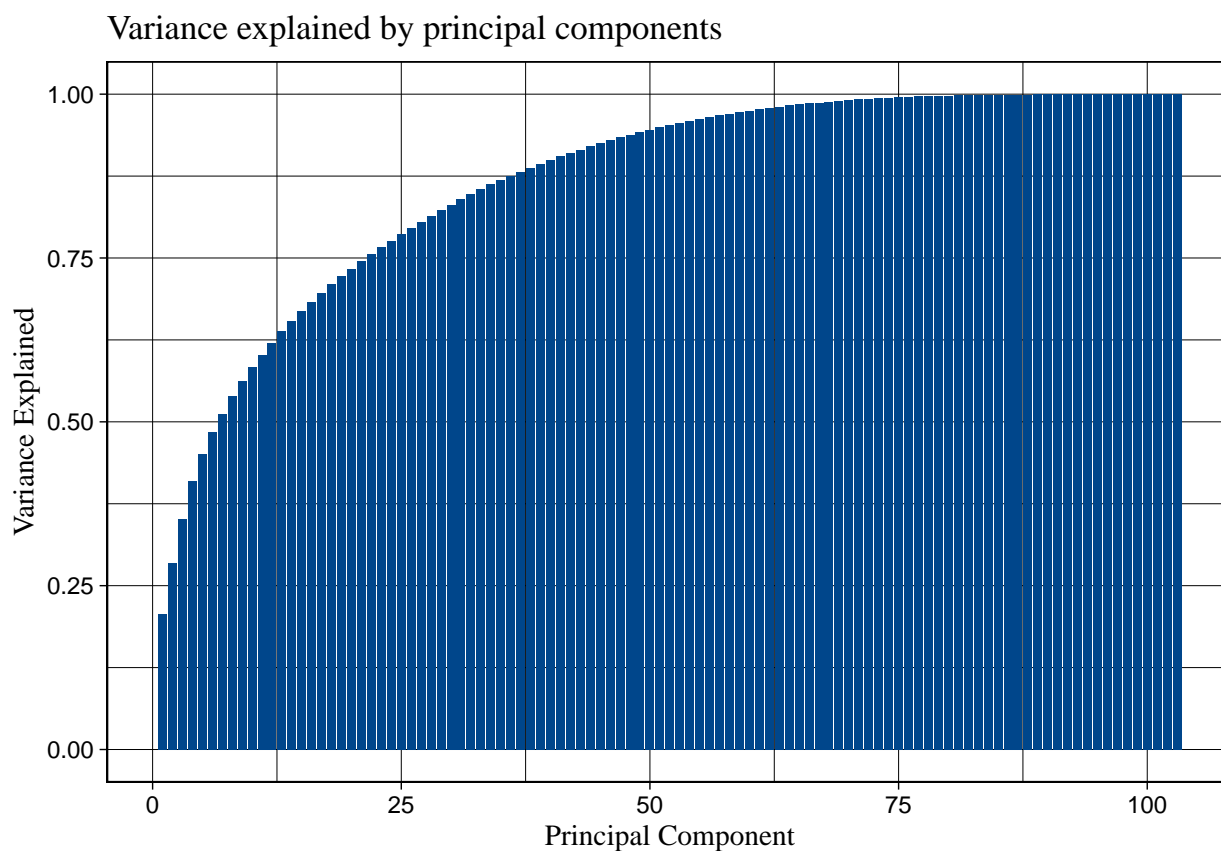
1. PCA

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.



```
# 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)"
```

```
# Informal way (90%)
```

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

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

```
# Biggest drop
```

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

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

```
# 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.

```
# 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.

```
# 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)
}

# 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)) %dorange% {

  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)) %dorn% {

  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)
  }

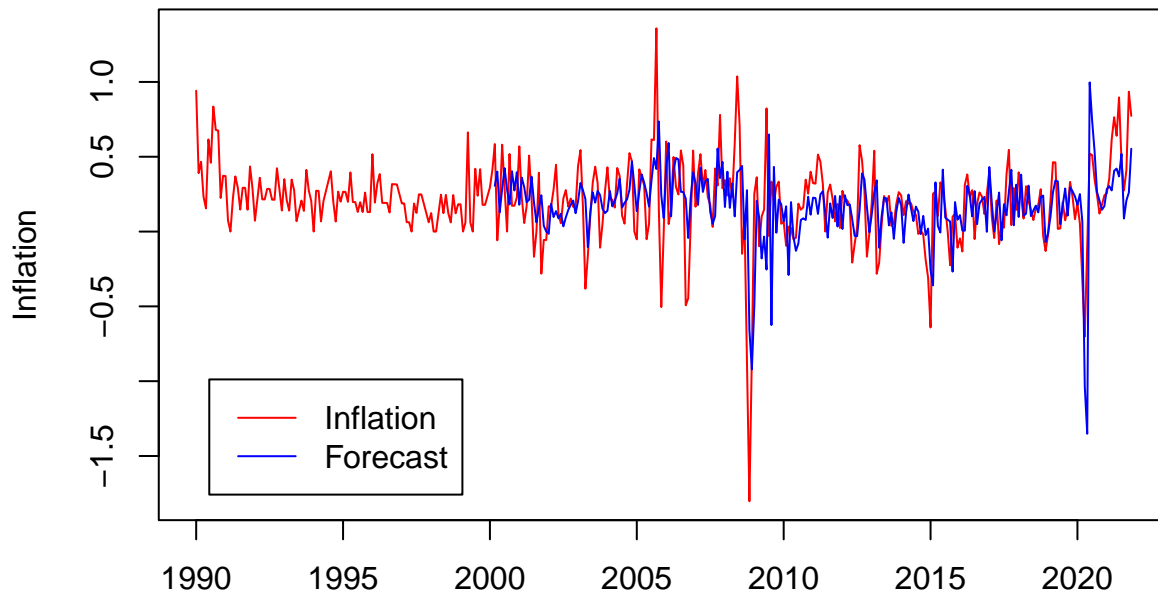
  p.opt = which.min(bic.table) # pick the optimal p

  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
}

```

AR+PC forecast



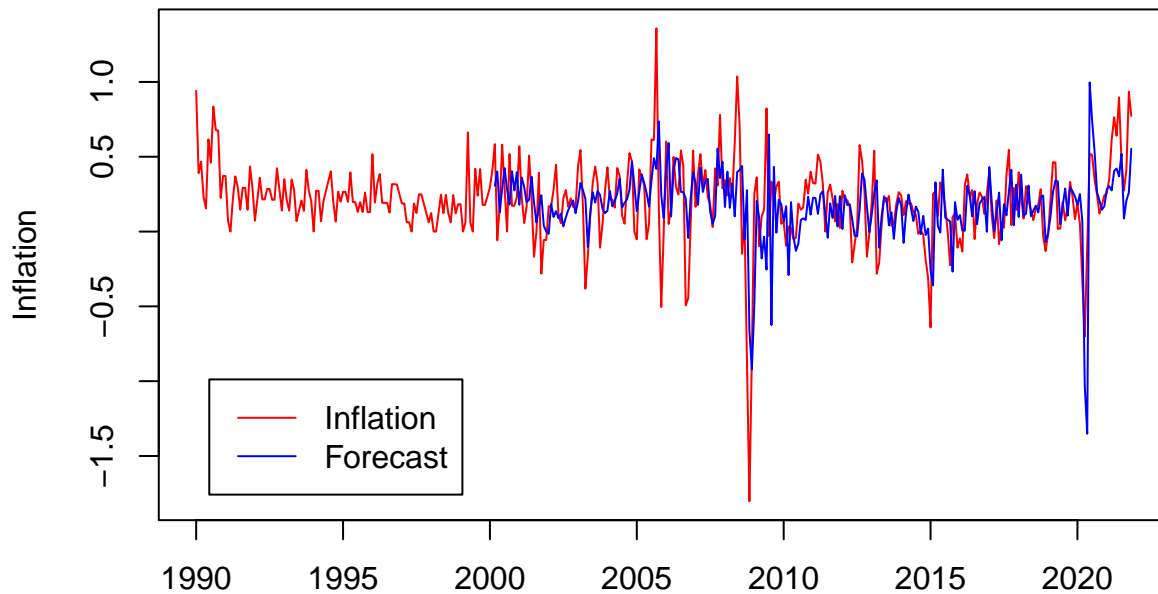
```
# Save forecasts
forecasts = cbind(AR = forecasts, AR_PC = forecast1) %>%
  as.ts()
```

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)
}
```

Ridge forecast



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

```

      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 embedded 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)

# Coefficients
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_coefficients = output$coef %>%
  reduce(cbind) %>%
  as.matrix()

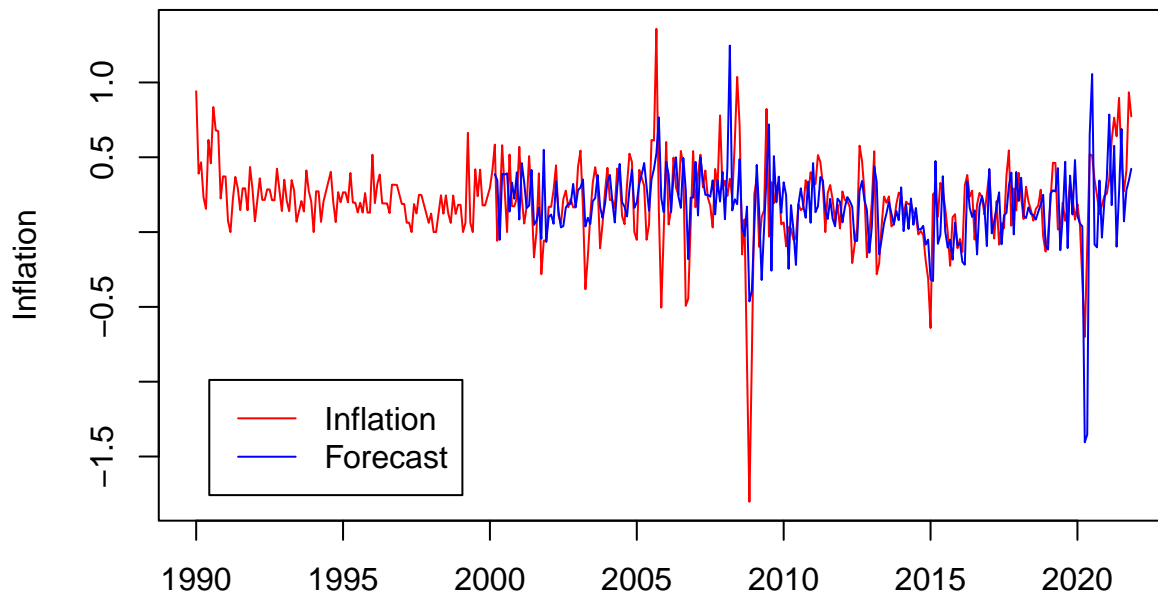
toc()

```

115.872 sec elapsed

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

Ridge forecast



Using 4 lags of CPI and no lags of other variables

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

LASSO Regression

```
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 embedded 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)
}

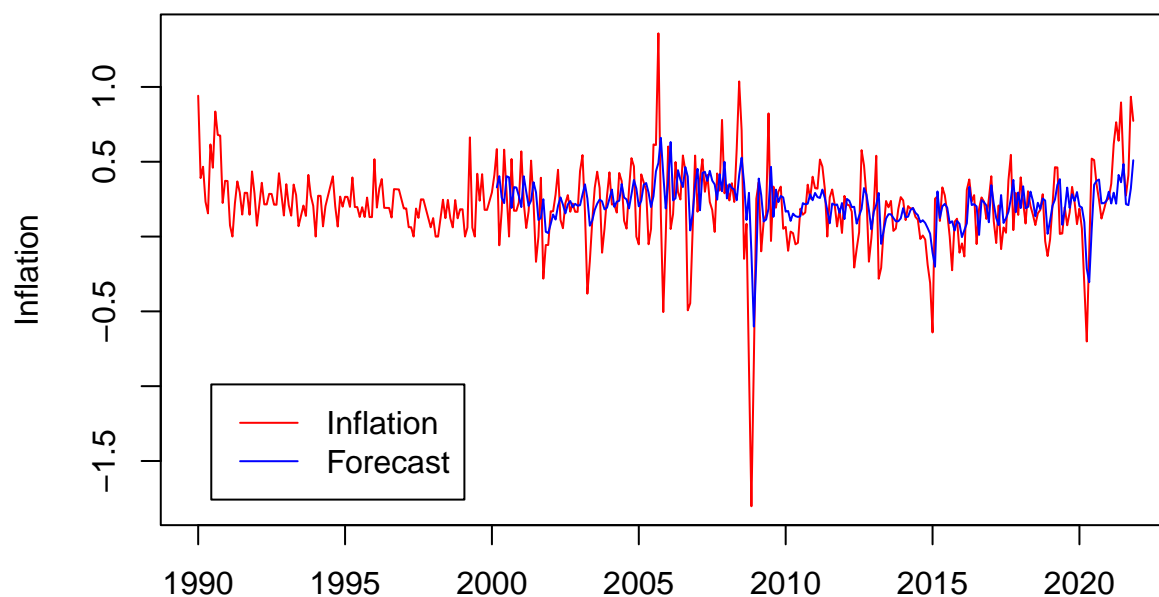
forecast1 = forecast1 %>%
  unlist() %>%
  ts(start = start(inflation) + c(0, rolling_window), frequency = frequency(inflation))

toc()

## 37.407 sec elapsed
beepr::beep()

```


LASSO forecast



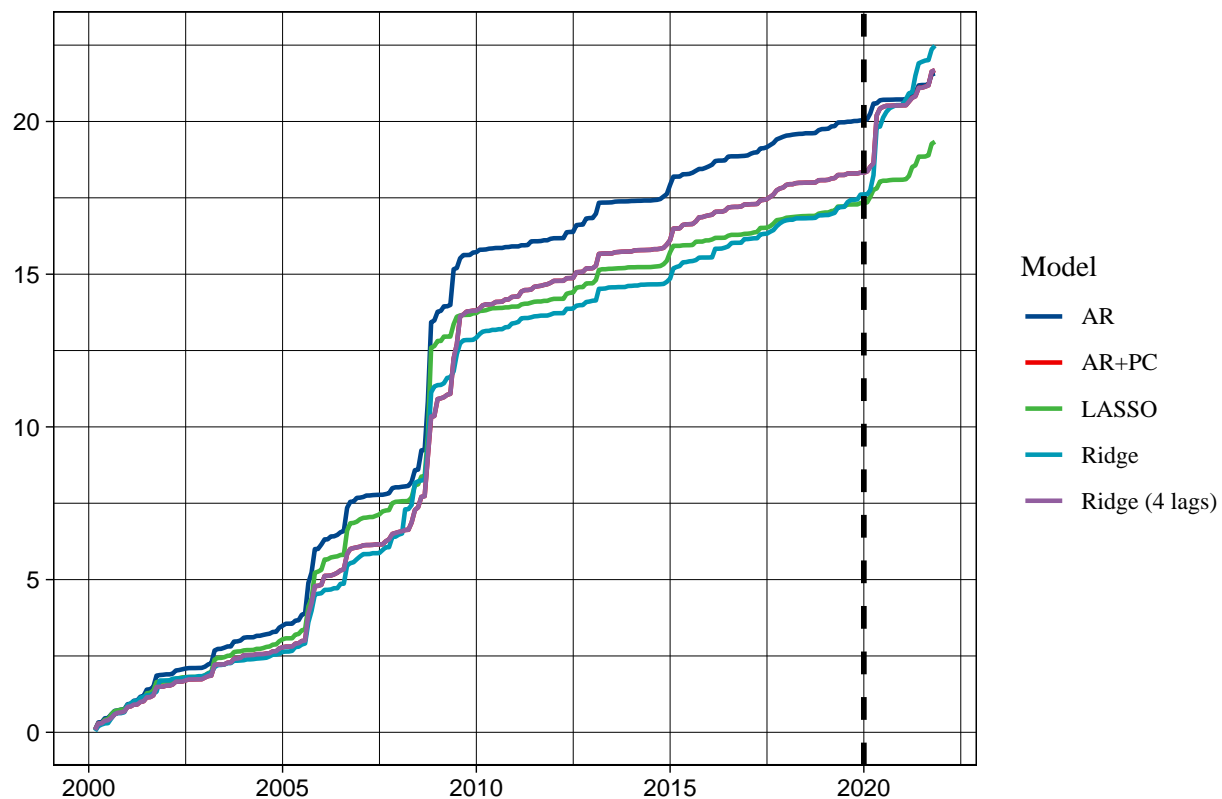
```
# Save forecasts
forecasts = cbind.zoo(forecasts, LASSO = forecast1) %>%
  as.ts()
```

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' ) )
```

Cumulative squared errors



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_coefficients)
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$Standard_Deviation[
"CPIAUCSL"]
ridge_coeff$Standard_Deviation[ridge_coeff$ridge_names == "CPIAUCSL.2"] <- ridge_coeff$Standard_Deviation[
"CPIAUCSL"]
ridge_coeff$Standard_Deviation[ridge_coeff$ridge_names == "CPIAUCSL.3"] <- ridge_coeff$Standard_Deviation[

```

```

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

ridge_coeff_std <- ridge_coeff

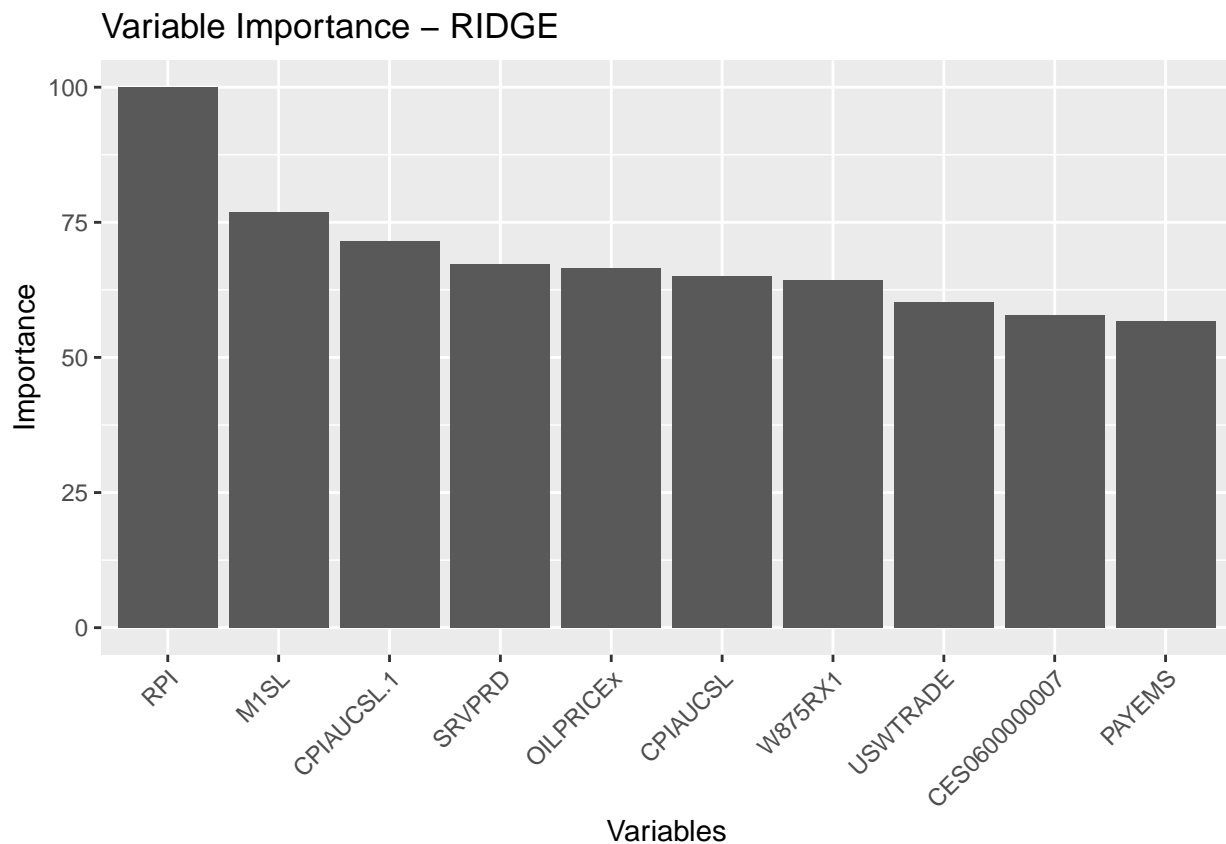
for (col in 2:262) {
  ridge_coeff_std[[col]] <- ridge_coeff_std[[col]] * ridge_coeff_std$Standard_Deviation
}

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.

```

# 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(.)))

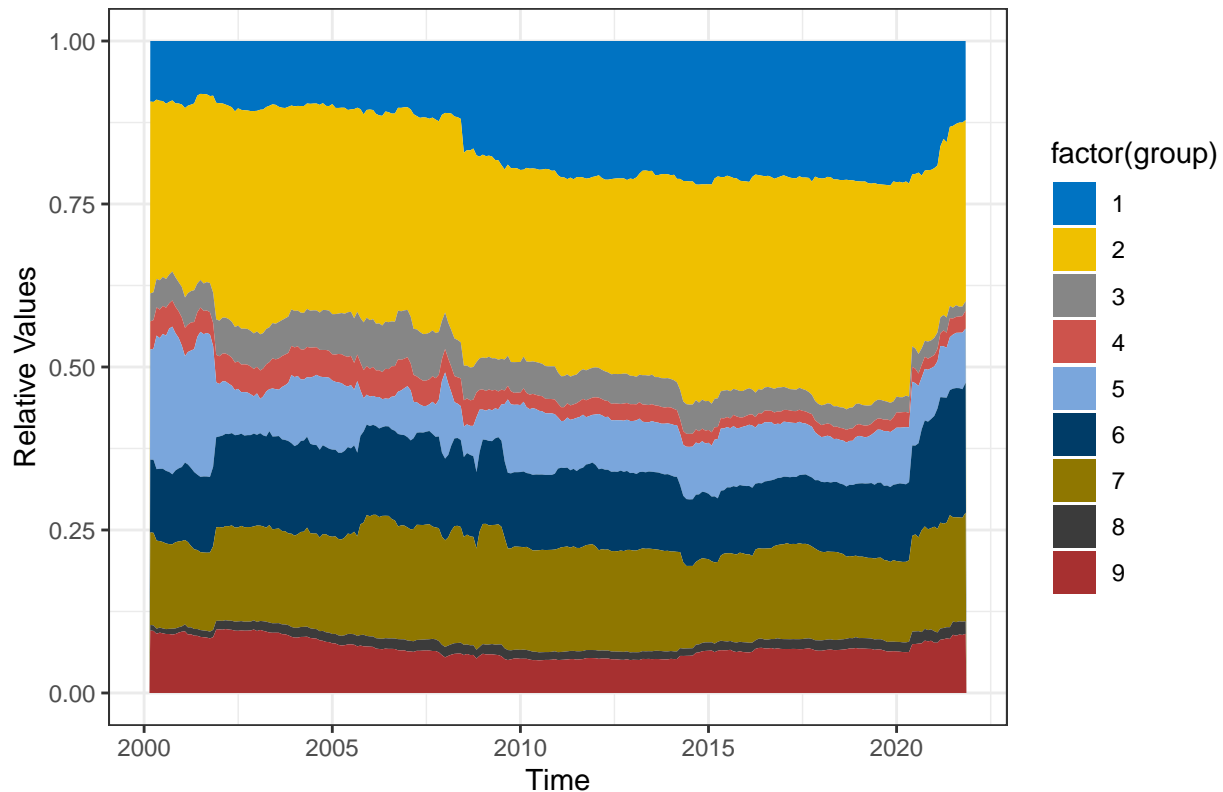
colnames(group_sums)[2:262] <- as.Date(time(forecast1))

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()

```

RIDGE – Group Importance over Time



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.

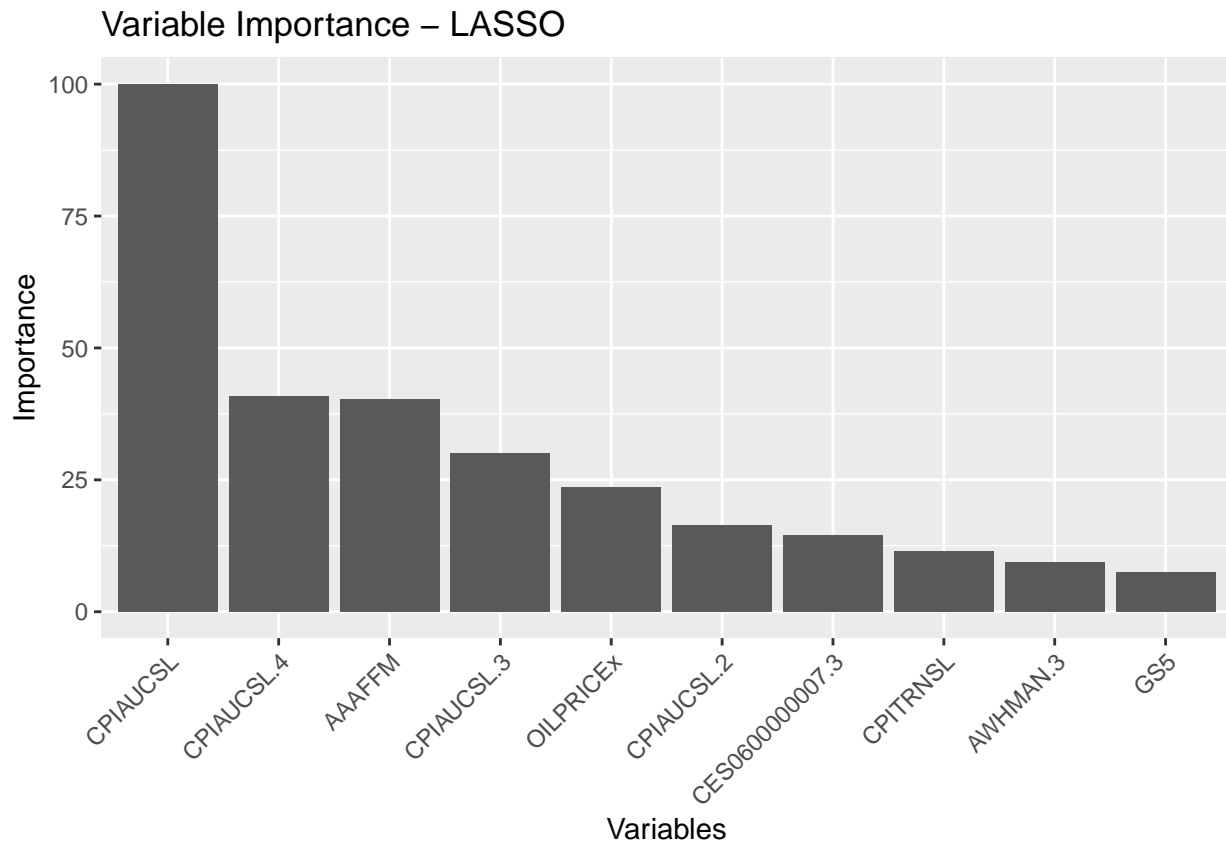
```
coeff_lasso <- coeff_lasso %>%
  mutate_all(~replace_na(., 0))

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 throughout 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.

```

# 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()

```


LASSO – Group Importance over Time



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 by 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]][1 + l]
  }
}

# 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 group

# 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

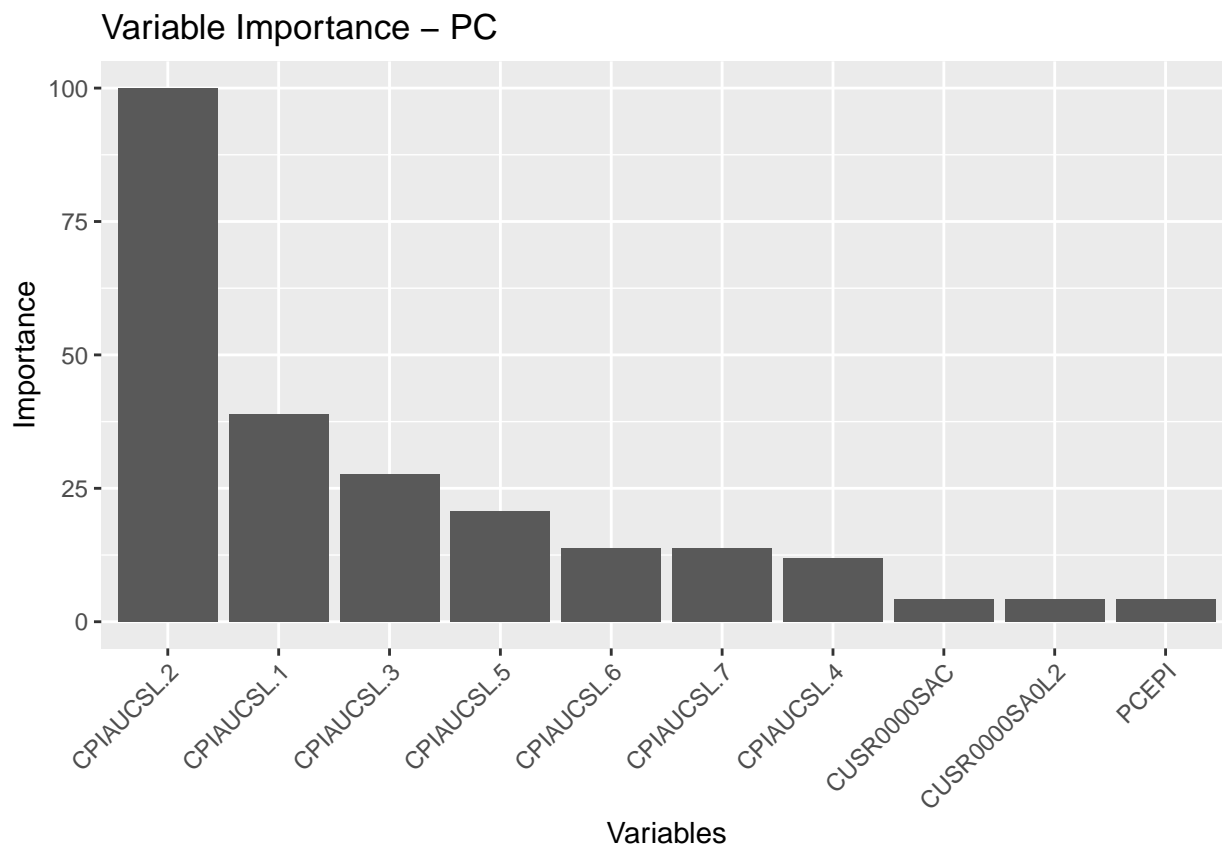
```

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_"))

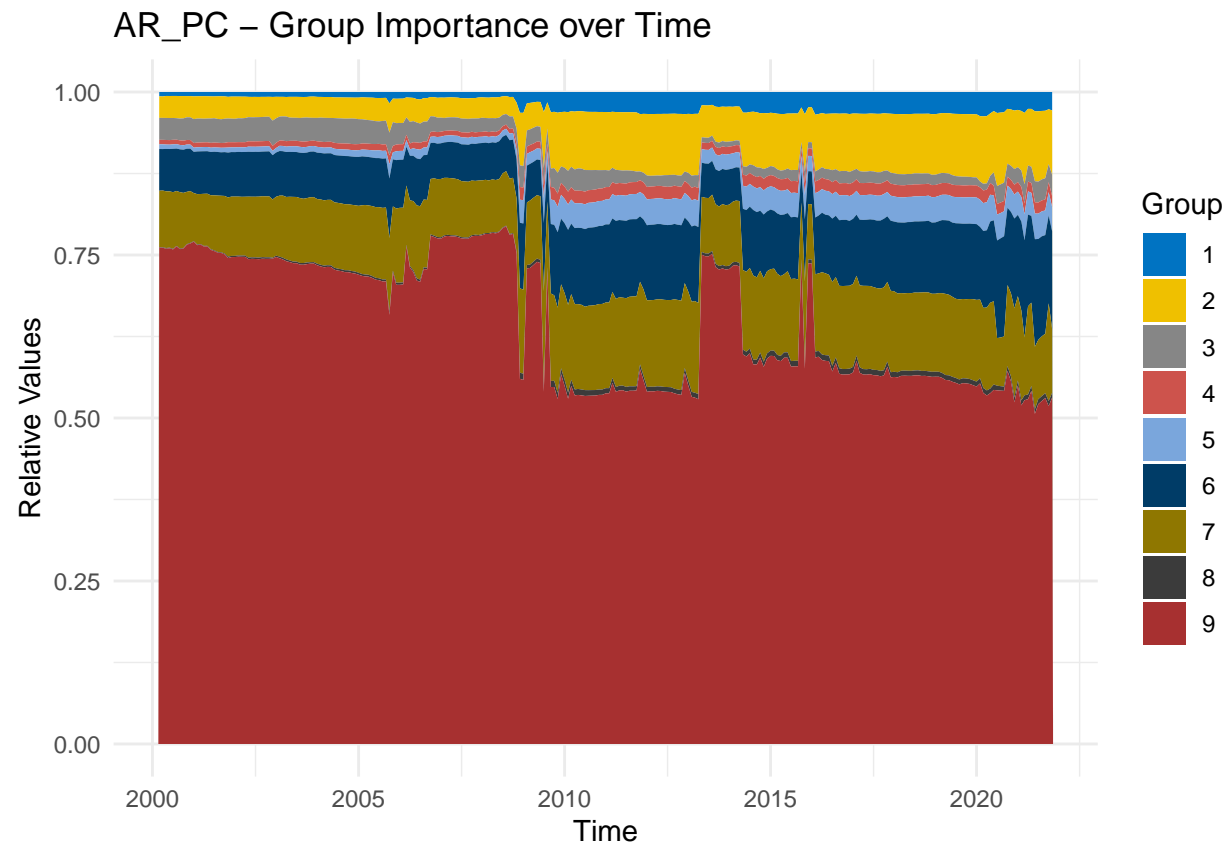
pc_rel_transposed <- as.data.frame((t(pc_rel)))

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()
```



```
In [ ]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
from tqdm import tqdm
from sklearn.metrics import mean_squared_error, r2_score
```

```
In [ ]: # hide warning messages
import warnings
warnings.filterwarnings("ignore")
```

```
In [ ]: # better plots
sns.set(rc={'figure.figsize':(12,8)});
```

```
In [ ]: directory = os.path.dirname(os.getcwd())
directory
```

```
Out[ ]: 'd:\\github\\AssignmentEconometricsIV'
```

Question

The third question consists of an inflation forecasting exercise using a large set of monthly macroeconomic variables and nonlinear models. As in the previous question, the forecasts are based on a rolling-window framework of fixed length of 492 observations, starting in January 1959. Therefore, the forecasts start on January 1990. The last forecasts are for November 2021. More specifically, the rolling window forecasting scheme can be described as follows:

1. Run all in-sample analysis and estimation using data from observation a to observation $a + 492 - 1$.
2. Compute the forecast for observation at position $a + 492$.
3. Set $a = a + 1$ and repeat the two steps above.

```
In [ ]: # read the data
input_path = f'{directory}\\data\\stacionarized_cpi.csv'
df = pd.read_csv(input_path)
df['date'] = pd.to_datetime(df['date'])
df = df.set_index('date')
```

```
In [ ]: df.head()
```

```
Out[ ]:
```

	RPI	W875RX1	DPCERA3M086SBEA	RETAILx	INDPRO	IPFPNSS	IPFINAL	IPCONGD	IP
date									
1959-03-01	0.643011	0.735934	0.941009	0.832120	1.430253	0.603609	0.489927	0.000000	

	RPI	W875RX1	DPCERA3M086SBEA	RETAILx	INDPRO	IPFPNSS	IPFINAL	IPCONGD	IF
date									
1959-04-01	0.649412	0.704864	-0.363947	0.061571	2.107741	1.433803	1.454234	1.565338	
1959-05-01	0.576311	0.661646	1.200535	0.780340	1.495024	0.826920	0.958304	0.476849	
1959-06-01	0.310244	0.297379	0.370829	0.906434	0.114438	0.703445	0.712642	-0.476849	
1959-07-01	-0.058921	-0.076384	-0.342687	-0.033018	-2.423797	0.116693	0.824692	1.305596	

5 rows × 104 columns

```
In [ ]: # set the amount of lags
lags = 4
```

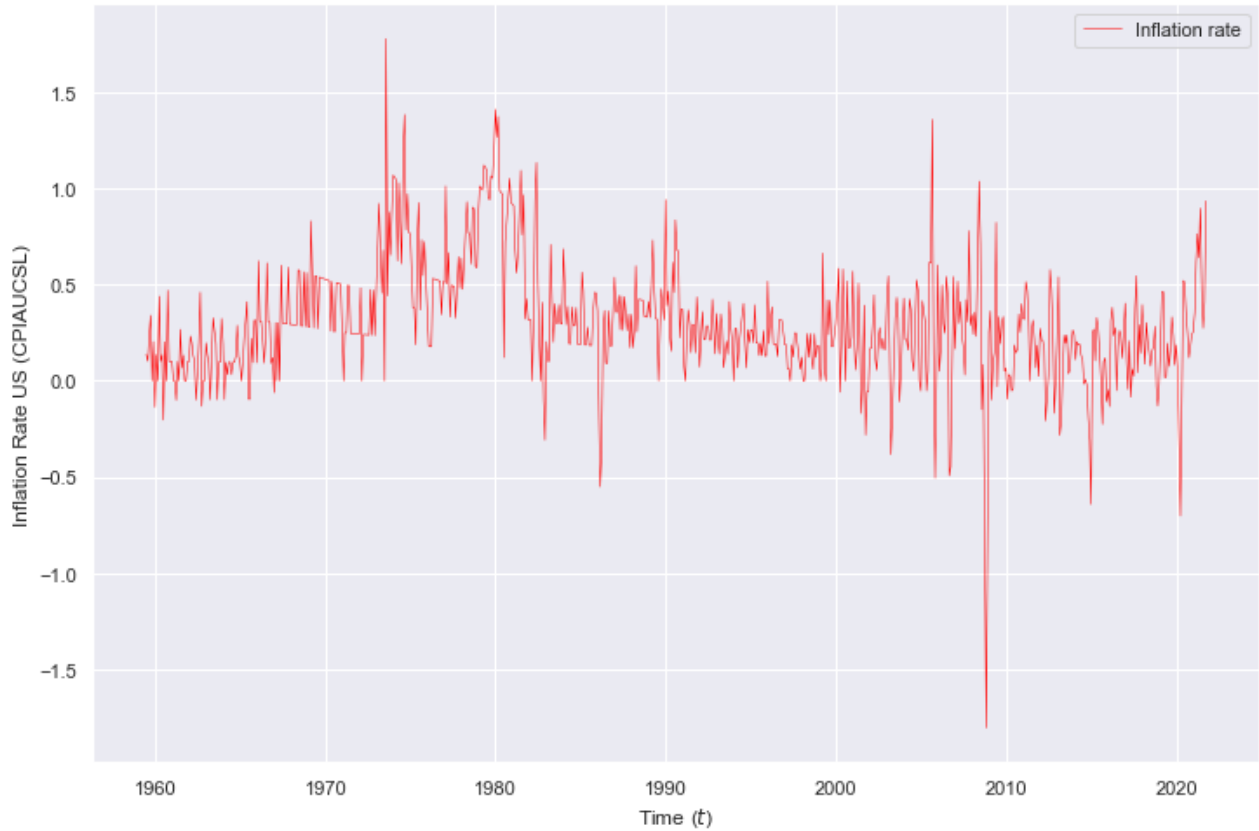
```
In [ ]: # create lagged variables
for col in df.columns:
    for i in range(lags):
        name_col = col + f"(-{i+1})"
        df[name_col] = df[col].shift(i+1)
```

```
In [ ]: # inflation rate ahead
df['CPIAUCSL(+1)'] = df['CPIAUCSL'].shift(-1)
```

```
In [ ]: # drop nan rows
df = df.dropna()
```

```
In [ ]: # plot Inflation rate of US
plt.plot(df['CPIAUCSL'], color='red', label='Inflation rate', linewidth = 0.5)

plt.xlabel(f'Time (${t$})')
plt.ylabel('Inflation Rate US (CPIAUCSL)')
plt.legend()
plt.show()
```



In []:

```
df
```

Out[]:

	RPI	W875RX1	DPCERA3M086SBEA	RETAILx	INDPRO	IPFPNSS	IPFINAL	IPCONGD
date								
1959-07-01	-0.058921	-0.076384	-0.342687	-0.033018	-2.423797	0.116693	0.824692	1.305596
1959-08-01	-0.563656	-0.574751	0.600333	0.636421	-3.446532	-0.702622	-0.234750	0.117877
1959-09-01	0.072136	0.000000	1.001845	-1.315700	-0.120914	-0.470898	-0.353804	-0.353743
1959-10-01	0.128006	0.115215	-0.683507	0.728802	-0.729019	-0.236695	-0.473264	-0.474023
1959-11-01	0.759485	0.671571	-0.044618	-2.670393	0.607511	-1.070659	-1.674509	-2.161044
...
2021-06-01	-0.268168	0.246839	0.588917	0.848804	0.546727	0.002528	0.114008	-0.202956
2021-07-01	0.803329	0.376098	-0.313442	-1.637075	0.767226	1.461057	1.668025	0.970731
2021-08-01	-0.041013	-0.094940	0.727968	1.152697	-0.136496	-0.232743	-0.485054	-0.407293
2021-09-01	-1.333562	0.169571	0.264718	0.738929	-1.018960	-0.235388	-0.539534	-0.555028

	RPI	W875RX1	DPCERA3M086SBEA	RETAILx	INDPRO	IPFPNSS	IPFINAL	IPCONGD
date								
2021-10-01	-0.226827	0.002789	0.740685	1.769110	1.664607	1.015773	1.119566	1.035674

748 rows × 521 columns

```
In [ ]: from functions.nonlinear_models import NN_forecast, LSTM_forecast
```

```
In [ ]: # set variables
y = df["CPIAUCSL(+1)"]
X = df.drop("CPIAUCSL(+1)", axis=1)
```

```
In [ ]: # add forecast columns
df["CPIAUCSL_estimated_NN"] = np.nan
df["CPIAUCSL_estimated_LSTM"] = np.nan
```

(a) (60 points)

Estimate a model based on a neural network specification and compute one step ahead forecasts. You are free to choose between shallow, deep, convolution or LSTM networks. However, you have to motivate your particular choice;

feedforward Neural Network model

Cunha Medeiros, Marcelo and Schütte, Erik Christian Montes and Soussi, Tobias Skipper, Global Inflation: Implications for forecasting and monetary policy (June 24, 2022). Available at SSRN: <https://ssrn.com/abstract=4145665> or <http://dx.doi.org/10.2139/ssrn.4145665>

Architecture:

- Optimizer: Adam (<https://arxiv.org/abs/1412.6980>)
- Layers: 3 (like the previous paper: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4145665)
- Activation Function: ReLu $f(x) = \max(x, 0)$
- Neurons: 100, 60 and 30 in the first, second and third layers respectively (based on <https://www.sciencedirect.com/science/article/abs/pii/B978008051433850015X>)

LSTM Neural Network model works as the same way, with LSTM acting like a neuron.

```
In [ ]: # set some parameters
rolling_window = 492 - lags
T = len(y) - rolling_window
```

```
In [ ]: # List of forecast squared errors
#errors_NN = []
```



```

errors_LSTM = []

for t in tqdm(range(T), desc='Processing for time'):
    # predict date
    date = y[[rolling_window+t]].index

    # estimation sets
    X_train = X[t:(rolling_window+t)]
    y_train = y[t:(rolling_window+t)]

    # forecast sets
    X_test = X.iloc[[rolling_window+t]]
    y_test = y[rolling_window+t]

    # estimations
    #y_pred_NN, error_pred_NN = NN_forecast(X_train, y_train, X_test, y_test)
    y_pred_LSTM, error_pred_LSTM = LSTM_forecast(X_train, y_train, X_test, y_test)

    # fill forecast columns
    #df["CPIAUCSL_estimated_NN"][date] = y_pred_NN
    df["CPIAUCSL_estimated_LSTM"][date] = y_pred_LSTM

    # append forecast squared errors
    #errors_NN.append(error_pred_NN)
    errors_LSTM.append(error_pred_LSTM)

```

Processing for time: 100%|██████████| 260/260 [14:31:51<00:00, 201.20s/it]

In []: `df[["CPIAUCSL(+1)", "CPIAUCSL_estimated_NN", "CPIAUCSL_estimated_LSTM"]][rolling_window`

Out[]: **CPIAUCSL(+1) CPIAUCSL_estimated_NN CPIAUCSL_estimated_LSTM**

date			
2000-03-01	-0.058514	0.242504	0.321759
2000-04-01	0.175234	0.124897	0.129800
2000-05-01	0.580720	0.217302	0.177507
2000-06-01	0.289519	0.527120	0.401178
2000-07-01	0.000000	0.295989	0.225920
...
2021-06-01	0.471599	0.368065	0.353107
2021-07-01	0.273614	-0.373261	0.011361
2021-08-01	0.410742	0.060732	0.363657
2021-09-01	0.934505	0.172199	0.187941
2021-10-01	0.773092	0.320927	0.294376

260 rows × 3 columns

In []: `output = f'{directory}\\output\\NN_predictions.csv'`
`df[["CPIAUCSL", "CPIAUCSL(+1)", "CPIAUCSL_estimated_NN", "CPIAUCSL_estimated_LSTM"]].to`

```
In [ ]: path = f'{directory}\\output\\NN_predictions.csv'
df = pd.read_csv(path, index_col=0)
df.index = pd.to_datetime(df.index)
```

```
In [ ]: df[rolling_window:]
```

```
Out[ ]:      CPIAUCSL  CPIAUCSL(+1)  CPIAUCSL_estimated_NN  CPIAUCSL_estimated_LSTM
```

date				
2000-03-01	0.584795	-0.058514	0.242504	0.321759
2000-04-01	-0.058514	0.175234	0.124897	0.129800
2000-05-01	0.175234	0.580720	0.217302	0.177507
2000-06-01	0.580720	0.289519	0.527120	0.401178
2000-07-01	0.289519	0.000000	0.295989	0.225920
...
2021-06-01	0.896742	0.471599	0.368065	0.353107
2021-07-01	0.471599	0.273614	-0.373261	0.011361
2021-08-01	0.273614	0.410742	0.060732	0.363657
2021-09-01	0.410742	0.934505	0.172199	0.187941
2021-10-01	0.934505	0.773092	0.320927	0.294376

260 rows × 4 columns

```
In [ ]: # prediction
prediction_dates = list(X.iloc[(rolling_window):].index)
```

```
In [ ]: # treat the error list
errors_NN_ = [item[0] for item in errors_NN]
errors_LSTM_ = [item[0] for item in errors_LSTM]

# compute cumulated mse
cum_errors_NN = list(np.cumsum(errors_NN_))
cum_errors_LSTM = list(np.cumsum(errors_LSTM_))
```

```
In [ ]: errors = {'Cumulated_MSE_NN': cum_errors_NN, 'Cumulated_MSE_LSTM': cum_errors_LSTM}
mse = pd.DataFrame(errors, index=prediction_dates)
```

```
In [ ]: mse
```

```
Out[ ]:      Cumulated_MSE_NN  Cumulated_MSE_LSTM
```

2000-03-01	0.090612	0.144607
------------	----------	----------

	Cumulated_MSE_NN	Cumulated_MSE_LSTM
2000-04-01	0.093146	0.146672
2000-05-01	0.225218	0.309253
2000-06-01	0.281672	0.321720
2000-07-01	0.369282	0.372760
...
2021-06-01	25.398293	20.114305
2021-07-01	25.816740	20.183083
2021-08-01	25.939247	20.185299
2021-09-01	26.520357	20.742657
2021-10-01	26.724810	20.971827

260 rows × 2 columns

```
In [ ]: output = f'{directory}\\output\\NN_MSE.csv'
mse.to_csv(output, sep=',', encoding='utf-8')
```

```
In [ ]: path = f'{directory}\\output\\NN_MSE.csv'
mse = pd.read_csv(path, index_col=0)
mse.index = pd.to_datetime(mse.index)
```

```
In [ ]: mse
```

```
Out[ ]: Cumulated_MSE_NN Cumulated_MSE_LSTM
```

2000-03-01	0.090612	0.144607
2000-04-01	0.093146	0.146672
2000-05-01	0.225218	0.309252
2000-06-01	0.281672	0.321720
2000-07-01	0.369282	0.372760
...
2021-06-01	25.398293	20.114305
2021-07-01	25.816740	20.183083
2021-08-01	25.939247	20.185299
2021-09-01	26.520357	20.742657
2021-10-01	26.724810	20.971827

260 rows × 2 columns

(b) (20 points)

Plot your forecasts against the four linear benchmarks from Question 2. Comment on your results.

```
In [ ]: path = '../output/forecasts.csv'
forecast_ln = pd.read_csv(path, index_col=0)
prediction_dates.append(pd.to_datetime('2021-11-01'))
forecast_ln.index = prediction_dates
```

```
In [ ]: forecast_ln
```

```
Out[ ]: forecasts.forecasts_Q2.AR forecasts.forecasts_Q2.AR_PC forecasts.forecasts_Q2.Ridge_4lags forecast
```

2000-03-01	0.347137	0.306083	0.358069
2000-04-01	0.424690	0.401117	0.359949
2000-05-01	0.184807	0.128173	0.357127
2000-06-01	0.250046	0.286717	0.358928
2000-07-01	0.416175	0.423749	0.361864
...
2021-07-01	0.404299	0.519179	0.239185
2021-08-01	0.190942	0.086930	0.233436
2021-09-01	0.205723	0.207572	0.232049
2021-10-01	0.365649	0.260807	0.233505
2021-11-01	0.602073	0.553538	0.237903

261 rows × 7 columns

```
In [ ]: fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(24, 16))

# True CPI
axes[0,0].plot(df['CPIAUCSL'][rolling_window:], color='cyan', label='True CPI rate', li
# NN forecasts
axes[0,0].plot(df["CPIAUCSL_estimated_NN"], color='blue', label='NN', linewidth=0.5)
# adding a vertical line at 2020, January
axes[0,0].axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')
# Labels
```

```

axes[0,0].set_xlabel('Time')
axes[0,0].set_ylabel('Forecast')
axes[0,0].set_title('Inflation Rate US (CPIAUCSL) Neural Network Forecast')
axes[0,0].legend()

# True CPI
axes[0,1].plot(df['CPIAUCSL'][rolling_window:], color='cyan', label='True CPI rate', li
# LSTM forecasts
axes[0,1].plot(df['CPIAUCSL_estimated_LSTM'], color='black', label='LSTM', linewidth=0.
# adding a vertical line at 2020, January
axes[0,1].axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')
# Labels
axes[0,1].set_xlabel('Time')
axes[0,1].set_ylabel('Forecast')
axes[0,1].set_title('Inflation Rate US (CPIAUCSL) LSTM Forecast')
axes[0,1].legend()

# True CPI
axes[1,0].plot(df['CPIAUCSL'][rolling_window:], color='cyan', label='True CPI rate', li
# AR
axes[1,0].plot(forecast_ln["forecasts.forecasts_Q2.AR"], color='purple', label='AR', li
# adding a vertical line at 2020, January
axes[1,0].axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')
# Labels
axes[1,0].set_xlabel('Time')
axes[1,0].set_ylabel('Forecast')
axes[1,0].set_title('Inflation Rate US (CPIAUCSL) AR Forecast')
axes[1,0].legend()

# True CPI
axes[1,1].plot(df['CPIAUCSL'][rolling_window:], color='cyan', label='True CPI rate', li
# AR+PC
axes[1,1].plot(forecast_ln["forecasts.forecasts_Q2.AR_PC"], color='green', label='AR+PC
# adding a vertical line at 2020, January
axes[1,1].axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')
# Labels
axes[1,1].set_xlabel('Time')
axes[1,1].set_ylabel('Forecast')
axes[1,1].set_title('Inflation Rate US (CPIAUCSL) AR+PC Forecast')
axes[1,1].legend()

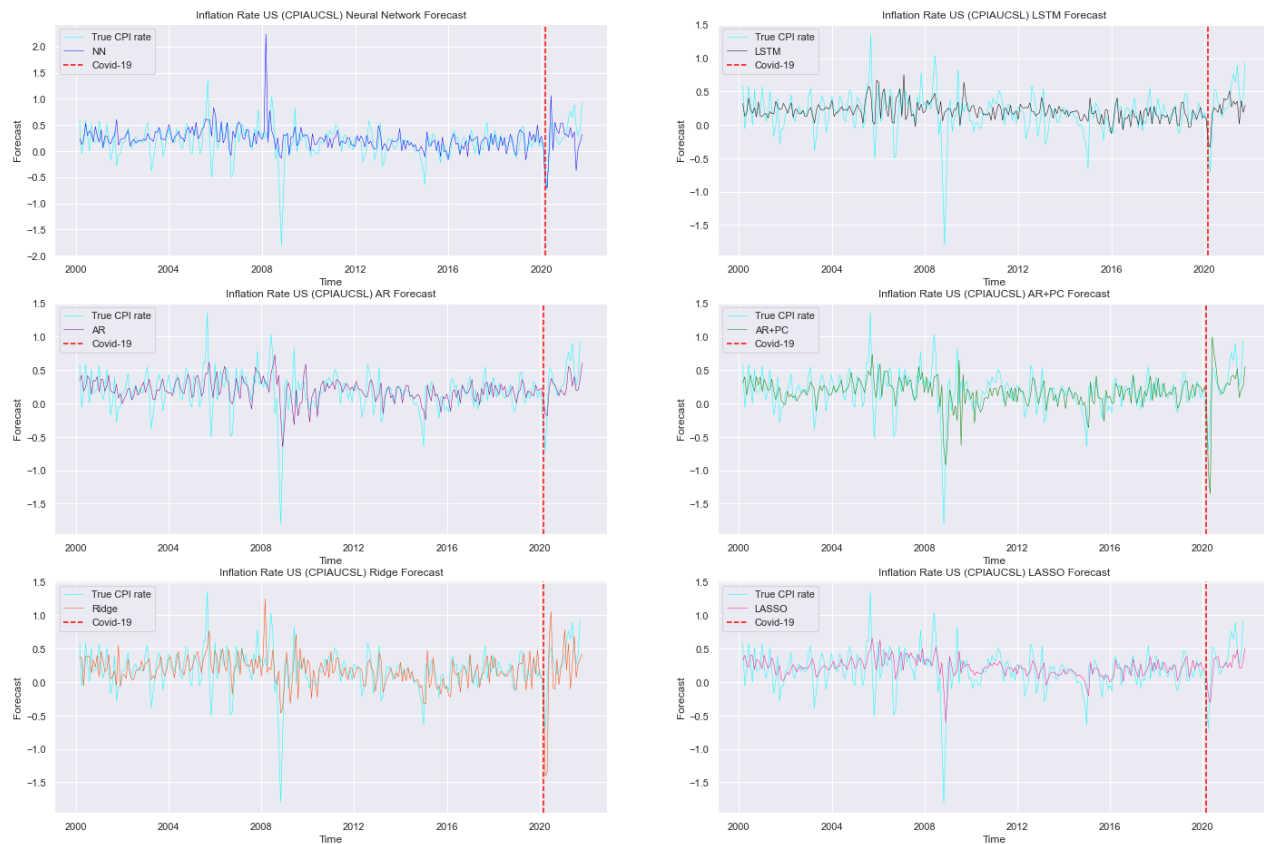
# True CPI
axes[2,0].plot(df['CPIAUCSL'][rolling_window:], color='cyan', label='True CPI rate', li
# Ridge
axes[2,0].plot(forecast_ln["forecasts.forecasts_Q2.Ridge"], color='orangered', label='R
# adding a vertical line at 2020, January
axes[2,0].axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')
# Labels
axes[2,0].set_xlabel('Time')
axes[2,0].set_ylabel('Forecast')
axes[2,0].set_title('Inflation Rate US (CPIAUCSL) Ridge Forecast')
axes[2,0].legend()

# True CPI
axes[2,1].plot(df['CPIAUCSL'][rolling_window:], color='cyan', label='True CPI rate', li
# LASSO
axes[2,1].plot(forecast_ln["forecasts.forecasts_Q2.LASSO"], color='deeppink', label='LA
# adding a vertical line at 2020, January
axes[2,1].axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')
# Labels

```

```
axes[2,1].set_xlabel('Time')
axes[2,1].set_ylabel('Forecast')
axes[2,1].set_title('Inflation Rate US (CPIAUCSL) LASSO Forecast')
axes[2,1].legend()
```

Out[]: <matplotlib.legend.Legend at 0x274f8d7bb50>



```
In [ ]: plt.plot(df['CPIAUCSL'][rolling_window:], color='cyan', label='True CPI rate', linewidth
plt.plot(df["CPIAUCSL_estimated_NN"], color='blue', label='NN', linewidth = 0.5)
plt.plot(df["CPIAUCSL_estimated_LSTM"], color='black', label='LSTM', linewidth = 0.5)

# adding a vertical line at 2020, January
plt.axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')

plt.xlabel('Time')
plt.ylabel('Inflation Rate US (CPIAUCSL) Forecasts')
plt.legend()
plt.show()
```



(c) (20 points)

Compute the mean squared error of the NN-based model and the benchmarks. Does the NN model outperform the linear alternative?

```
In [ ]: path = '../output/cum_error.csv'
cum_mse = pd.read_csv(path, index_col=0)
cum_mse["date"] = pd.to_datetime(cum_mse["date"])
cum_mse.set_index("date", inplace=True)
```

```
In [ ]: cum_mse
```

```
Out[ ]:
```

	AR	AR+PC	Ridge (4 lags)	Ridge	LASSO	RF (4 lags)	RF
date							
2000-03-01	0.056481	0.077681	0.051405	0.039072	0.065451	0.087837	0.086347
2000-04-01	0.289967	0.288942	0.226516	0.202956	0.278926	0.309426	0.268405
2000-05-01	0.290059	0.291156	0.259602	0.254299	0.285988	0.313885	0.269424
2000-06-01	0.399404	0.377594	0.308793	0.290743	0.413642	0.416084	0.330281
2000-07-01	0.415446	0.395612	0.314027	0.299689	0.426557	0.417525	0.341706
...
2021-07-01	20.777157	21.104799	29.034805	21.953238	18.848510	19.511802	18.618540
2021-08-01	20.783992	21.139650	29.036419	21.994010	18.851793	19.551800	18.643675

	AR	AR+PC	Ridge (4 lags)	Ridge	LASSO	RF (4 lags)	RF
date							
2021-09-01	20.826024	21.180928	29.068351	22.011110	18.892047	19.590289	18.671658
2021-10-01	21.149622	21.634798	29.559752	22.359695	19.267982	20.023417	19.094611
2021-11-01	21.178869	21.683002	29.846179	22.482222	19.337216	20.144240	19.209014

261 rows × 7 columns

In []:

```
# plot of cumulated MSE
plt.plot(mse["Cumulated_MSE_NN"], color='blue', label="NN", linewidth = 0.5)

# plot of cumulated MSE
plt.plot(mse["Cumulated_MSE_LSTM"], color='black', label="LSTM", linewidth = 0.5)

# plot of cumulated MSE
plt.plot(cum_mse["AR"], color='purple', label="AR", linewidth = 0.5)

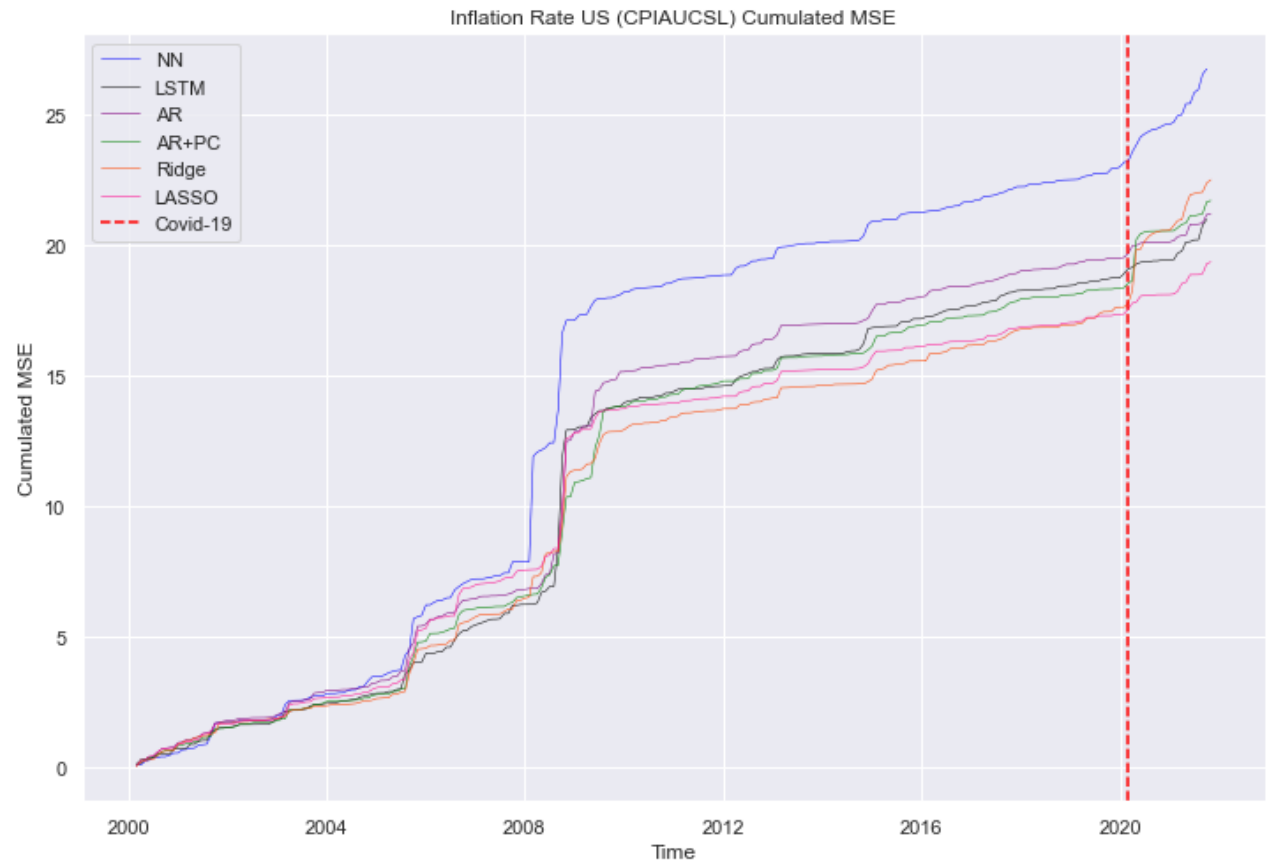
# plot of cumulated MSE
plt.plot(cum_mse["AR+PC"], color='green', label="AR+PC", linewidth = 0.5)

# plot of cumulated MSE
plt.plot(cum_mse["Ridge"], color='orangered', label="Ridge", linewidth = 0.5)

# plot of cumulated MSE
plt.plot(cum_mse["LASSO"], color='deeppink', label="LASSO", linewidth = 0.5)

# adding a vertical line at 2020, January
plt.axvline(x=mse.index[240], color='red', linestyle='--', label='Covid-19')

plt.xlabel('Time')
plt.ylabel('Cumulated MSE')
plt.title('Inflation Rate US (CPIAUCSL) Cumulated MSE')
plt.legend()
plt.show()
```

Trabalho - Econometria IV

Guilherme Luz, Guilherme Masuko, Caio Garzeri

August 2023

```
library(lubridate) # for handling dates
library(randomForest) # Random Forest implementation of the original Fortran code by Brieman (2001)
library(ranger) # Faster implementation of Random Forest
```

Question 3

Item D

In order to include the lags of the variables as covariates, we need to do an embedding process. **explicar**. (We do this inside the rolling window loop to avoid ‘cheating’).

After this process, we can use the usual IID bootstrap, since we are interested in direct forecasting.

```
# 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)
}
```

```
n_lags = 4

# Rolling window forecasting
rolling_window <- 492

# Random Forest parameters
p = (1+n_lags)*ncol(data) # number of variables
mtry = ((1/3)*p) %>% round() # number of variables randomly selected
num.trees = 500 # number of trees
min.bucket = 5 # minimal number of observations in each leave (terminal node)
```

```

set.seed(1430)

forecast1 = list()

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

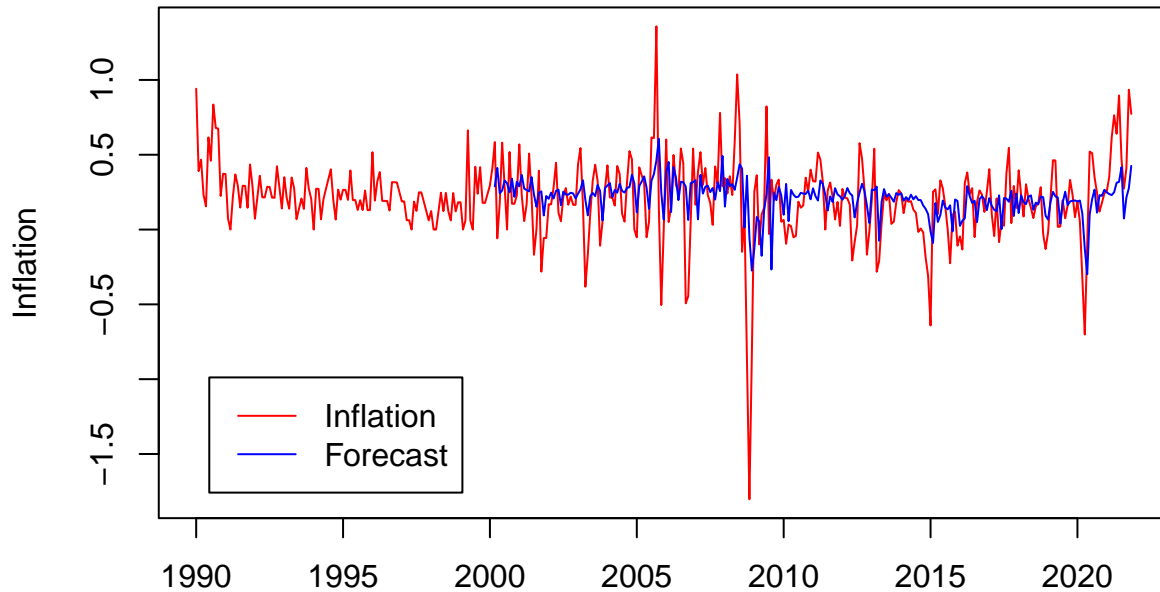
  # Random forest estimation
  RF = ranger(inflation.ahead ~.,
    data = RF_data %>% na.omit(),
    oob.error = T,
    # Parameters below are set previously
    mtry = mtry,
    num.trees = num.trees,
    min.bucket = min.bucket)

  # Prediction
  new = RF_data %>% select(-inflation.ahead) %>% tail(1)
  forecast1[a] = predict(RF, data = new)
}

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

```

RF forecast



Using 4 lags of all variables

```
n_lags = 0

# Rolling window forecasting
rolling_window <- 492

# Random Forest parameters
p = (1+n_lags)*ncol(data) # number of variables
mtry = ((1/3)*p) %>% round() # number of variables randomly selected
num.trees = 500 # number of trees
min.bucket = 5 # minimal number of observations in each leave (terminal node)

set.seed(1430)

forecast1 = list()

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

  # get the window for training the model
  train = data[a:(a+rolling_window-1), ] %>% select(-CPIAUCSL)
  train_cpi = data[a:(a+rolling_window-1), ] %>% select(CPIAUCSL)
```

```

# embed
RF_data = my_embed(train, n_lags = n_lags)
cpi_lags = my_embed(train_cpi, n_lags = 4)
# bind the embeded columns with the one-step-ahead inflation
RF_data = bind_cols(inflation.ahead = lead(inflation[a:(a+rolling_window-1)]),
                    cpi_lags, RF_data)

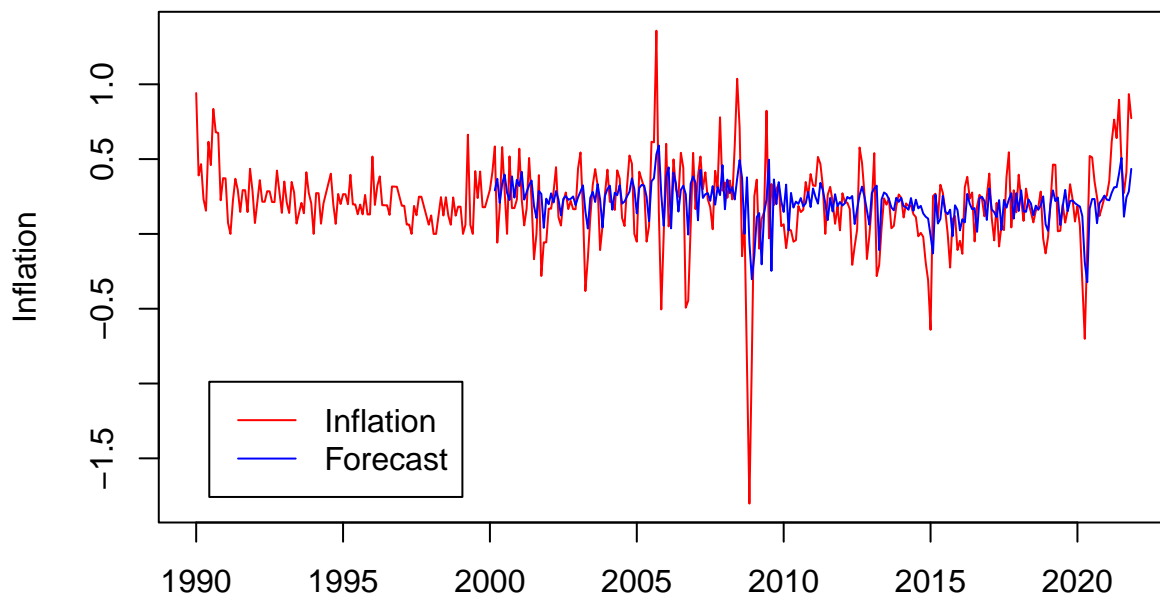
# Random forest estimation
RF = ranger(inflation.ahead ~.,
            data = RF_data %>% na.omit(),
            oob.error = T,
            # Parameters below are set previously
            mtry = mtry,
            num.trees = num.trees,
            min.bucket = min.bucket)

# Prediction
new = RF_data %>% select(-inflation.ahead) %>% tail(1)
forecast1[a] = predict(RF, data = new)
}

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

```

RF forecast



Using 4 lags of CPI and no lags of other variables

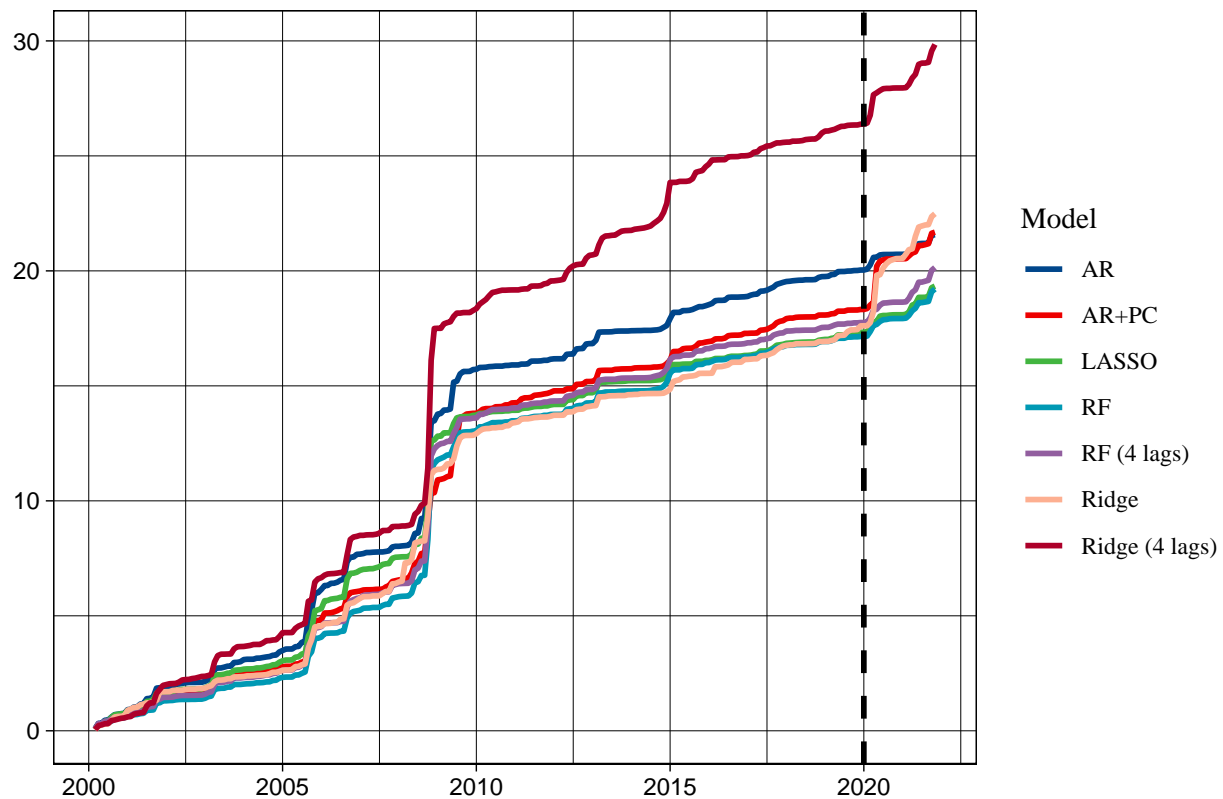
Item F

```

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

```

Cumulative squared errors



```

cum_error %>%
  tail(1) %>%
  select(-date) %>%
  mutate_all(~./tail(cum_error, 1)$AR) %>%
  t() %>%
  data.frame() %>%
  setNames("MSE") %>%
  kbl(digits = 4, booktabs = T)

```

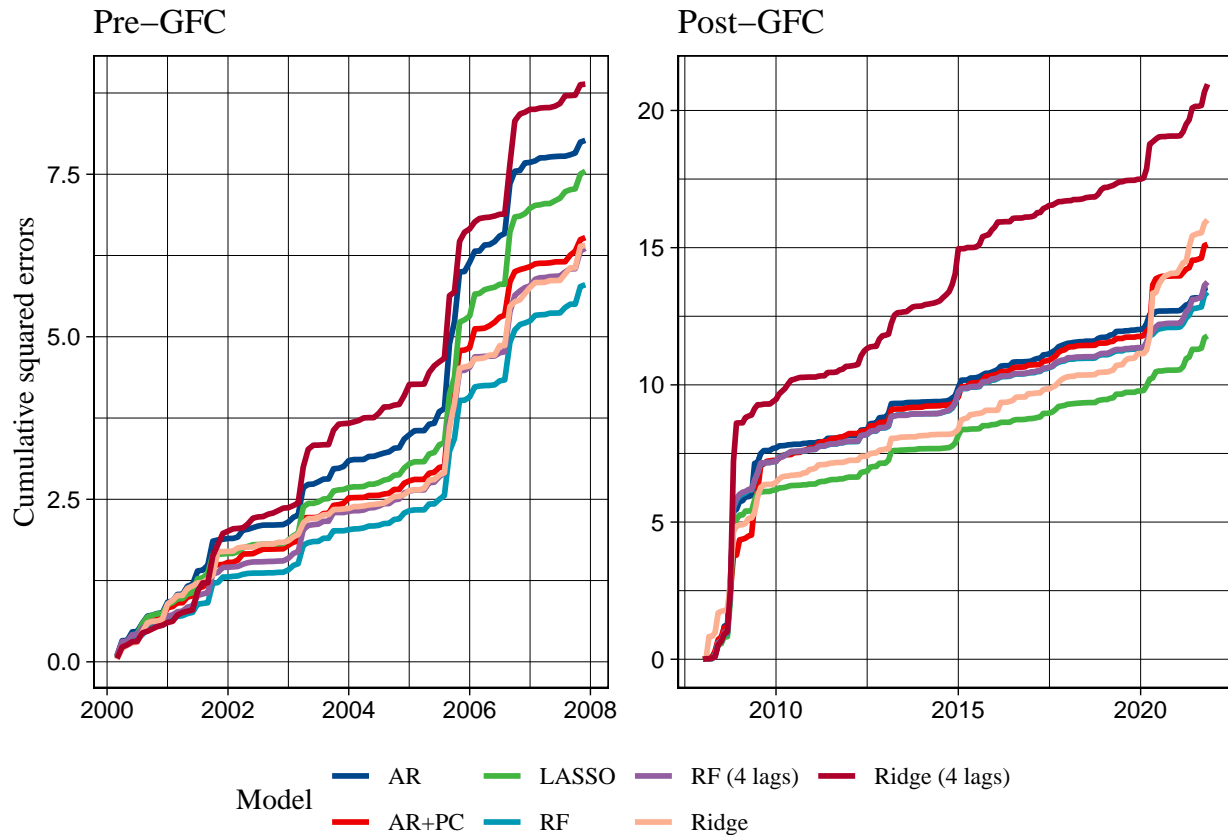
	MSE
AR	1.0000
AR+PC	1.0051
Ridge (4 lags)	1.3836
Ridge	1.0422
LASSO	0.8964
RF (4 lags)	0.9338
RF	0.8905

```
# Plot for subsample
split = 2008

plot_pre = cum_error %>%
  filter(year(date) < split) %>%
  gather(key = model, value = error, -date) %>%
  ggplot(aes(x = date, y = error, color = model)) + geom_line(size = 1) +
  scale_color_lancet() + labs(title = "Pre-GFC", color = "Model",
  y = "Cumulative squared errors", x = NULL)

plot_post = cum_error %>%
  filter(!(year(date) < split)) %>%
  mutate_at(vars(-date), ~. - first(.)) %>%
  gather(key = model, value = error, -date) %>%
  ggplot(aes(x = date, y = error, color = model)) + geom_line(size = 1) +
  scale_color_lancet() + labs(title = "Post-GFC", color = "Model",
  y = NULL, x = NULL)

lemon::grid_arrange_shared_legend(plot_pre, plot_post, ncol = 2)
```



```
# Save
write.csv(forecasts, file = "output/forecasts.csv")
write.csv(cum_error, file = "output/cum_error.csv")
```

Item E

```
forecasts = forecasts %>%
  bind_cols(cum_error$date) %>%
  setNames(c("AR", "AR+PC", "Ridge (4 lags)", "Ridge", "LASSO",
    "RF (4 lags)", "RF", "date"))
```

```
# Inflation ts to data.frame
forecasts = data.frame(inflation = inflation) %>%
  bind_cols(date = zoo::as.Date.yearmon(time(inflation))) %>%
  inner_join(forecasts, by = "date")
```

```
# Plot forecasts
```

```
forecasts %>%
  select(-`Ridge (4 lags)`) %>%
  gather(key = model, value = pred, -date, -inflation) %>%
  gather(key = type, value = value, -date, -model) %>%
  ggplot(aes(x = date, y = value, color = type)) + geom_line() +
  labs(color = NULL, y = NULL, x = NULL) + scale_color_manual(values = c("black",
    "#ED0000FF"), labels = c("Inflation", "Forecast")) + facet_wrap(~model,
    ncol = 2) + theme(legend.position = "top")
```


— Inflation — Forecast

