Panel Data Econometrics: Common Factor Analysis for Empirical Researchers

R software companion guide

Steven Parker

February 2020

Contents

1 Basics

The purpose of this guidebook is to enable R users the ability to use the tools and techniques discussed in the book ***Panel Data Econometrics: Common Factor Analysis for Empirical Researchers*** (Sul, 2019). R is a free software for statistical analysis available from www.r-project.org (R Core Team, 2017). The R workflow is based upon the idea of replicability and R's use is growing in economics. See Racine (2017) for a discussion of the workflow as it applies to economics.

The first 2 chapters are not intended to teach R but provide a minimal introduction to some packages and instructions which were used to edit the data into usable formats. A good introductory tutorial on R can be found here Wickham and Grolemund (2016). The remaining chapters are a rough/basic guide for replicating the applications from the text. Importantly, this guidebook only helps you implement the software that is provided with the text on the text's website in R. The text is still needed to understand many of the details.

You will need to load the accompanying package **`PDEwCF`**. In the first instance, to load the **`PDEwCF`** package you will need to install if from the local drive using `install` then selecting `package archive file`. Then navigate to the location of the **`PDEwCF`**tar file. Alternatively, the command `install.packages("location/PDEwCF_0.1.0.tar.gz", repos = NULL, type = "source")` can be used from the cursor, where location is the *location* of the file. The PDEwCF package contains the data in `RData` format and direct ports of all the functions created by *Donggyu Sul* for his book *Panel Data Econometrics: Common Factor Analysis for Empirical Researchers*.

1.1 Loading CSV data

This is a package name **`package`** and this is a `command` or `result` from R.

The following code snippet shows how to load the data and organsie it for graphing. Note that the data is now in CSV format so that we can use **readr**. First we need to load **readr** into the workspace with the `library` command. This makes the function `read_csv` available. The input to this function is the location of the file and the number of rows to skip when reading the file into the workspace. The symbol `<-` means assign.

The data we load is the personal consumption expenditure (PCE) price index that has been produced by the Bureau of Economic Analysis (BEA). The 'MATn46_t39.csv' includes annual PCE prices for 46 detailed items from 1978 to 2016 (n = 46, T = 39). This data is used in Chapter 3.

```r
library(readr)
mat <- read_csv("data/MATn46_t39.csv", skip = 2)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `%` = col_logical(),
##   `@` = col_logical()
## )

## See spec(...) for full column specifications.
```

The notifications above tell us that the data was mostly considered of double type (numeric). This should be checked as ocassionally missing values are coded in as `NA` and you will need to adjust the `readr::read_csv` command to include `NA`s.

```
## # A tibble: 5 x 8
##   `%`     `?` `Food and nonal~ `Alcoholic beve~ `Food produced ~ Clothing
##   <lgl> <dbl>            <dbl>            <dbl>            <dbl>    <dbl>
```

```
## 1 NA      1978            171.            23.8              1.2    80.2
## 2 NA      1979            190.            26.8              1.3    85.5
## 3 NA      1980            208             30                1.2    91.1
## 4 NA      1981            222.            32.4              1.2    99.4
## 5 NA      1982            231.            34.7              1.1   103.
## # ... with 2 more variables: `Footwear 2` <dbl>, Housing <dbl>
```

A `?` before the name of a function gives information about the function and its inputs. So to obtain information about the `read_csv` function we can type `?read_csv` in the console. After running the above commands the data is loaded into the workspace. The data is in a tibble, a type of dataframe. We also look at the data using the function `head` which shows the first 6 rows by default. The output shows that the some editing is required.

From here we can clean the data as the column names have unnecessary characters. We use the `str_replace` function from the **`stringr`** package. `colnames` is a function that obtains the column names of a dataframe. The first instance obtains the column names while the last instance assigns the updated names. Note that `#` indicates a comment and everything after is ignored. The `str_replace` inputs are the string, find value and the replacement value.

```r
library(stringr)


# rename columns
mat_nms <- colnames(mat)
mat_nms <- str_replace(mat_nms, "\\?", "Year")
mat_nms <- str_replace(mat_nms, "@", "Empty")
mat_nms <- str_replace(mat_nms, "%", "Empty2")
colnames(mat) <- mat_nms
```

We can save this data into R format using the command `save(data, file="location/name.RData")`.

Use the command `load(file="location/name.RData")` to load `.Rdata` files into R.

2 Tidyverse for data manipulations

The tidyverse (Wickham, 2017) is a collection of packages, including those used in chapter 1, that makes it easy to manipulate data. The key package is **`dplyr`** which provides a collection of verbs (i.e. `mutate`, `summarize`, `rename`) for manipulating data. While some of the code looks longer than in the text, but it has an advantage that it is very easy to read. The pipe function `%>%` is used heavily and it tells R *then do this*. Usually you will not need to input the data once the data is referenced in the first line of a series of commands (see the code snippets). The pipe function works with dataframes and tibbles only.

We will start here with a clean workspace.[1]

Load data and packages.[2] We leave the details of the loaded packages shown so that you can see the packages we used, in case there are any future issues with the code.[3] To load data from a package, once the package is loaded, use the `data` command.

```
library(tidyverse)
library(PDEwCF)


# load data
data("MATn46_t39")
# use below if you have converted the data to r already and
# are not using the package data.


# load("data/MATn46_t39.RData")
```

Note the data is now sorted alphabetically. So the outputs are in a slightly different order

[1] Use the `rm` command. `ls()` is list of all objects in the environment.

[2] In this text we only load them once, but if you are working through this in separate sessions you will need to load the packages each session.

[3] Occasionally packages change giving unintended results, e.g. when a package is updated a function no longer exists or compatible. An example would be PHTT for panel data analysis with heterogeneous time trends (Bada and Liebl, 2014).

than in the text.

We show a small snippet of the data below using the `head` command. In this chapter we will show how to go from the csv file to this file.

```
head(MATn46_t39)
```

```
## # A tibble: 6 x 3
##    Year inflation                                                          price
##   <dbl> <chr>                                                              <dbl>
## 1  1978 Food and nonalcoholic beverages purchased for off-premises consum~  171.
## 2  1979 Food and nonalcoholic beverages purchased for off-premises consum~  190.
## 3  1980 Food and nonalcoholic beverages purchased for off-premises consum~  208
## 4  1981 Food and nonalcoholic beverages purchased for off-premises consum~  222.
## 5  1982 Food and nonalcoholic beverages purchased for off-premises consum~  231.
## 6  1983 Food and nonalcoholic beverages purchased for off-premises consum~  239.
```

2.1 Manipulating the data with dplyr

In this section we rely on the pipe indicator `%>%` from the **`magrittr`** package and the verbs from the **`dplyr`** package for data transformations. The structure of piped command is data use a pipe use a command use a more commands.

Once again we load the PCE data into the work space and rename the columns (As shown in chapter 1). This time we dont need to call the individual libraries as they are part of the `tidyverse` package. The data in mat will look like what was seen previously and the first 5 entries of mat is shown.

```r
mat <- read_csv("data/MATn46_t39.csv", skip = 2)
# rename columns
mat_nms <- colnames(mat)
mat_nms <- str_replace(mat_nms, "\\?", "Year")
mat_nms <- str_replace(mat_nms, "@", "Empty")
mat_nms <- str_replace(mat_nms, "%", "Empty2")
colnames(mat) <- mat_nms
```

```
## # A tibble: 5 x 8
##   Empty2  Year `Food and nonal~ `Alcoholic beve~ `Food produced ~ Clothing
##   <lgl>  <dbl>            <dbl>            <dbl>            <dbl>    <dbl>
## 1 NA      1978             171.             23.8              1.2     80.2
## 2 NA      1979             190.             26.8              1.3     85.5
## 3 NA      1980             208              30                1.2     91.1
## 4 NA      1981             222.             32.4              1.2     99.4
## 5 NA      1982             231.             34.7              1.1    103.
## # ... with 2 more variables: `Footwear 2` <dbl>, Housing <dbl>
```

In the following code we use the verb `select` to select a column which contains the text "empty". In Chapter 1 we renamed some extra columns empty so here we wish to remove them as the there is a `-` sign before the `contains` command. Then next we wish to `gather` the data. This creates a long panel, or stacked panel. The `key` is the value to stack on, `value` is the name of the new variable created holding our data values (prices in this case), and we do not wish to include year, so we negate it with `-`. This `gather` command is similar to reshape in *Stata* or *Matlab*. The opposite of `gather` is `spread`. When we gather the data it is called narrow while spread data is called wide (Wide data is like a matrix of T rows and N columns).

Table 1: The inflation data in narrow format

Year	inflation	price
1978	Food and nonalcoholic beverages purchased for off-premises consumption	171.1
1979	Food and nonalcoholic beverages purchased for off-premises consumption	190.3
1980	Food and nonalcoholic beverages purchased for off-premises consumption	208.0
1981	Food and nonalcoholic beverages purchased for off-premises consumption	221.7
1982	Food and nonalcoholic beverages purchased for off-premises consumption	231.3

```
# make the data tidy (narrow format)
MATn46_t39 <- mat %>%
  dplyr::select(-contains("Empty")) %>%
  gather(key = inflation, value = price, -Year)
```

2.2 Summarizing data

In the following code snippet we use the verbs groupby and summarise to summarise the data by groups. In this case we want to see a summary of the data by time. To see by inflation component we would use `group_by(inflation)`.

```
by_time <- MATn46_t39 %>%
  group_by(Year) %>%
  summarise(Mean = mean(price),
            Median = median(price),
            Min = min(price),
            Max = max(price),
            Std.dev = sd(price),
            IQR = IQR(price),
            N = n())
```

Table 2: Yearly summary of the inflation data

Year	Mean	Median	Min	Max	Std.dev	IQR	N
1978	30.86	17.15	0.2	194.5	41.20	23.38	46
1979	34.39	18.45	0.4	218.0	46.25	25.85	46
1980	38.00	20.20	0.7	246.6	51.80	25.83	46
1981	42.09	22.00	0.8	277.9	57.36	27.10	46
1982	44.98	23.10	1.0	303.0	61.27	28.52	46

2.3 Plotting

We can plot the data using `ggplot`. In this example I select the first 4 subindices using the `filter` command. The symbol `|` means or and `==` is a test of equality. Other useful symbols are `!=` not equal and `&` for and. The filter command is very flexible for data manipulation for groups. With these 4 series we plot them as a time-series. Time series are best plot with lines so we use the `geom_line`.To plot the data you need to use `aes` for the aesthetics. In this case I assign Year to the x axis, the value of the index to the y axis and the inflation component as the color. `theme_minimal` removes some chart junk.

```
# plot a subsample of the data
MATn46_t39 %>%
  filter(inflation == "Food and nonalcoholic beverages purchased for off-premises consum
           inflation == "Alcoholic beverages purchased for off-premises consumption" |
           inflation == "Food produced and consumed on farms"  |
           inflation ==  "Clothing" ) %>%
  ggplot() +
  geom_line(aes(x = Year, y = price, color = inflation)) + theme_minimal()
```

`ggplot` allows for a large amount of customization to the graph objects through `theme` commands.

We can also use other chart types such as bar charts (`geom_bar`) and histograms

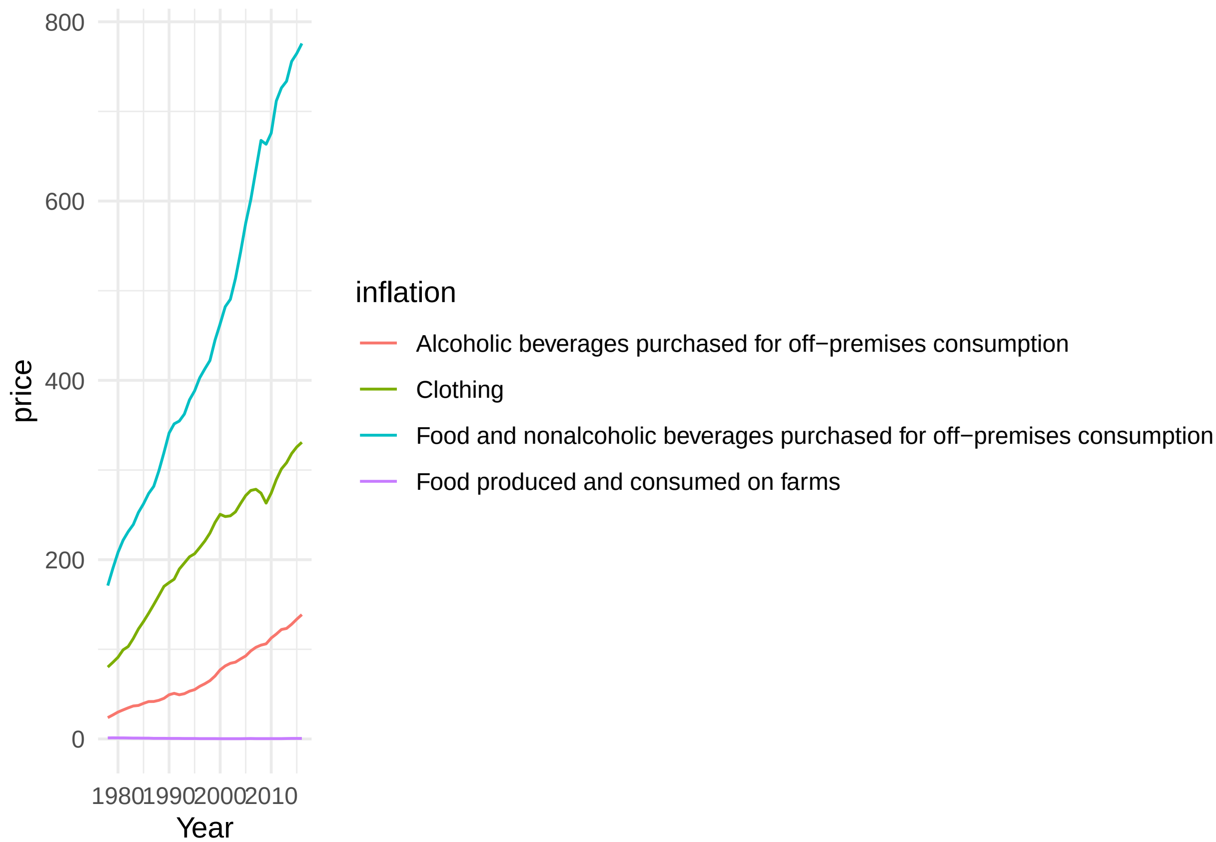


Figure 1: Quick time series plot of some inflation series

(`geom_histogram`) for example, but below we demonstrate the use of kernel densities with `geom_density`.

```
MATn46_t39 %>%
  filter(inflation == "Food and nonalcoholic beverages purchased for off-premises consum
           inflation == "Alcoholic beverages purchased for off-premises consumption" |
           inflation == "Food produced and consumed on farms"  |
           inflation ==  "Clothing" ) %>%
  ggplot() +
  geom_density(aes(x = price, color = inflation)) +
  theme_minimal() +
  facet_wrap(~inflation, scales = "free")  + # creates a grid of plots
  theme(legend.position = "none") # removes legend
```

2.4 Some tips about dplyr verbs

The **dplyr** verbs `select` and `lag` should be prefaced with `dplyr::` so that in the case of other packages that use these verbs as functions the verbs you need from **dplyr** will be used rather than those from other packages. These are the 2 main cases of I have found. When the packages load, the conflicts warning provides some ideas of potential clashes (see warnings and messages above).

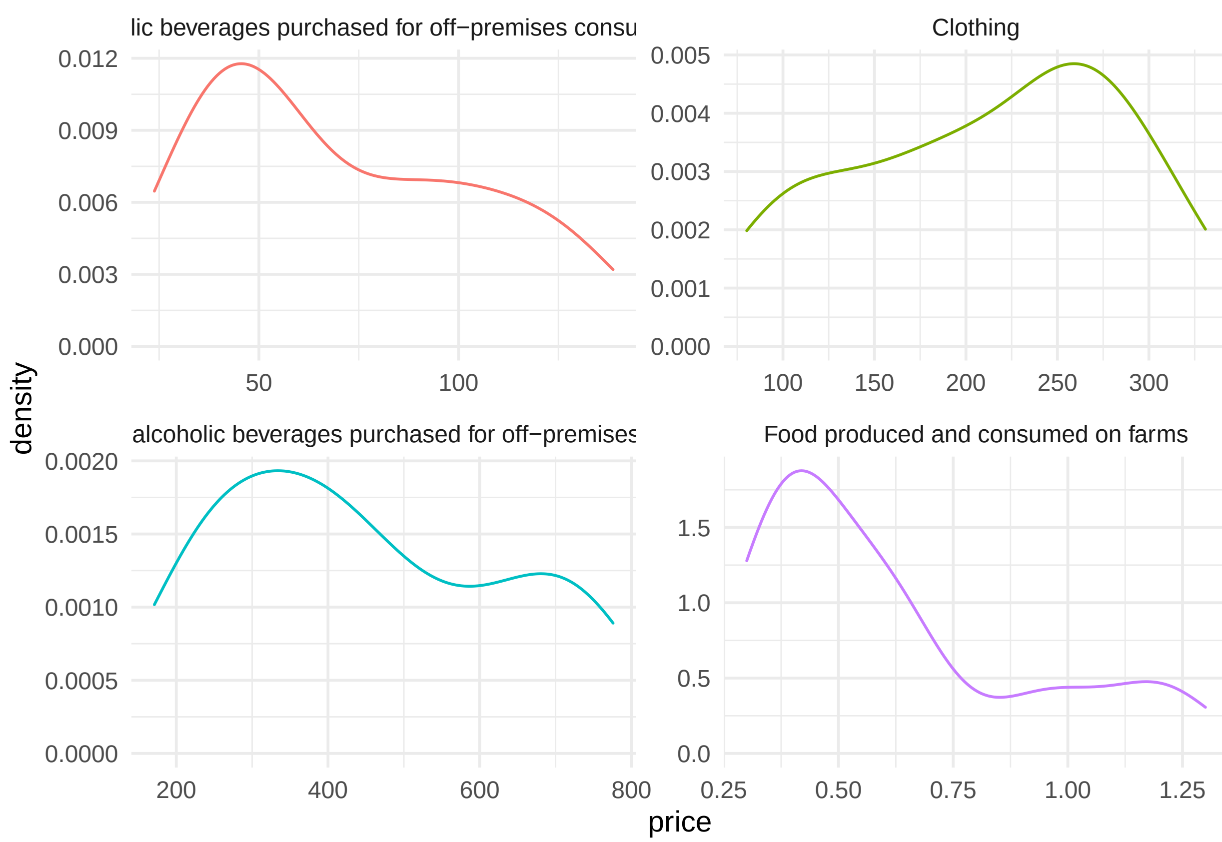


Figure 2: Quick density plot of some inflation series

3 Factor number identification

In this chapter we replicate the practice exercises at the end of the chapter. In terms of coding we introduce the loop programming structure and functions.

3.1 Crime rates

We start with the first practice exercise which uses the crime dataset. We load the data and set up a list (`panel_list`) for our loop. The data are shown. The labels in `panel_list` match those in the column `Crime_Type`.

```
# load the data
# load("data/crime_data.RData")
data("crime_data")
panel_list <- c("violent", "murder", "robbery",
                "rape", "assault", "property",
                "burglary", "larceny", "motorv")
```

Inspect the data with `head(crime_data)`.

```
## # A tibble: 5 x 5
##    Year Crime_Type State   lncrime    id
##   <dbl> <chr>      <chr>     <dbl> <int>
## 1  1965 violent    Alabama    8.84     1
## 2  1966 violent    Alabama    9.00     1
## 3  1967 violent    Alabama    9.04     1
## 4  1968 violent    Alabama    9.02     1
## 5  1969 violent    Alabama    9.09     1
```

We then create the variables named in table 3.1 of the text (Sul, 2019) and a list of their names. The data is spread

```
crime_data <- crime_data %>%
  group_by(Crime_Type, State) %>%
  mutate(dlncrime = lncrime - dplyr::lag(lncrime, 1),  # dy
         lncrime_plus = lncrime/sd(lncrime, na.rm = TRUE), # y+
         dlncrime_plus = dlncrime/sd(dlncrime, na.rm = TRUE)) %>%  # dy+
  ungroup() %>%
  gather(key = variable, value = vals, -Year, -Crime_Type, -State, -id) %>%
  dplyr::select(-id) %>%
  spread(key = State, value = vals) %>%
  arrange(Crime_Type)

var_list <- c("lncrime", "lncrime_plus", "dlncrime", "dlncrime_plus")
```

```
## # A tibble: 8 x 8
##    Year Crime_Type variable      Alabama  Alaska Arizona Arkansas California
##   <dbl> <chr>      <chr>           <dbl>   <dbl>   <dbl>    <dbl>      <dbl>
## 1  1965 assault    dlncrime       NA      NA      NA       NA          NA
## 2  1965 assault    dlncrime_plus  NA      NA      NA       NA          NA
## 3  1965 assault    lncrime         5.00    4.44    4.74     4.56        4.96
## 4  1965 assault    lncrime_plus   15.5     8.91   14.1     12.3        12.9
## 5  1966 assault    dlncrime        0.175 -0.0359  0.0720    0.195       0.107
## 6  1966 assault    dlncrime_plus   1.74  -0.290   0.777     2.09        1.40
## 7  1966 assault    lncrime         5.18    4.41    4.81     4.76        5.07
## 8  1966 assault    lncrime_plus   16.0     8.84   14.3     12.8        13.2
```

In the following code snippet we set up a matrix called `store_crime_ic` to store the output from the two loops. The purpose of the loops is loop through each panel then each variable and then apply the `BaiNgIC()` function to find the number of factors for each variable. The loop structure in R uses the command `for(i in 1:n)`, telling the counter to go from 1 to n, where `:` is a sequence indicator. The `{ }` group the code to be run through the loop. An example is shown below.

```
for(i in 1:5){
  # some code here
  print(i) # prints loop counter i
}
```

R also provides an alternative to loops: the `apply` function. Details can be found with `?apply`. The following code has comments explaining each line. To access output from a function which has multiple outputs we use the `$`. In the code below, `BaiNgIC(dat)$ic2` access the Bai Ng information criteria number 2.

```
store_crime_ic <- matrix(NA, nrow = length(panel_list), ncol = 4)

# loop through the crime panels
for(i in 1:length(panel_list)){
  # use this value in the next loop to filter the data
  filt_val = panel_list[i]
  # loop through the four variables
  for(v in 1:length(var_list)){
    # get the variable to use for input
    v_val = var_list[v]

    # organize the data to fit the input for the function BaiNgIC
```

```
    dat <- crime_data %>%
      # jointly filter the panel and variable
      filter(Crime_Type == filt_val & variable == v_val) %>%
      # remove unnecessary columns
      dplyr::select(-Year, -Crime_Type, -variable) %>%
      na.omit()    # remove NA values

# find the number of factors based on the IC2 information criteria and store it
    store_crime_ic[i, v] <- BaiNgIC(dat)$ic2  # this is a scalar output
  } # close loop v

} # close loop i

tab31 <- data.frame(store_crime_ic)
colnames(tab31) <- var_list
tab31 <- cbind(panel_list, tab31)
```

The output from this code is Table 3.1 of the text. There is a small difference in column 4 (Δy_{it}), the violent factor number is 2 instead of 1 as in the text.

3.2 Price indices

First load the data. This data is called `PCE.csv` in the text. We load the version which is included in the package using `data()` we worked on early.

```
# load the data
data("MATn46_t39")
```

Table 3: Table 3.1

Series	y_{it}	y_{it}^{+}	Δy_{it}	Δy_{it}^{+}
violent	6	8	2	1
murder	5	2	3	1
robbery	7	7	3	1
rape	7	6	1	1
assault	6	8	1	1
property	7	7	1	1
burglary	8	8	2	1
larceny	6	7	1	1
motorv	8	8	1	1

```
inflation_data <- MATn46_t39 %>%
  group_by(inflation) %>%
  mutate(dp = (price - dplyr::lag(price))/ dplyr::lag(price), # create inflation
      dp1 = (dp - lag(dp)  ), # whiten data
      pdp = dp/sd(dp, na.rm = TRUE), # standardize
      ddp = dp1/sd(dp1, na.rm = TRUE)) %>% # standardize whitened data
  ungroup()
```

To undertake the rolling window analysis we set up a function. The structure is name of the function then keyword `function` with the required inputs in `()`. Setting an input = to a value, e.g `roll_window = 25` sets a default value. Just like loops, the internal code is found between `{}`. This function repeats much of the material in the previous loops but adds a new filtering proecess based on the years. As such this function is set up requiring a `start_year` input. We follow the text and use 1978, and window size `roll_window = 25`. Importantly, the last line of code is a `return()`, this returns the output you want.

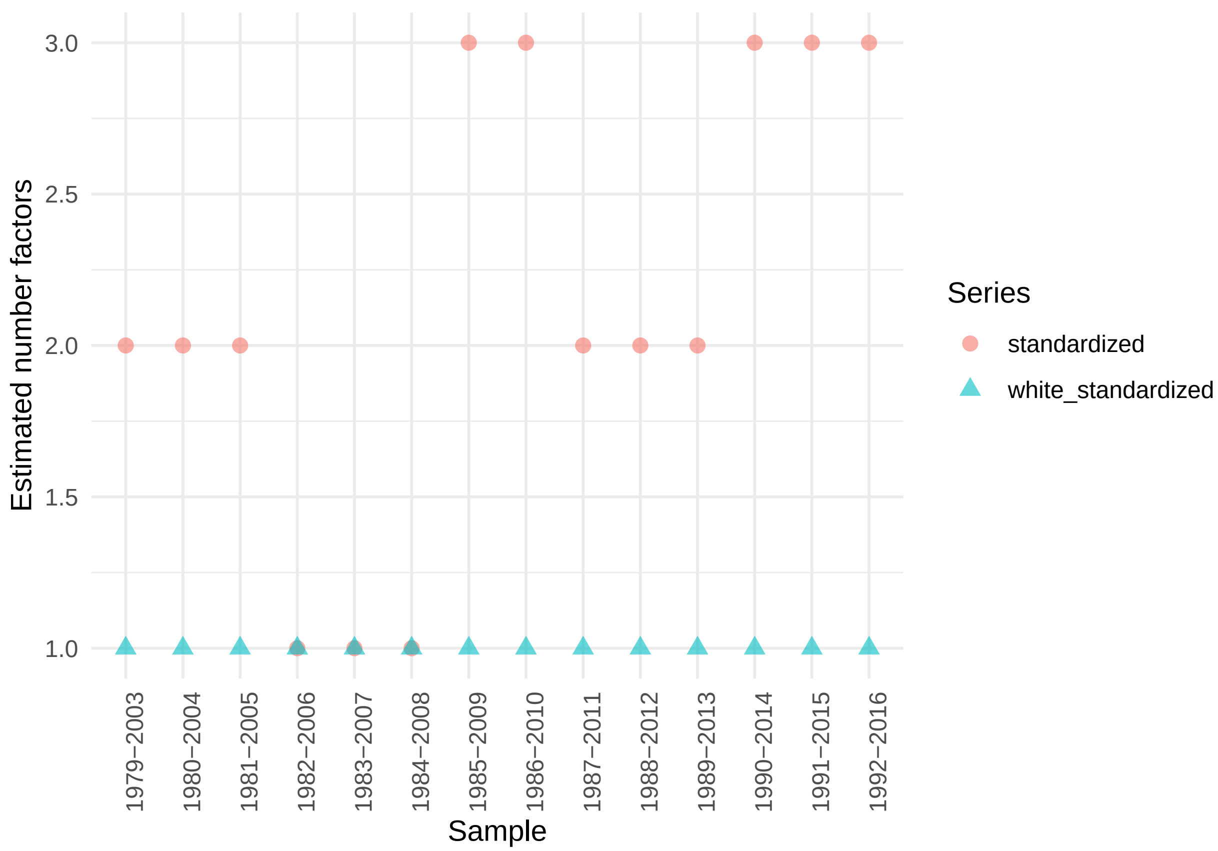
```
# undertake a window analysis


ic2_window_analysis <- function(xdat, start_year, roll_window = 25, white_coef = 0.5){
```

```
store_dxs <- as.numeric() # for results
store_xxs <- as.numeric()
year_labs <- as.character()
labstore <- as.character()
maxyear <- max(xdat$Year, na.rm = TRUE)
WW <- maxyear - roll_window - start_year + 1
xdat2 <- xdat %>%
  dplyr::select(Year, inflation, price) %>%
  group_by(inflation) %>%
  # then compute measures
  mutate(dp = (price - dplyr::lag(price))/ dplyr::lag(price)) %>% # create inflation
  ungroup()

for(w in 1:(WW)){
  Year_1 = start_year + w
  Year_25 = Year_1 + roll_window - 1
  year_labs <- paste0(as.character(Year_1), "-", as.character(Year_25))

  # get years of data
  xdat3 <- xdat2 %>%
    filter(between(Year, Year_1, Year_25))  %>%
    group_by(inflation) %>%
    mutate(dx = dp - (white_coef * lag(dp, 1)), # whiten data
           xxs = dx/sd(dx, na.rm = TRUE),       # standardize whitened data
           dxs = dp/sd(dp, na.rm = TRUE)) %>%   # standardize
    ungroup()
  xdat4 <- xdat3 %>%
```

```
      dplyr::select(Year, inflation, dxs) %>%
      spread(key = inflation, value = dxs) %>%
      na.omit() %>%
      dplyr::select(-Year)

    # the ic
    store_dxs[w] <-  BaiNgIC(xdat4)$ic2

    xdat5 <- xdat3 %>%
      dplyr::select(Year, inflation, xxs) %>%
      spread(key = inflation, value = xxs) %>%
      na.omit() %>%
      dplyr::select(-Year)
    store_xxs[w] <-  BaiNgIC(xdat5)$ic2
    labstore[w] <- year_labs

  } # w
  out <- cbind.data.frame(sample = labstore,
                          standardized = store_dxs,
                          white_standardized = store_xxs)
  return(out)
}  # close function
```

Here we use the function to replicate the text figure 3.4. `gather` the `ic2_robust` data for convenience as it allows the aesthetics (`aes`) to be easily applied.

```r
ic2_robust <- ic2_window_analysis(inflation_data, 1978, 25)


ic2_robust %>%
  gather(key = Series, value = number_of_factors, -sample) %>%
  # -sample leaves sample as its own column
  ggplot() +
  geom_point(aes(x = sample, y = number_of_factors,
                 shape = Series, color = Series), size = 2.5, alpha = 0.6) +
  # some minor dressing
  theme_minimal() +
  labs(y = "Estimated number factors", x = "Sample") +
  theme(axis.text.x=element_text(angle=90,hjust=1))
```



4 Decomposition of panel

In this chapter we practice with common factors estimation.

The function `pc` operates exactly as the Matlab version in the text, which is explained in section 4.6.1.

4.1 Principal component estimation

Load the data if needed using `data(MATn46_t39)`.

```
# prepare data for analysis
inflation_data <- MATn46_t39 %>%
  group_by(inflation) %>%
  mutate(dp = (price - dplyr::lag(price))/ dplyr::lag(price), # create inflation
       dp1 = (dp - lag(dp)  ), # whiten data
       pdp = dp/sd(dp, na.rm = TRUE), # standardize
       ddp = dp1/sd(dp1, na.rm = TRUE)) %>% # standardize whitened data
  ungroup()
```

```
## # A tibble: 4 x 7
##    Year inflation                                  price      dp     dp1   pdp    ddp
##   <dbl> <chr>                                      <dbl>   <dbl>   <dbl> <dbl>  <dbl>
## 1  1978 Food and nonalcoholic beverages purc~  171. NA       NA       NA    NA
## 2  1979 Food and nonalcoholic beverages purc~  190.  0.112   NA        4.74 NA
## 3  1980 Food and nonalcoholic beverages purc~  208   0.0930 -0.0192   3.93 -0.887
## 4  1981 Food and nonalcoholic beverages purc~  222.  0.0659 -0.0271   2.78 -1.25
```

Note, the data is now sorted by the inflation components so the output is in a different order than the text. We use the `BaiNgIC` from the previous chapter to find the number of factors the whitened data.

```r
# Spread the data wide
ddp <- inflation_data %>%
  dplyr::select(Year, inflation, ddp) %>%
  na.omit() %>%
  spread(key = inflation, value = ddp) %>%
  dplyr::select(-Year)

# Use Bai Ng information criteria
k <- BaiNgIC(ddp)$ic2
```

There is $k = 1$ factor. Now estimate the factors with `pc` on the standardized inflation rates.

```r
pdp <- inflation_data %>%
  dplyr::select(Year, inflation, pdp) %>%
  na.omit() %>%
  spread(key = inflation, value = pdp) %>%
  dplyr::select(-Year)

# pc estimation
fact.est.pdp <- pc(pdp, k)

f.pdp <- fact.est.pdp$f1      # factor
l.pdp <- fact.est.pdp$lambda1 # loadings
e.pdp <- fact.est.pdp$v1      # R2
v.pdp <- fact.est.pdp$e1      # fixed effect + idiosyncratic term
```

4.2 Standardisation and estimation of PC factors

In this section we use a new data set. The data is from the 'int99.cvs' file which has the one-month interest rates for 28 industrial countries from January 1999. to November 2015. Load the data.

```
data("MATint99")
```

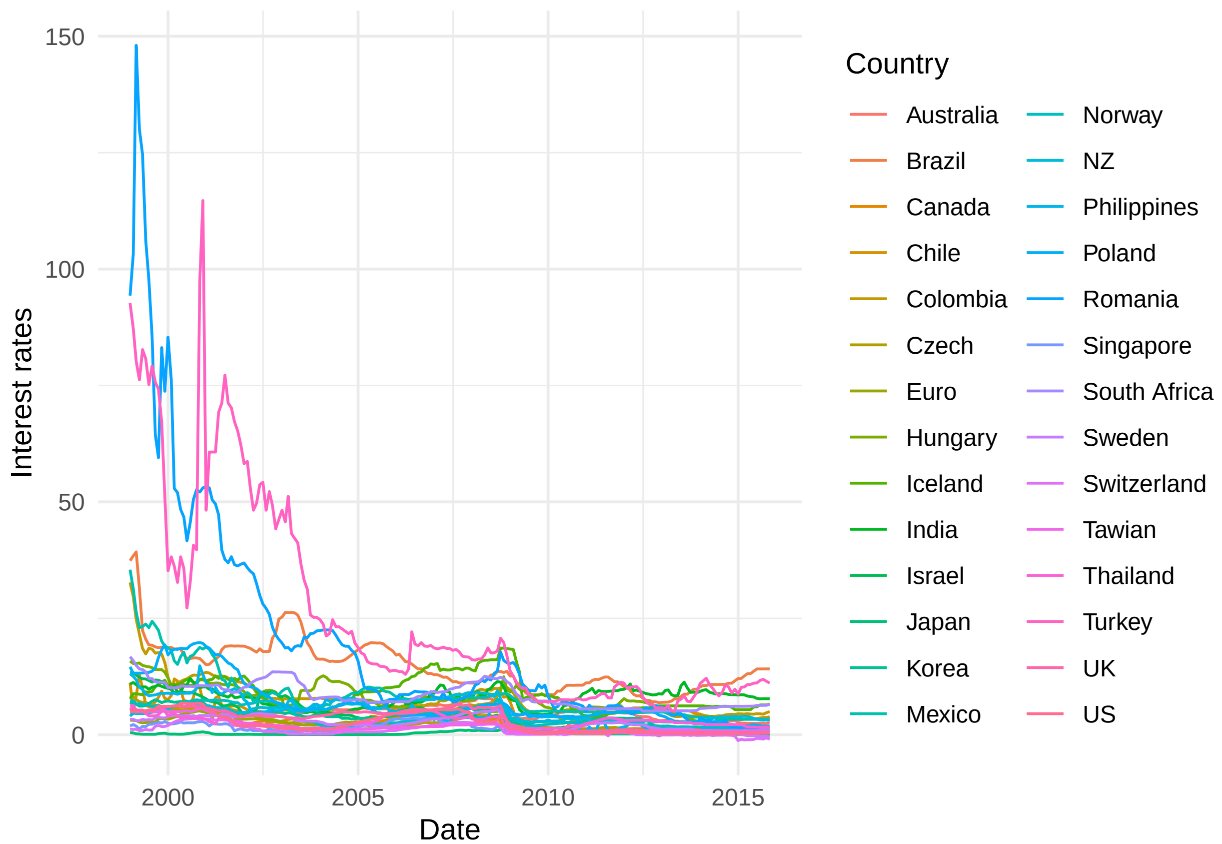


Figure 3: Interest rate data

```
# find country with greatest variance
var.big <- MATint99 %>%
  group_by(Country) %>%
  mutate(var= var(Interestrate),
```

```r
        n = n()) %>%
  ungroup() %>%
  mutate(max = ifelse(max(var) == var, 1, 0)) %>%
  filter(max == 1) %>%
  dplyr::select(Country) %>%
  unique() %>%
  as.character()

# raw data
sq <- MATint99 %>%
  dplyr::select(Date, Country, Interestrate) %>%
  na.omit() %>%
  spread(key = Country, value = Interestrate) %>%
  dplyr::select(-Date)

fact.est.sq <- pc(sq, 1, stand = FALSE)

# repeat with standardization
sq1 <- MATint99 %>%
  group_by(Country) %>%
  mutate(sq1 = Interestrate / sd(Interestrate)) %>%
  ungroup() %>%
  dplyr::select(Date, Country, sq1) %>%
  na.omit() %>%
  spread(key = Country, value = sq1) %>%
  dplyr::select(-Date)
```

```r
# pc estimation standarized data
fact.est.sq1 <- pc(sq1, 1)


Label.int.rate <- paste0("Interest rate - ", var.big)
# prepare data for plotting
ff.dta <- MATint99 %>%
  filter(Country == var.big) %>%
  bind_cols(f1 = -fact.est.sq$f1, f2 = -fact.est.sq1$f1) %>%
  dplyr::select(-Country) %>%
  gather(key = col, value = ff, -Date) %>%
  group_by(col) %>%
  mutate(ff =  ff- mean(ff),
         ff = ff/sd(ff),
  # Labels for plots
         col = ifelse(col == "Interestrate",
                      Label.int.rate, ifelse(col == "f2",
                                             "PC Factor after standardization",
                                             "PC Factor before standardization"))) %>%
  ungroup()


# Plot both cases here
ff.dta %>%
  ggplot() +
  geom_line(aes(x = Date, y = ff, color = col)) +
  geom_point(aes(x = Date, y = ff, color = col, shape = col)) +
  theme_minimal() +
  labs(y = "") + theme(legend.title = element_blank())
```

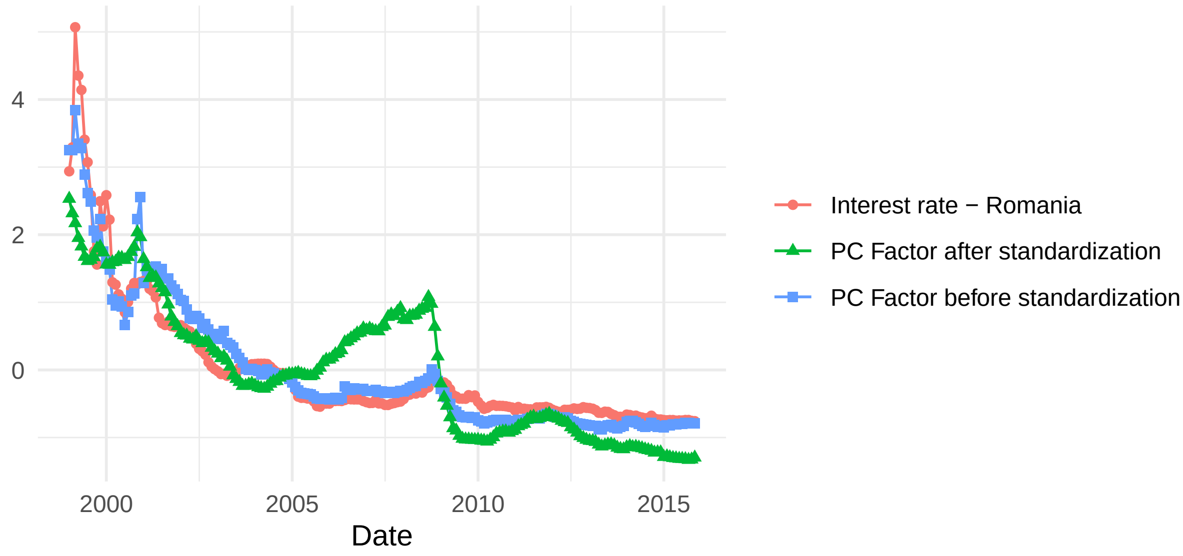


Figure 4: Figure 4.5

Note the Matlab and R code standardize the data inside the program, so figure 4.5 in the text may be different from the reproduction here.

5 Identification of common factor

5.1 Identifying common factors

We are back to our familiar inflation dataset. We prepare in the same manner as previously. We use new names with the created variables to keep it aligned with the text. You will also need to load this dataset `MATn3_t39`. It has three inflation components—service, durable, nondurable—and is shown below.

```
data("MATn3_t39")
```

```
## # A tibble: 5 x 3
##    Year inflation price
##   <dbl> <chr>     <dbl>
## 1  1978 Durable    90.3
## 2  1979 Durable    96.3
## 3  1980 Durable   105.
## 4  1981 Durable   112.
## 5  1982 Durable   116.
```

We introduce a new function `bind_rows` which appends a dataframe with another. The columns should have the same names. This saves us some processing and we can recover the variables using the list term `index_3_list` in a `dplyr::filter` command.

```
index_3_list <- c("Durable", "Nondurable", "Service")

# prepare data for analysis
inflation_data <- MATn46_t39 %>%
  bind_rows(MATn3_t39) %>%
```

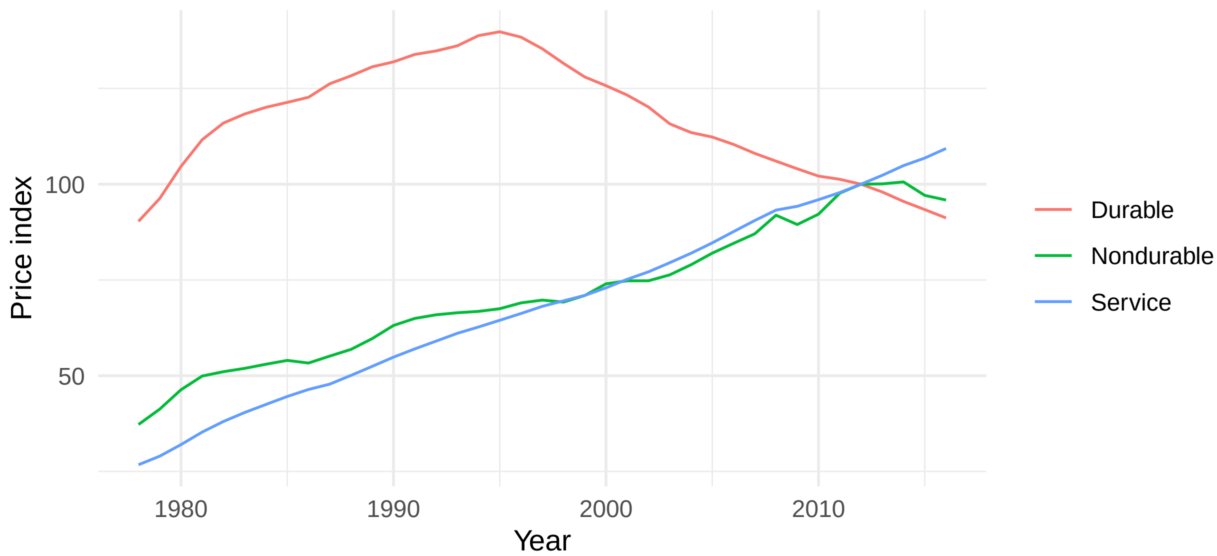


Figure 5: Three sub inflation indices

```
  group_by(inflation) %>%
  mutate(dp = (price - dplyr::lag(price))/ dplyr::lag(price), # create inflation
       d2p = (dp - lag(dp)  ),
       dps = dp/sd(dp, na.rm = TRUE),
       d2ps = d2p/sd(d2p, na.rm = TRUE)) %>%
  rename(p1 = price) %>%
  ungroup()


# spread and find factors
dps <- inflation_data %>%
  dplyr::select(Year, inflation, dps) %>%
  filter(!inflation %in% index_3_list) %>% # remove the extra 3 categories
  na.omit() %>%
  spread(key = inflation, value = dps) %>%
  dplyr::select(-Year)
```

```r
k_dps <- BaiNgIC(dps, 8)$ic2

d2ps <- inflation_data %>%
  dplyr::select(Year, inflation, d2ps) %>%
  filter(!inflation %in% index_3_list) %>% # remove the extra 3 categories
  na.omit() %>%
  spread(key = inflation, value = d2ps) %>%
  dplyr::select(-Year)

k_d2ps <- BaiNgIC(d2ps, 8)$ic2

# for later
n <- ncol(d2ps)
bigT <- nrow(d2ps)
```

5.2 Leadership model

We get the names of the sub-indices to identify which sub-index may be a latent factor and then bind it to the output.

```r
store_bn <- matrix(NA, nrow = n, ncol = 2)
nms <- colnames(d2ps)
for(i in 1:n){
  if(i == 1){
    xx <- d2ps[,2:n]
  }
  if(i >1 & i == n){
```

Table 4: Latent factor identification for inflation indices data

Name	Column	Factors
Accommodations 17	1	1
Alcoholic beverages purchased for off-premises consumption	2	1
Clothing	3	0
Commercial and vocational schools 15	4	1
Educational books	5	1
Financial services	6	1
Food and nonalcoholic beverages purchased for off-premises consumption	7	1
Food produced and consumed on farms	8	1
Food services	9	1
Footwear 2	10	0
Foreign travel by U.S. residents	11	1
Furniture, furnishings, and floor coverings 5	12	1

```
    xx <- d2ps[,-i]
  }
  if(i == n){
    xx <- d2ps[,1:(n-1)]
  }
  x1 <- cbind.data.frame(1,d2ps[,i])
  x1 <- as.matrix(x1)
  prx <- defactor(as.matrix(xx), as.matrix(x1))

  store_bn[i,1] <- i
  store_bn[i,2] <- BaiNgIC(prx, 8)$ic2
}
colnames(store_bn) <- c("Column", "Factors")

out <- cbind.data.frame(Name = nms, store_bn)
```

The robust check is computed below. Note that we now store the subsamples (SS) horizontally.

```
store_bn <- matrix(NA, nrow = n, ncol = 11)
nms <- colnames(d2ps)
for(i in 1:n){
  if(i == 1){
    xx <- d2ps[,2:n]
  }
  if(i >1 & i == n){
    xx <- d2ps[,-i]
  }
  if(i == n){
    xx <- d2ps[,1:(n-1)]
  }
  store_bn[i,1] <- i
  for(j in 1:10){
      x2 <- xx[j:bigT,]
      x3 <- cbind.data.frame(1,d2ps[j:bigT,i])
      x3 <- as.matrix(x3)
      prx <- defactor(as.matrix(x2), as.matrix(x3))

      store_bn[i,j + 1] <- BaiNgIC(prx, 8)$ic2
  } # j
} # i


colnames(store_bn) <- c("Column", paste0("SS", 1:10))
```

Table 5: Robust check for latent factor identification for inflation indices data

Name	Column	SS1	SS2	SS3	SS4	SS5	SS6	SS7	SS8	SS9	SS10
Accommodations 17	1	1	1	1	1	1	1	1	1	1	1
Alcoholic beverages purchased for off-premises consumption	2	1	1	1	1	1	1	1	1	1	1
Clothing	3	0	0	0	0	0	1	1	1	1	1
Commercial and vocational schools 15	4	1	1	1	1	1	1	1	1	1	1
Educational books	5	1	1	1	1	1	1	1	1	1	1
Financial services	6	1	1	1	1	1	1	1	1	1	1
Food and nonalcoholic beverages purchased for off-premises consumption	7	1	1	1	1	1	1	1	1	1	1
Food produced and consumed on farms	8	1	1	1	3	1	1	1	1	1	1
Food services	9	1	1	1	1	1	1	1	1	1	1
Footwear 2	10	0	0	0	1	1	1	1	1	1	1
Foreign travel by U.S. residents	11	1	1	1	1	1	1	1	1	1	1
Furniture, furnishings, and floor coverings 5	12	1	1	1	1	1	1	1	1	1	1

```r
out_robust <- cbind.data.frame(Name = nms, store_bn)
```

The results are slightly different but the interpretation is the same as in the text.

5.3 Multiple variables as single factor

Early we loaded the three subindices (durable, nondurable and services) into R. They are now in the `inflation_data`.

```r
d2p <- inflation_data %>%
  dplyr::select(Year, inflation, d2p) %>%
  filter(!inflation %in% index_3_list) %>%
  na.omit()


md2p <- d2p %>%
  group_by(Year) %>%
  summarise(md2p = mean(d2p))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
d3p <- inflation_data %>%
  dplyr::select(Year, inflation, dp3 = dp, d2p3 = d2p, d23ps = d2ps) %>%
  filter(inflation %in% index_3_list) %>%
  na.omit()
```

Now we use the data to examine the possibility of a common factor.

```
d2p3 <- d3p %>%
  na.omit() %>%
  dplyr::select(Year, inflation, d2p3) %>%
  spread(key = inflation, value = d2p3) %>%
  dplyr::select(-Year) # make as mat

store_m_bn <- matrix(NA, nrow = 10, ncol = 5)

for(j in 1:10){
  store_m_bn[j, 1] <- j
  md2p_mat <- as.matrix(md2p[j:bigT,])
  d2ps_mat <- as.matrix(d2ps[j:bigT,])

# try various combinations

  x1 <- cbind.data.frame(1,d2p3[j:bigT,]) %>% as.matrix() # all three vars as in text
  x12 <- cbind.data.frame(1,d2p3[j:bigT,-3]) %>% as.matrix()
  x13 <- cbind.data.frame(1,d2p3[j:bigT,-2]) %>% as.matrix()
  x23 <- cbind.data.frame(1,d2p3[j:bigT,-1]) %>% as.matrix()

  hx = x1 %*% solve(crossprod(x1)) %*% t(x1) %*% md2p_mat
```

```r
  hx12 = x12 %*% solve(crossprod(x12)) %*% t(x12) %*% md2p_mat
  hx13 = x13 %*% solve(crossprod(x13)) %*% t(x13) %*% md2p_mat
  hx23 = x23 %*% solve(crossprod(x23)) %*% t(x23) %*% md2p_mat

  x1 <- cbind.data.frame(1,hx) %>% as.matrix()
  x12 <- cbind.data.frame(1,hx12) %>% as.matrix()
  x13 <- cbind.data.frame(1,hx13) %>% as.matrix()
  x23 <- cbind.data.frame(1,hx23) %>% as.matrix()

  prx <- d2ps_mat - x1 %*% solve(crossprod(x1)) %*% t(x1) %*% d2ps_mat
  prx12 <- d2ps_mat - x12 %*% solve(crossprod(x12)) %*% t(x12) %*% d2ps_mat
  prx13 <- d2ps_mat - x13 %*% solve(crossprod(x13)) %*% t(x13) %*% d2ps_mat
  prx23 <- d2ps_mat - x23 %*% solve(crossprod(x23)) %*% t(x23) %*% d2ps_mat

  store_m_bn[j, 2] <- BaiNgIC(prx12, 8)$ic2
  store_m_bn[j, 3] <- BaiNgIC(prx13, 8)$ic2
  store_m_bn[j, 4] <- BaiNgIC(prx23, 8)$ic2
  store_m_bn[j, 5] <- BaiNgIC(prx, 8)$ic2
} # j

colnames(store_m_bn) <- c("Sub sample", paste(index_3_list[c(1,2)], collapse = ", "),
                          paste(index_3_list[c(1,3)], collapse = ", "),
                          paste(index_3_list[c(2,3)], collapse = ", "),  "All variables"

out_multiple <- cbind.data.frame(store_m_bn)
```

Table 6: Examination of multiple variables as single factor for inflation data

Sub sample	Durable, Nondurable	Durable, Service	Nondurable, Service	All variables
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	1	1	1
6	1	1	1	1
7	1	1	1	1
8	1	1	1	1
9	1	1	1	1
10	1	1	1	1

6 Static and dynamic relationships

Start by loading the data into the workspace (environment).

```
# load data
data("spot")
data("price")
```

Here we obtain the US price data and join it to the main price data. We join with the `left_join()` command. The tidyverse contains a number of *join* functions based upon SQL join commands. We use a left join, joining by date: we match the dates on the `us_dta` to the dates of the `price` data. We alos do some manipulations to standardiz e the data.

```
# price data
us_dta <- price %>%
  filter(Country == "US") %>%
  dplyr::select(Date, US = price)


p_dta <- price %>%
```

```
  filter(Country != "US") %>%
  left_join(us_dta, by = "Date") %>%
  mutate(p = log(price/US)) %>%
  group_by(Country) %>%
  mutate(dp = p - dplyr::lag(p, 1),
         d2p = dp -  dplyr::lag(dp, 1),
         sdp = dp/sd(dp, na.rm = TRUE),
         sd2p = d2p/sd(d2p, na.rm = TRUE))
```

6.1 Common-dynamic relationship

We follow the method employed in the book here. First we make the individual data matrices.

```
dp_dta <- p_dta %>%
  dplyr::select(Date, Country, dp) %>%
  ungroup() %>%
  spread(key = Country, value = dp) %>%
  na.omit()

d2p_dta <- p_dta %>%
  dplyr::select(Date, Country, d2p) %>%
  ungroup() %>%
  spread(key = Country, value = d2p) %>%
  na.omit()

sdp_dta <- p_dta %>%
  dplyr::select(Date, Country, sdp) %>%
```

```
  ungroup() %>%
  spread(key = Country, value = sdp) %>%
  na.omit()

sd2p_dta <- p_dta %>%
  dplyr::select(Date, Country, sd2p) %>%
  ungroup() %>%
  spread(key = Country, value = sd2p) %>%
  na.omit()

# make the spot rates data
s_dta <- spot %>%
  filter(Date <= max(dp_dta$Date)) %>%
  mutate(s = log(price)) %>%
  group_by(Country) %>%
  mutate(ds = s - dplyr::lag(s, 1)) %>%
  dplyr::select(Date, Country, s) %>%
  ungroup()

euro_dta <- s_dta %>%
  filter(Country == "EURO") %>%
  dplyr::select(Date, EURO = s)

deu_dta <- s_dta %>%
  filter(Country != "EURO") %>%
  left_join(euro_dta, by = "Date") %>%
  mutate(seu = s - EURO) %>%
```

```r
  group_by(Country) %>%
  mutate(deu = seu - dplyr::lag(seu, 1)) %>%
  dplyr::select(Date, Country, deu) %>%
  spread(key = Country, value = deu) %>%
  na.omit()

ds_dta <- spot %>%
  filter(Date <= max(dp_dta$Date)) %>%
  mutate(s = log(price)) %>%
  group_by(Country) %>%
  mutate(ds = s - dplyr::lag(s, 1)) %>%
  dplyr::select(Date, Country, ds) %>%
  spread(key = Country, value = ds)  %>%
  dplyr::select(-US) %>%
  na.omit()

# clean up workspace
rm( "euro_dta", "s_dta", "spot", "us_dta", "price")
```

Now, we estimate the regression coefficients for the cross-section averaged data.

```r
# Estimation the common-dynamic relationship

# csa based factors
mdus = apply(ds_dta[-1], 1, mean)
mdeu =apply(deu_dta[-1], 1, mean)
mdp =apply(dp_dta[-1], 1, mean)
```

Table 7: Beta estimates of the common-dynamic relationship

Factors	Coefs	beta	sig	tstat
2	(Intercept)	-0.002	0.001	-1.335
	mdp	2.559	0.619	4.137
3	(Intercept)	-0.002	0.001	-1.169
	mdp	2.520	0.675	3.731
	mdeu	-0.193	0.081	-2.399

```
dta.1 <- cbind.data.frame(mdus, mdp)

dta.2 <- cbind.data.frame(mdus, mdp, mdeu)

beta_2_facts <- olshac(dta.1, 6)

betas_3_facts <- olshac(dta.2, 6)
```

6.2 Idio-dynamic relationship

First identify the number of factors in the price data.

```
T2 <- dim(dp_dta)[1]

n2 <- dim(dp_dta)[2]


# loop through dataframe in samples of 50, whiten data, find IC2

pp_dta <- p_dta %>%

  dplyr::select(-sdp, -sd2p) %>%

  filter(Date > min(p_dta$Date)) %>%

  group_by(Country) %>%

  mutate(rn = row_number()) %>% # for each country

  ungroup() %>%

  dplyr::select(rn, everything())
```

```r
mx_rn <- max(pp_dta$rn)

ic_store <- matrix(NA, 50, 3)
colnames(ic_store) <- c("Sample", "sdp", "sd2p")

for(i in 1:50){
  tmp_sdp_dta <- pp_dta  %>%
    filter(rn >= i) %>%
    group_by(Country) %>%
    mutate(sdp = dp/sd(dp, na.rm = TRUE)) %>%
    dplyr::select(Date, Country, sdp) %>%
    ungroup() %>%
    spread(key = Country, value = sdp) %>%
    dplyr::select(-Date )

  tmp_sd2p_dta <- pp_dta  %>%
    filter(rn >= i+1) %>%
    group_by(Country) %>%
    mutate(sd2p = d2p/sd(d2p, na.rm = TRUE)) %>%
    dplyr::select(Date, Country, sd2p) %>%
    ungroup() %>%
    spread(key = Country, value = sd2p) %>%
    dplyr::select(-Date )

  ic_store[i,1] <- i
  ic_store[i,2] <- BaiNgIC(tmp_sdp_dta)$ic2
  ic_store[i,3] <- BaiNgIC(tmp_sd2p_dta)$ic2
```

Table 8: Number of factors in the price data

Factors	sdp	sdp2
1	41	50
2	9	

```
}


tab_factors_sdp <- summary(as.factor(ic_store[,2]))
tab_factors_sdp2 <- summary(as.factor(ic_store[,3]))
```

We don't show the full table of `ic_store`, only the summary of the results.

We compute the coefficients for one factor and two factors below.

```
xx1 <- cbind.data.frame(1, mdp, mdus)
xx1 <- as.matrix(xx1)
xtx1 <- solve(crossprod(xx1))
dsc <- as.matrix(ds_dta[,2:ncol(ds_dta)]) - xx1 %*% xtx1 %*% t(xx1) %*% as.matrix(ds_dt
dpc <- as.matrix(dp_dta[,2:ncol(dp_dta)]) - xx1 %*% xtx1 %*% t(xx1) %*% as.matrix(dp_dt
bcce1 <- sum(sum(dsc*dpc))/sum(sum(dpc*dpc))

xx2 <- cbind.data.frame(1, mdp, mdus, mdeu)
xx2 <- as.matrix(xx2)
xtx12 <- solve(crossprod(xx2))

dsc <- as.matrix(ds_dta[,2:ncol(ds_dta)]) - xx2 %*% xtx12 %*% t(xx2) %*% as.matrix(ds_d
dpc <- as.matrix(dp_dta[,2:ncol(dp_dta)]) - xx2 %*% xtx12 %*% t(xx2) %*% as.matrix(dp_d
bcce2 <- sum(sum(dsc*dpc))/sum(sum(dpc*dpc))
```

The results of `bcce1` are 0.71 and `bcce2` are 0.671.

6.3 GHS method

This is a direct port of the GHS three factor code.

```
sds1 <- ds_dta %>%
  gather(key = Country, value = ds, -Date) %>%
  group_by(Country) %>%
  mutate(m = mean(ds),
         sd= sd(ds),
         sds = (ds - m)/sd) %>%
  dplyr::select(Country, Date, sds) %>%
  spread(key = Country, value = sds) %>%
  dplyr::select(-Date) %>% as.matrix()

sdp1 <- dp_dta %>%
  gather(key = Country, value = dp, -Date) %>%
  group_by(Country) %>%
  mutate(sdp = (dp - mean(dp))/sd(dp)) %>%
  dplyr::select(Country, Date, sdp) %>%
  spread(key = Country, value = sdp) %>%
  dplyr::select(-Date) %>% as.matrix()

# get factors
fs <- pc(sds1, 2)$f1
fp <- pc(sdp1, 1)$f1
```

```r
# create
ymat <- ds_dta[ ,-1 ]
xmat <- cbind(1, fs, fp)
y_3 <- defactor(y_mat = ymat, x_mat = xmat)
ymat <- dp_dta[ ,-1 ]
x_3 <- defactor(y_mat = ymat, x_mat = xmat)

be1 <- sum(sum(x_3*y_3))/sum(sum(x_3*x_3))

xmat <- cbind(1, fs, fp, mdp, mdus, mdeu)
ymat <- ds_dta[ ,-1 ]
y_3b <- defactor(y_mat = ymat, x_mat = xmat)

ymat <- dp_dta[ ,-1 ]
x_3b <- defactor(y_mat = ymat, x_mat = xmat)

be3 <- sum(sum(x_3b*y_3b))/sum(sum(x_3b*x_3b))

# Panel Robust t-ratio =====
re <- y_3b - x_3b * be3
re = re * x_3b
xq <- apply(re, 2, sum)
xq <-  crossprod(xq)
invxx <- 1/sum(sum(x_3b * x_3b));
xq <- invxx %*% xq %*% invxx
xq <- sqrt(diag(xq))
tr <- be3 / xq
```

Table 9: GHS estimates of the idio-dynamic relationship

b	tstat
0.531	4.384

```
ghs_3_facts <- cbind(b = be3, tstat = tr)
```

6.4 Iterative PC: Bai's Estimator

The loop to iterate Bai's fixed effects estimator are shown. There is 500 iterations (`iters`) and the tolerance is set to 10^-10 (`tol`).

```
bb <- be3
ds_mat <- as.matrix(ds_dta[ ,-1 ])
dp_mat <- as.matrix(dp_dta[ ,-1 ])


tol <- 10^-10
iters <- 500
for(i in 1:iters){
  rs <- ds_mat - dp_mat * bb
  rss <- stand_mat(rs)
  k3 = BaiNgIC(rss, 8)$ic2
  fr <- pc(rs, k3)$f1
  ff <- cbind(1, fr)
  # ff <- cbind(1, fr, mdp, mdus, mdeu)
  y4 <- defactor(ds_mat, ff)
  x4 <- defactor(dp_mat, ff)
```

```
  be4 <- sum(sum(x4*y4))/sum(sum(x4*x4))
  if((be4 - bb) < tol){
    break
  }
  bb <- be4
  }
```

After i = 19 iterations be4 = 0.749. Now we repeat the the procedure including the cross-section averages starting with `bb = be4`.

```
for(i in 1:iters){
  rs <- ds_mat - dp_mat * bb
  rss <- stand_mat(rs)
  k3 = BaiNgIC(rss, 8)$ic2
  fr <- pc(rs, k3)$f1
  # ff <- cbind(1, fr)
  ff <- cbind(1, fr, mdp, mdus, mdeu)
  y4 <- defactor(ds_mat, ff)
  x4 <- defactor(dp_mat, ff)

  be4 <- sum(sum(x4*y4))/sum(sum(x4*x4))
  if((be4 - bb) < tol){
    break
  }
  bb <- be4
  }
```

The results of the estimators are summarized below.

Table 10: Summary of the estimates of the idio-dynamic relationship

CCE 2 factors	CCE 3 factors	GHS 3 factors	IFE PC only factors	IFE PC and CSA factors
0.71	0.671	0.531	0.749	0.719

6.5 PLM package for CCE regressions

We can use the **plm** package (Croissant and Millo, 2008) to compute the simple pooled CCE model. First we organize the data into the format for `plm` and then use the `pdata.frame` command to tell R that we have a panel data for plm. Since we have heterogeneity only in the factor loadings we use the pooled version, `model="p"`, in the command `pcce` with formula `ds ~ dp`.

```
library(plm)
```

```
##
## Attaching package: 'plm'

## The following objects are masked from 'package:dplyr':
##
##     between, lag, lead
```

```
dta_dp_n <- dp_dta %>%
  gather(key = Country, value = dp, -Date)

deu_dta_n <- deu_dta %>%
  gather(key = Country, value = deu, -Date)

ds_dta_n  <- ds_dta %>%
  gather(key = Country, value = ds, -Date) %>%
```

```r
  left_join(dta_dp_n, by = c("Date", "Country")) %>%
  left_join(deu_dta_n, by = c("Date", "Country")) %>%
  pdata.frame(index = c("Country", "Date") )

mod_pcce <- pcce(ds ~ dp, data = ds_dta_n, model="p")

# for mean group use model = "mg"
# mod_mgcce <- pcce(ds ~ dp, data = ds_dta_n, model="mg")

summary(mod_pcce)
```

```
## Common Correlated Effects Pooled model
##
## Call:
## pcce(formula = ds ~ dp, data = ds_dta_n, model = "p")
##
## Balanced Panel: n = 27, T = 202, N = 5454
##
## Residuals:
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.1146363 -0.0107980 -0.0002515  0.0000000  0.0100780  0.1935946
##
## Coefficients:
##    Estimate Std. Error z-value  Pr(>|z|)
## dp  0.70966    0.13067  5.4309 5.606e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## Total Sum of Squares: 4.2012
## Residual Sum of Squares: 2.1156
## HPY R-squared: 0.48663
```

7 Convergence

The first example in this chapter analyses weak σ convergence of the crime data. The crime data is kept as csv files and has not been converted to R format. The first code chunk reads in the violent crime data into R and processes it for the `weak_converge` function.

7.1 weak σ-convergence test

```
violent_mat <- read_csv("data/violent_mat.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )

## See spec(...) for full column specifications.
```

```
violent_mat_nms <- colnames(violent_mat)
violent_mat_nms <- str_replace( violent_mat_nms, "%", "Year")
colnames(violent_mat) <- violent_mat_nms

# organize data
violent_dat <- violent_mat %>%
  gather(key = State, value = Crimes, - Year) %>%
  mutate(lncrimes = log(Crimes)) %>%
  dplyr::select(-Crimes) %>%
  spread(key = State, value = lncrimes)  %>%
```

```
  dplyr::select(-Year) %>%
  data.frame()
```

The `weak_converge` takes data as a `data.frame` rather than a matrix. We store the output from this function into a matrix called `store_mat` with standard errors computed with 3 and 4 lags.

```
wc_3_out <- weak_converge(violent_dat, 3)
wc_4_out <- weak_converge(violent_dat, 4)

store_mat <- matrix(NA, nrow = 9, ncol = 6)
store_mat[1,1] <- round(100 * wc_3_out$phi, 3)
store_mat[1,2] <- round(wc_3_out$tstats, 3)
store_mat[1,3] <- round(wc_4_out$tstats, 3)

# final file list
final_list <- c("violent_dat")
panel_list <- c("violent")
```

The process can be repeated using the PC factors. We only show the case for violent crimes.

```
out_pc_violent <- pc(violent_dat, 1)
f1 <- as.matrix(out_pc_violent$f1) # pc factors
xx <- matrix(c(matrix(1, nrow = nrow(violent_dat)), f1), ncol = 2)
b <- solve(t(xx) %*% xx) %*% t(xx)  %*% as.matrix(violent_dat)
b2 <- t(as.matrix(b[2,]))
z <- violent_dat - f1 %*% b2
```

Table 11: (#tab:ch0703,)Summary of the weak sigma-convergence tests with various crime rates

Series	CSA phi x 100	t (lags = 3)	t (lags = 4)	PC phi x 100	t (lags = 3)	t (lags = 4)
violent	-1.491	-4.504	-4.120	-0.779	-1.910	-1.744
murder	-0.447	-5.402	-5.237	-0.282	-2.440	-2.287
robbery	-0.518	-2.204	-2.026	-0.487	-2.309	-2.126
rape	-0.382	-9.667	-8.900	-0.163	-2.490	-2.279
assault	-0.277	-2.344	-2.153	-0.222	-2.064	-1.892
property	-0.192	-5.749	-5.276	-0.190	-5.855	-5.370
burglary	-0.490	-4.668	-4.264	-0.401	-3.592	-3.280
larceny	-0.240	-6.321	-5.794	-0.176	-10.703	-10.154
motorv	-0.189	-2.097	-1.968	-0.242	-2.630	-2.449

```
wc_violent_3 <- weak_converge(z, 3)

wc_violent_4 <- weak_converge(z, 4)

store_mat[1,4] <- round(100 * wc_violent_3$phi, 3)

store_mat[1,5] <- round(wc_violent_3$tstats, 3)

store_mat[1,6] <- round(wc_violent_4$tstats, 3)
```

We repeat this process (load data, process, test, store) in a loop for which only the output is shown. If you use a loop, you will need to set a common location for all the csv files and make sure they are all set up the same: consistent headings and columns.

In the next part we wish to explore some of the properties of the data.

```
store_dta_props <- matrix(NA, nrow = length(final_list), ncol = 4)

r <- 1


for(i in final_list){

  data_rel <- get(i)  # assigns i to data_rel


  # how many negative
```

```r
my <- apply(data_rel, 2, min)
percent_neg <- mean(ifelse(my < 0, 1, 0))
store_dta_props[r, 2] <- percent_neg

# number trending
trd <- 1:nrow(data_rel)
ols_store <- matrix(NA, nrow = ncol(data_rel), ncol = 3)
for(ii in 1:ncol(data_rel)){
  mod1dta <- cbind.data.frame(y = data_rel[,ii], trd) # make dataframe
  mod1 <- lm(mod1dta)
  res <- mod1$residuals
  sig <- NeweyWestvcov(res, 3)
  xx <- model.matrix(mod1)
  xtx1 <- solve(crossprod(xx))
  sig <- c(sig) * diag(xtx1)
  tstat <- mod1$coefficients/sqrt(sig)
  ols_store[ii, 1] <- ii
  ols_store[ii, 2] <- mod1$coefficients[2]
  ols_store[ii, 3] <- tstat[2]
}
lessthan <- mean(ifelse(ols_store[,3] <= -1.65, 1, 0))
store_dta_props[r, 1] <- lessthan
modlogt <- logtregress(data_rel)
store_dta_props[r, 3] <- modlogt$b
store_dta_props[r, 4] <- modlogt$`t stat`

r <- r + 1
```

Table 12: Partial replication of table 7.3

Series	t less than -1.65 (%)	y less than 0 (%)
violent	0.00	0.00
murder	0.66	0.10
robbery	0.12	0.00
rape	0.02	0.02
assault	0.02	0.00
property	0.18	0.00
burglary	0.00	0.00
larceny	0.08	0.00
motorv	0.40	0.00

```
}
```

7.2 Replication of economic transition and growth

Last we replicate the example of relative convergence given in the **ConvergenceClubs** package (Sichera and Pizzuto, 2018) which is a replication of Phillips and Sul (2009).

```
library(ConvergenceClubs)


data("filteredGDP")
clubs <- findClubs(filteredGDP, dataCols=2:35, unit_names = 1, refCol=35,time_trim = 1/3
summary(clubs)
```

```
## Number of convergence clubs: 7
## Number of divergent units: 0
##
##          | # of units  | beta     | std.err  | tvalue   | cstar
## -------- ------------- ---------- ---------- ---------- --------
```

```
## club1 | 50 | 0.382 | 0.041 | 9.282 | 0
## club2 | 30 | 0.24 | 0.035 | 6.904 | 0
## club3 | 21 | 0.11 | 0.032 | 3.402 | 0
## club4 | 24 | 0.131 | 0.064 | 2.055 | 0
## club5 | 14 | 0.19 | 0.111 | 1.701 | 0
## club6 | 11 | 1.003 | 0.166 | 6.024 | 0
## club7 | 2 | -0.47 | 0.842 | -0.559 | 0
```

Appendix

A list of the functions in the **`PDEwCF`** package is provided. They are direct ports of the code provided on the text's website at https://personal.utdallas.edu/~dxs093000/book/panel.htm. They contain minimal documentation and the users should see the text for usage. They are also specifically designed to replicate the text and the estimators in chapter 6 are not multiple regressors.

1. NeweyWestvcov
2. olshac
3. weak_converge
4. pc
5. BaiNgIC
6. defactor
7. stand_mat

Data includes

1. assault_dat
2. burglary_dat
3. crime_data
4. larceny_dat
5. MATint99
6. MATn3_t39
7. MATn46_t39
8. motorv_dat
9. murder_dat
10. price

11. property_dat
12. rape_dat
13. robbery_dat
14. spot
15. violent_dat

References

Bada, O., Liebl, D., 2014. PHTT: Panel data analysis with heterogeneous time trends in r. Journal of Statistical Software, Articles 59, 1–33. https://doi.org/10.18637/jss.v059.i06

Croissant, Y., Millo, G., 2008. Panel data econometrics in R: The plm package. Journal of Statistical Software, Articles 27, 1–43. https://doi.org/10.18637/jss.v027.i02

Phillips, P.C.B., Sul, D., 2009. Economic transition and growth. Journal of Applied Econometrics 24, 1153–1185. https://doi.org/10.1002/jae.1080

Racine, J.S., 2017. Energy, economics, replication & reproduction. Energy Economics. https://doi.org/%2010.1016/j.eneco.2017.06.027

R Core Team, 2017. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Sichera, R., Pizzuto, P., 2018. ConvergenceClubs: Finding convergence clubs, (r package).

Sul, D., 2019. Panel data econometrics: Common factor analysis for empirical researchers. Taylor & Francis.

Wickham, H., 2017. Tidyverse: Easily install and load the 'tidyverse'.

Wickham, H., Grolemund, G., 2016. R for data science. Import, tidy, transform, visualize, and model data. "O'Reilly Media".