BAN430 Forecasting

Sondre Hølleland

9.8.2022

Table of contents

Introduction

In this course you will learn about forecasting methods.

Progress plan

Week number	Subject	Reading material	First lecture	Second lecture
2	Introduction and R	Chapter 1	Introduction lecture BAN430	Basic R / recap
3	Time series graphics	Chapter 2	Graphics	
4	Time series decomposition	Chapter 3	Time series decomposition	
5	Time series features	Chapter 4	Time series features	
6	Forecasters toolbox	Chapter 5	Forecasters toolbox	
7	Judgmental forecasts	Chapter 6	Judgmental forecasts	
8	Regression models	Chapter 7	Regression models	
9	ARIMA models	Chapter 9	ARIMA models	
10	ARIMA models	Chapter 9		
11	Volatility forecasting	Course website		
12	Some practicle forecasting issues	Chapter 13		
13	Preparations for the exam		Discussion lecture/exercise seminar	
14	Home exam			

Literature

FPP Third edition

We will use the textbook by Hyndman and Athanasopoulos (2018), i.e. the online version which can be accessed at https://otexts.com/fpp3/.

Curriculum

Textbook (Hyndman and Athanasopoulos (2018)) chapters 1-9 and 13. Additional notes by lecturer on volatility forecasting. All the material on this website.

1 R and Rstudio

In this course, we will be using R and Rstudio to e.g. visualize time series, estimate model parameters, forecast, etc. It is therefore essential to have some basic knowledge of how to write an R script and how to read in data and do some simple data manipulation for preparing the data for different time series analysis. Hopefully, most of you have some experience with R and Rstudio before. If you have, this will be a short recap, if not this will be a very short introduction covering the most basic operations.

Installing R and Rstudio

1. Install R:

   - Go to: cran.uib.no
   - Press download R for Linux/MacOS/Windows
   - Press base
   - Download R-4.x.x for Linux/MacOS/Windows
   - Run the installation using default options

2. Install Rstudio

   - Go to: rstudio.com
   - Select Rstudio desktop
   - Press Download Rstudio desktop
   - Select the Rstudio desktop with open source licence, which is free

   - Select the version for your operating system
   - Run the installation using default settings

3. Open Rstudio and check that it works (it should start without any error messages).
4. Install the R-package of the book "fpp3".

   - In Rstudio, select Tools -> Install packages -> write "fpp3" and make sure install dependencies is marked. Press Install. You can also run the following code in the console

```r
install.packages("fpp3", dependencies = TRUE)
```

Other useful packages to install are

```r
install.packages("tidyverse", dependencies = TRUE)
install.packages("readxl")
```

(will add other packages as we go)

R recap

If R and Rstudio are completely new tools for you, this section will probably not be detailed enough to get you started, but fear not. There are lots of good and useful online material for learning basic R. One possibility is to work through the first section of (chapter 1-8) of the book R for data science by Wickham and Grolemund (2016) available online. There is also a free Coursera course on R programming, recommended by the textbook authors.

We will mostly be using the tidyverse approach to doing data manipulation. This is in line with what you learn in courses like BAN400 R programming for Data Science or BAN420 Introduction to R and also with what the authors of the textbook does (Hyndman and Athanasopoulos (2018)).

Say you are given an .xlsx file (MS excel format) of daily prices of an US 10 year Treasury bond. The excel file contains several sheets with the

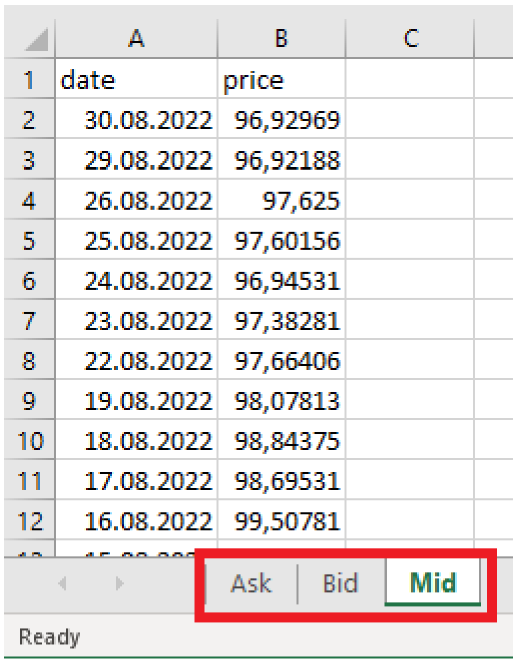
1. Closing ask price ("Ask")
2. Closing bid price ("Bid")
3. Closing mid price ("Mid")

Each contains two columns: date and price. In the figure below we have taken a screen shot of the Mid sheet.

You are interesting in reading in the closing mid price. To read in this data, you may use the following code.

```r
library(fpp3)    # loading textbook package
library(tidyverse)
library(readxl) # loading package for reading excel files
dat <- read_excel("data/US10YTRR.xlsx", sheet = "Mid")
head(dat) # printing out the first 6 rows
```

```
# A tibble: 6 x 2
  date                  price
  <dttm>                <dbl>
```



	A	B	C
1	date	price	
2	30.08.2022	96,92969	
3	29.08.2022	96,92188	
4	26.08.2022	97,625	
5	25.08.2022	97,60156	
6	24.08.2022	96,94531	
7	23.08.2022	97,38281	
8	22.08.2022	97,66406	
9	19.08.2022	98,07813	
10	18.08.2022	98,84375	
11	17.08.2022	98,69531	
12	16.08.2022	99,50781	

Figure 1.1: US 10-year Treasury bonds index collected from the Refinitiv Eikon data base.

```
1 2022-08-30 00:00:00  96.9
2 2022-08-29 00:00:00  96.9
3 2022-08-26 00:00:00  97.6
4 2022-08-25 00:00:00  97.6
5 2022-08-24 00:00:00  96.9
6 2022-08-23 00:00:00  97.4
```

The sheet argument specifies which sheet in the excel file we want to read. The read_excel function is also quite smart so it recognizes that the date column is a date and automatically format it accordingly. It is however perhaps not so useful to also include the time of the day (all is 00:00:00), so let us remove this part.

```r
dat %>%
  mutate(date = as.Date(date))
```

```
# A tibble: 8,804 x 2
   date        price
   <date>      <dbl>
 1 2022-08-30   96.9
 2 2022-08-29   96.9
 3 2022-08-26  97.6
 4 2022-08-25  97.6
 5 2022-08-24  96.9
 6 2022-08-23  97.4
 7 2022-08-22  97.7
 8 2022-08-19  98.1
 9 2022-08-18  98.8
10 2022-08-17  98.7
# ... with 8,794 more rows
```

Here I have used the mutate function. This is a function we use to either mutate an existing column or create a new one. In this case we mutated the date column transforming it to a "Date" object. We could also be intersted in adding a column for which year the observation is from.

```
dat %>%
  mutate(date = as.Date(date),
         year = year(date))
```

```
# A tibble: 8,804 x 3
   date       price  year
   <date>     <dbl> <dbl>
 1 2022-08-30  96.9  2022
 2 2022-08-29  96.9  2022
 3 2022-08-26  97.6  2022
 4 2022-08-25  97.6  2022
 5 2022-08-24  96.9  2022
 6 2022-08-23  97.4  2022
 7 2022-08-22  97.7  2022
 8 2022-08-19  98.1  2022
 9 2022-08-18  98.8  2022
10 2022-08-17  98.7  2022
# ... with 8,794 more rows
```

Here we have used the year function from the *lubridate* package, which is loaded with the *fpp3* package. The operator *%>%* is used to add operations to the data manipulation pipeline in the given order. We start with the data object (a tibble) and add a mutate operation to that where we first transform the date column and add a year column. Now that we are pleased with our pipeline, let us save this to the *dat* object.

```
dat <- dat %>%
  mutate(date = as.Date(date),
         year = year(date))
dat %>% glimpse()
```

```
Rows: 8,804
Columns: 3
$ date  <date> 2022-08-30, 2022-08-29, 2022-08-26, 2022-08-25, 2022-08-24, 202~
$ price <dbl> 96.92969, 96.92188, 97.62500, 97.60156, 96.94531, 97.38281, 97.6~
$ year  <dbl> 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022~
```

The glimpse function summarizes the tibble/data frame.

filter and select

Now, the data ranges from/to

```
range(dat$date)
```

```
[1] "1987-08-03" "2022-08-30"
```

but say you only want to use data from 2010 onwards. To do this, we use the filter function. This function is useful for selecting rows that fulfil some condition, in this case year >= 2010. Let us make a pipeline for this

```
dat %>%
  filter(year >= 2010)
```

```
# A tibble: 3,178 x 3
   date       price  year
   <date>     <dbl> <dbl>
 1 2022-08-30  96.9  2022
 2 2022-08-29  96.9  2022
 3 2022-08-26  97.6  2022
 4 2022-08-25  97.6  2022
 5 2022-08-24  96.9  2022
 6 2022-08-23  97.4  2022
 7 2022-08-22  97.7  2022
 8 2022-08-19  98.1  2022
 9 2022-08-18  98.8  2022
10 2022-08-17  98.7  2022
# ... with 3,168 more rows
```

Since 2022 is not a complete year (in the data), you also don't want observations after 2021. Then you can add this as an extra condition.

```
dat %>%
  filter(year >= 2010, year <=2021)
```

```
# A tibble: 3,012 x 3
   date       price  year
   <date>     <dbl> <dbl>
 1 2021-12-31  98.8  2021
 2 2021-12-30  98.8  2021
 3 2021-12-29  98.4  2021
 4 2021-12-28  99.0  2021
 5 2021-12-27  99.1  2021
 6 2021-12-23  98.9  2021
 7 2021-12-22  99.3  2021
 8 2021-12-21  99.2  2021
 9 2021-12-20  99.5  2021
10 2021-12-17  99.7  2021
# ... with 3,002 more rows
```

Alternatively, you can use the between function

```
dat %>%
  filter(between(year, 2010, 2021))
```

which will produce the same result. Another useful function is called select. While filter is used on the rows of your data, select is for columns. Say we don't need the year column after having filtered out the years we don't want. We can then either select the columns we want to keep

```
dat %>%
  filter(between(year, 2010, 2021)) %>%
  select(date, price)
```

or remove the columns we do not want

```
dat %>%
  filter(between(year, 2010, 2021)) %>%
  select(-year)
```

```
# A tibble: 3,012 x 2
   date       price
   <date>     <dbl>
 1 2021-12-31  98.8
 2 2021-12-30  98.8
 3 2021-12-29  98.4
 4 2021-12-28  99.0
 5 2021-12-27  99.1
 6 2021-12-23  98.9
 7 2021-12-22  99.3
 8 2021-12-21  99.2
 9 2021-12-20  99.5
10 2021-12-17  99.7
# ... with 3,002 more rows
```

group_by and summarize

Say we are interested in calculating the yearly mean price. In the tidyverse pipeline this means we want to group our observations according to year and summarize by year the mean of the observations. We will filter to avoid having the first and last years that are incomplete.

```
dat %>%
  filter(between(year, 1988, 2021)) %>%
  group_by(year) %>%
  summarize(meanPrice = mean(price))
```

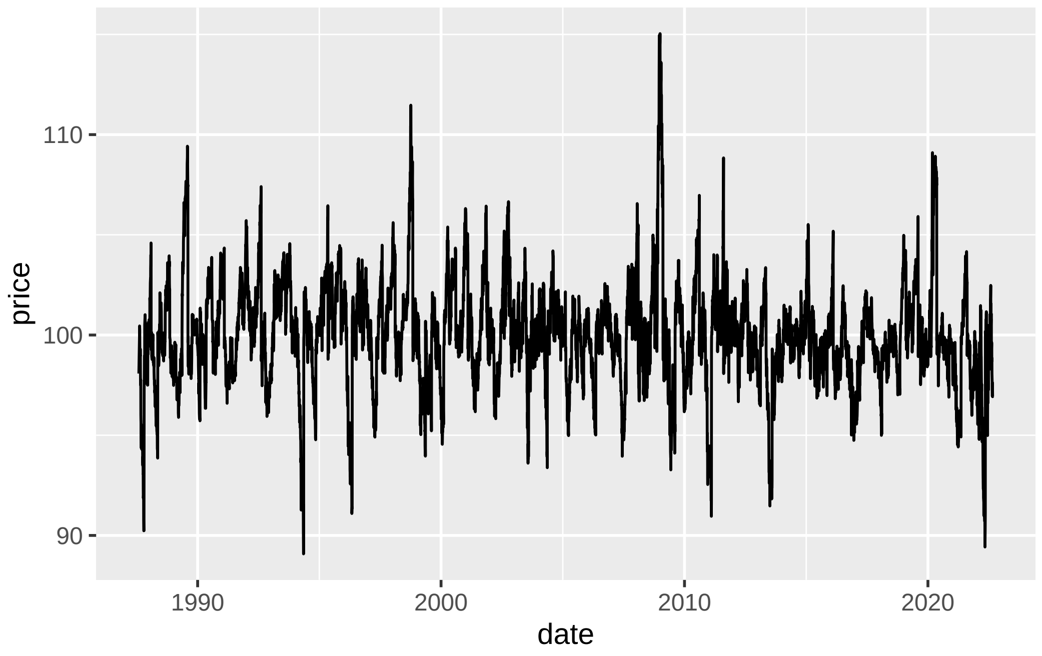
```
# A tibble: 34 x 2
    year meanPrice
   <dbl>     <dbl>
 1  1988      99.7
 2  1989     101.
 3  1990     100.
 4  1991     100.
 5  1992     100.
 6  1993     102.
 7  1994      98.3
 8  1995     102.
 9  1996      99.8
10  1997      99.9
# ... with 24 more rows
```

This pipeline could be read as first we take out observations prior to 1988 and after 2021, then we group the observations according to year and summarize the mean price by year. Note that this operation will delete any columns that are not in the group_by or being calculated in the summarize.

ggplot

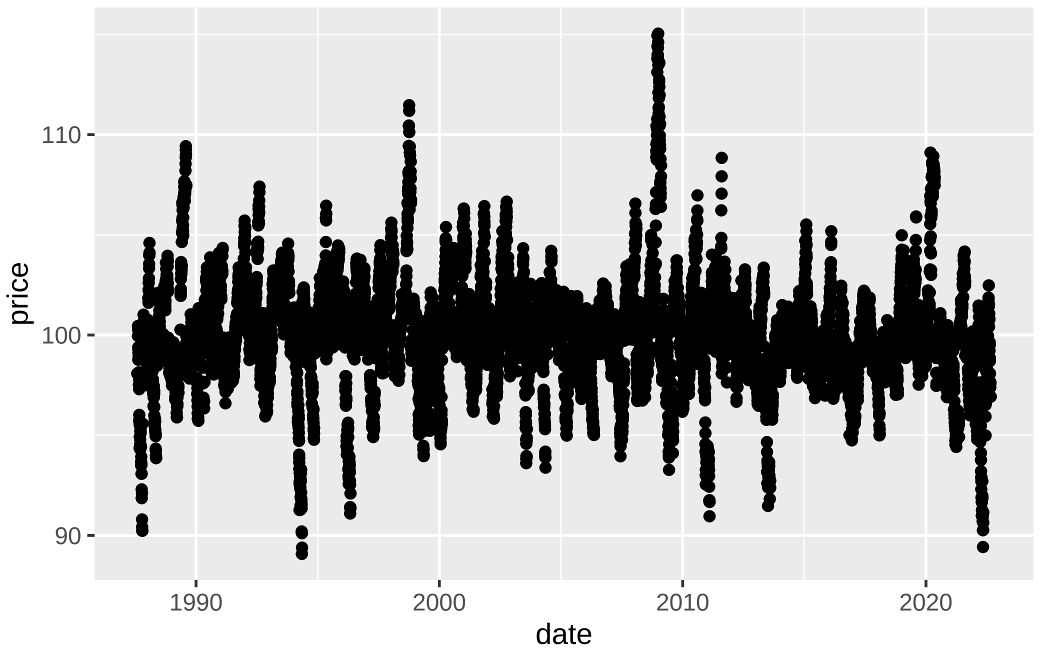
Plotting a data frame is convenient to do using the ggplot2 package. This will (when used appropriately) produce beautiful figures. Let us plot the time series at hand. The ggplot2 follows the same logic with a pipeline, but instead of the %>% operator, we add elements to the figure using +. We need to specify the data object and the name of the x and y columns to be plotted. Everything in the figure that is to vary based on values in the data frame needs to be wrapped in a aes (aesthetic) function (here the x and y arguments). By adding the geom_line() we insert a line.

```
ggplot(data = dat,
       aes(x=date, y = price)) +
  geom_line()
```



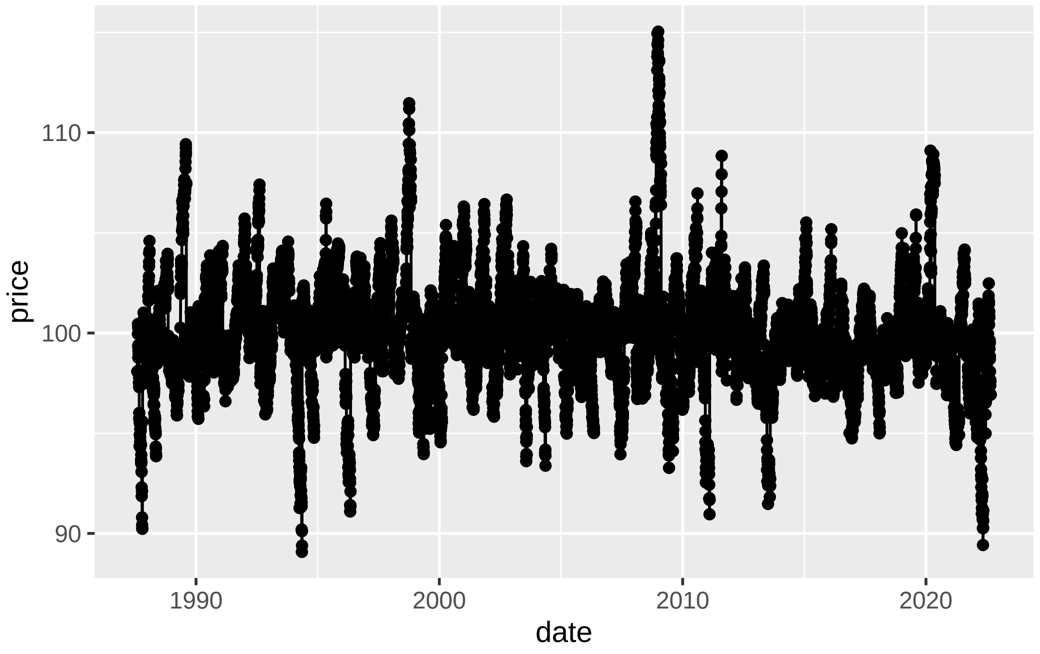
We could instead add geom_point()

```
ggplot(data = dat,
       aes(x=date, y = price)) +
  geom_point()
```



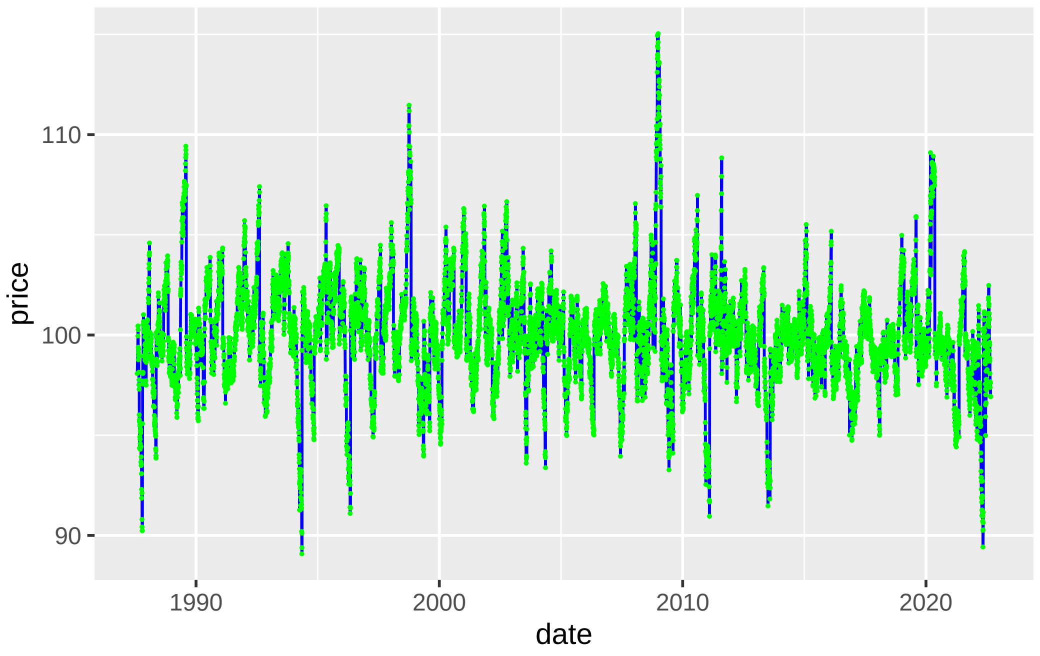
or do both

```
ggplot(data = dat,
       aes(x=date, y = price)) +
  geom_line() +
  geom_point()
```



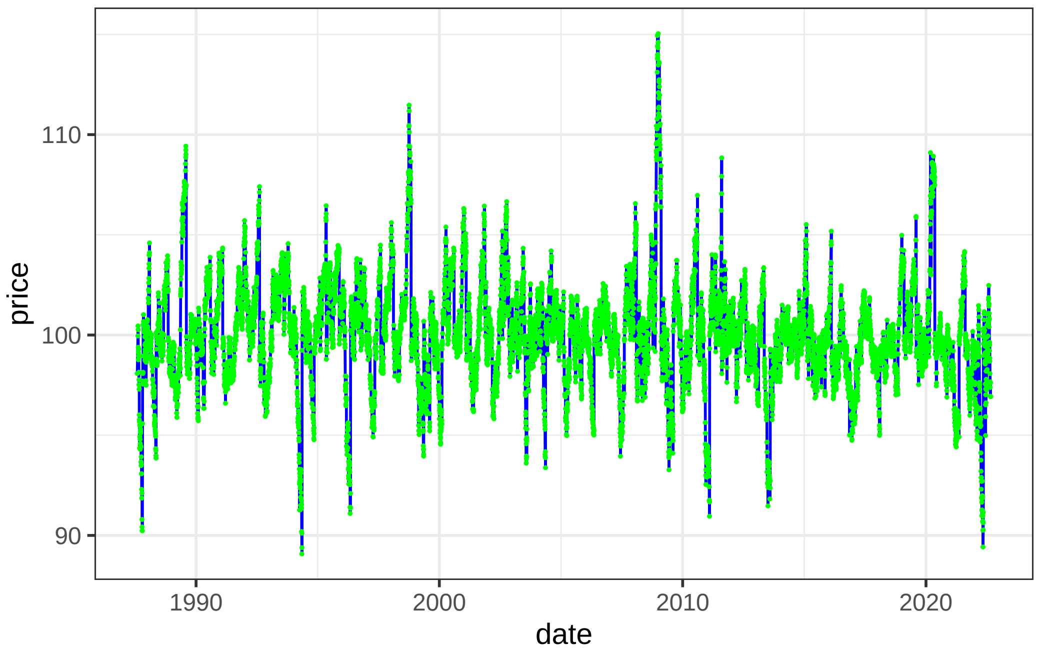
We can change the colors and decrease the size of the points:

```
ggplot(data = dat,
       aes(x=date, y = price)) +
  geom_line(color = "blue") +
  geom_point(color = "green", size = .2)
```



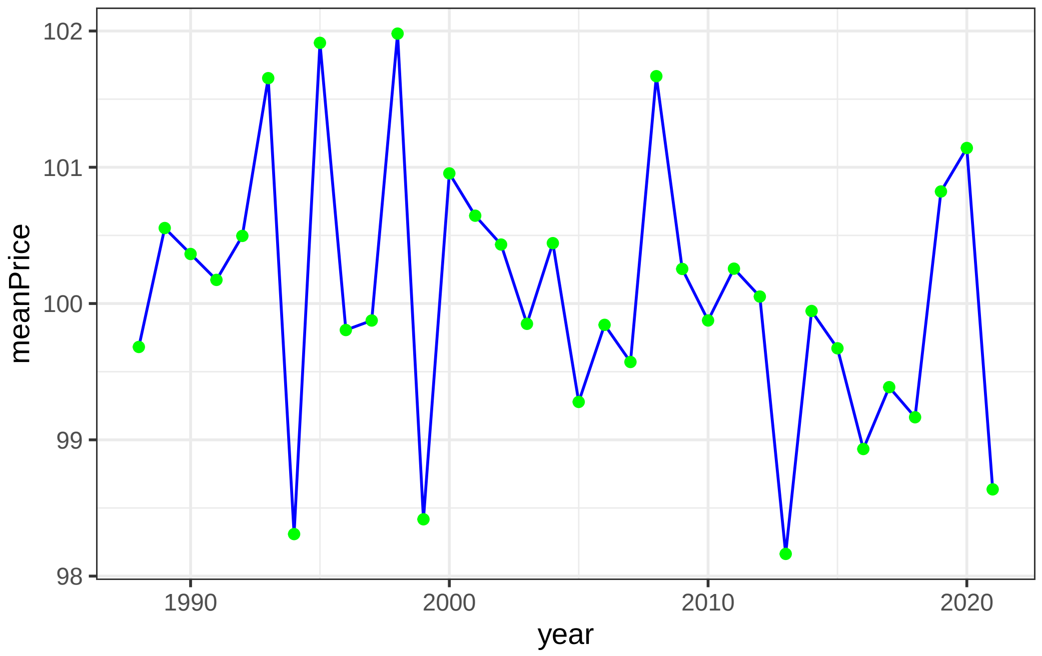
Or maybe we do not want to use the default theme: –>

```
ggplot(data = dat,
       aes(x=date, y = price)) +
  geom_line(color = "blue") +
  geom_point(color = "green", size = .2) +
  theme_bw()
```



We can also include the plotting in our data manipulation pipeline. For instance, lets summarize the data by year and plot the resulting yearly time series.

```
dat %>%
  filter(between(year, 1988, 2021)) %>%
  group_by(year) %>%
  summarize(meanPrice = mean(price)) %>%
  # adding plotting to pipeline:
  ggplot(aes(x=year, y = meanPrice)) +
  geom_line(color = "blue") +
  geom_point(color = "green") +
  theme_bw()
```



Epilogue

We cannot illustrate all aspects here, but you will learn new elements by studying examples throughout the course. This recap is mostly for remembering the basics of data manipulation in R and simple plotting. As you will see in the continuation, the coding is not much more complex then what you have seen here and the fpp3 package uses the same type of logic and syntax as the tidyverse. There will however be some new functions specific for time series analysis that you will need to learn.

Exercises

1. Set working directory.
2. Load the data.
3. Filter away observations prior to 2010.
4. Remove columns except .. and time
5. Summarize data to monthly means
6. Make a plot with time on x-axis and monthly means on y-axis
7. Save the figure to file.

2 Time series basics

2.1 What is a time series

2.2 Time series notation

2.3 Stationarity

2.4 Autocorrelation

2.5 White noise

2.6 Time series graphics

3 Decomposition

In this chapter we will consider different adjusments and transformations one can do prior to a model task. Then we move on to techniques for decomposing a time series into a trend-cycle, season and remainder component.

Calender adjustments

In the example in the video above, we are not interested in a proxy for working days per month, and to avoid the effect of this we use the mean (average) production per working day within each month instead of total production per month. The code to generate the example can be found below:

```r
library(lubridate)
library(tidyverse)
library(fpp3)

# ggplot theme:
theme_set(
  theme_bw() +
    theme(panel.grid.minor = element_blank(),
          panel.grid.major = element_blank())
)

# Daily production:
dat <- tibble(
  date = seq(as.Date("2015-01-01"), as.Date("2019-12-31"), by = "1 day"),
  price = pi
) %>%
  #Removing the weekends:
  filter(wday(date, week_start = 1) %in% 1:5) %>%
  #Note: We do not remove public holidays, and the worker never takes a day off
  mutate(YearMonth = yearmonth(date))

# -- TOTAL PRODUCTION FIGURE --
```

```r
dat %>%
  group_by(YearMonth) %>%
  summarize(`Total production` = sum(price)) %>%
  as_tsibble(index = "YearMonth") %>%
  ggplot(aes(x = YearMonth,
             y = `Total production`)) +
  geom_point(color = "skyblue") +
  geom_line(color = "skyblue") +
  scale_y_continuous(breaks = seq(60, 100, 5),
                     labels = paste0("$",seq(60, 100, 5),"k"),
                     limits = c(60,75))
# -- MEAN PRODUCTION FIGURE --
dat %>% group_by(YearMonth) %>%
  summarize(`Mean production` = mean(price)) %>%
  as_tsibble(index = "YearMonth") %>%
  ggplot(aes(x = YearMonth,
             y = `Mean production`)) +
  geom_point(color = "skyblue") +
  geom_line(color="skyblue")+
  scale_y_continuous(breaks = seq(3, 4, .02),
                     labels = paste0("$",seq(3, 4, .02),"k"))
```

Population adjustment

Adjusting for population size is usually a good idea when studying a quantity that is affected by it. The most obvious example is to study GDP (Gross Domestic Product) per capita instead of GDP.

```r
# --- Setting up the data --
scandinaviaUSA <- global_economy %>%
  filter(Country %in% c("Norway","Sweden","Denmark", "United States"))
scandinaviaUSA %>% head()

# --- GDP by country in $US --
scandinaviaUSA %>%
  autoplot(GDP)+
  labs(title= "GDP", y = "$US")

# --- Population by country  --
scandinaviaUSA %>%
  autoplot(Population)+
  labs(title= "Population", y = "Number of people")

# --- Population by country  (log-scale on y-axis) --
scandinaviaUSA %>%
  autoplot(Population)+
  scale_y_log10()+
  labs(title= "Population (log-scale)",y = "Number of people")

# --- GDP per capita by country ---
scandinaviaUSA %>%
  autoplot(GDP/Population) +
  labs(title= "GDP per capita = GDP / Population", y = "$US")
```

Inflation adjustment

Adjusting for inflation is a simple way of taking into account that 5$ in 1950 would get you much more than 5$ would today. This compensation is usually done by a consumer price index, which is standardized to a specific year (in the video below we show examples with 2010 and 2015 as reference years).

Let Y_t denote the raw time series and $Y_t^\star$ the inflation adjusted. Let CPI_t denote a relevant consumer price index defined to be 100 in the reference year $t^\star$. Then

$$Y_t^\star = Y_t \cdot \frac{100}{\text{CPI}_t}.$$

More generally, we can choose the reference year $t^\star$ and write this as

$$Y_t^\star = Y_t \cdot \frac{\text{CPI}_{t^\star}}{\text{CPI}_t}.$$

The inflation adjusted series is then measured in the unit “year $t^\star$-money”.

```
# --- Inflation adjusted GDP per capita by country ---
scandinaviaUSA %>%
  autoplot(GDP/Population *100 / CPI) +
  labs(title= "GDP per capita = GDP / Population", y = "$US")

# --- CPI FOR NORWAY (data from Statistics Norway)---
CPI <- read.csv("data/CPI_norway.csv", sep = ";") %>% as_tibble() %>%
  select(1:2) %>%
```

```r
  rename(Year = X,CPI = Y.avg2) %>%
  mutate(Year = as.numeric(Year), CPI = as.numeric(CPI))%>%
  filter(Year < 2022) %>%
  as_tsibble(index = Year)

# --- CPI figure ---
CPI %>%
  autoplot(CPI, color = "blue", lwd = 1.2) +
  labs(title= "Consumer Price Index", y = "NOK",
       subtitle = "Data source: Statistics Norway")+
  geom_hline(yintercept = 100, lty = 2) + geom_vline(xintercept = 2015, lty = 2)+
  scale_x_continuous(breaks = seq(1925,2025,10))+
  scale_y_continuous(breaks = seq(0,120,10))

# --- BIG MAC price index ---
bigMac <- read_csv("https://raw.githubusercontent.com/TheEconomist/big-mac-data/master/out
norBigMac <- bigMac %>%
  filter(name %in% c("Norway")) %>%
  mutate(Year = lubridate::year(date)) %>%
  as_tsibble(index = "date")%>%
  filter(Year <2022) %>%
  left_join(CPI, by = "Year")

# --- BIG MAC price index figure ---
norBigMac %>%
  autoplot(local_price) +
  labs(title= "Big Mac price in Norway", y = "NOK",
       subtitle = "Data source: The Economist") +
  geom_smooth(method = "lm", se=FALSE)

#--- Inflation adjusted BIG MAC price index figure ---
norBigMac %>%
  mutate(cpiAdjusted =local_price / CPI * 100)  %>%
  as_tsibble(index = date) %>%
  autoplot(cpiAdjusted)+
  labs(title= "Inflation adjusted Big Mac price in Norway", y = "NOK (2015)",
       subtitle = "Data sources: The Economist (big mac index), Statistics Norway (CPI)")
  geom_smooth(method = "lm", se=FALSE)
```

Mathematical Transformations

In many situations it can be necessary to do a mathematical transformation of a time series. There can be different reasons for doing so, but a main one is to make it stationary (or at least more stationary). For instance, if you see that the variation increases or decreases with the level of the series. The most common transformation (for positive time series) is probably using the logarithm. It is often effective and interpretable as changes in the log value correspond to relative changes in the original scale. We write the transformed series, w_t, as $w_t = \log y_t$, where y_t is the original time series.

The textbook also mentions power transformations of the form $w_t = y_t^p$ (squarte roots - $p = \frac{1}{2}$, cube roots - $p = \frac{1}{3}$, etc). These are not as common to use, but there are situations where these may be better than the logarithm.

A family of transformations (including log- and a class of power transformations) is the Box-Cox transform. For any value of $\lambda \in \mathbb{R}$,

$$w_t = \begin{cases} \log(y_t), & \text{if } \lambda = 0; \\ (y_t^\lambda - 1)/\lambda, & \text{otherwise.} \end{cases}$$

As you can see, if $\lambda = 0$ we have a simple natural logarithm transform. This version of the Box-Cox transform is also defined for negative values of y_t as long as $\lambda > 0$.

The book has a very nice shiny app for experimenting with different values of λ on a time series of gas production in Australia. We have borrowed it below, but you find it also here. They write that: *" A good value of* λ is one which makes the size of the seasonal variation about the same across the whole series, as that makes the forecasting model simpler." This pretty much sums up why one does mathematical transformations as a preprocessing step before fitting a model - it makes the model simpler.

For financial assets, such as stocks, it is often better to consider the returns rather than the price series. This is also a mathematical transformation and involves differencing. First order differencing means subtracting the previous observation from the present, i.e. $w_t = y_t - y_{t-1}$. Taking differences is an effective way of potentially making a non-stationary time series stationary. E.g. if a time series has a linear trend: $Y_t = \alpha t + Z_t$, where α is a real constant and Z_t is a white noise, we get that

$$W_t = Y_t - Y_{t-1} = \alpha t + Z_t - \alpha(t-1) - Z_{t-1} = Z_t - Z_{t-1} + \alpha,$$

effectively removing the trend in the transformed series. We will return to this when considering ARIMA models.

There are different definitions of returns, but the most common ones are the standard returns, r_t, and log-returns, lr_t, defined respectively by

$$r_t = \frac{y_t - y_{t-1}}{y_{t-1}},$$
$$\mathrm{lr}_t = \log y_t - \log y_{t-1} = \log \frac{y_t}{y_{t-1}}.$$

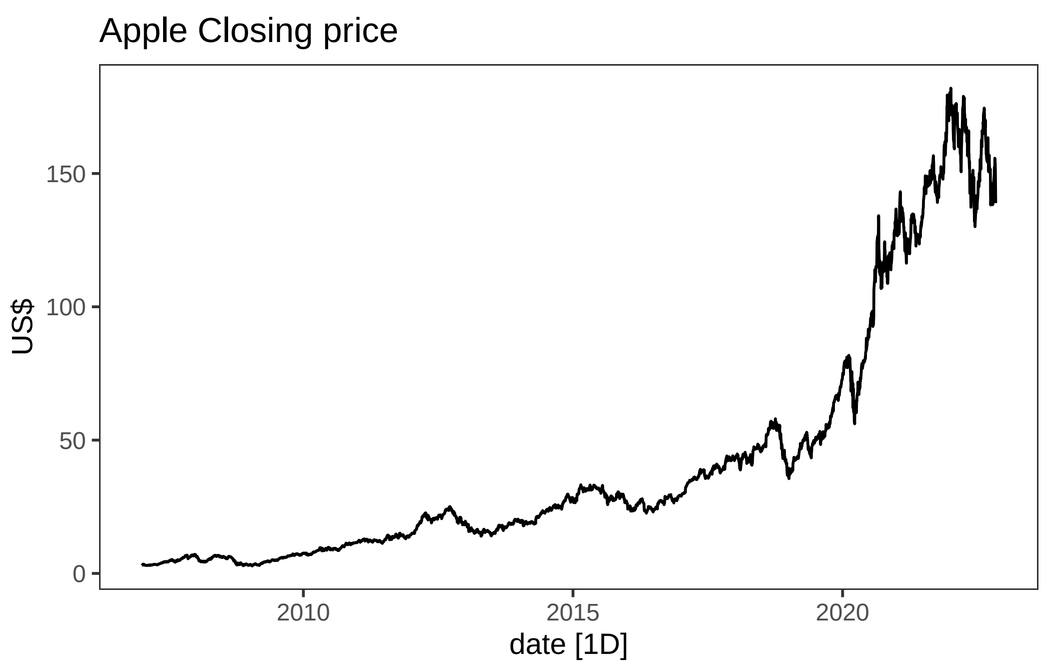
A daily return series for a stock usually has expectation close to zero and little autocorrelation, which can be convenient in many situations. However, they are typically hetereoskedastic (non-constant variance) and the squared returns are often autocorrelated. We will come back to this, when discussing volatility forecasting towards the end of the course.

Code on mathematical transformations

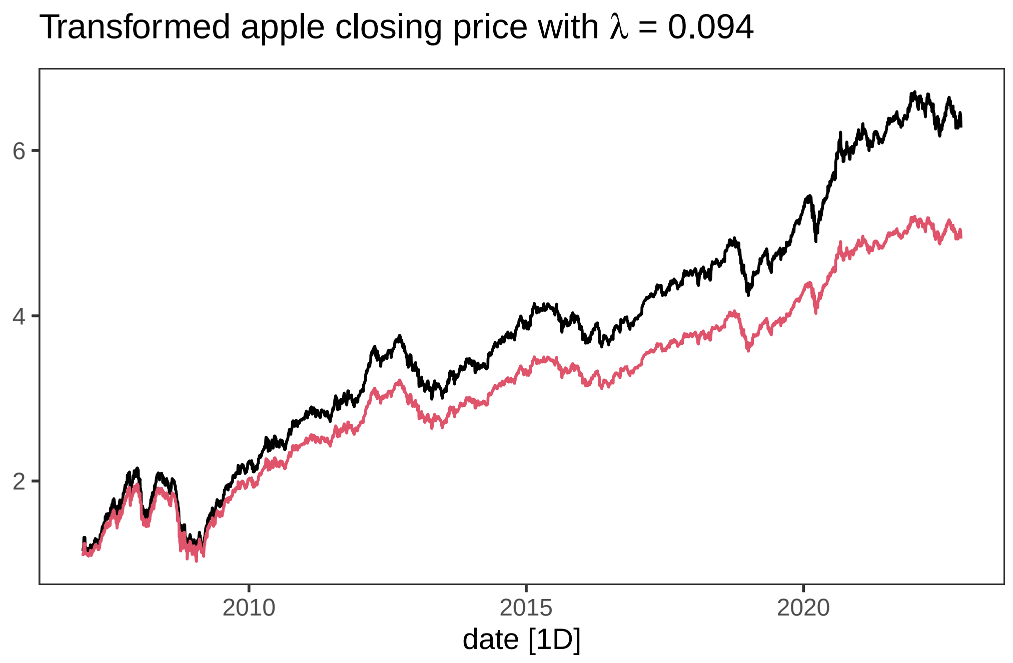
```
# Package for downloading stock data (primarily from Yahoo! Finance)
library(quantmod)

# -- Download the data: --
getSymbols("AAPL")
```

```
[1] "AAPL"
```

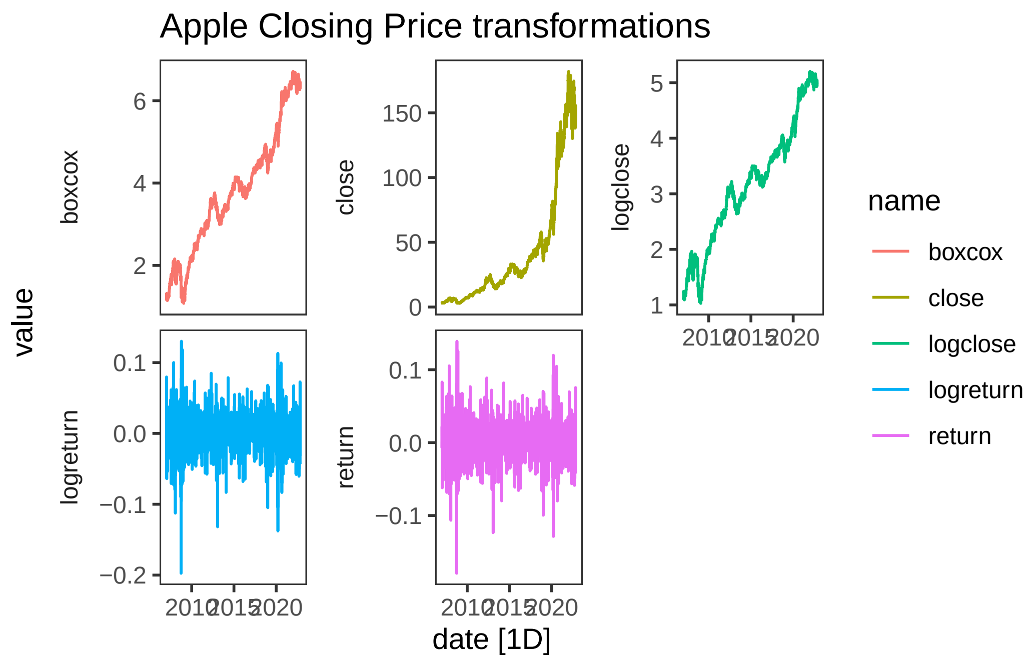
```
# -- Extract the closing price and create a tsibble: --
close.price <- tibble(
  close = as.numeric(AAPL$AAPL.Close),
  date  = time(AAPL)
) %>% as_tsibble(index = date)

# -- Plot closing price: --
close.price %>% autoplot(close) +
  labs(title = "Apple Closing price",
       y     = "US$")
```



```
# -- Adding transformations : --
close.price <- close.price %>%
  mutate(logclose  = log(close), # log-transform
         logreturn = c(NA, diff(logclose)), # log returns
         return    = c(NA, diff(close)/close[-nrow(close.price)]) # Returns
         )


# -- Box-cox-transform --
lambda <- close.price %>%
  features(close, features = guerrero) %>%
  pull(lambda_guerrero)
close.price <- close.price %>%
  mutate(boxcox = box_cox(close,lambda))
close.price %>%
  autoplot(boxcox) +
  labs(y = "",
       title = latex2exp::TeX(paste0(
         "Transformed apple closing price with $\\lambda$ = ",
         round(lambda,3))))+
  # Adding red curve with log-transform
  geom_line(aes(y=logclose), col = 2)
```



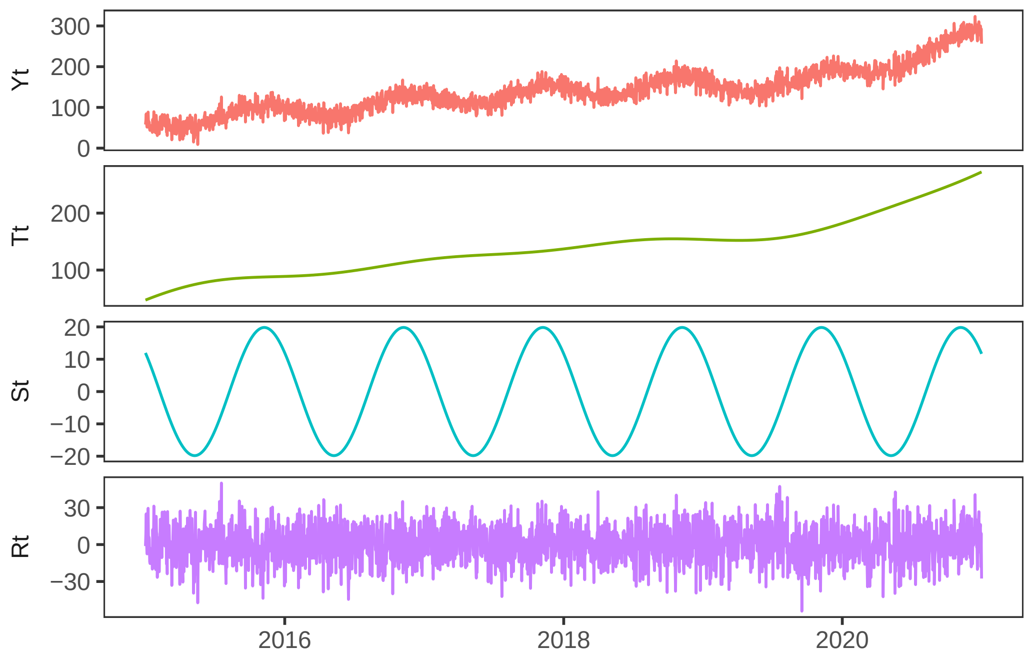
```
# -- Plotting the different transformations --
close.price %>%
  pivot_longer(-date) %>%
  autoplot(value) +
  facet_wrap(~name, scales="free_y", strip.position = "left")+
  labs(title = "Apple Closing Price transformations") +
  theme(strip.placement = "outside")
```



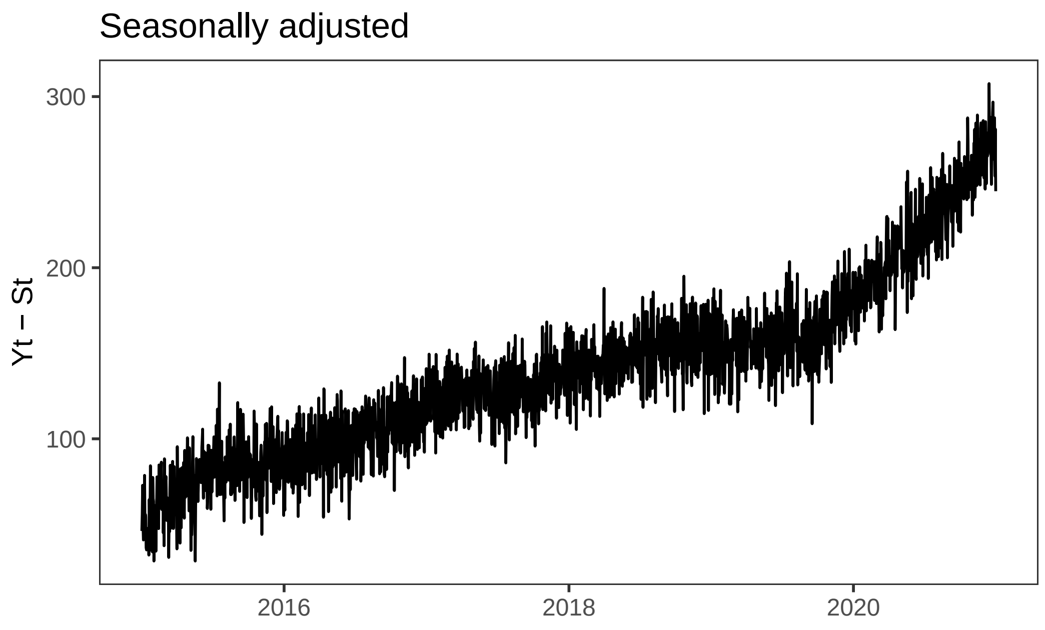
Time series components and Seasonal adjustment

Code on mathematical transformations

```
set.seed(1344)
library(tidyverse)
library(fpp3)
dat <- tibble(
  date = seq(as.Date("2015-01-01"), as.Date("2020-12-31"), by = "1 day"),
  t = 1:length(date),
  Tt = 100+9*t/300 + .5*((t-1000)/200)^3+(.87*cos(2*pi*t/1200)+.42*sin(2*pi*t/600))*11,
  St = (.94*cos(2*pi*t/365) -1.2*sin(2* pi*t/365))*13,
  Rt = rnorm(length(date), sd = 15),
  Yt = Tt + St +  Rt
)

# -- Plotting Y and its components --
dat %>%
  pivot_longer(cols = -c(date,t)) %>%
  mutate(name = factor(name, levels = c("Yt", "Tt", "St", "Rt"))) %>% # to order the panel
  ggplot(aes(x=date, y = value, col = name)) +
  geom_line() + facet_wrap(~name, ncol = 1, scales = "free_y", strip.position = "left") +
  theme(strip.placement = "outside", axis.title = element_blank(), legend.position = "none
```



```
# -- Plotting seasonally adjusted Y --
ggplot(dat, aes(x = date, y = Yt-St)) +
  geom_line() +
  labs(title = "Seasonally adjusted", x = "")
```

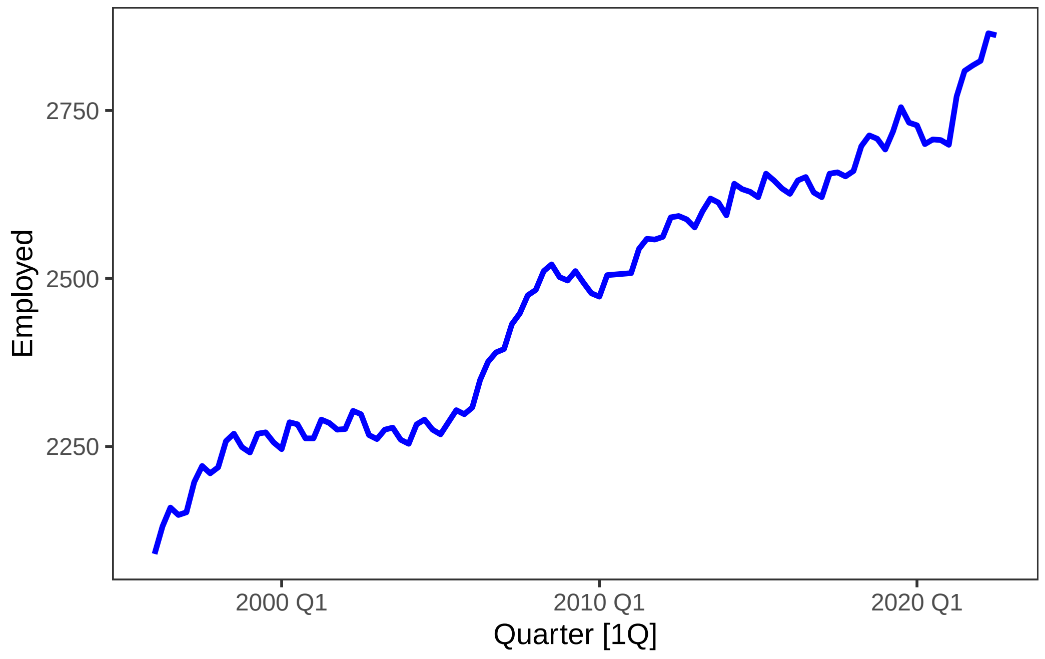


Moving averages

Code

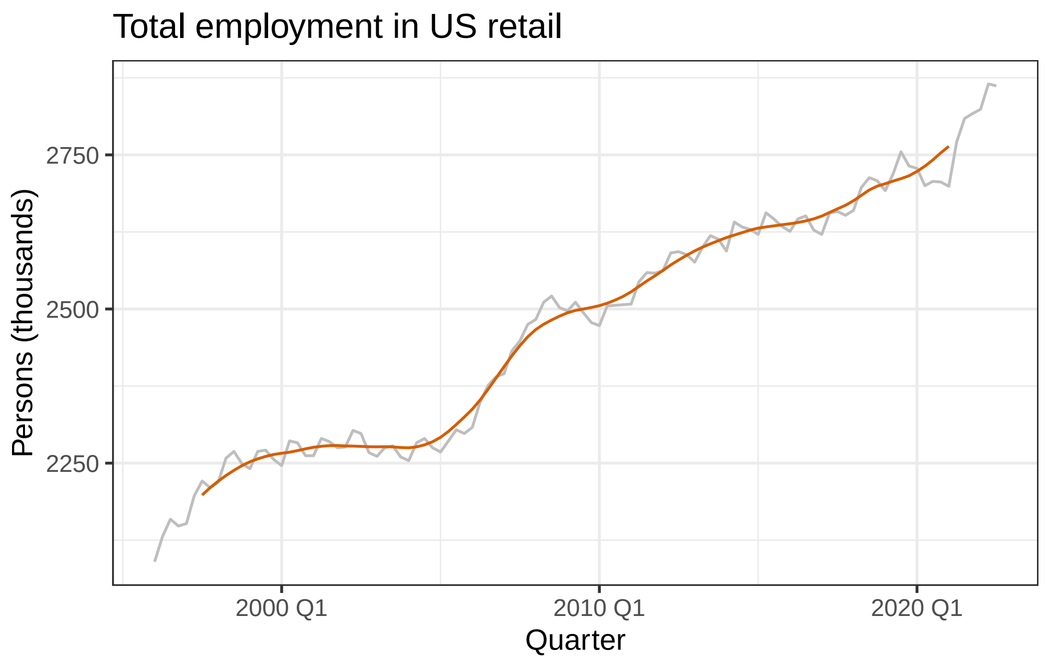
```
library(fpp3)
library(tidyverse)

# ggplot theme:
theme_set(theme_bw() +
            theme(panel.grid.major = element_blank(),
                  panel.grid.minor = element_blank()))

# -- Read in data: --
dat <- readxl::read_excel(
  "data/NorwayEmployment_15-74years_bySex.xlsx") %>%
  as_tibble() %>%
  mutate(Quarter = str_replace(Quarter, "K","Q"),
         Quarter = yearquarter(Quarter))
names(dat)[3] <- "Employed"

# -- Aggregating from Employed by sex to total --
```

```r
dat <- dat %>%
  group_by(Quarter) %>%
  summarize(Employed = sum(Employed)) %>%
  as_tsibble(index = Quarter) # Time series table
dat %>%
  autoplot(Employed, lwd = 1, colour = "blue")
```



```r
dat <- dat %>%
  mutate(
    `12-MA` = slider::slide_dbl(Employed, mean,
                                .before = 5, .after = 6, .complete = TRUE),
    `2x12-MA` = slider::slide_dbl(`12-MA`, mean,
                                  .before = 1, .after = 0, .complete = TRUE)
  )
dat %>%
  ggplot(aes(x=Quarter, y =Employed))+
  geom_line(colour = "gray") +
  geom_line(aes(y = `2x12-MA`), colour = "#D55E00") +
  theme_bw()+
  labs(y = "Persons (thousands)",
       title = "Total employment in US retail")
```

```
Warning: Removed 12 row(s) containing missing values (geom_path).
```

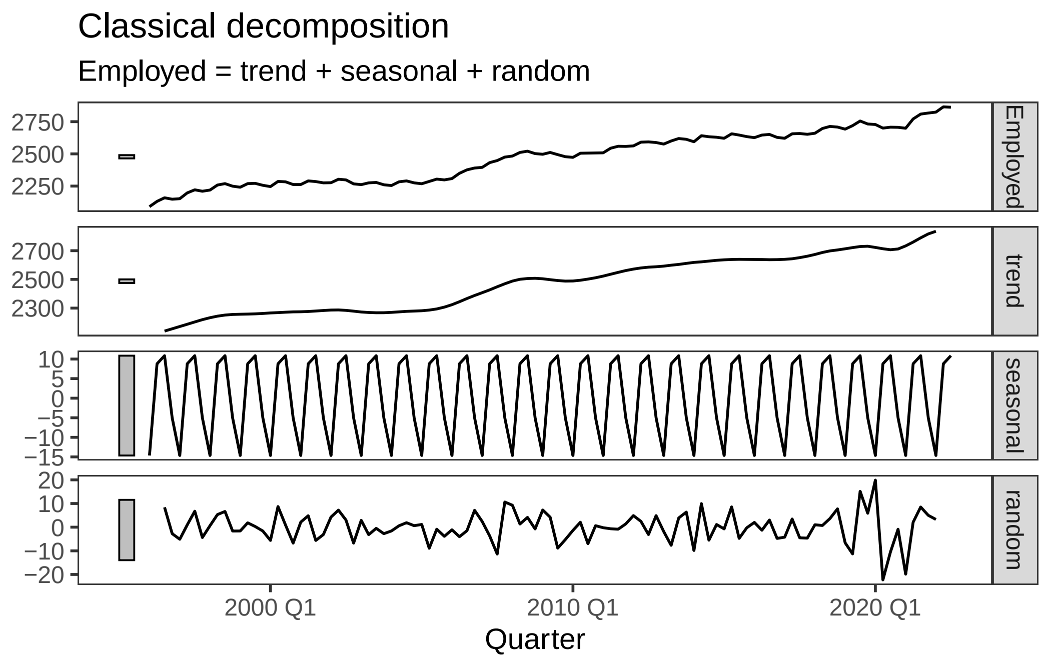


Classical decomposition

Code

```
dat %>%
  model(
    classical_decomposition(Employed, type = "additive")
  ) %>%
  components() %>%
  autoplot()
```

```
Warning: Removed 2 row(s) containing missing values (geom_path).
```

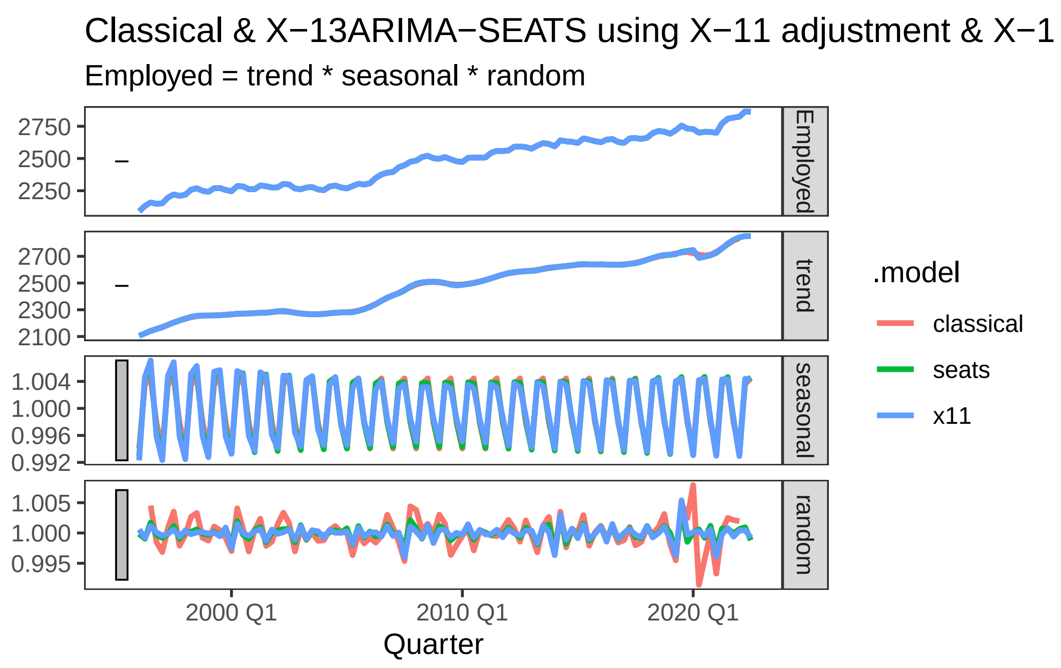


3.1 Statistics agencies: X11 and SEATS

Code

```
dat %>%
  model(
    classical = classical_decomposition(Employed,
                                        type = "multiplicative"),
    x11 = X_13ARIMA_SEATS(Employed ~ x11()),
    seats = X_13ARIMA_SEATS(Employed ~ seats())
  ) %>%
  components() %>%
  mutate(random = ifelse(.model == "classical",
                         random,
                         irregular)) %>%
  autoplot(lwd = 1)
```

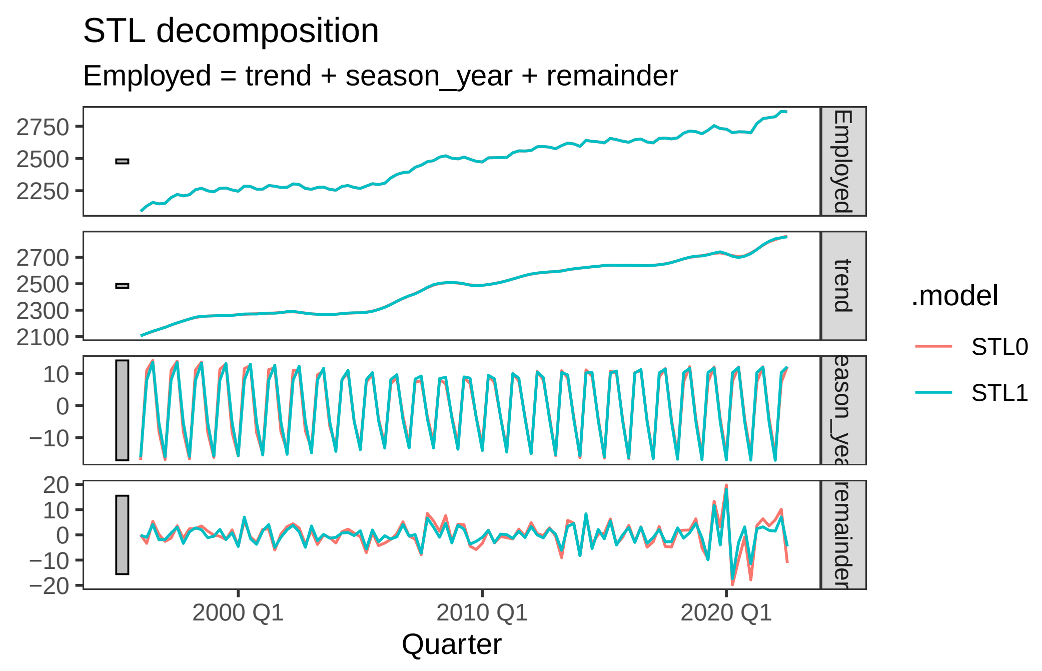
```
Warning: Removed 2 row(s) containing missing values (geom_path).
```



STL: Seasonal and Trend decomposition using Loess

Code

```
dat %>%
  model(
    STL0 = STL(Employed),
    STL1 = STL(Employed ~ trend(window = 5) + # default 7
                 season(window = 19),          # default 11
               robust = FALSE)
  ) %>%
  components() %>%
  autoplot()
```

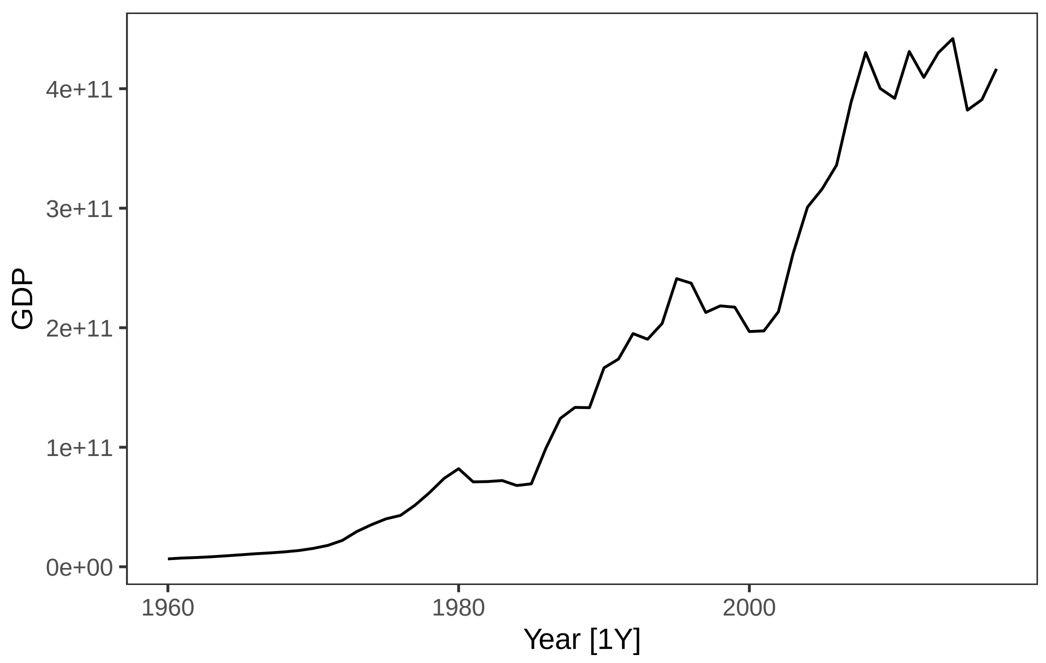


Exercises

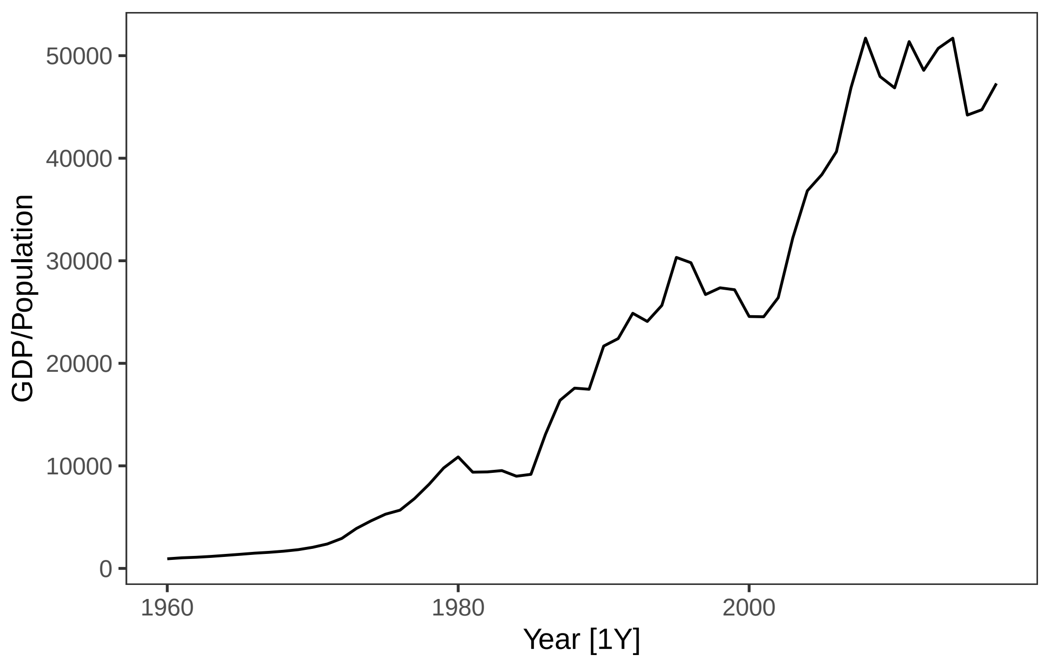
1. Use the global_economy data, select a country (e.g. Austria). Plot GDP, GDP per capita and GDP per capita inflation adjusted, GDP inflation adjusted.

Solution

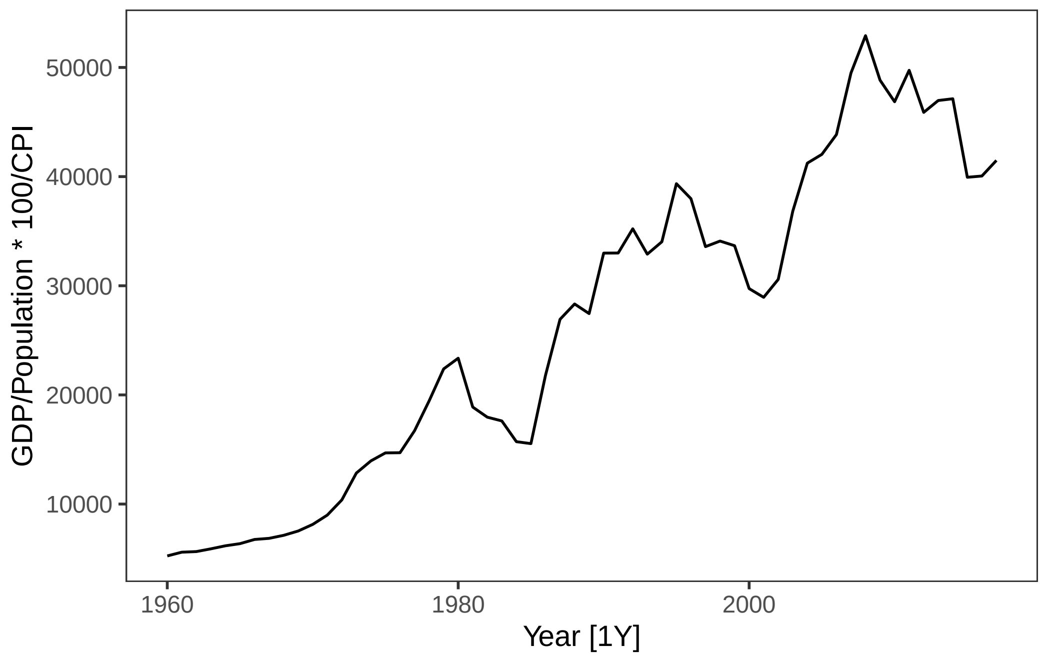
```r
library(fpp3)
library(tidyverse)
dat <- global_economy %>%
  filter(Country == "Austria")
dat %>% autoplot(GDP)
```



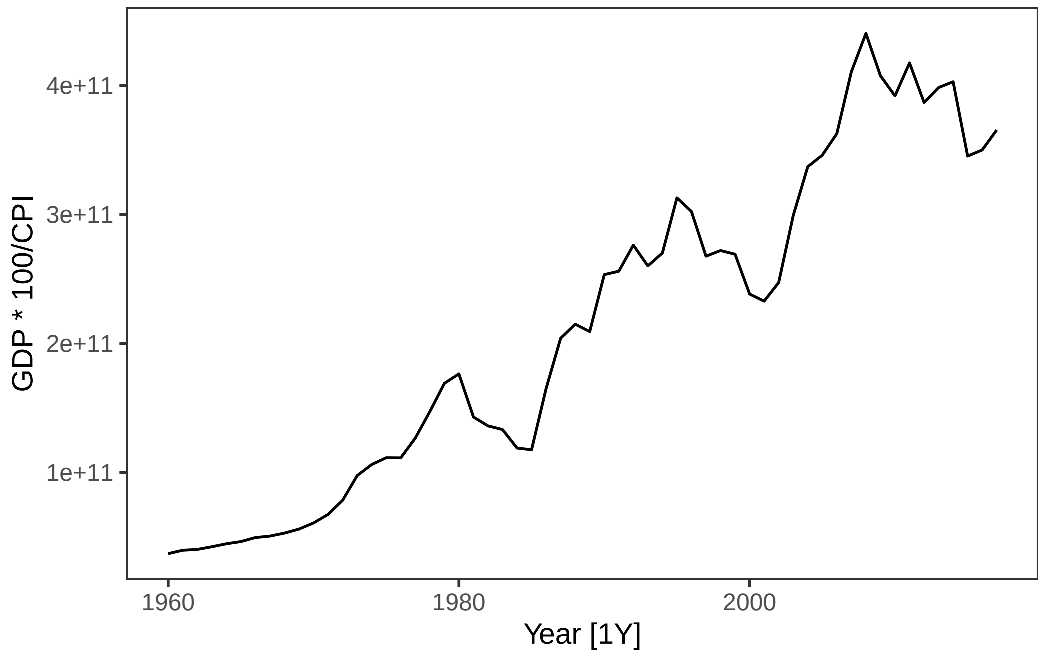
```
dat %>% autoplot(GDP/Population)
```



```r
dat %>% autoplot(GDP/Population * 100/CPI)
```



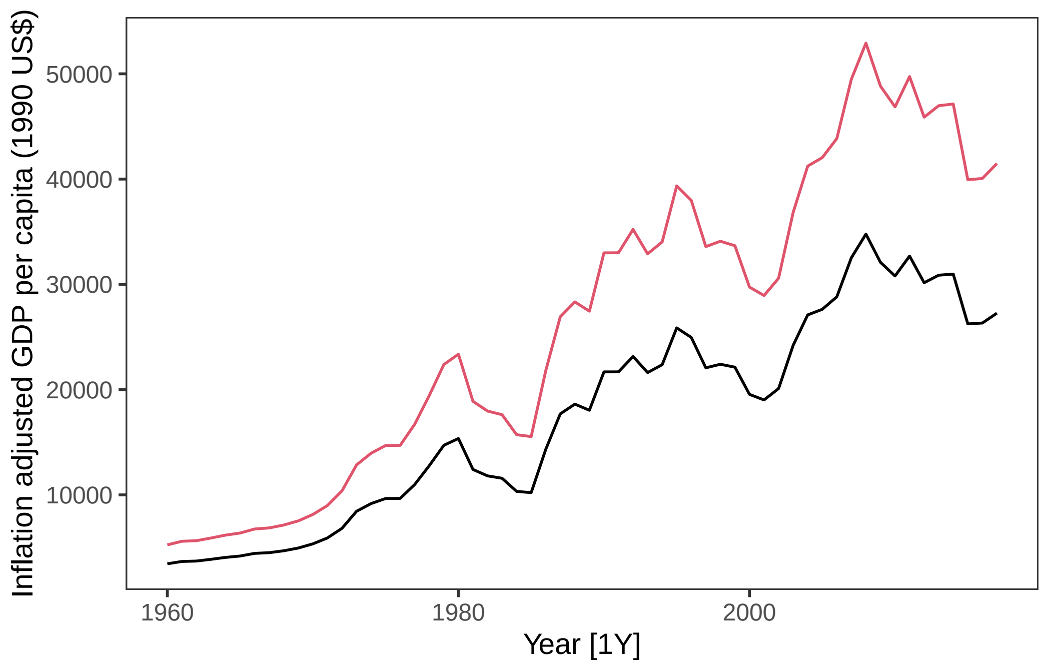
```r
dat %>% autoplot(GDP * 100/CPI)
```



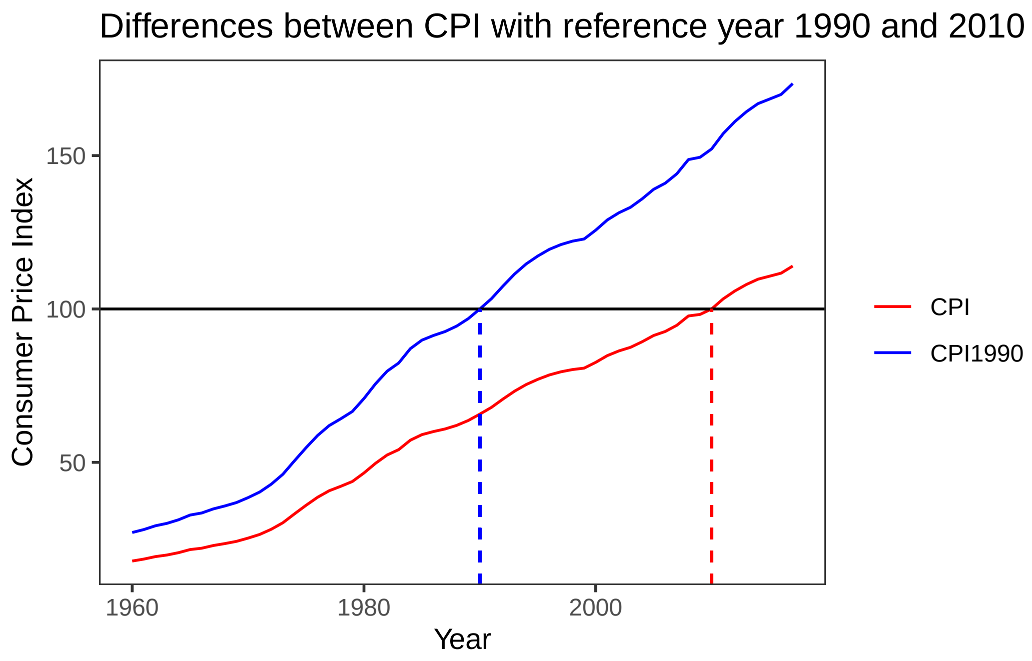
2. In the global_economy data set, the CPI has a reference year of 2010. Do the necessary changes to inflation adjust GDP per capita with 1990 as reference year.

Solution

```
library(fpp3)
library(tidyverse)
dat <- global_economy %>%
  filter(Country == "Austria")

# -- Extracting the CPI in 1990: --
cpi1990 <- dat %>% filter(Year ==1990) %>% pull(CPI)

# -- Transforming such that CPI1990 is 100 in 1990: --
dat <- dat %>% mutate(CPI1990 = CPI / cpi1990 * 100)

# -- Plotting Inflation adjusted GDP per capita: --
dat %>% autoplot(GDP/Population * 100/CPI1990) +
  labs(y = "Inflation adjusted GDP per capita (1990 US$)") +
  geom_line(aes(y= GDP/Population * 100/CPI), col = 2)
```



```
# -- Comparing the two CPIs: --
dat %>%
  pivot_longer(cols = c(CPI,CPI1990)) %>%
  ggplot(aes(x = Year, y = value, col = name)) + geom_line()+
  geom_hline(yintercept = 100) +
  scale_color_manual(values = c("red","blue"))+
  geom_segment(x = 1990,xend = 1990, y = -Inf, yend =100, lty = 2, col = "blue")+
  geom_segment(x = 2010,xend = 2010, y = -Inf, yend =100, lty = 2, col = "red")+
  labs(title = "Differences between CPI with reference year 1990 and 2010",
       y = "Consumer Price Index") +
  theme(legend.title = element_blank())
```



3. Use the quantmod package to download data for another stock. Will a log transformation do a good job for this as well?
4. Use the guerrero feature to select a λ for the Box-Cox transformation on the data from the previous exercise.

4 Time series features

5 Forecasters toolbox

6 Judgemental forecast

7 Regression models

8 ARIMA models

Shiny app for playing with ACF and ARMA models

The following shiny app has been developed by Sondre Hølleland and being administred at https://sholleland.shinyapps.io/ban430_shinyapps. Due to restrictions relating to available computing hours on the free shinyapps account, the app may not work. You may then copy the code below to run the shiny app locally on your own computer. Remember that understanding the details of this code is not necessary.

```r
library(shiny)
library(tidyverse)
library(fpp3)
library(ggpubr)
sbp.width <- 3

theme_set(theme_bw() + theme(panel.grid.major = element_blank(),
                             panel.grid.minor = element_blank()))
# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("ARMA models"),
    tabsetPanel(
      tabPanel("AR(1)",

               sidebarLayout(
                 sidebarPanel(width = sbp.width,
                              uiOutput("artext"),
                   sliderInput("arphi",
                               "phi:",
                               min = -.99,
                               max = .99,
                               value = .7,
                               step = .01),
                   sliderInput("arsigma",
                               "sigma:",
                               min = 0,
                               max = 10,
                               value = 1,
                               step = .2),
                   numericInput("arseed",
                                "Seed:",
                                value = 1234,
                                min = 0,
                                max = 9999,
                                step = 1),
                   numericInput("arn",
                                "Sample size:",
                                value = 500,
                                min = 100,
                                max = 5000,
                                step = 100)
                 ),

                 # Show a plot of the generated distribution
                 mainPanel(
                   plotOutput("arPlot", height = "600px")
                 )
               )),
      tabPanel("MA(1)",

               sidebarLayout(
                 sidebarPanel(width = sbp.width,
                              uiOutput("matext"),
                   sliderInput("matheta",
                               "theta:",
                               min = -1,
                               max = 1,
                               value = .7,
                               step = .01),
                   sliderInput("masigma",
                               "sigma:",
                               min = 0,
                               max = 10,
                               value = 1,
                               step = .2),
                   numericInput("maseed",
                                "Seed:",
                                value = 1234,
                                min = 0,
                                max = 9999,
                                step = 1),
                   numericInput("man",
                                "Sample size:",
                                value = 500,
                                min = 100,
                                max = 5000,
                                step = 100)
                 ),

                 # Show a plot of the generated distribution
                 mainPanel(
                   plotOutput("maPlot", height = "600px")
                 )
               )),
      tabPanel("ARMA(1,1)",

               sidebarLayout(
                 sidebarPanel(width = sbp.width,
                              uiOutput("armatext"),
                   sliderInput("armaphi",
                               "phi:",
                               min = -.99,
                               max = .99,
                               value = .7,
                               step = .01),
                   sliderInput("armatheta",
                               "theta:",
                               min = -1,
                               max = 1,
                               value = .7,
                               step = .01),
                   sliderInput("armasigma",
                               "sigma:",
                               min = 0,
                               max = 10,
                               value = 1,
                               step = .2),
                   numericInput("armaseed",
                                "Seed:",
                                value = 1234,
                                min = 0,
                                max = 9999,
                                step = 1),
                   numericInput("arman",
                                "Sample size:",
                                value = 500,
                                min = 100,
                                max = 5000,
                                step = 100)
                 ),

                 # Show a plot of the generated distribution
                 mainPanel(
                   plotOutput("armaPlot", height = "600px")
                 )
               ))))
    # Sidebar with a slider input for number of bins



# Define server logic required to draw a histogram
server <- function(input, output) {
    output$artext <- renderUI({
      withMathJax("Model: $$Y_t=\\phi \\,Y_{t-1}+Z_t,\\quad Z_t\\sim \\text{i.i.d N}(0,\\s
    })
    output$matext <- renderUI({
      withMathJax("Model: $$Y_t=\\theta\\, Z_{t-1}+Z_t,\\quad Z_t\\sim \\text{i.i.d N}(0,\
    })
    output$armatext <- renderUI({
      withMathJax("Model: $$Y_t=\\phi\\, Y_{t-1}+\\theta \\,Z_{t-1}+Z_t,\\quad Z_t\\sim \\
    })
    output$arPlot <- renderPlot({
      set.seed(input$arseed)
      burnin <- 200
      x <- rnorm(input$arn+burnin, sd = input$arsigma)
      for(i in 2:length(x))
        x[i] <- input$arphi*x[i-1]+rnorm(1, sd = input$arsigma)
      df <- tsibble(x = x[burnin+1:(length(x)-burnin)],
                    t = 1:(length(x)-burnin),
                    index = "t")
      # Theoretical acf:

      theoretical.correlations <-
        tibble(lag = 1:29,
               acf = ARMAacf(ar = c(input$arphi), lag.max = 29, pacf =FALSE)[-1],
               pacf = ARMAacf(ar = c(input$arphi), lag.max = 29, pacf =TRUE))
      # generate bins based on input$bins from ui.R
       ggarrange(
         df %>% autoplot() + scale_y_continuous("AR(1) series")+
           scale_x_continuous("Time index", expand = c(0,0)),
         ggarrange(df %>% ACF() %>% autoplot()+
                     scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                     ylab("Sample PACF"),
         df %>% PACF() %>% autoplot()+
           scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks = seq(0,30,
           ylab("Sample PACF"),
         theoretical.correlations %>% ggplot(aes(x=lag))+
           geom_segment(aes(x=lag,xend=lag,y=0,yend=acf))+geom_hline(yintercept=0)+
           scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks = seq(0,30,
           scale_y_continuous("Theoretical ACF"),
         theoretical.correlations %>% ggplot(aes(x=lag))+
           geom_segment(aes(x=lag,xend=lag,y=0,yend=pacf))+geom_hline(yintercept=0)+
           scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks = seq(0,30,
           scale_y_continuous("Theoretical PACF"),
         nrow = 2, ncol = 2),
         ncol = 1, nrow = 2, heights = c(1,2))
    })

    output$maPlot <- renderPlot({
      set.seed(input$maseed)
      burnin <- 200
      z <- rnorm(input$man+burnin, sd = input$masigma)
      x <- numeric(input$man+burnin)
      for(i in 2:length(x))
        x[i] <- input$matheta*z[i-1]+z[i]
      df <- tsibble(x = x[burnin+1:(length(x)-burnin)],
                    t = 1:(length(x)-burnin),
                    index = "t")

      theoretical.correlations <-
        tibble(lag = 1:29,
               acf = ARMAacf(ma = c(input$matheta), lag.max = 29, pacf =FALSE)[-1],
               pacf = ARMAacf(ma = c(input$matheta), lag.max = 29, pacf =TRUE))
      # generate bins based on input$bins from ui.R
      ggarrange(
        df %>% autoplot() + scale_y_continuous("MA(1) series")+
          scale_x_continuous("Time index", expand = c(0,0)),
        ggarrange(df %>% ACF() %>% autoplot()+
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                                         ylab("Sample ACF"),
                  df %>% PACF() %>% autoplot()+
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                                         ylab("Sample PACF"),
                  theoretical.correlations %>% ggplot(aes(x=lag))+
                    geom_segment(aes(x=lag,xend=lag,y=0,yend=acf))+geom_hline(yintercept=0
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                    scale_y_continuous("Theoretical ACF"),
                  theoretical.correlations %>% ggplot(aes(x=lag))+
                    geom_segment(aes(x=lag,xend=lag,y=0,yend=pacf))+geom_hline(yintercept=
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                    scale_y_continuous("Theoretical PACF"),
                  nrow = 2, ncol = 2),
        ncol = 1, nrow = 2, heights = c(1,2))
    })
    output$armaPlot <- renderPlot({
      set.seed(input$armaseed)
      burnin <- 200
      #sdZ = sqrt(input$armasigma *(1-input$armaphi^2)/(1+2*input$armaphi*input$armatheta
      z <- rnorm(input$arman+burnin, sd = input$armasigma)
      x <- numeric(input$arman+burnin)
      for(i in 2:length(x))
        x[i] <- input$armaphi*x[i-1]+input$armatheta*z[i-1]+z[i]
      df <- tsibble(x = x[burnin+1:(length(x)-burnin)],
                    t = 1:(length(x)-burnin),
                    index = "t")

      theoretical.correlations <-
        tibble(lag = 1:29,
               acf = ARMAacf(ar = c(input$armaphi), ma = c(input$armatheta), lag.max = 29,
               pacf = ARMAacf(ar = c(input$armaphi), ma = c(input$armatheta), lag.max = 29
      # generate bins based on input$bins from ui.R
      ggarrange(
        df %>% autoplot() + scale_y_continuous("MA(1) series")+
          scale_x_continuous("Time index", expand = c(0,0)),
        ggarrange(df %>% ACF() %>% autoplot()+
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                                         ylab("Sample ACF"),
                  df %>% PACF() %>% autoplot()+
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                                         ylab("Sample PACF"),
                  theoretical.correlations %>% ggplot(aes(x=lag))+
                    geom_segment(aes(x=lag,xend=lag,y=0,yend=acf))+geom_hline(yintercept=0
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                    scale_y_continuous("Theoretical ACF"),
                  theoretical.correlations %>% ggplot(aes(x=lag))+
                    geom_segment(aes(x=lag,xend=lag,y=0,yend=pacf))+geom_hline(yintercept=
                    scale_x_continuous("Lag", expand = c(0,0),limit=c(0.5,29.5), breaks =
                    scale_y_continuous("Theoretical PACF"),
                  nrow = 2, ncol = 2),
        ncol = 1, nrow = 2, heights = c(1,2))
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```

9 Volatility forecasting

10 Practicle forecasting issues

```r
1+2
```

```
[1] 3
```

References

Hyndman, Rob J, and George Athanasopoulos. 2018. *Forecasting: Principles and Practice*. OTexts. https://otexts.com/fpp3/.

Wickham, Hadley, and Garrett Grolemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. " O'Reilly Media, Inc.".