Assignment 2

Group 4

2024-09-16

```
options(contrasts = c("contr.sum", "contr.poly"))
require("ggplot2")
require("dplyr")
require("ppcor")
require("caret")
require("tidyr")
require("stringr")
require("lubridate")
require("tsibble")
require("ggfortify")
require("gridExtra")
require("reshape2")

library(imputeTS)   # Time series missing value imputation

library(jsonlite)   # handle JSON data returned by Frost
library(tidyr)      # unpack data from JSON format
library(tidyverse)  # data manipulation with mutate etc, string formatting
library(lubridate)  # process date and time information
library(tsibble)    # special tibbles for time series
library(fpp3)       # autoplot() and gg_season() for time series
library(readr)      # to read the Frost client ID from file
```

Task 1: Dimension reduction on air quality data

Part A: Get

- Obtain data from https://archive.ics.uci.edu/dataset/360/air+quality.
- Provide a brief description of the data based on the information from the website.

```
airquality <- read.table("AirQualityUCI.csv", sep=";", dec=",", header= T)

airqual <- airquality %>%
  dplyr::select(-c("X", "X.1")) %>%
  na.omit() %>%
  mutate(timedate = dmy_hms(paste(Date, Time))) %>%
  dplyr::select(-c("Time", "Date")) %>%
  relocate(timedate) %>%
  as_tibble()

summary(airqual)
```

```
##      timedate                          CO.GT.          PT08.S1.CO.
```

```
##  Min.   :2004-03-10 18:00:00   Min.   :-200.00   Min.   :-200
##  1st Qu.:2004-06-16 05:00:00   1st Qu.:   0.60   1st Qu.: 921
##  Median :2004-09-21 16:00:00   Median :   1.50   Median :1053
##  Mean   :2004-09-21 16:00:00   Mean   : -34.21   Mean   :1049
##  3rd Qu.:2004-12-28 03:00:00   3rd Qu.:   2.60   3rd Qu.:1221
##  Max.   :2005-04-04 14:00:00   Max.   :  11.90   Max.   :2040
##     NMHC.GT.         C6H6.GT.        PT08.S2.NMHC.       NOx.GT.
##  Min.   :-200.0   Min.   :-200.000   Min.   :-200.0   Min.   :-200.0
##  1st Qu.:-200.0   1st Qu.:   4.000   1st Qu.: 711.0   1st Qu.:  50.0
##  Median :-200.0   Median :   7.900   Median : 895.0   Median : 141.0
##  Mean   :-159.1   Mean   :   1.866   Mean   : 894.6   Mean   : 168.6
##  3rd Qu.:-200.0   3rd Qu.:  13.600   3rd Qu.:1105.0   3rd Qu.: 284.0
##  Max.   :1189.0   Max.   :  63.700   Max.   :2214.0   Max.   :1479.0
##   PT08.S3.NOx.     NO2.GT.         PT08.S4.NO2.   PT08.S5.O3.
##  Min.   :-200   Min.   :-200.00   Min.   :-200   Min.   :-200.0
##  1st Qu.: 637   1st Qu.:  53.00   1st Qu.:1185   1st Qu.: 700.0
##  Median : 794   Median :  96.00   Median :1446   Median : 942.0
##  Mean   : 795   Mean   :  58.15   Mean   :1391   Mean   : 975.1
##  3rd Qu.: 960   3rd Qu.: 133.00   3rd Qu.:1662   3rd Qu.:1255.0
##  Max.   :2683   Max.   : 340.00   Max.   :2775   Max.   :2523.0
##        T                  RH               AH
##  Min.   :-200.000   Min.   :-200.00   Min.   :-200.0000
##  1st Qu.:  10.900   1st Qu.:  34.10   1st Qu.:   0.6923
##  Median :  17.200   Median :  48.60   Median :   0.9768
##  Mean   :   9.778   Mean   :  39.49   Mean   :  -6.8376
##  3rd Qu.:  24.100   3rd Qu.:  61.90   3rd Qu.:   1.2962
##  Max.   :  44.600   Max.   :  88.70   Max.   :   2.2310
```

```r
head(airqual)
```

```
## # A tibble: 6 x 14
##   timedate            CO.GT. PT08.S1.CO. NMHC.GT. C6H6.GT. PT08.S2.NMHC. NOx.GT.
##   <dttm>               <dbl>       <int>    <int>    <dbl>         <int>   <int>
## 1 2004-03-10 18:00:00    2.6        1360      150     11.9          1046     166
## 2 2004-03-10 19:00:00    2          1292      112      9.4           955     103
## 3 2004-03-10 20:00:00    2.2        1402       88      9             939     131
## 4 2004-03-10 21:00:00    2.2        1376       80      9.2           948     172
## 5 2004-03-10 22:00:00    1.6        1272       51      6.5           836     131
## 6 2004-03-10 23:00:00    1.2        1197       38      4.7           750      89
## # i 7 more variables: PT08.S3.NOx. <int>, NO2.GT. <int>, PT08.S4.NO2. <int>,
## #   PT08.S5.O3. <int>, T <dbl>, RH <dbl>, AH <dbl>
```

Contains the responses of a gas multisensor device deployed on the field in an Italian city. Hourly responses averages are recorded along with gas concentrations references from a certified analyzer. Multivariate (15) and time series Has missing values.

Hints

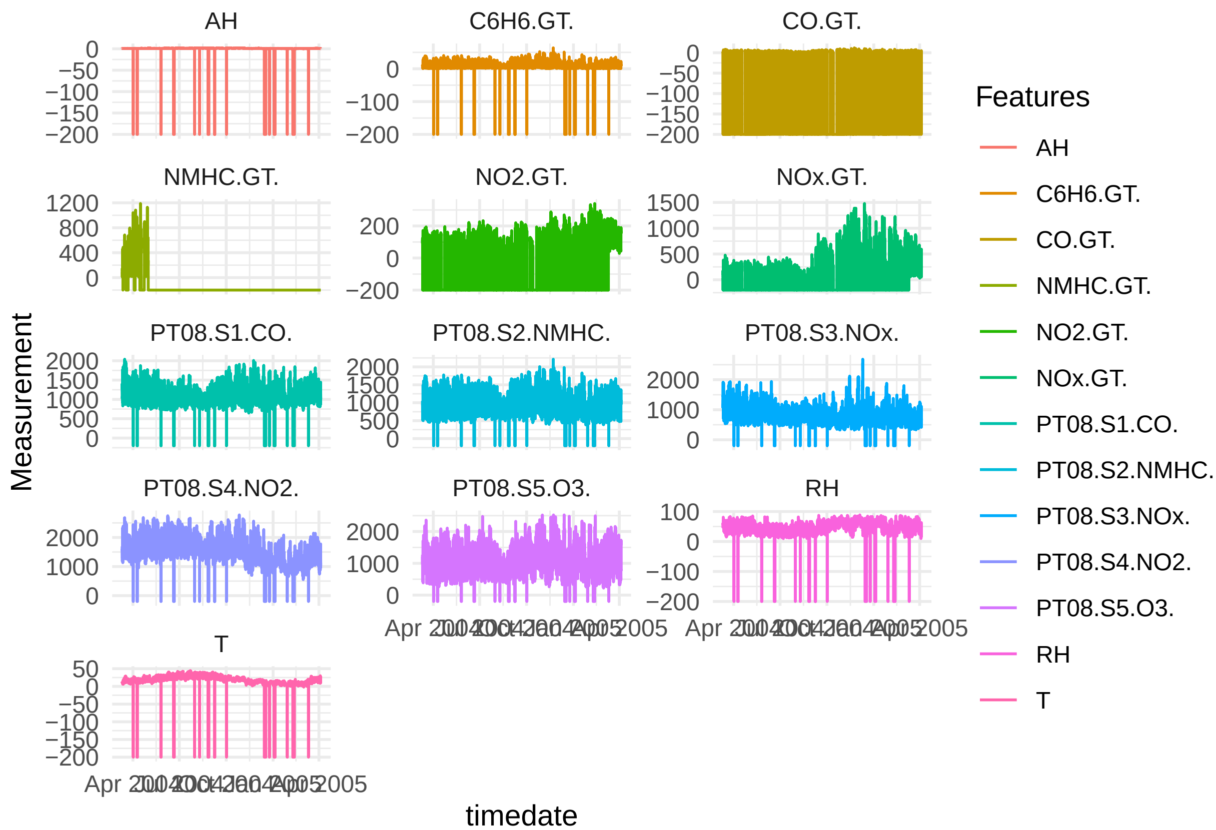
- See options of `read.table()` for correct import

Part B: Import and Visualize

- Load the data and convert to tsibble.
  - Make sure dates and hours are converted into proper time objects
  - Remove incomplete days at beginning and end of data
- Plot the data as is, preferably as multiple panels in a single plot

- Describe the data. What is most striking?

```
airqual %>%
  pivot_longer(!timedate, names_to="Features", values_to="Measurement") %>%
  ggplot(aes(x =  timedate, y = Measurement, col = Features)) +
    geom_line() +
    theme_minimal() +
  facet_wrap(~ Features, scales = "free_y", ncol = 3)
```



There seems to be multiple -200 in all the numerical (integer and Categorical) data that seems to be outliers or missing values. Should probably be removed or imputed from the dataset.

Part C: PCA of data as is

- Perform PCA on the data as prepared in B

```
pc <- prcomp(airqual[,-1])
summary(pc)
```

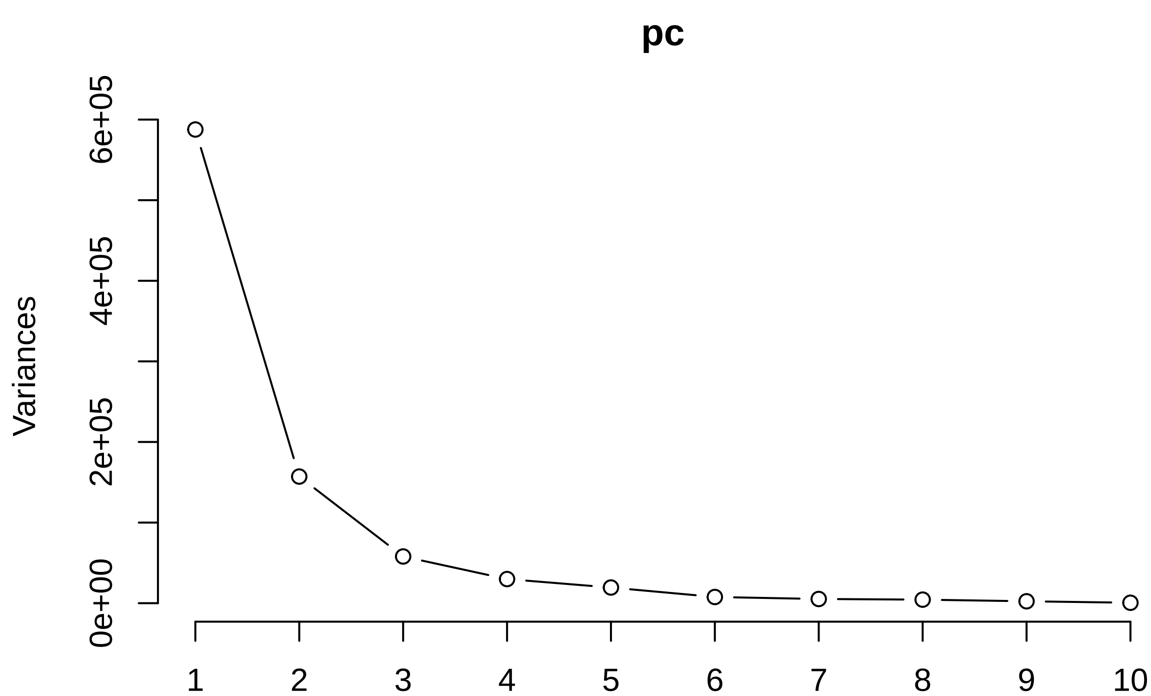
```
## Importance of components:
##                              PC1      PC2       PC3       PC4       PC5      PC6
## Standard deviation     766.6076 396.3514 240.73023 172.96207 139.37119 87.72389
## Proportion of Variance   0.6736   0.1801   0.06643   0.03429   0.02226  0.00882
## Cumulative Proportion    0.6736   0.8537   0.92012   0.95441   0.97668  0.98550
##                             PC7      PC8      PC9     PC10     PC11    PC12
## Standard deviation     72.12259 66.31383 48.62607 23.26879 11.78504 2.29019
## Proportion of Variance  0.00596  0.00504  0.00271  0.00062  0.00016 0.00001
## Cumulative Proportion   0.99146  0.99650  0.99921  0.99983  0.99999 1.00000
```

```
##                              PC13
## Standard deviation     0.7725
## Proportion of Variance 0.0000
## Cumulative Proportion  1.0000
```

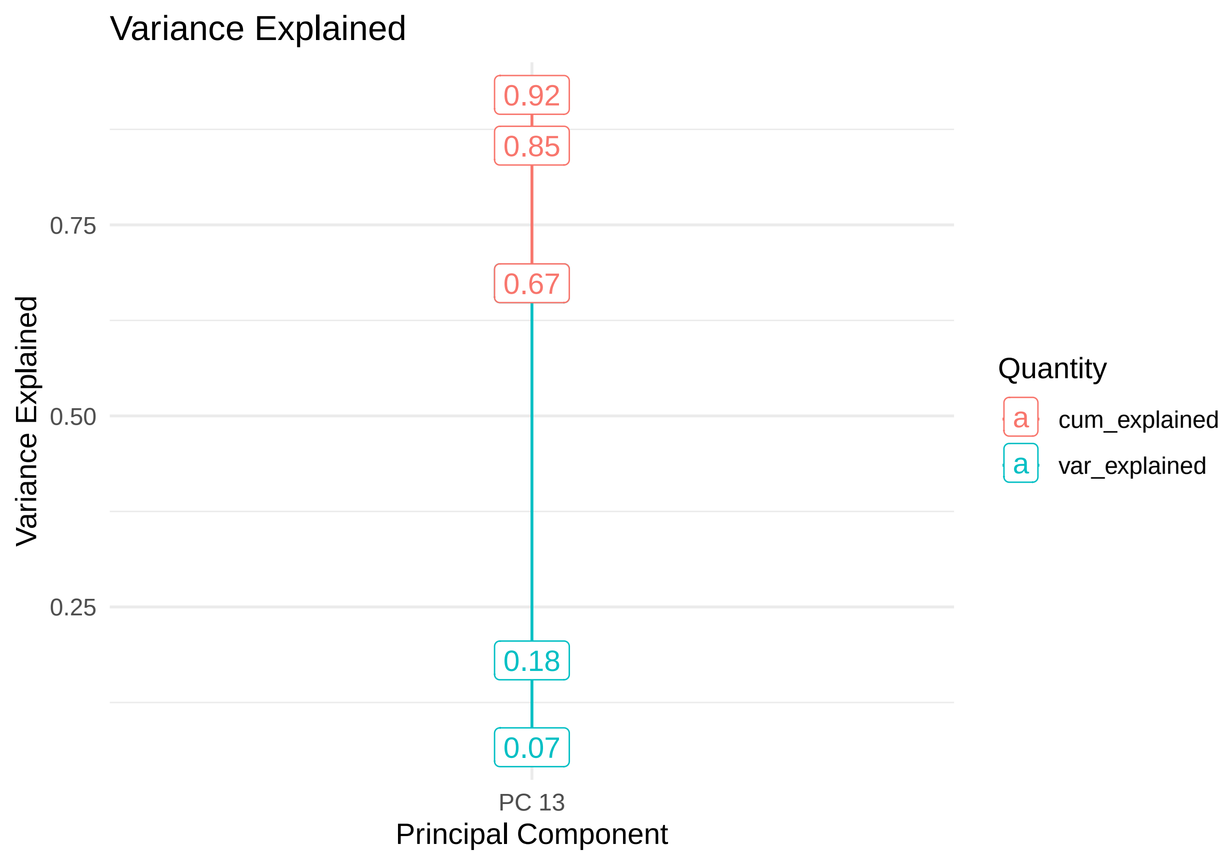
- Create a screeplot and create biplots for 1st and 2nd and for 2nd and 3rd PCs

Screeplot

```r
plot(pc, type = "l")
```



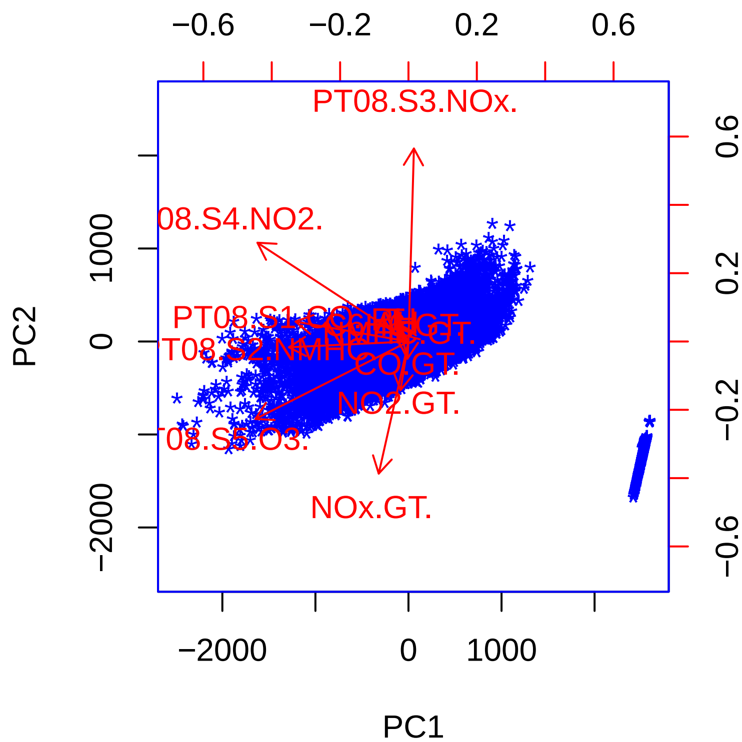

```r
pc_v <- data.frame(PC = paste0("PC ", ncol(pc$x)),
                   var_explained = pc$sdev^2 / sum(pc$sdev^2)) %>%
          mutate(cum_explained = cumsum(var_explained))

pp <- pc_v[1:3,] %>%
  pivot_longer(!PC, names_to="Quantity", values_to="Explained") %>%
  ggplot(aes(x = PC, y = Explained, color=Quantity, group=Quantity))+
  geom_line() + geom_point() +
  theme_minimal() +
  labs(title = "Variance Explained", x = "Principal Component",
       y = "Variance Explained")


pp + geom_label(aes(label = round(Explained, 2)))
```



0.91 of variance is explained in the first 3 principal components

Biplots

```
biplot(pc, scale=0, col=c('blue', 'red'), xlabs=rep('*', nrow(pc$x[, 1:3])))
```



- Can clearly see the -200 outliers
- PRO8.S3.NOx positivly correlated between PC1 and PC2
- NOX.GT. opposite
- Plot the scores for the PCs
- Comment on the results. Can you relate some features to your observations in part B?

Score plot

```
d_pc <- data.frame(Time=airqual[,1], pc$x[,1:3])
head(d_pc)
```

```
##              timedate        PC1      PC2      PC3
## 1 2004-03-10 18:00:00 -530.23707 228.6828 177.0456
## 2 2004-03-10 19:00:00 -212.28933 377.7837 210.4454
## 3 2004-03-10 20:00:00 -309.51936 313.9656 247.2225
## 4 2004-03-10 21:00:00 -396.57641 230.7925 242.9392
## 5 2004-03-10 22:00:00 -192.91228 317.7539 298.5658
## 6 2004-03-10 23:00:00   27.54921 441.3118 357.9443
```

Hints

- `ggfortify` provides `autoplot()` for PCA results for ggplot-style biplots
- To plot the scores, you can use the same code as for plotting the original data

Part D: Missing values

- Identify missing values in the time series

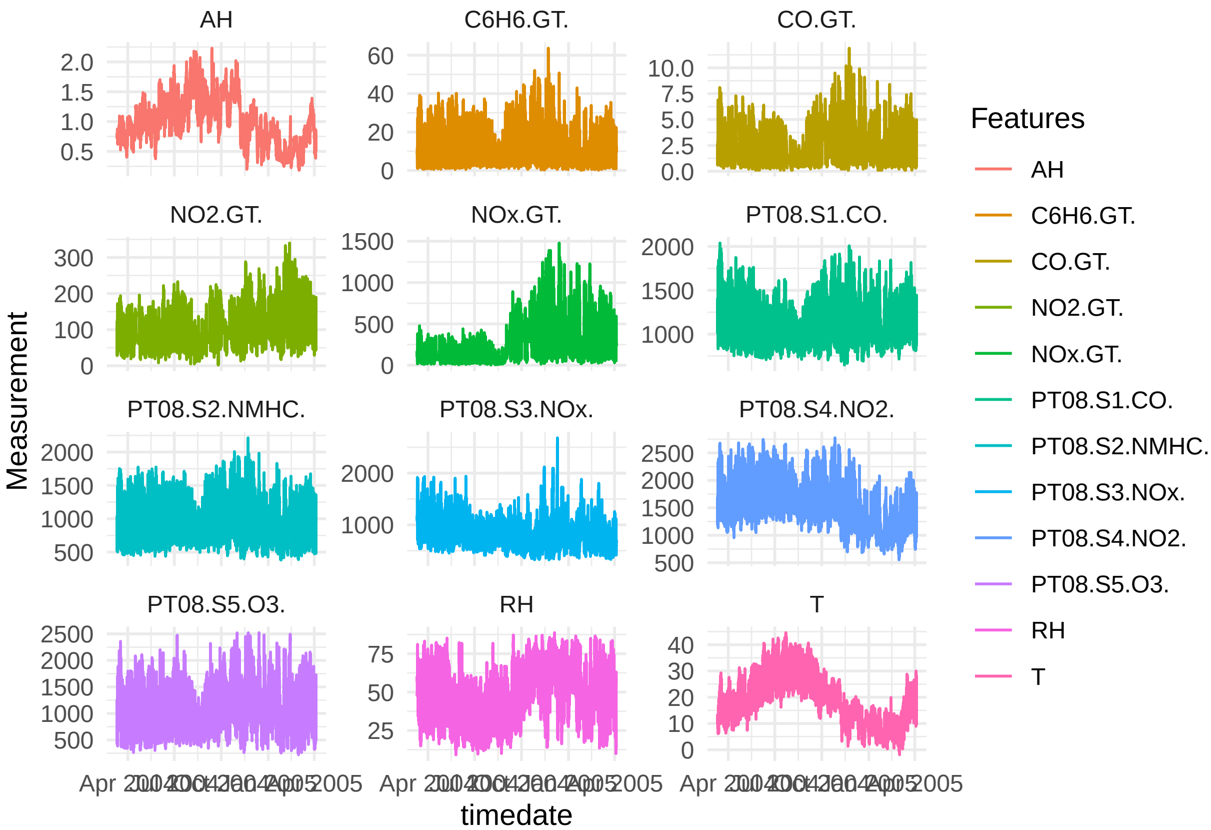
The website says that all -200 are NA

- Investigate to which degree missing values occur at the same time for multiple sensors
- Is one or are multiple sensors behaving peculiarly? How would you handle this?

NHMC.GT has almost all missing values after a certain time, as seen in the visualizion, therfore remove to not skew the result away from real data.

- Discuss options for handling missing values: (a) drop all time points containing any missing value, (b) impute values for missing values. In case of (b) choose a method for imputation. Justify your decisions.
- **(a)** By dropping all NA we only get real data to do anaylizis on, but we miss out on number of datapoints so our overall accuracy becomes lower. In addition it is not possible as we need a regular time axis.
- **(b)** By imputing we can still use rows that are mostly NA free while having the model be skewed as little as possible by these values.
  - Choose to set the NA to the mean value of the overall dataset, therby making the datapoint the most probable if we were to get it randomly. By ussing rolling average we also get the most probable next step from the previous k steps, therfore making it more locally accurate than just using mean.
- At the end of this step, you should have a version of the data containing only valid values. Plot these data as in Part B.

```r
airqual_c <- airqual %>%
  dplyr::select(-c("NMHC.GT."))

airqual_c[airqual_c == -200] = NA

#airqual_c <- imputeTS::na_mean(airqual_c, option = "mean")
# Missing Value Imputation by Weighted Moving Average, Simple moving average
# Plain rolling average
airqual_c <- imputeTS::na_ma(airqual_c, k = 4, weighting = "simple")

airqual_c %>%
  pivot_longer(!timedate, names_to="Features", values_to="Measurement") %>%
  ggplot(aes(x =  timedate, y = Measurement, col = Features)) +
  geom_line() +
  theme_minimal() +
  facet_wrap(~ Features, scales = "free_y", ncol = 3)
```



Hints

- Remember `imputeTS`
- You can apply its functions to an entire dataframe, will be done column-wise

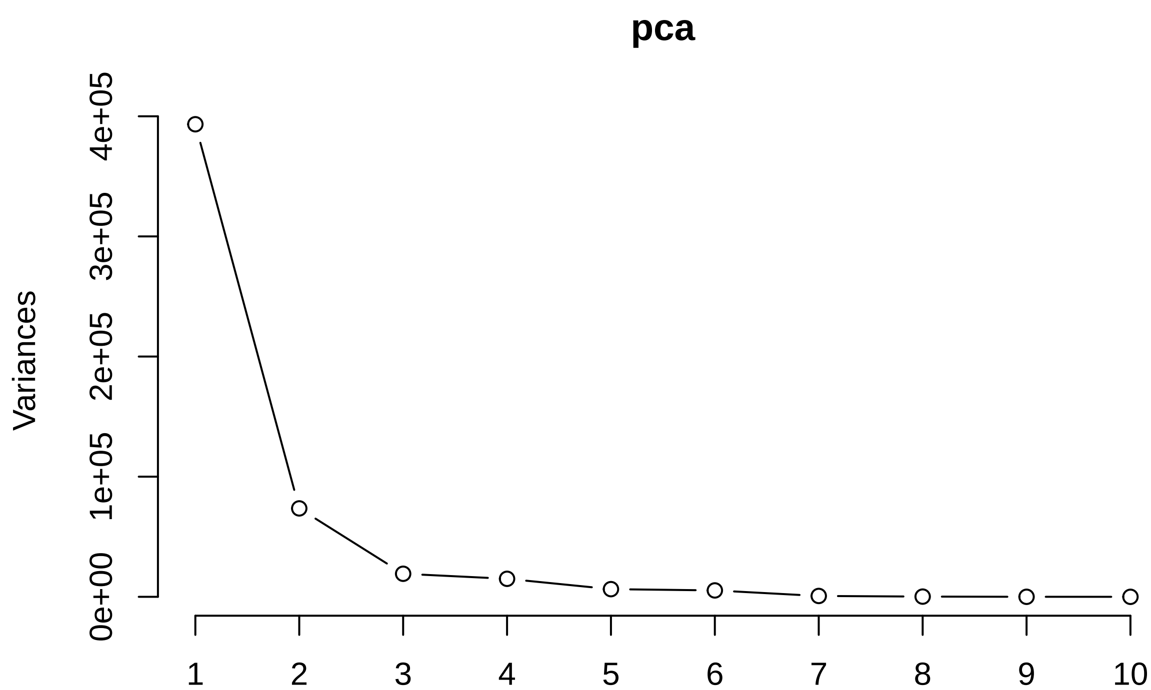
Part E: PCA of cleaned data

- Perform PCA on the data as prepared in D

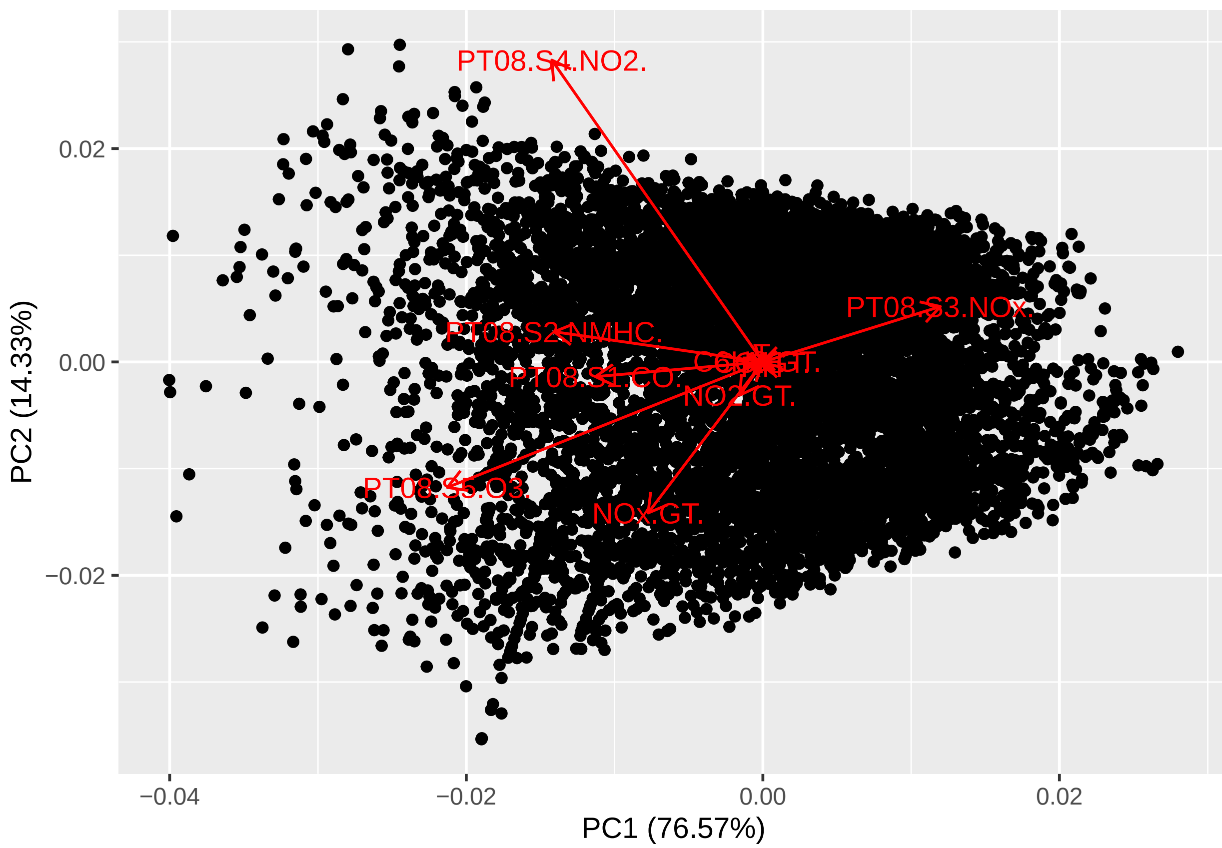
```
pca <- prcomp(airqual_c[,-1]) #, center = TRUE, scale. = TRUE
```

- Create a screeplot and biplots for 1st/2nd, 2nd/3rd, 3rd/4th PC

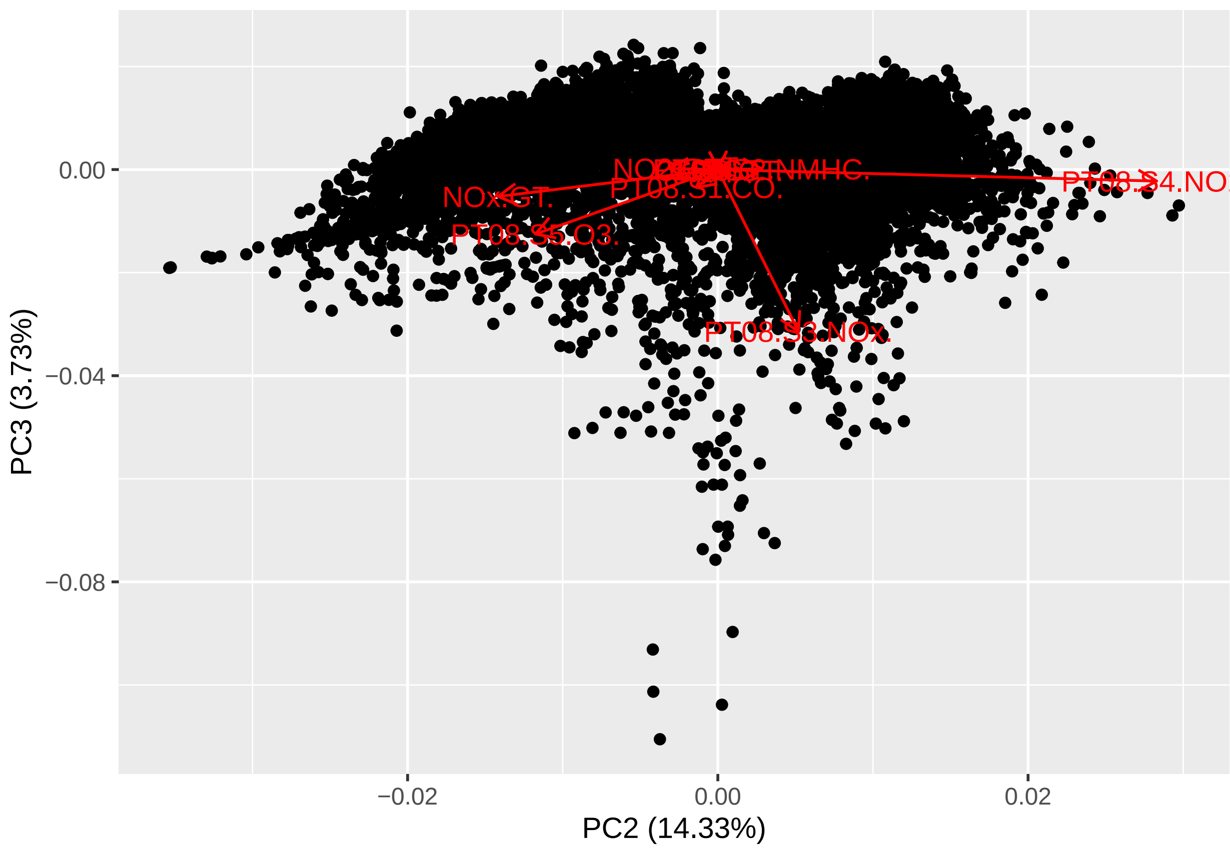
```
# screeplot
pca <- prcomp(airqual_c[,-1])
plot(pca, type = "l")
```



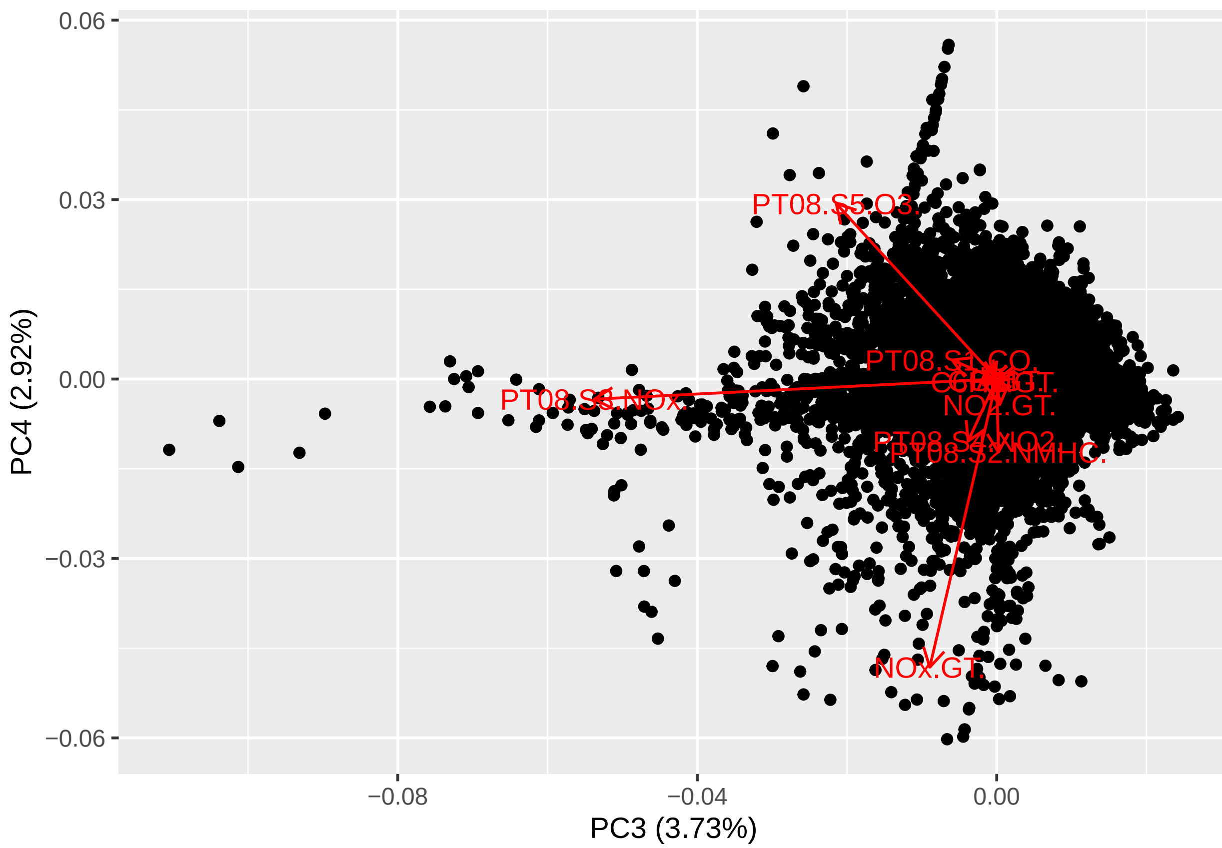

```r
# biplots
autoplot(pca, x=1,y=2, loadings=TRUE, loadings.label=TRUE)
```




```r
autoplot(pca, x=2,y=3, loadings=TRUE, loadings.label=TRUE)
```




```r
autoplot(pca, x=3,y=4, loadings=TRUE, loadings.label=TRUE)
```



Biplot red arrow original axis of data

- Compute total variance explained by 1st, 1st and 2nd, 1st to 3rd, ... PCs

```
#Variance explained
summary(pca)$importance[3,]
```

```
##     PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10
## 0.76570 0.90897 0.94631 0.97548 0.98791 0.99826 0.99964 0.99997 1.00000 1.00000
##    PC11    PC12
## 1.00000 1.00000
```

- Choose how many PCs to keep and transform data back to original sample space

Keep to 0.99 mark, so PC1 to PC6

```
t <- airqual_c$timedate
x_1 <- pca$x[, 1:6] %*% t(pca$rotation[, 1:6])
x_2 <- t(pca$center + pca$scale * t(x_1))
```

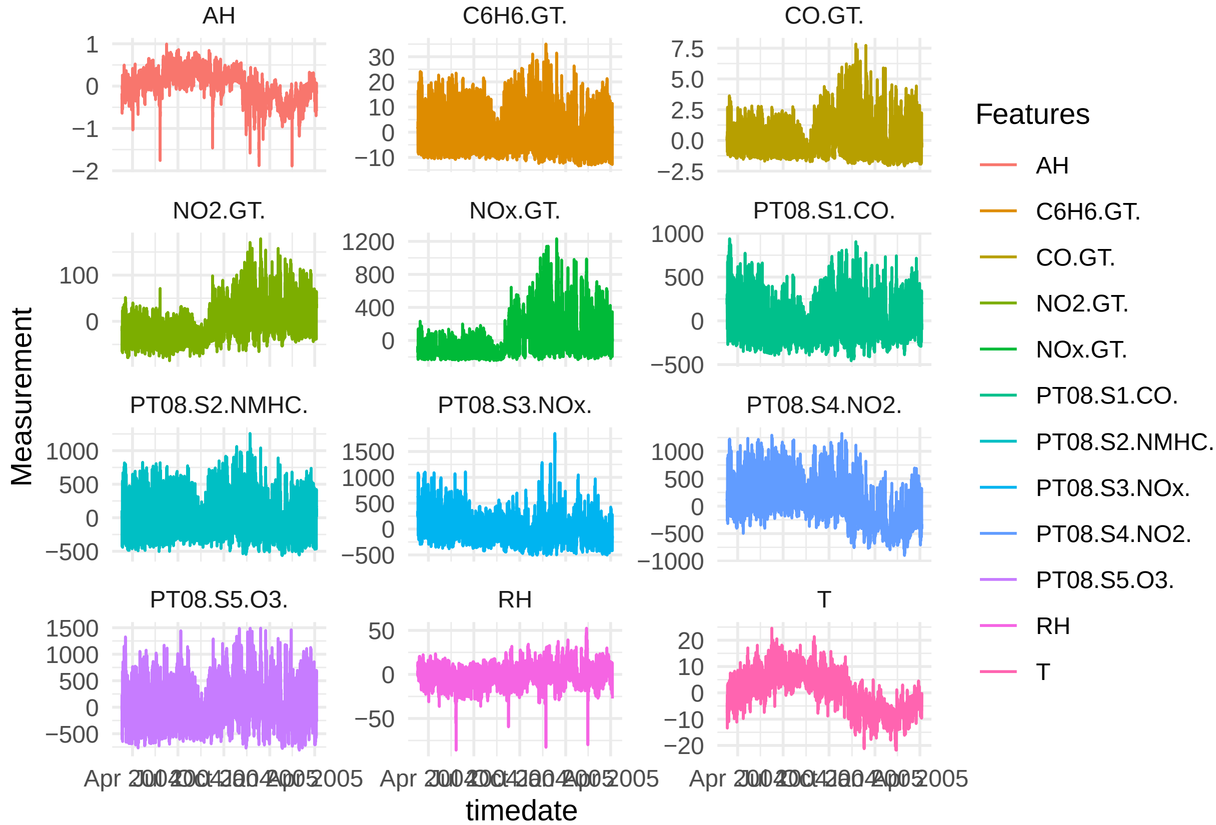
$$S = XL \Leftrightarrow SL^{-1} = XLL^{-1} \Leftrightarrow X = SL^{-1} = SL^T$$

S : Scores L : Loadings X : Original matrix

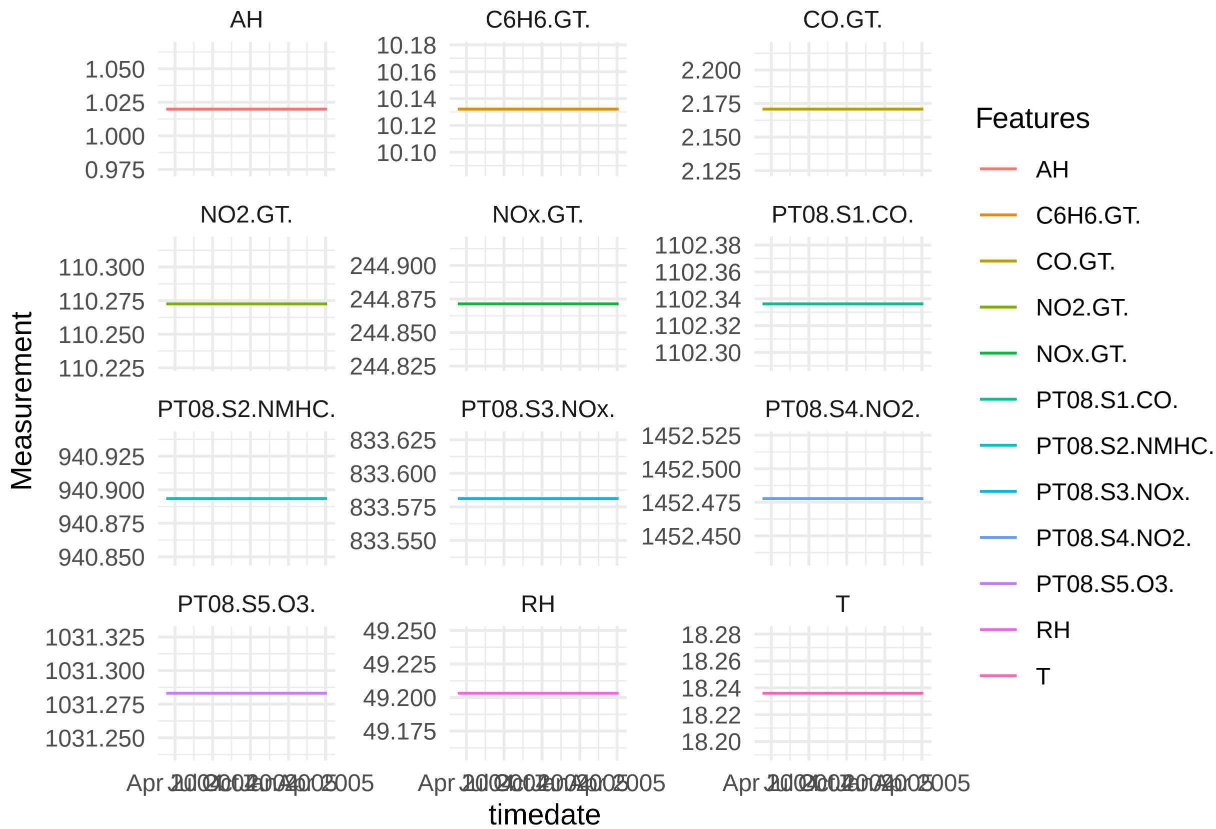
- Plot the result against the cleaned data, compare and discuss

```
data.frame(timedate = airqual_c$timedate, x_1) %>%
  pivot_longer(!timedate, names_to="Features", values_to="Measurement") %>%
  ggplot(aes(x =  timedate, y = Measurement, col = Features)) +
  geom_line() +
```

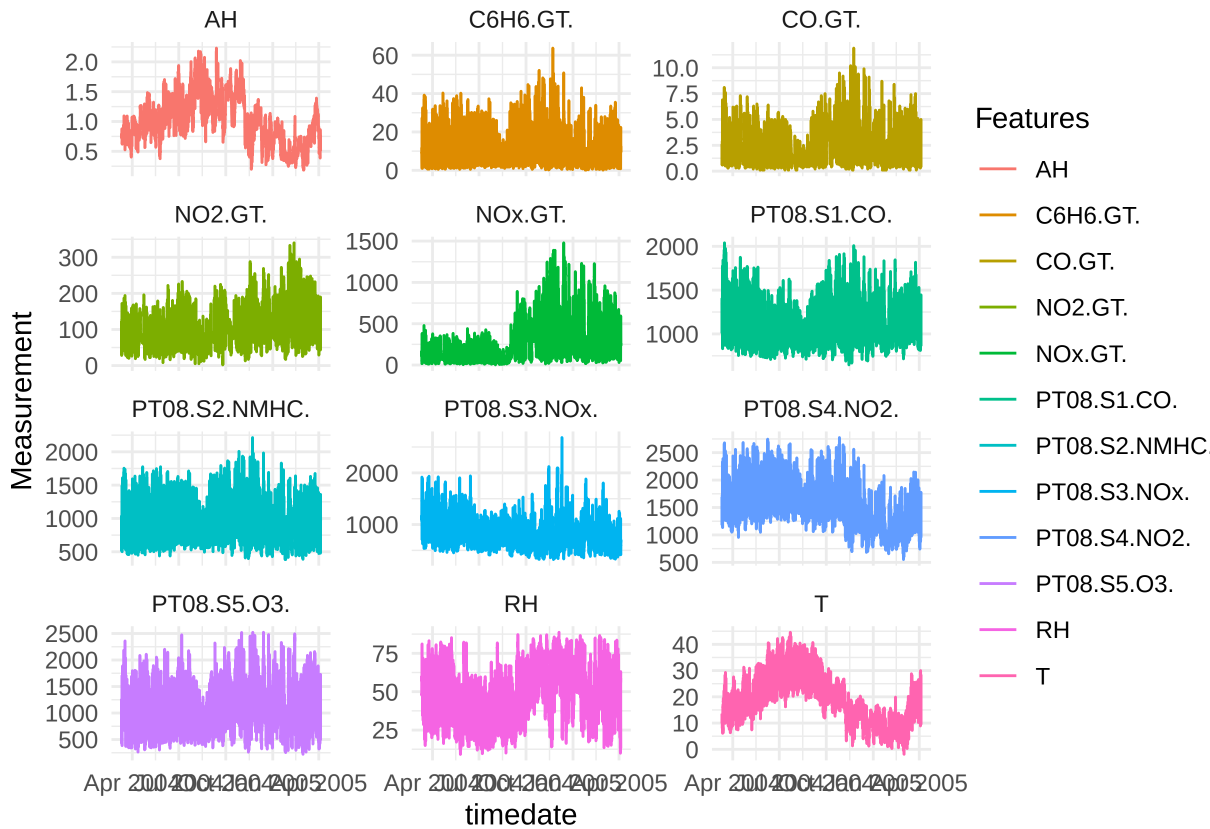
```
  theme_minimal() +
  facet_wrap(~ Features, scales = "free_y", ncol = 3)
```




```
data.frame(timedate = airqual_c$timedate, x_2) %>%
  pivot_longer(!timedate, names_to="Features", values_to="Measurement") %>%
  ggplot(aes(x =  timedate, y = Measurement, col = Features)) +
  geom_line() +
  theme_minimal() +
  facet_wrap(~ Features, scales = "free_y", ncol = 3)
```




```r
airqual_c %>%
  pivot_longer(!timedate, names_to="Features", values_to="Measurement") %>%
  ggplot(aes(x =  timedate, y = Measurement, col = Features)) +
  geom_line() +
  theme_minimal() +
  facet_wrap(~ Features, scales = "free_y", ncol = 3)
```

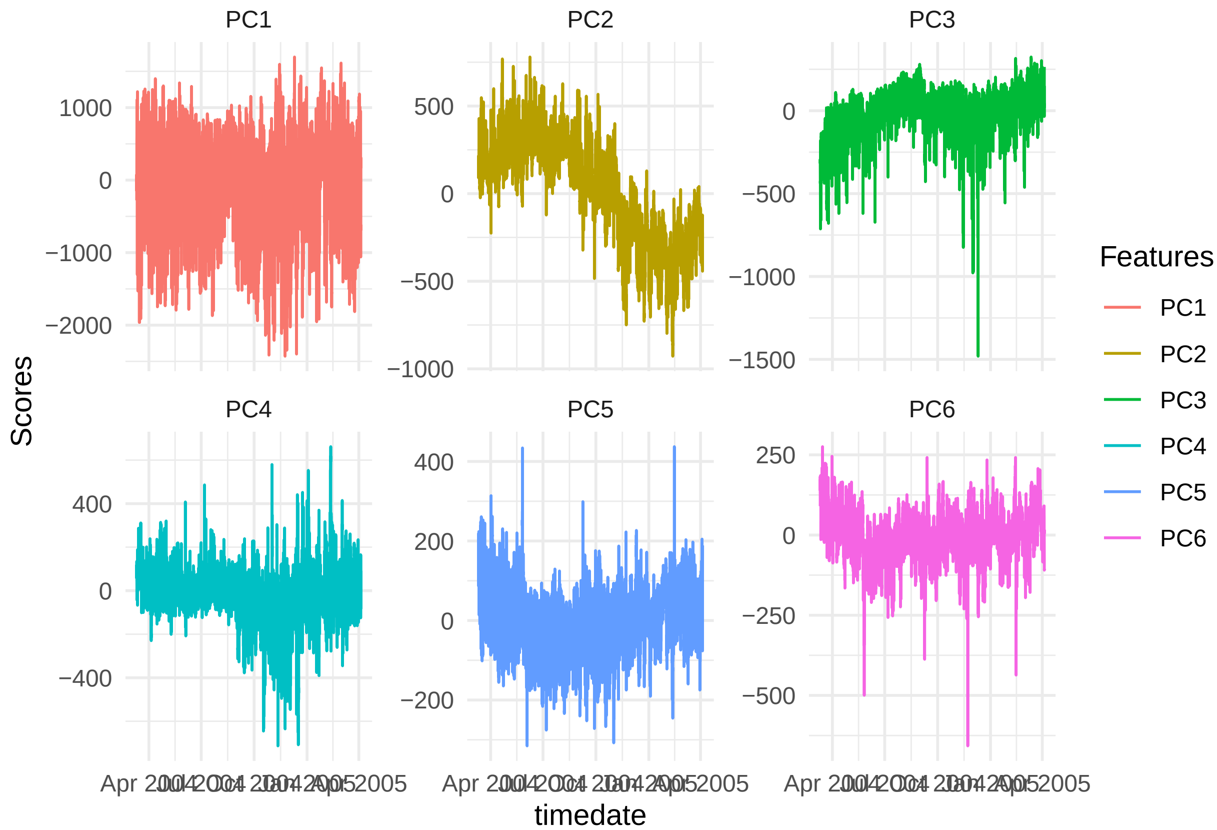


Looks mostly the same, but deviates slightly as we removed all after PC7.

- Also plot the scores, zoom in to short time intervals and look at periodicity

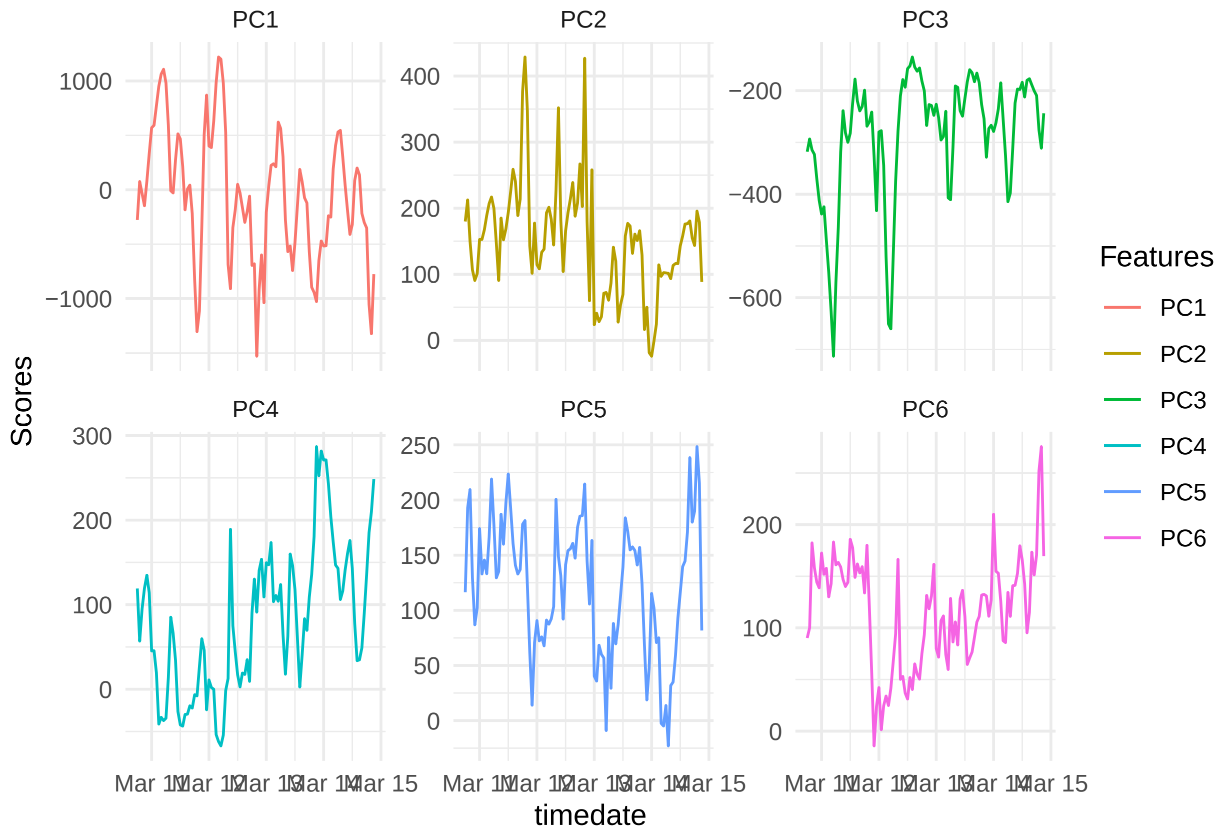
Whole

```
data.frame(timedate = airqual_c$timedate, pca$x[,1:6]) %>%
  pivot_longer(!timedate, names_to="Features", values_to="Scores") %>%
  ggplot(aes(x =  timedate, y = Scores, col = Features)) +
  geom_line() +
  theme_minimal() +
  facet_wrap(~ Features, scales = "free_y", ncol = 3)
```



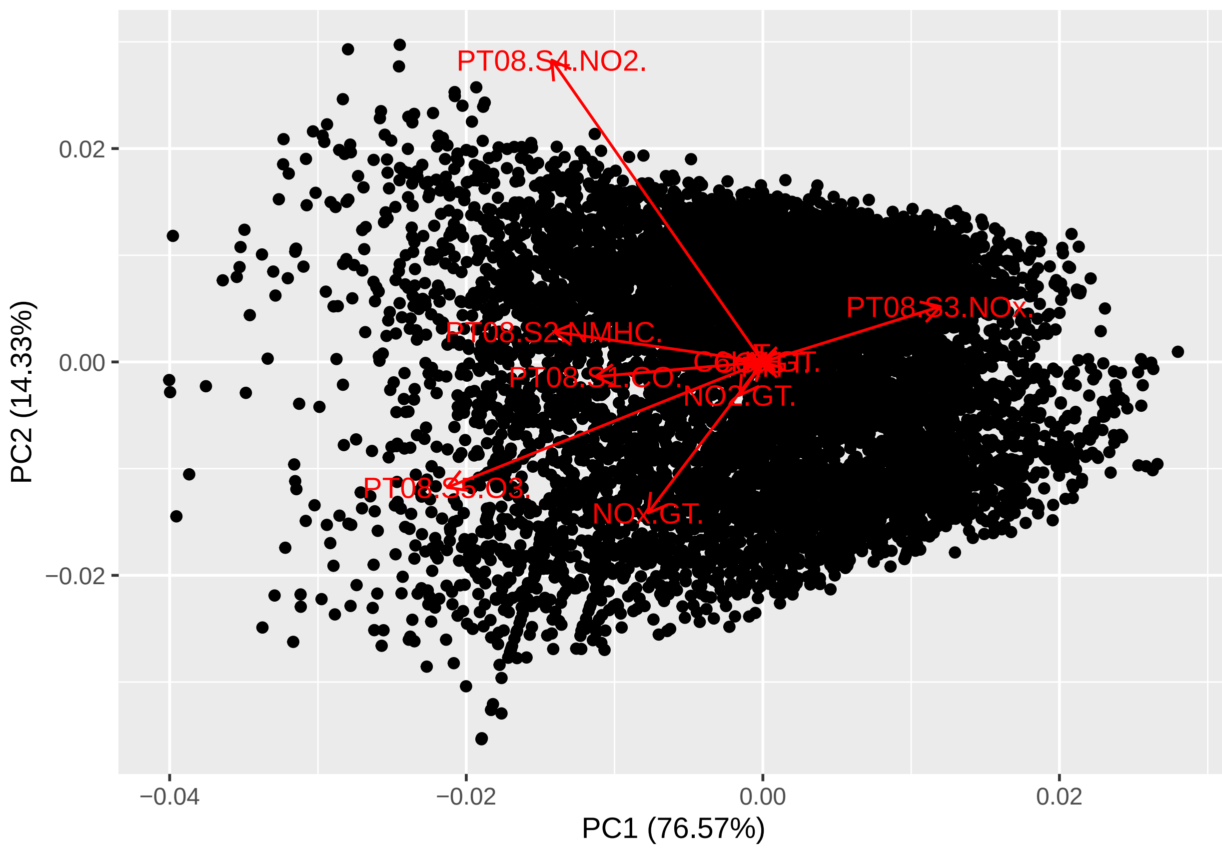
Zoomed

```
data.frame(timedate = airqual_c$timedate[1:100], pca$x[1:100,1:6]) %>%
  pivot_longer(!timedate, names_to="Features", values_to="Scores") %>%
  ggplot(aes(x =  timedate, y = Scores, col = Features)) +
  geom_line() +
  theme_minimal() +
  facet_wrap(~ Features, scales = "free_y", ncol = 3)
```



- Can you interpret certain PCs?

```
autoplot(pca, x=1,y=2, loadings=TRUE, loadings.label=TRUE)
```



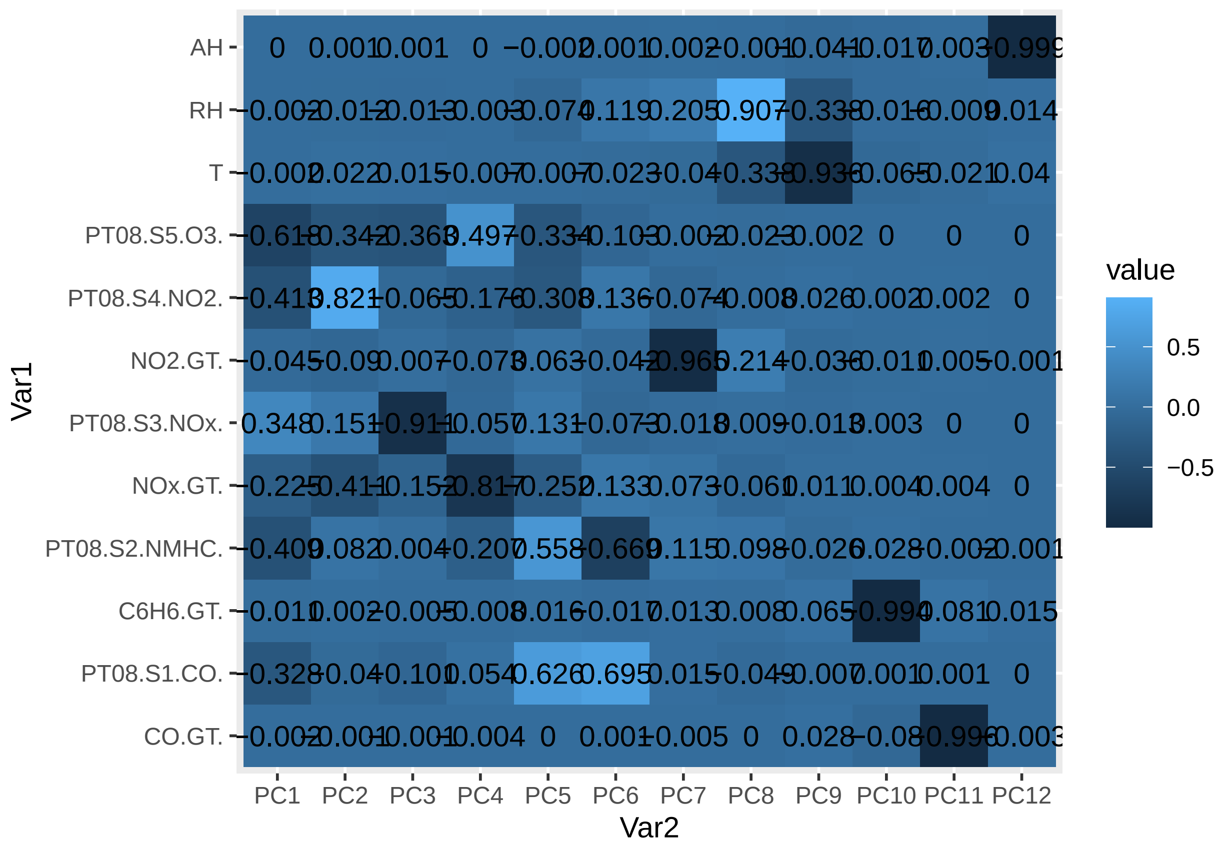

```r
data.frame(pca$rotation)
```

```
##                        PC1           PC2           PC3           PC4
## CO.GT.        -0.0018481988 -0.0007049452 -0.0014077536 -0.0043626668
## PT08.S1.CO.   -0.3276959483 -0.0400440912 -0.1009593770  0.0537630030
## C6H6.GT.      -0.0111497193  0.0023373054 -0.0045012384 -0.0078980451
## PT08.S2.NMHC. -0.4085995243  0.0821947866  0.0037025890 -0.2073677556
## NOx.GT.       -0.2247579859 -0.4110090314 -0.1516711696 -0.8168812736
## PT08.S3.NOx.   0.3475027920  0.1508453517 -0.9111126863 -0.0570822819
## NO2.GT.       -0.0452313753 -0.0903548818  0.0065398011 -0.0728416677
## PT08.S4.NO2.  -0.4131515654  0.8212890426 -0.0647945566 -0.1760872468
## PT08.S5.O3.   -0.6176661288 -0.3415204901 -0.3633580682  0.4970953230
## T             -0.0021215778  0.0223286223  0.0146078597 -0.0068587449
## RH            -0.0019124088 -0.0116763960 -0.0125220967 -0.0031264697
## AH            -0.0001449796  0.0009679888  0.0006321034 -0.0003673498
##                        PC5          PC6          PC7           PC8
## CO.GT.         0.0002686689  0.000757493 -0.005358532  6.260908e-06
## PT08.S1.CO.    0.6259430252  0.695331376  0.014987171 -4.858268e-02
## C6H6.GT.       0.0163460121 -0.017403082  0.013341054  7.606726e-03
## PT08.S2.NMHC.  0.5577650907 -0.669313381  0.114878545  9.781349e-02
## NOx.GT.       -0.2518500677  0.132534035  0.073466372 -6.084502e-02
## PT08.S3.NOx.   0.1312507403 -0.072666427 -0.017993617  9.409716e-03
## NO2.GT.        0.0633409652 -0.041565772 -0.965138888  2.139533e-01
## PT08.S4.NO2.  -0.3082878182  0.135577313 -0.073581584 -8.415517e-03
## PT08.S5.O3.   -0.3340438218 -0.102865230 -0.001629221 -2.331928e-02
## T             -0.0070602392 -0.022954286 -0.040469919 -3.383485e-01
```

```
## RH            -0.0740118957  0.119022190  0.205178331  9.073875e-01
## AH            -0.0021484402  0.001182079  0.002142285 -1.130502e-03
##                         PC9          PC10          PC11          PC12
## CO.GT.          0.028421694 -0.0798924563 -0.9963659489 -3.164062e-03
## PT08.S1.CO.    -0.007152100  0.0011586314  0.0008637521 -2.323526e-04
## C6H6.GT.        0.065223128 -0.9939169941  0.0814893649  1.470071e-02
## PT08.S2.NMHC.  -0.025579860  0.0277063380 -0.0023210316 -1.066060e-03
## NOx.GT.         0.011434225  0.0038849340  0.0041500425  2.374803e-04
## PT08.S3.NOx.   -0.013083805  0.0034508345  0.0002142262 -3.947865e-04
## NO2.GT.        -0.035903855 -0.0106244017  0.0054654120 -8.600059e-04
## PT08.S4.NO2.    0.025888303  0.0016380067  0.0020691370  4.622803e-04
## PT08.S5.O3.    -0.002484763 -0.0002733566 -0.0004846717  7.147926e-05
## T              -0.936057264 -0.0653693331 -0.0213934536  4.010257e-02
## RH             -0.337718440 -0.0162504714 -0.0093587511  1.391709e-02
## AH             -0.041331962 -0.0172440281  0.0033651508 -9.989843e-01
```

```r
melt(pca$rotation) %>%
  ggplot(aes(Var2, Var1)) +
  geom_tile(aes(fill = value)) +
  geom_text(aes(fill = value, label = round(value, 3)))
```

```
## Warning in geom_text(aes(fill = value, label = round(value, 3))): Ignoring
## unknown aesthetics: fill
```



By looking at the loadings of PC1 one can see that PT08.S5.O3 is the most important feature with respect to explainable variance. Scores determine witch which weight each PC enter into each row of X

Task 2: STL and correlation on weather data

Part A: Data collection for a single station

Based on material from the lectures, write an R function that can obtain a daily average temperature series for a meteorological station from the Norwegian Met Institute's Frost service. The function shall return a tsibble.

```
.client_id <- str_trim(read_file("client_id.txt"))

# Server to collect data from and resource we want from server
server <- "frost.met.no"
resource <- "observations/v0.jsonld"

# Station(s) we want data for. SN17850 is the station ID for Ås (Blindern is SN18700)
sources <- 'SN17850'

# Type of data we want, P1D means daily data
elements <- 'mean(air_temperature P1D)'

# Time range we want data for
reference_time <- '1874-01-01/2023-12-31'

# Specify that we want mean temperature calculated from midnight to midnight
timeoffsets <- 'PT0H'


.query_url <- str_glue("https://{.client_id}@{server}/{resource}?sources={sources}&referencetime={refere
```

```
# Set this to TRUE to generate a json file
get_data_from_frost = TRUE
weather_file = "weather_data_json.rds.bz2"
if ( get_data_from_frost ) {
  raw_data <- try(fromJSON(URLencode(.query_url), flatten=TRUE))

  if (class(raw_data) != 'try-error') {
    print("Data retrieved from frost.met.no!")
    write_rds(raw_data, weather_file, compress="bz2", text=TRUE)  # JSON represents data as text
    print(str_glue("Raw data (JSON) written to '{weather_file}'"))
  } else {
    print("Error: the data retrieval was not successful!")
  }
} else {
  raw_data <- read_rds(weather_file)
  print(str_glue("Raw data (JSON) read from '{weather_file}'"))
}
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'weather_data_json.rds.bz2'
```

```
df <- unnest(raw_data$data, cols = c(observations))
```

```
head(df)
```

```
## # A tibble: 6 x 14
##   sourceId  referenceTime        elementId value unit  timeOffset timeResolution
##   <chr>     <chr>                <chr>     <dbl> <chr> <chr>      <chr>
```

```
## 1 SN17850:0 1874-01-01T00:00:00~ mean(air~   3.2 degC  PT0H       P1D
## 2 SN17850:0 1874-01-02T00:00:00~ mean(air~   4.1 degC  PT0H       P1D
## 3 SN17850:0 1874-01-03T00:00:00~ mean(air~   2.5 degC  PT0H       P1D
## 4 SN17850:0 1874-01-04T00:00:00~ mean(air~   0.3 degC  PT0H       P1D
## 5 SN17850:0 1874-01-05T00:00:00~ mean(air~  -3.9 degC  PT0H       P1D
## 6 SN17850:0 1874-01-06T00:00:00~ mean(air~   1.9 degC  PT0H       P1D
## # i 7 more variables: timeSeriesId <int>, performanceCategory <chr>,
## #   exposureCategory <chr>, qualityCode <int>, level.levelType <chr>,
## #   level.unit <chr>, level.value <int>
```

```
df |> dplyr::select(referenceTime, value) |>
      mutate(referenceTime=as.Date(referenceTime)) |>
      rename(Date=referenceTime, Temp=value) |>
      as_tsibble(index = Date) -> dc
head(dc)
```

```
## # A tsibble: 6 x 2 [1D]
##   Date        Temp
##   <date>     <dbl>
## 1 1874-01-01   3.2
## 2 1874-01-02   4.1
## 3 1874-01-03   2.5
## 4 1874-01-04   0.3
## 5 1874-01-05  -3.9
## 6 1874-01-06   1.9
```

Part B: Data preparation for a single station

- Identify gaps in the time series.

```
has_gaps(dc)
```

```
## # A tibble: 1 x 1
##   .gaps
##   <lgl>
## 1 TRUE
```

```
gaps <- count_gaps(dc)
head(gaps)
```

```
## # A tibble: 6 x 3
##   .from      .to           .n
##   <date>     <date>     <int>
## 1 1874-08-01 1874-08-31    31
## 2 1876-01-01 1877-07-31   578
## 3 1879-01-01 1879-01-31    31
## 4 1880-10-31 1880-10-31     1
## 5 1881-05-01 1881-05-31    31
## 6 1900-12-31 1900-12-31     1
```

- Assume that gaps up to 31 days are acceptable. Find the earliest date in the time series such that all following data have no gaps longer than 31 days. Limit the time series to this.

```
cutoff_date <-  as.character(tail(gaps[gaps$.n >= 31,], n=1)$.to)
cutoff_date
```

```
## [1] "1988-06-17"
```

identify that all dates after, even if 31 is included, is 1988-06-17

```
dc_cut <- dc %>% tsibble::filter_index(cutoff_date ~ .)
head(dc_cut)
```

```
## # A tsibble: 6 x 2 [1D]
##   Date        Temp
##   <date>     <dbl>
## 1 1988-06-18  18.7
## 2 1988-06-21  19.7
## 3 1988-06-22  18
## 4 1988-06-23  21.2
## 5 1988-06-25  21.3
## 6 1988-06-26  23
```

- Create a regular time series by filling gaps in the tsibble with n/a-s.

```
dc_cut_filled <- tsibble::fill_gaps(dc_cut, .full = TRUE)

head(dc_cut_filled)
```

```
## # A tsibble: 6 x 2 [1D]
##   Date        Temp
##   <date>     <dbl>
## 1 1988-06-18  18.7
## 2 1988-06-19  NA
## 3 1988-06-20  NA
## 4 1988-06-21  19.7
## 5 1988-06-22  18
## 6 1988-06-23  21.2
```

- Impute values for the n/a-s. Justify your choice of imputation method.

Choose to impute using simple rolling average, as you get the adventage of getting the most probable next step as well as being locally probable as opposed to using to mean or median for the whole dataset.

```
dc_cut_filled_imp <- imputeTS::na_ma(dc_cut_filled, k = 4, weighting = "simple")
```

- You should now have a regular time series with only numeric values.

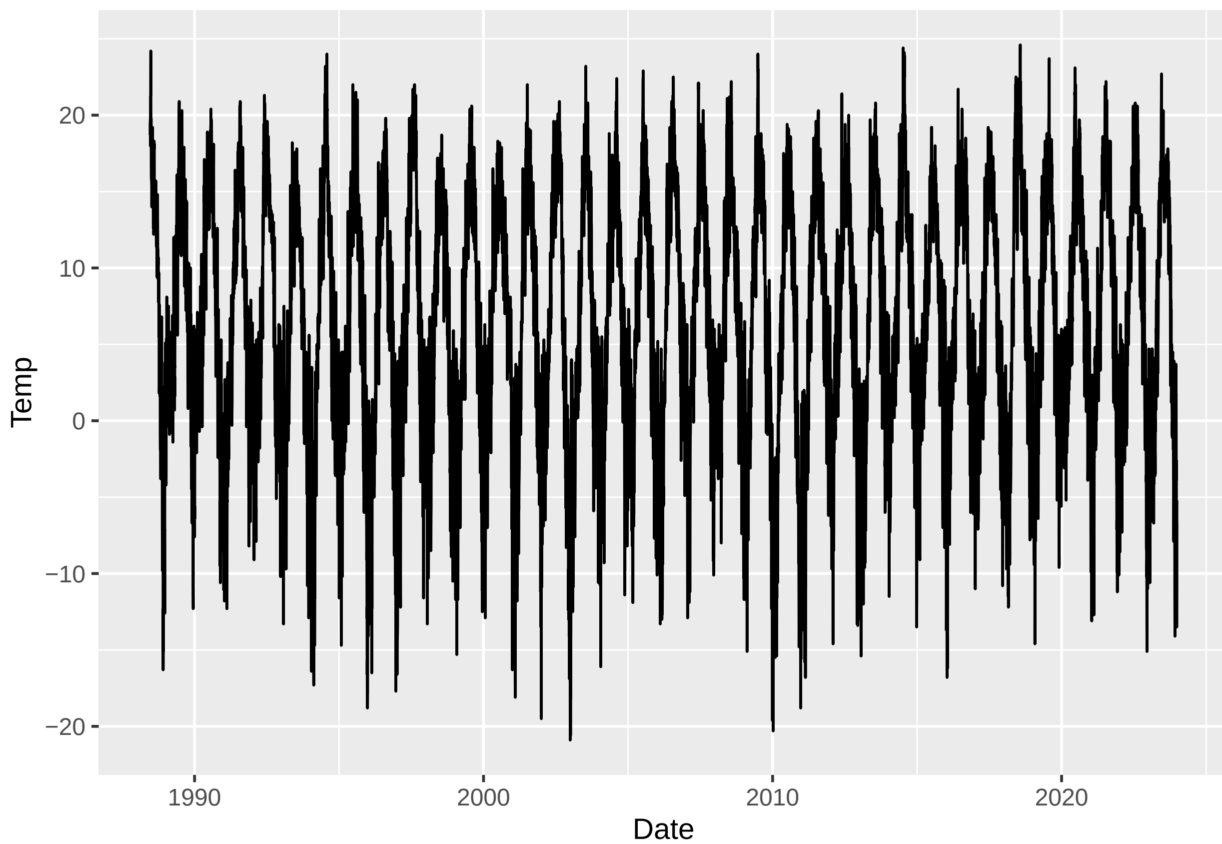
```
tsibble::has_gaps(dc_cut_filled_imp)
```

```
## # A tibble: 1 x 1
##   .gaps
##   <lgl>
## 1 FALSE
```

```
is.numeric(dc_cut_filled_imp$Temp)
```

```
## [1] TRUE
```

```
dc_cut_filled_imp %>%
  ggplot(aes(Date, Temp)) +
  geom_line()
```



- Remove all data for 29 February so all years have data for exactly 365 days.

```
dc_cut_filled_imp_noLeap <- dc_cut_filled_imp %>%
  dplyr::filter(!(month(Date) == 2 & day(Date) == 29))
head(dc_cut_filled_imp_noLeap)
```

```
## # A tsibble: 6 x 2 [1D]
##   Date        Temp
##   <date>     <dbl>
## 1 1988-06-18  18.7
## 2 1988-06-19  19.4
## 3 1988-06-20  19.4
## 4 1988-06-21  19.7
## 5 1988-06-22  18
## 6 1988-06-23  21.2
```

- Combine all this code into a function for re-use later. The function should receive the original tsibble from part A as input and return a new tsibble.

```
timeseries_cleaner <- function(table) {
  gaps <- count_gaps(table)
  cutoff_date <-  as.character(tail(gaps[gaps$.n >= 31,], n=1)$.to)
  dc_cut <- dc %>% tsibble::filter_index(cutoff_date ~ .)
  dc_cut_filled_imp <- imputeTS::na_ma(dc_cut_filled, k = 4, weighting = "simple")
  dc_cut_filled_imp_noLeap <- dc_cut_filled_imp %>%
    dplyr::filter(!(month(Date) == 2 & day(Date) == 29))

  return(dc_cut_filled_imp_noLeap)
```

```
}
```

```
identical(timeseries_cleaner(dc), dc_cut_filled_imp_noLeap)
```

```
## [1] TRUE
```

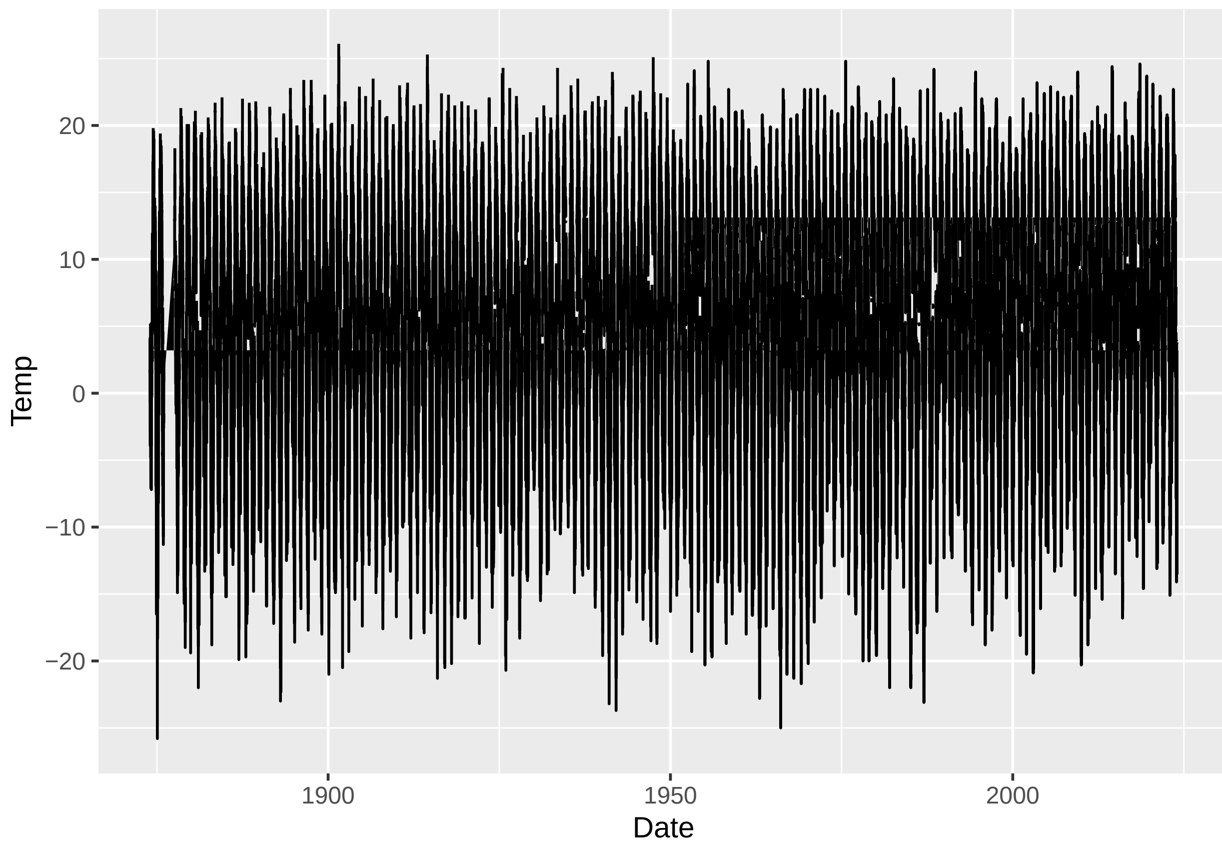
Hints

- tidyverse provides functions such as has_gaps() and count_gaps()

Part C: Exploratory analysis for a single station

- Plot the temperature data as function of time

```
dc %>%
  ggplot(aes(Date, Temp)) +
  geom_line()
```

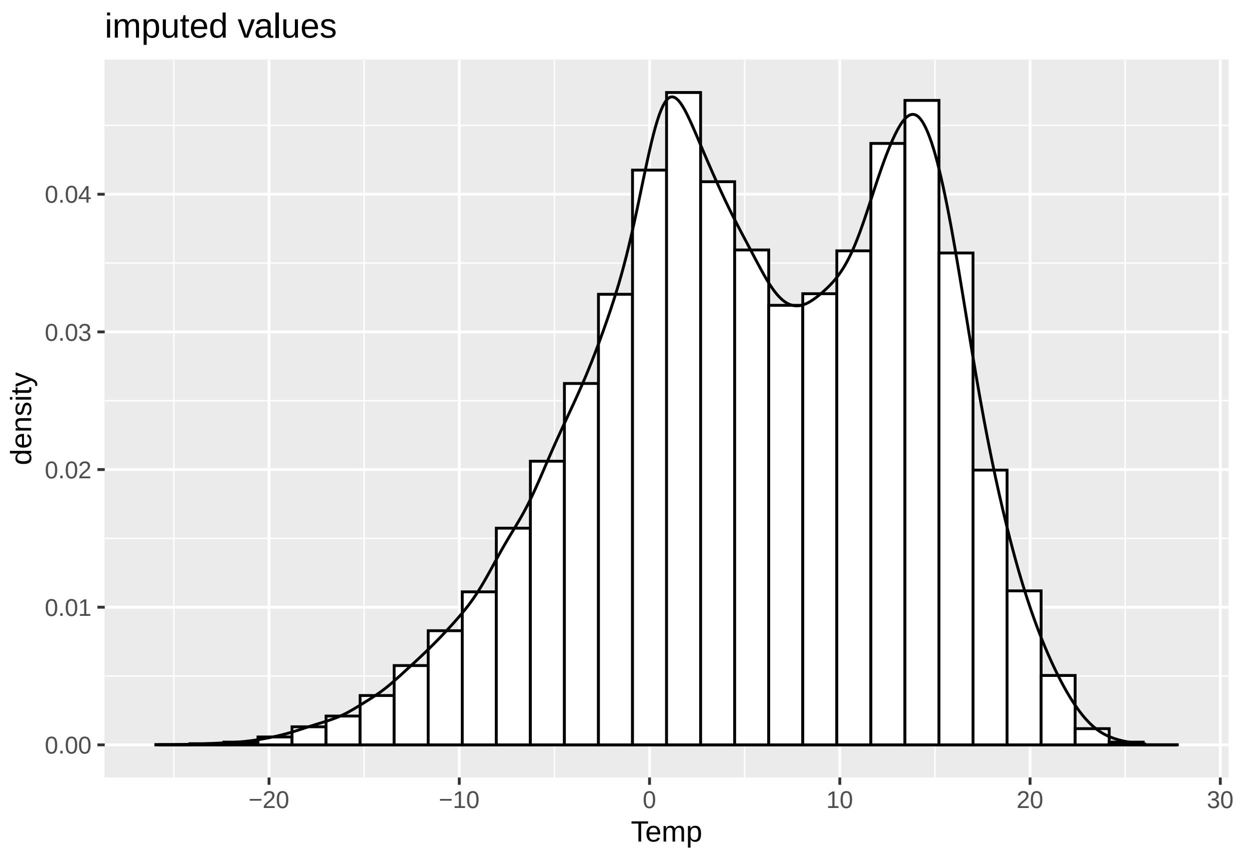


- Create density plots of original data and data with imputed values

```
tsibble::fill_gaps(dc, .full = TRUE) %>%
  imputeTS::na_ma(airqual_c, k = 4, weighting = "simple") %>%
  ggplot(aes(Temp)) +
  geom_histogram(aes(y = ..density..), fill = "white", color="black") +
  stat_density(kernel = "gaussian", fill = NA, colour = "black") +
  ggtitle("imputed values")
```

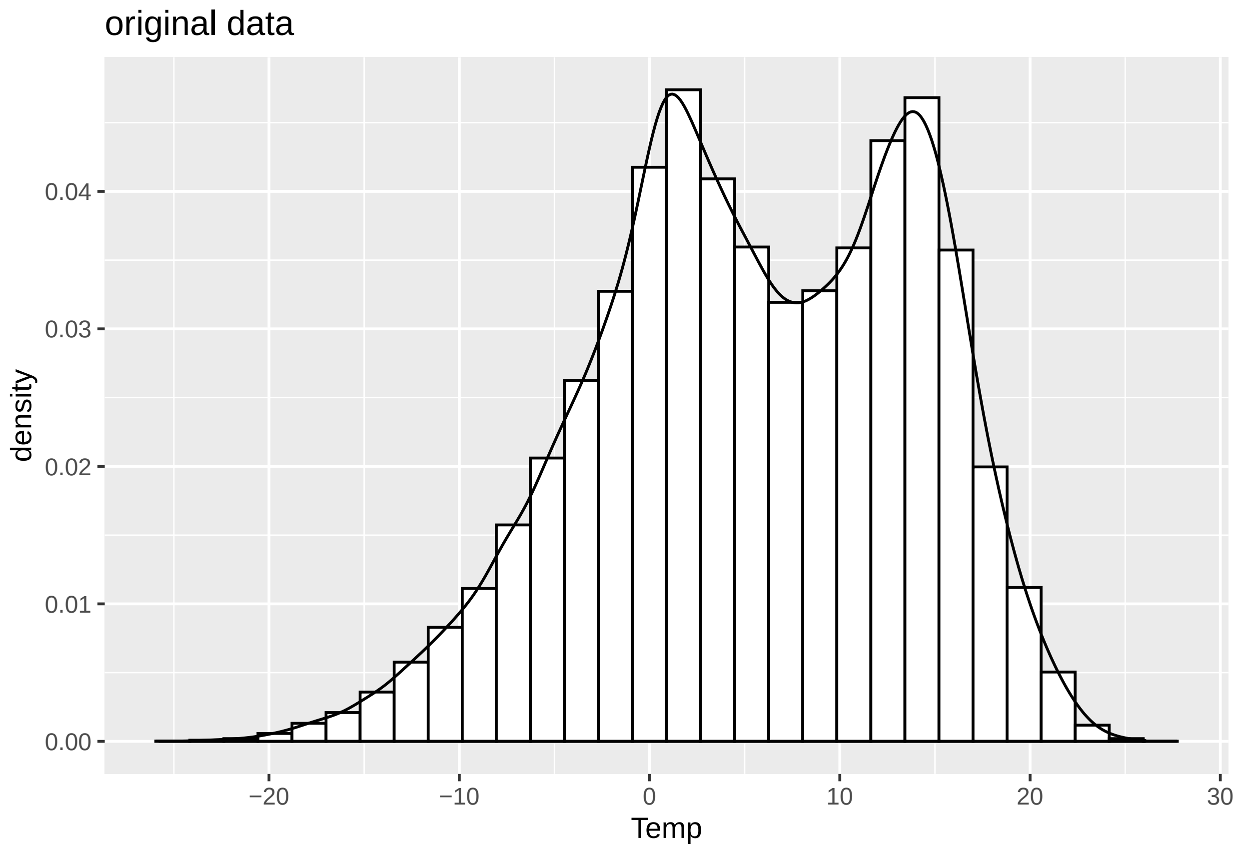
```
## Warning: na_ma: No imputation performed for column 2 of the input dataset.
```

```
##                   Reason: default method not implemented for type 'list'

## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 1147 rows containing non-finite outside the scale range
## (`stat_bin()`).

## Warning: Removed 1147 rows containing non-finite outside the scale range
## (`stat_density()`).
```




```
dc %>%
  ggplot(aes(Temp)) +
  geom_histogram(aes(y = ..density..), fill = "white", color="black") +
  stat_density(kernel = "gaussian", fill = NA, colour = "black") +
  ggtitle("original data")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Temprature seems to be bimodal

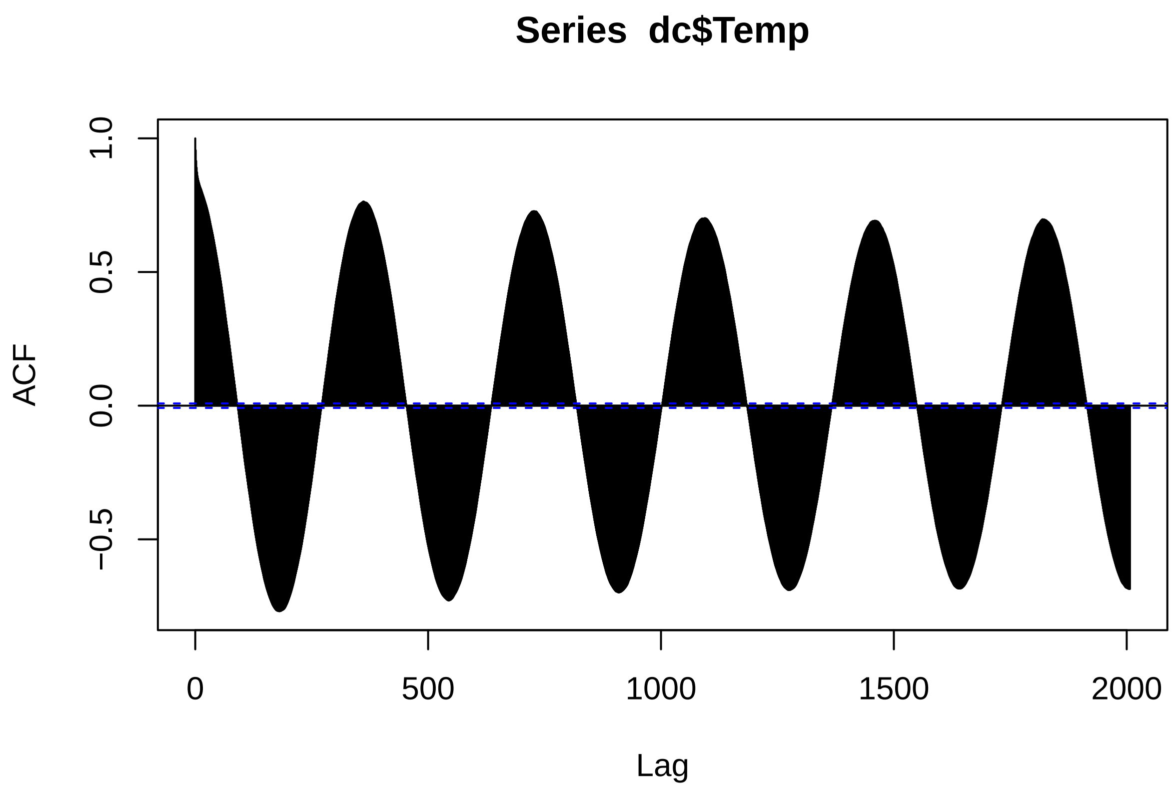
- Turn the temperature data into a timeseries (ts) object

```
ts_dc <- ts(dc) #tsibble::fill_gaps(dc, .full = TRUE)
head(ts_dc)
```

```
## Time Series:
## Start = 1
## End = 6
## Frequency = 1
##      Date Temp
## 1 -35063  3.2
## 2 -35062  4.1
## 3 -35061  2.5
## 4 -35060  0.3
## 5 -35059 -3.9
## 6 -35058  1.9
```

- Plot the autocorrelation function for lags up to 5.5 years; describe and discuss your observations

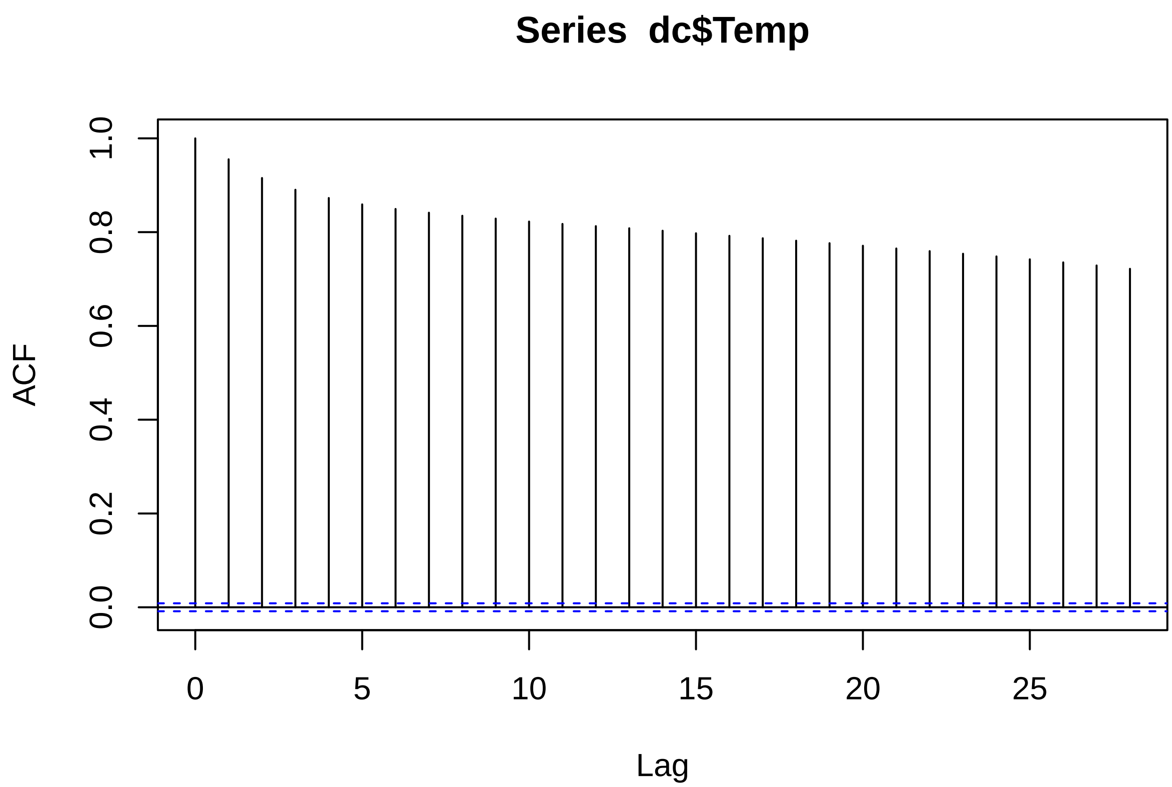
```
acf(dc$Temp, lag.max = 365*5.5)
```



correlation between a time series and its own lagged version Clear pattern in regression error, can be fixed. Breaks the assumption of normally distributed errors.

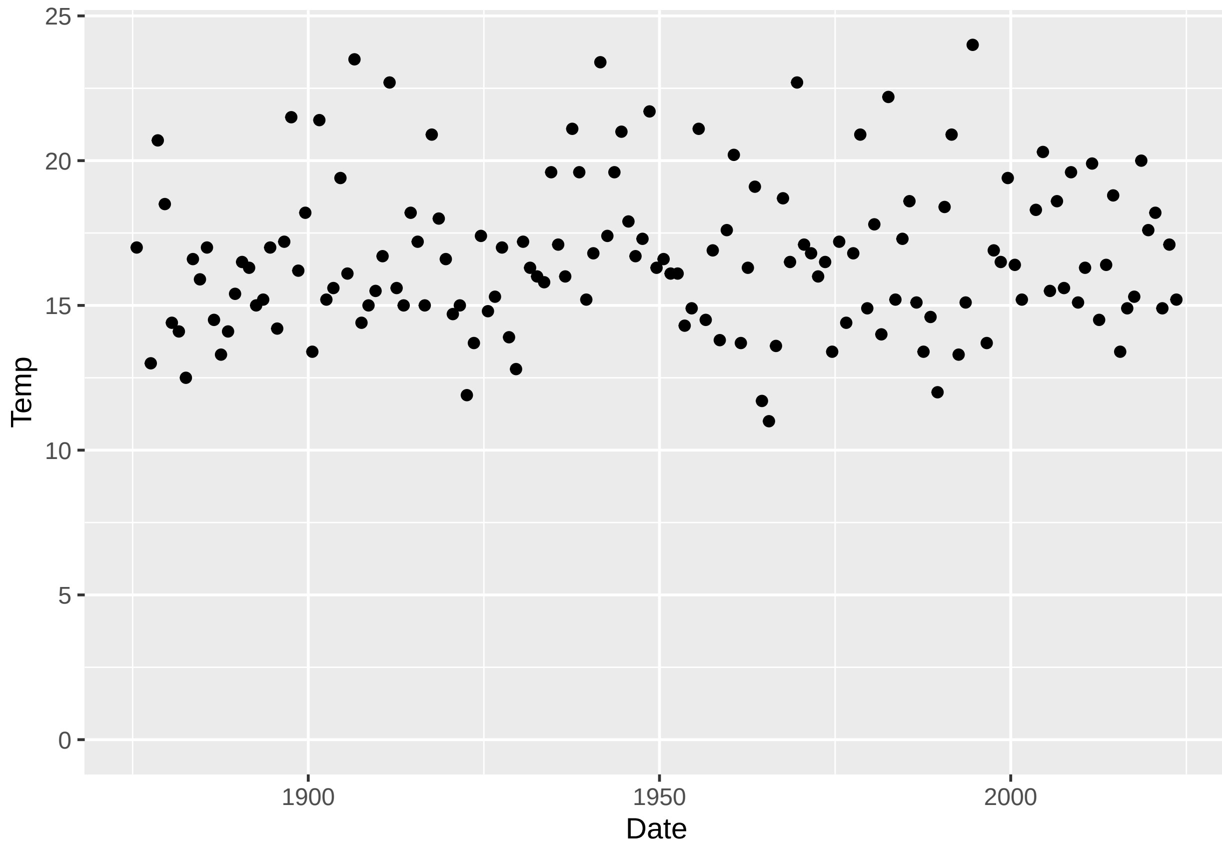
- Also plot the ACF only for short lags, up to four weeks

```
acf(dc$Temp, lag.max = 7*4)
```



- Select some days distributed throughout the year and plot temperature as function of year for, e.g., 1 October, as a scatter plot. This plot can be useful to choose the seasonality window later (see Figs 7 and 8 in Cleveland et al, 1990)

```
dc %>%
  dplyr::filter((month(Date) == 8 & day(Date) == 1)) %>%
  ggplot(aes(Date, Temp)) +
  geom_point() +
  ylim(0, NA)
```



Part D: STL analysis

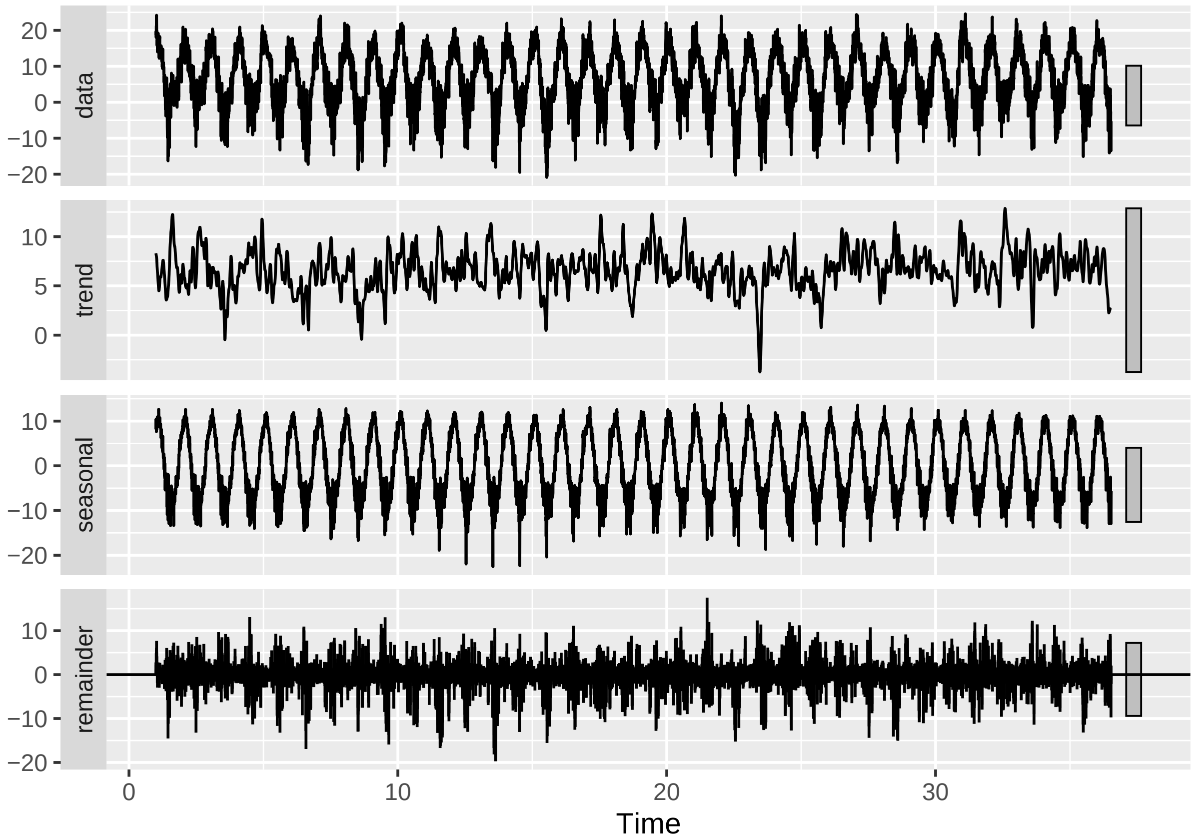
Sesonal Trend Residual, has inner and outer loop from loess using smoothing inner loop: updates tred and seasonality, use hyperparams to sepperate trend and seasonality, trend superposition seasonailty increase in curve outer loop: compute robustness weights, assesing the residuals. Loess : local regression, idea: for every data point, fit a polynominal to make a linear fit, qudaratic fit Iteratively improve fit assigning reducd weight to outliers. Use loess to estimate sesonal and tred components, and residuals

- Perform STL on the data. Explore different values for the seasonality and trend windows (remember that we want to look at trends over many years!), the choice between robust STL or not, and possibly the lowpass filter window. Describe your observations. It might be interesting to look at the ACF of the remainder in the STL result.

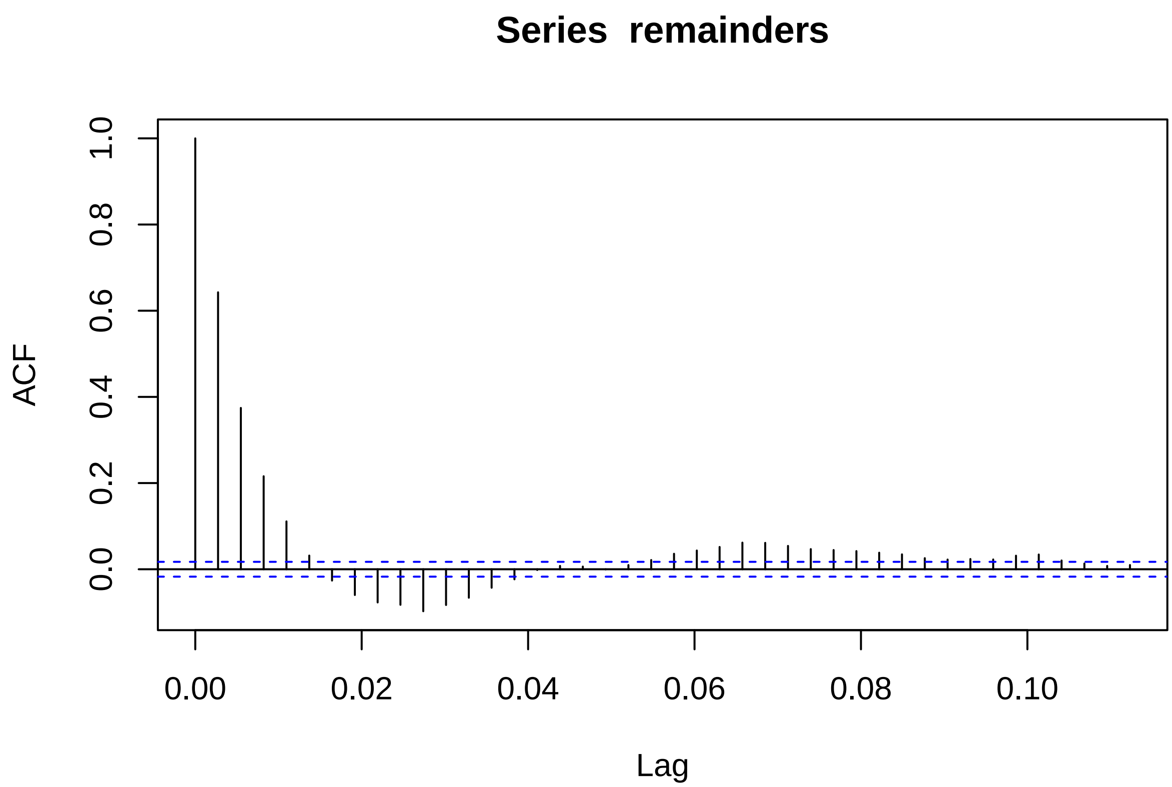
If the autocorrelations are small and close to zero, it indicates that the remainders are uncorrelated (resemble white noice)

```
# Trend over years
dc_c_ts <- ts(dc_cut_filled_imp_noLeap$Temp, frequency = 365) #365 Leap years removed, no need for 365...
```

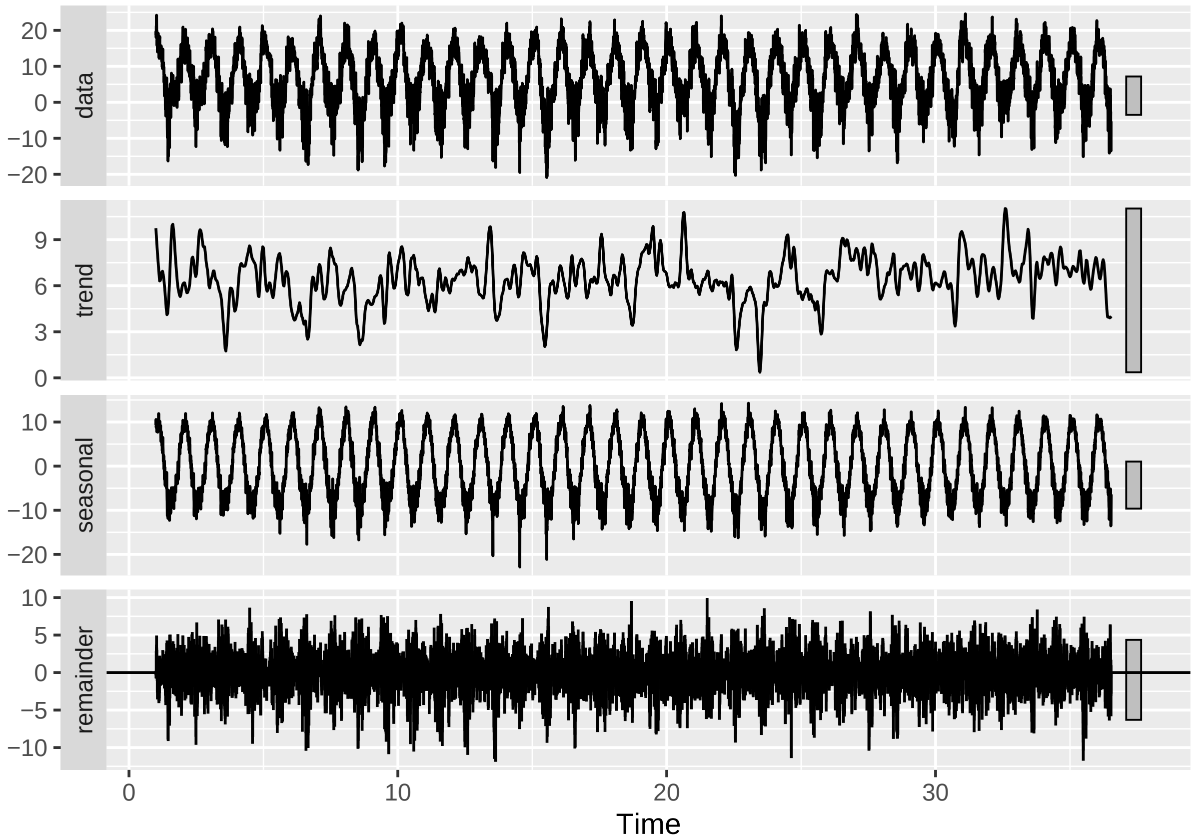
```
dc_stl <- stl(dc_c_ts, s.window=7, t.window=35, robust=TRUE)
autoplot(dc_stl)
```



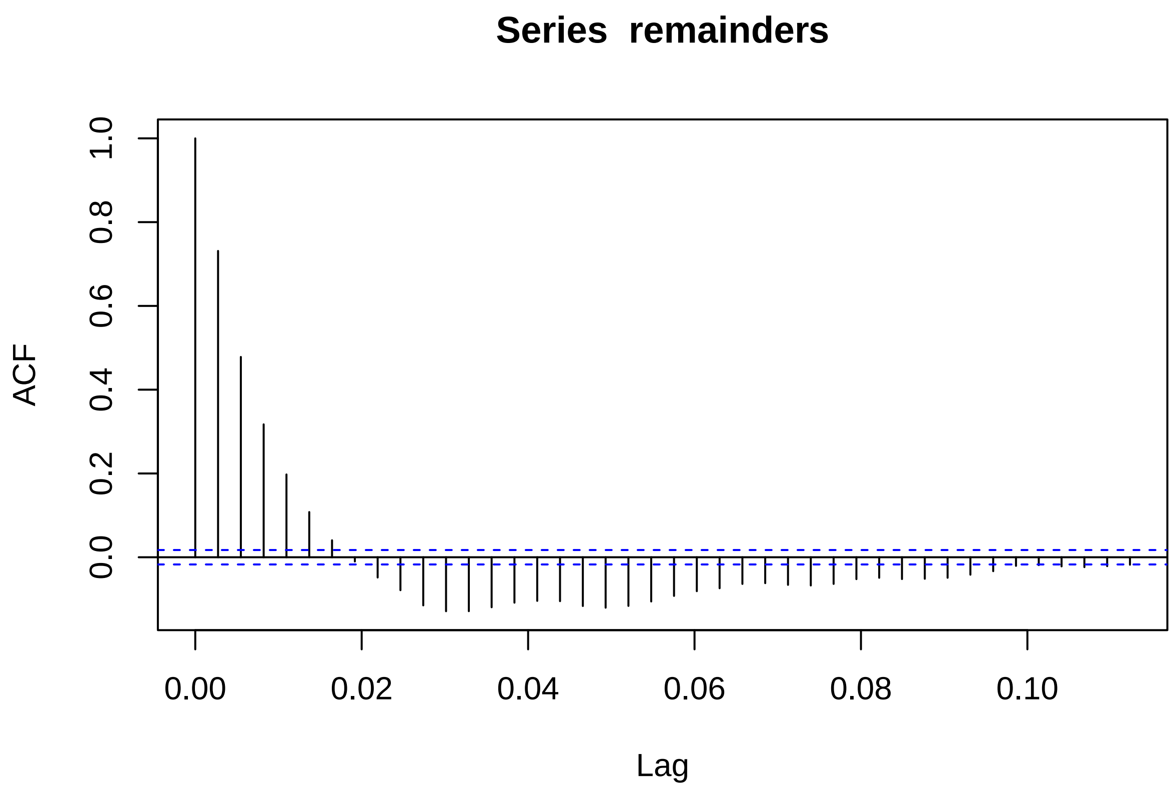

```r
# autocorrelation on remainders
remainders <- dc_stl$time.series[, "remainder"]
acf(remainders)
```




```
dc_stl <- stl(dc_c_ts, s.window=7, t.window=100, robust=FALSE)
autoplot(dc_stl)
```



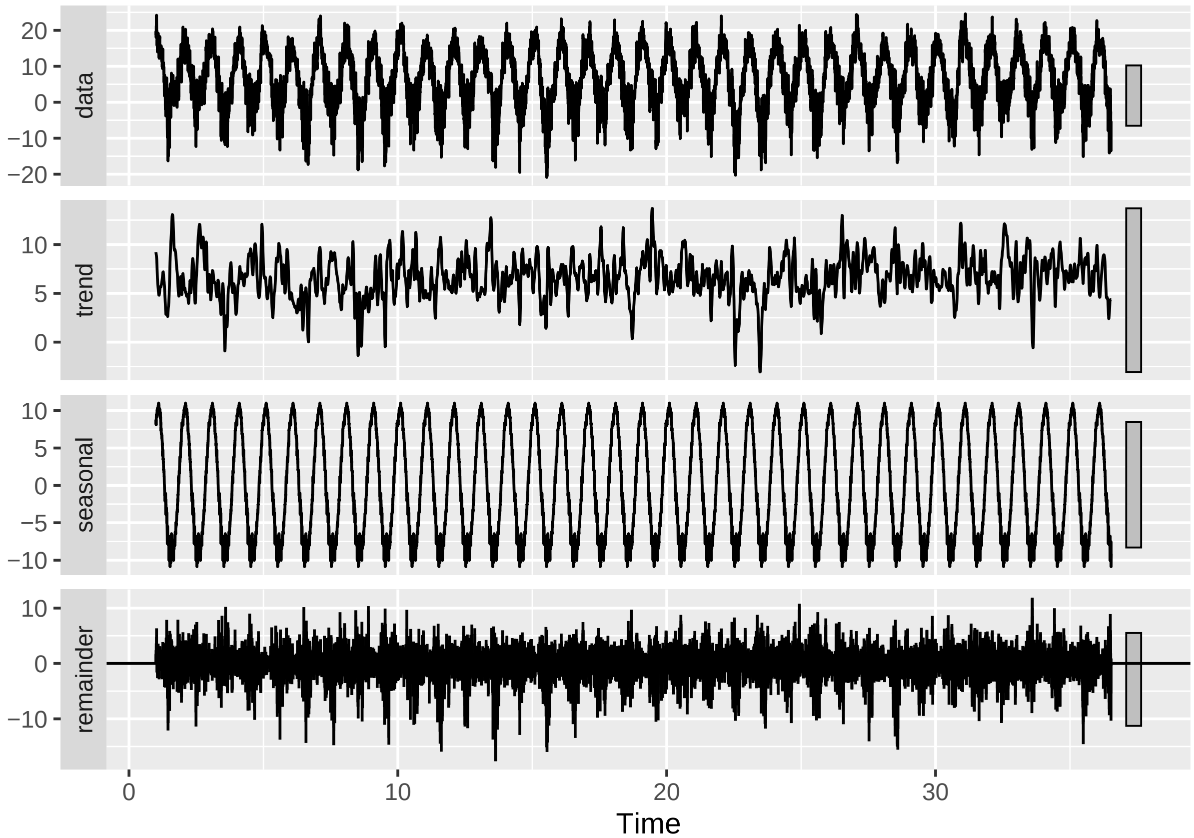

```r
# autocorrelation on remainders
remainders <- dc_stl$time.series[, "remainder"]
acf(remainders)
```



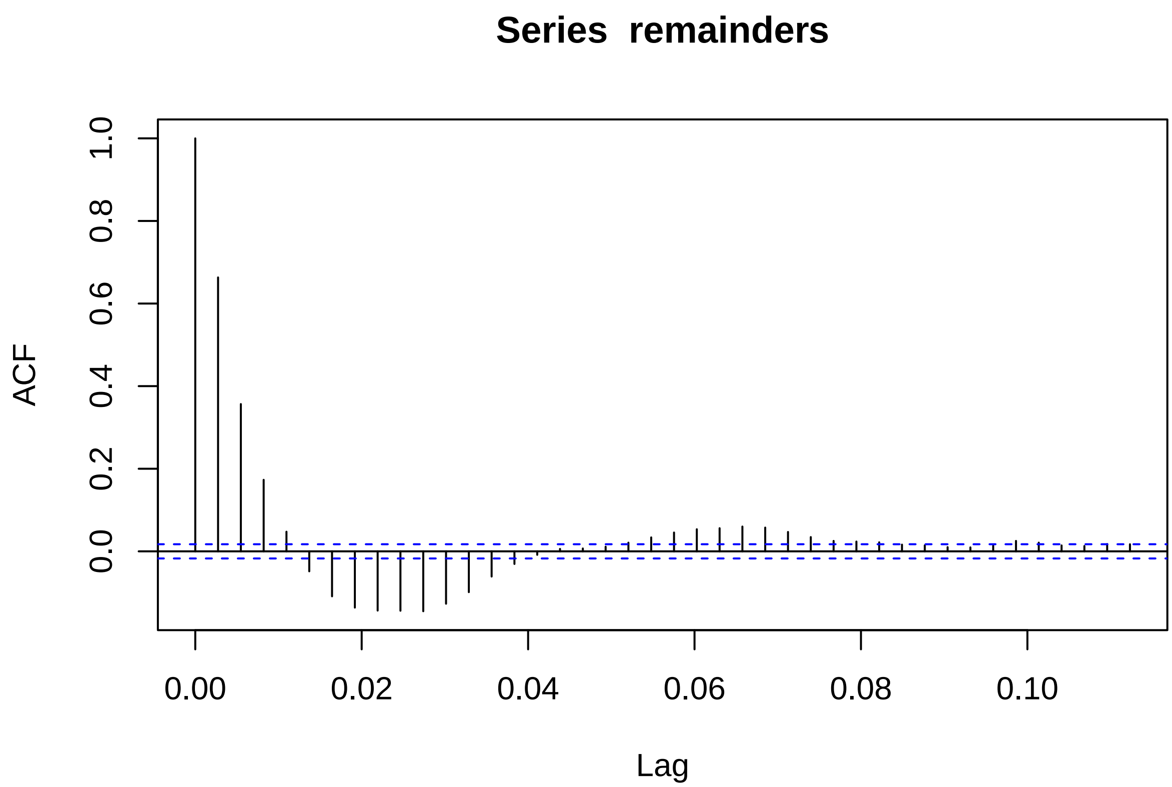
Having robust off seems to make greater remainders than without.

Wide sesonality window

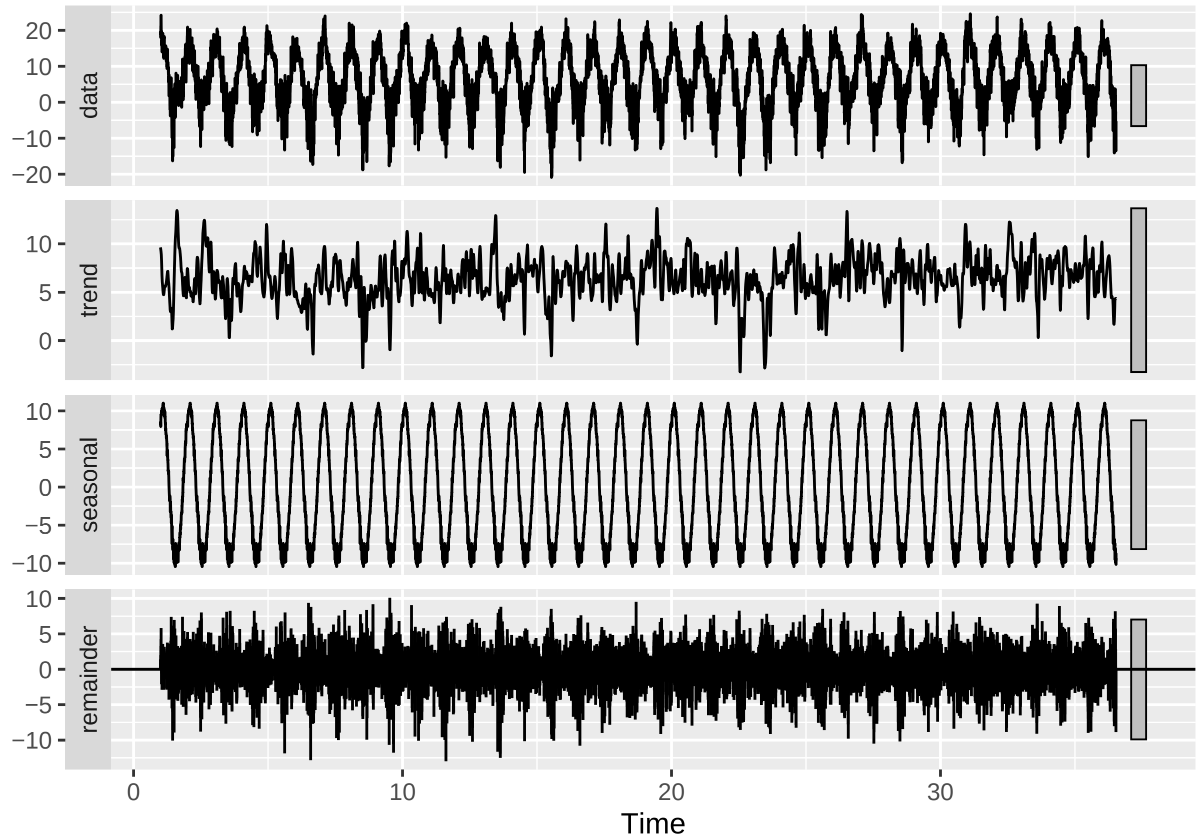
```
# STL wide seasonality window
dc_stl <- stl(dc_c_ts, s.window="periodic", t.window=35, robust=TRUE)
autoplot(dc_stl)
```



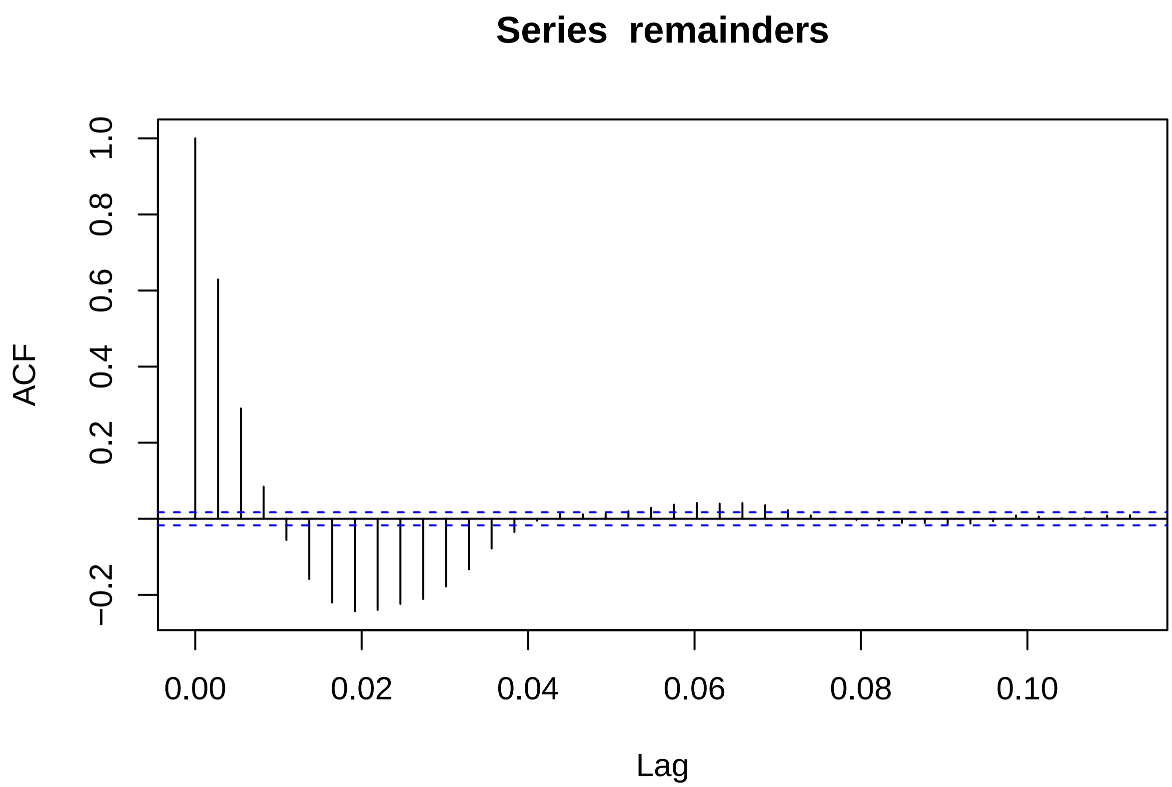

```r
# autocorrelation on remainders
remainders <- dc_stl$time.series[, "remainder"]
acf(remainders)
```




```r
# STL wide seasonality window
dc_stl <- stl(dc_c_ts, s.window="periodic", t.window=35, robust=FALSE)
autoplot(dc_stl)
```

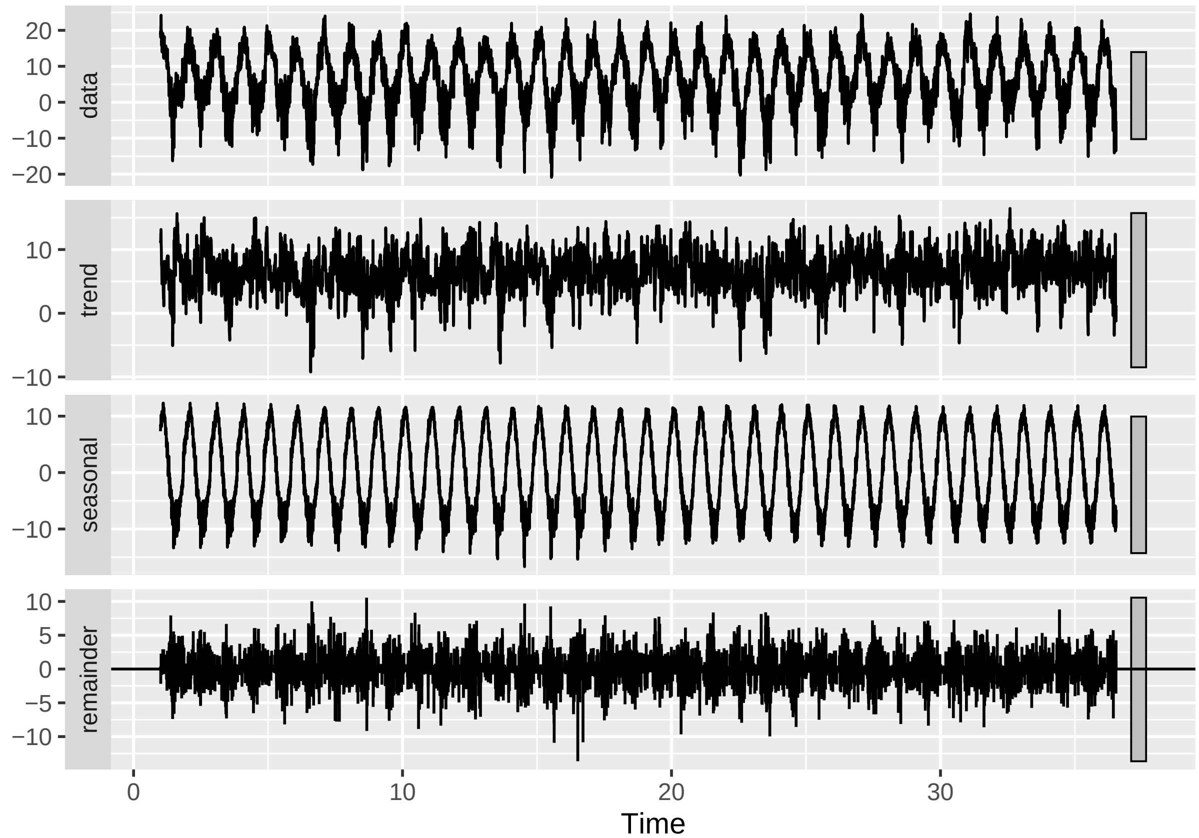



```
# autocorrelation on remainders
remainders <- dc_stl$time.series[, "remainder"]
acf(remainders)
```

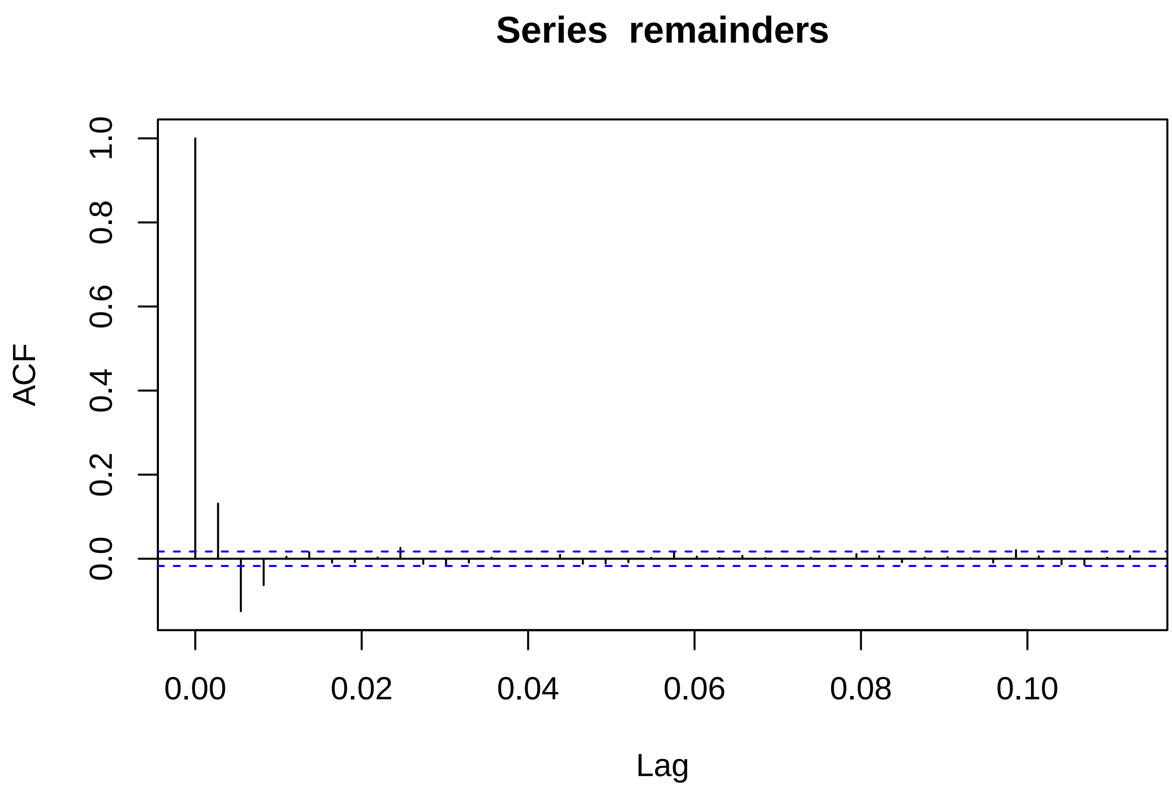


STL short seasonality window

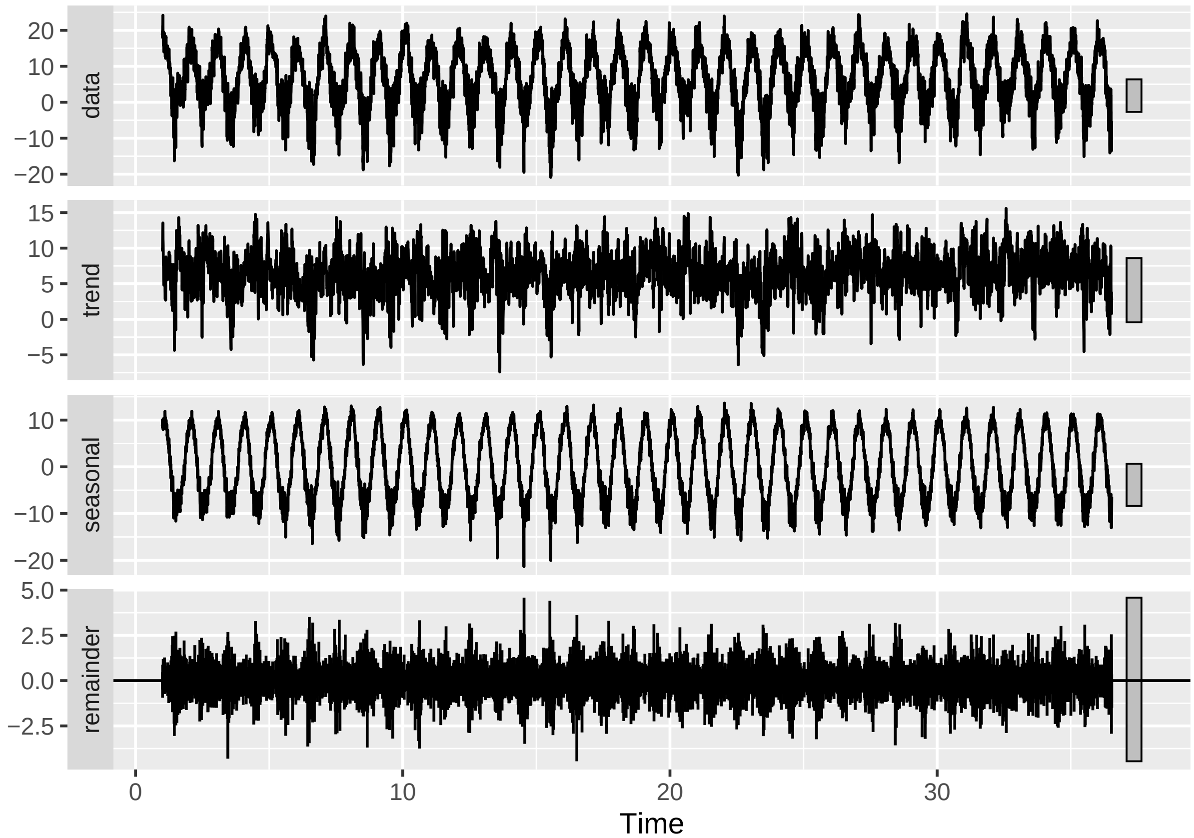
```
dc_stl <- stl(dc_c_ts, s.window=7, t.window=5, robust=TRUE)
autoplot(dc_stl)
```



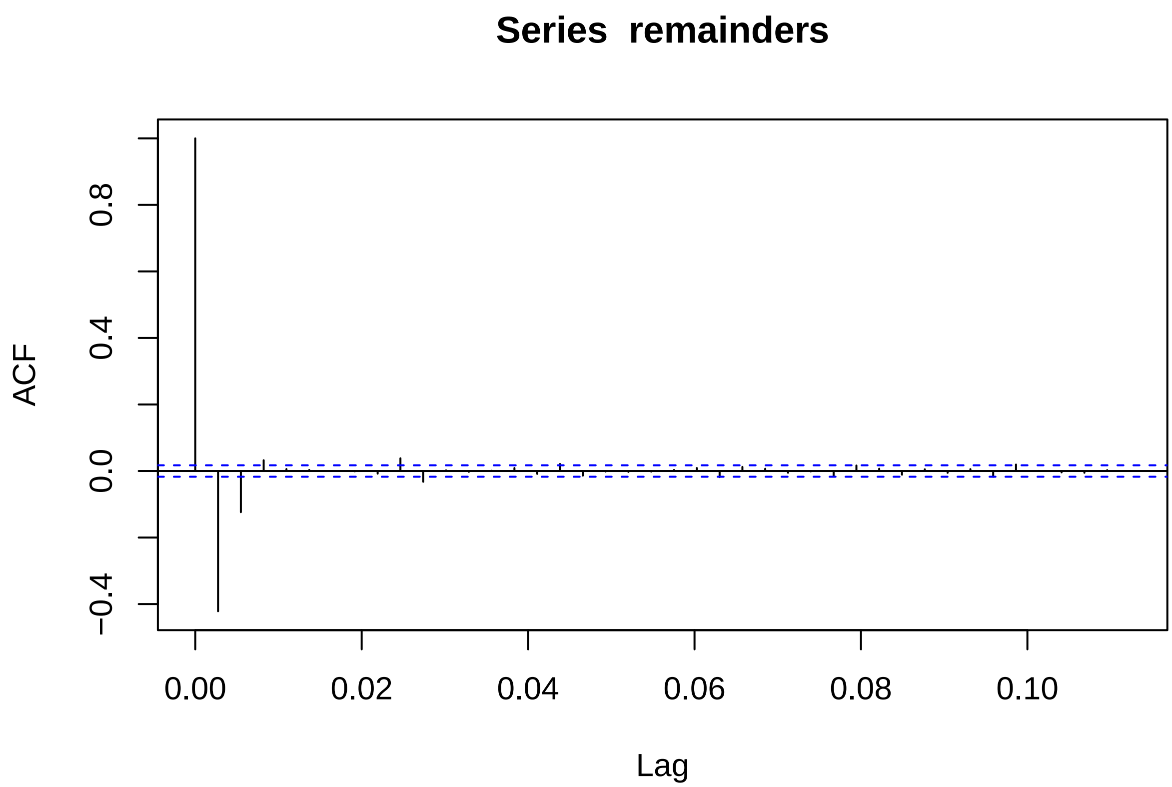

```r
# autocorrelation on remainders
remainders <- dc_stl$time.series[, "remainder"]
acf(remainders)
```




```r
dc_stl <- stl(dc_c_ts, s.window=7, t.window=5, robust=FALSE)
autoplot(dc_stl)
```




```r
# autocorrelation on remainders
remainders <- dc_stl$time.series[, "remainder"]
acf(remainders)
```



- Consult the original STL paper by Cleveland et al. (1990) for suggestions on how to choose STL parameters.

`s.window = "periodic"` means take all the data into account and wrap around , will make the seasonal component perioduc, aka amplitude will not scale over time. Do not use if the seasonality changes significantly over time.

If you assume that the seasonal patter is constant through time, you should set `s.window` to a be a big odd number, so that you use your entire data to perform your analysis. If seasonal patterns evolve quickly, you should use a small number but bigger than 7 and odd, therby only using the most recent data such that the analysis is not affected by old seasonal pattern that are not relevant anymore.

$n_{(s)}$: the smoothing parameter for the seasonal component "must be carefully tailored to each application"(Cleveland et al. p.9, (1990)). In the `stl()` function those parameters are `s.window` and `s.degree`.

Assume yearly periodicity with daily data, therefore `n_(p) = frequency = 365`

`s.window` is the seasonal smoothing parameter

- "We will always take $n_{(s)}$ to be odd."(Cleveland et al. p.15, (1990))
- "we also want $n_{(s)}$ to be at least 7."(Cleveland et al. p.15, (1990))
- "The choice of appropriate variation depend critically on the characteristics of the series."

diagnostic method:

- seasonal-diagnostic plot : "helps us decide how much of the variation in the data other than trend should go into the seasonal component and how much in the remainder"(Cleveland et al. p.15, (1990))

"Without robustness, the seasonal component is distorted. This is illustrated by seasonal-diagnostic display"(Cleveland et al. p.15, (1990))

- s.window : Should be odd and at least 7, according to Cleveland et al.
- s.degree : Should be zero or none
- t.window : span in lags of the loess window for trend extraction, should be odd.
-
- Based on your analysis, can you suggest a set of STL parameters to use for further work?

Since the data seems to be seasonally consistent over the years, I would recommend using a big odd number for `s.window` or just `periodic`

Part E: Multiple station analysis

- Obtain data from eight more stations. Two should be in the same part of Norway as the station from part A; then choose three stations each from two other parts of Norway. Data should cover several decades at least, so look for stations with long series.

```
.stations_url = str_glue("https://{.client_id}@frost.met.no/sources/v0.jsonld")
raw_stations <- fromJSON(URLencode(.stations_url), flatten=TRUE)
df_stations <- data.frame(raw_stations)
```

```
# 2 from Ås SN17850

unique(df_stations[,"data.county"])
```

```
##  [1] "ROGALAND"             "INNLANDET"            "VESTLAND"
##  [4] "BUSKERUD"             "TRØNDELAG"            "OSLO"
##  [7] "VESTFOLD"             NA                     "SVALBARD"
## [10] "MØRE OG ROMSDAL"      "NORDLAND"             "AGDER"
## [13] "AKERSHUS"             "NORSKEHAVET"          "TROMS"
## [16] "ØSTFOLD"              "NORDSJØEN"            "TELEMARK"
## [19] "KONTINENTALSOKKELEN" "FINNMARK"             "BARENTSHAVET"
## [22] "JAN MAYEN"
```

```
head(df_stations["data.county" == "AKERHUS", ])
```

```
##  [1] X.context                  X.type
##  [3] apiVersion                 license
##  [5] createdAt                  queryTime
##  [7] currentItemCount           itemsPerPage
##  [9] offset                     totalItemCount
## [11] currentLink                data..type
## [13] data.id                    data.name
## [15] data.shortName             data.country
## [17] data.countryCode           data.masl
## [19] data.validFrom             data.county
## [21] data.countyId              data.municipality
## [23] data.municipalityId        data.ontologyId
## [25] data.stationHolders        data.externalIds
## [27] data.wigosId               data.wmoId
## [29] data.icaoCodes             data.shipCodes
## [31] data.geometry..type        data.geometry.coordinates
## [33] data.geometry.nearest
## <0 rows> (or 0-length row.names)
```

```
COUNTYS = c("AKERSHUS", "OSLO", "FINNMARK", "INNLANDET")   # replace with names of "fylker" you are int

stations <- unnest(raw_stations$data, cols='id') |>
  dplyr::select(id, validFrom, country, county, municipality, name, masl, `@type`) |>
```

```
  mutate(validFrom=as.Date(validFrom)) |>
  filter(`@type` == "SensorSystem" & validFrom <= "1950-01-01" & country == "Norge" & county %in% COUNT
```

```
cut_stations <- rbind(
  stations[stations$county == "AKERSHUS",][1:2,],
  stations[stations$county == "OSLO",][1:3,],
  stations[stations$county == "FINNMARK",][1:3,]
  )
cut_stations
```

```
## # A tibble: 8 x 8
##   id      validFrom  country county   municipality name              masl `@type`
##   <chr>   <date>     <chr>   <chr>    <chr>        <chr>            <int> <chr>
## 1 SN19710 1913-01-01 Norge   AKERSHUS ASKER        ASKER              163 Sensor~
## 2 SN17850 1874-01-01 Norge   AKERSHUS ÅS           ÅS                  92 Sensor~
## 3 SN18700 1931-01-01 Norge   OSLO     OSLO         OSLO - BLINDERN     94 Sensor~
## 4 SN18810 1860-01-01 Norge   OSLO     OSLO         OSLO - BYGDØY ~     18 Sensor~
## 5 SN18950 1927-08-01 Norge   OSLO     OSLO         TRYVANNSHØGDA      514 Sensor~
## 6 SN93700 1868-01-01 Norge   FINNMARK KAUTOKEINO   KAUTOKEINO         307 Sensor~
## 7 SN99370 1940-11-01 Norge   FINNMARK SØR-VARANGER KIRKENES LUFTH~     89 Sensor~
## 8 SN93900 1913-04-01 Norge   FINNMARK KAUTOKEINO   SIHCCAJAVRI        382 Sensor~
```

- Preprocess the data as described in Part B. Find the latest starting date of any series and create a multivariate time series with data from all nine stations starting at this date.

```
get_dc_from_frost <- function(sources, frost = TRUE) {
  # Set this to TRUE to generate a json file
  weather_file = paste(sources, "_weather_data_json.rds.bz2", sep="")
  get_data_from_frost = frost #file.exists(file = weather_file)

  if ( get_data_from_frost ) {
    raw_data <- try(fromJSON(URLencode(.query_url), flatten=TRUE))

    if (class(raw_data) != 'try-error') {
      print("Data retrieved from frost.met.no!")
      write_rds(raw_data, weather_file, compress="bz2", text=TRUE)  # JSON represents data as text
      print(str_glue("Raw data (JSON) written to '{weather_file}'"))
    } else {
      print("Error: the data retrieval was not successful!")
    }
  } else {
    raw_data <- read_rds(weather_file)
    print(str_glue("Raw data (JSON) read from '{weather_file}'"))
  }

  df <- unnest(raw_data$data, cols = c(observations))

  df |> dplyr::select(referenceTime, value) |>
        mutate(referenceTime=as.Date(referenceTime)) |>
        rename(Date=referenceTime, Temp=value) |>
        as_tsibble(index = Date) -> dc

  return(dc)
}
```

```r
cut_stations
```

```
## # A tibble: 8 x 8
##   id      validFrom  country county   municipality name             masl `@type`
##   <chr>   <date>     <chr>   <chr>    <chr>        <chr>           <int> <chr>
## 1 SN19710 1913-01-01 Norge   AKERSHUS ASKER        ASKER             163 Sensor~
## 2 SN17850 1874-01-01 Norge   AKERSHUS ÅS           ÅS                 92 Sensor~
## 3 SN18700 1931-01-01 Norge   OSLO     OSLO         OSLO - BLINDERN    94 Sensor~
## 4 SN18810 1860-01-01 Norge   OSLO     OSLO         OSLO - BYGDØY ~    18 Sensor~
## 5 SN18950 1927-08-01 Norge   OSLO     OSLO         TRYVANNSHØGDA     514 Sensor~
## 6 SN93700 1868-01-01 Norge   FINNMARK KAUTOKEINO   KAUTOKEINO        307 Sensor~
## 7 SN99370 1940-11-01 Norge   FINNMARK SØR-VARANGER KIRKENES LUFTH~    89 Sensor~
## 8 SN93900 1913-04-01 Norge   FINNMARK KAUTOKEINO   SIHCCAJAVRI       382 Sensor~
```

```r
# col_name <- paste("Temp", as.character(cut_stations$name[1], sep="."))
# dc_stations <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[2]))) %>%
#   setNames("Date", col_name)
# for (row in cut_stations[-1,]) {
#   new_row <- timeseries_cleaner(get_dc_from_frost(as.character(row$id))) %>%
#     dplyr::select("Temp") %>%
#     setNames(paste("Temp", as.character(row$name), sep="."))
#   cbind(dc_stations, new_row)
# }
```

```r
dc_stations <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[1])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN19710_weather_data_json.rds.bz2'
```

```r
col2 <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[2])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN17850_weather_data_json.rds.bz2'
```

```r
col3 <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[3])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN18700_weather_data_json.rds.bz2'
```

```r
col4 <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[4])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN18810_weather_data_json.rds.bz2'
```

```r
col5 <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[5])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN18950_weather_data_json.rds.bz2'
```

```r
col6 <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[6])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN93700_weather_data_json.rds.bz2'
```

```r
col7 <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[7])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN99370_weather_data_json.rds.bz2'
```

```r
col8 <- timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[8])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN93900_weather_data_json.rds.bz2'
```

```r
akershus_asker_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[1])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN19710_weather_data_json.rds.bz2'
```

```r
akershus_ås_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[2])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN17850_weather_data_json.rds.bz2'
```

```r
oslo_blindern_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[3])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN18700_weather_data_json.rds.bz2'
```

```r
oslo_bygdøy_ås_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[4])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN18810_weather_data_json.rds.bz2'
```

```r
finnmark_kautokeino_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[5])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN18950_weather_data_json.rds.bz2'
```

```r
finnmark_sørVaranger_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[6])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN93700_weather_data_json.rds.bz2'
```

```r
innlandet_nordOdal_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[5])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN18950_weather_data_json.rds.bz2'
```

```r
innlandet_øysteSlidre_dc_c <-  timeseries_cleaner(get_dc_from_frost(as.character(cut_stations$id[6])))
```

```
## [1] "Data retrieved from frost.met.no!"
## Raw data (JSON) written to 'SN93700_weather_data_json.rds.bz2'
```

- Obtain the cross-correlation matrix between the nine stations. Is there any structure in this 9x9 matrix?
- Perform STL individually on each of the nine stations using the parameters from part D. Compare the resulting trends. Are all STL results of equal quality?

Hints

You can get a list of all available stations from Frost using

```r
#.stations_url = str_glue("https://{.client_id}@frost.met.no/sources/v0.jsonld")
#raw_stations <- fromJSON(URLencode(.stations_url), flatten=TRUE)
```

To limit this to stations with actual data, starting at least as early as 1950, coming from only some parts of Norway relevant columns, and limiting to relevant columns, filter the raw data as

```r
# COUNTYS = c(fylke1, fylke2, etc)   # replace with names of "fylker" you are interested in
#
```

```
# stations <- unnest(raw_stations$data, cols='id') |>
#   select(id, validFrom, country, county, municipality, name, masl, `@type`) |>
#   mutate(validFrom=as.Date(validFrom)) |>
#   filter(`@type` == "SensorSystem" & validFrom <= "1950-01-01" & country == "Norge" & county %in% COU
```

Part F (bonus): PCA

- Perform PCA on the multivariate time series.