timeseries analysis : seasonality etc.

Andreas Hadjiprocopis

February, 2018

```
#!/usr/bin/env Rscript
```

```
source('lib/UTIL.R');
source('lib/DATA.R');
source('lib/IO.R');
source('lib/TS.R');
source('lib/MC.R');
source('lib/SEASON.R');
source('lib/MIXTURES.R');

infile='cleaned_data/dat1.eliminateNA.csv'

dat1 <- data.frame(read_data(
    filename=infile
))
```

```
## read_data(): data read from file 'cleaned_data/dat1.eliminateNA.csv'.
```

```
if( is.null(dat1) ){
    cat("call to read_data() has failed for file '",infile,"'.\n", sep='')
    quit(status=1)
}
dummy <- remove_columns(inp=dat1, colnames_to_remove=c('id'))
dat1_noid <- dummy[['retained']]
dat1_id <- dummy[['removed']]
```

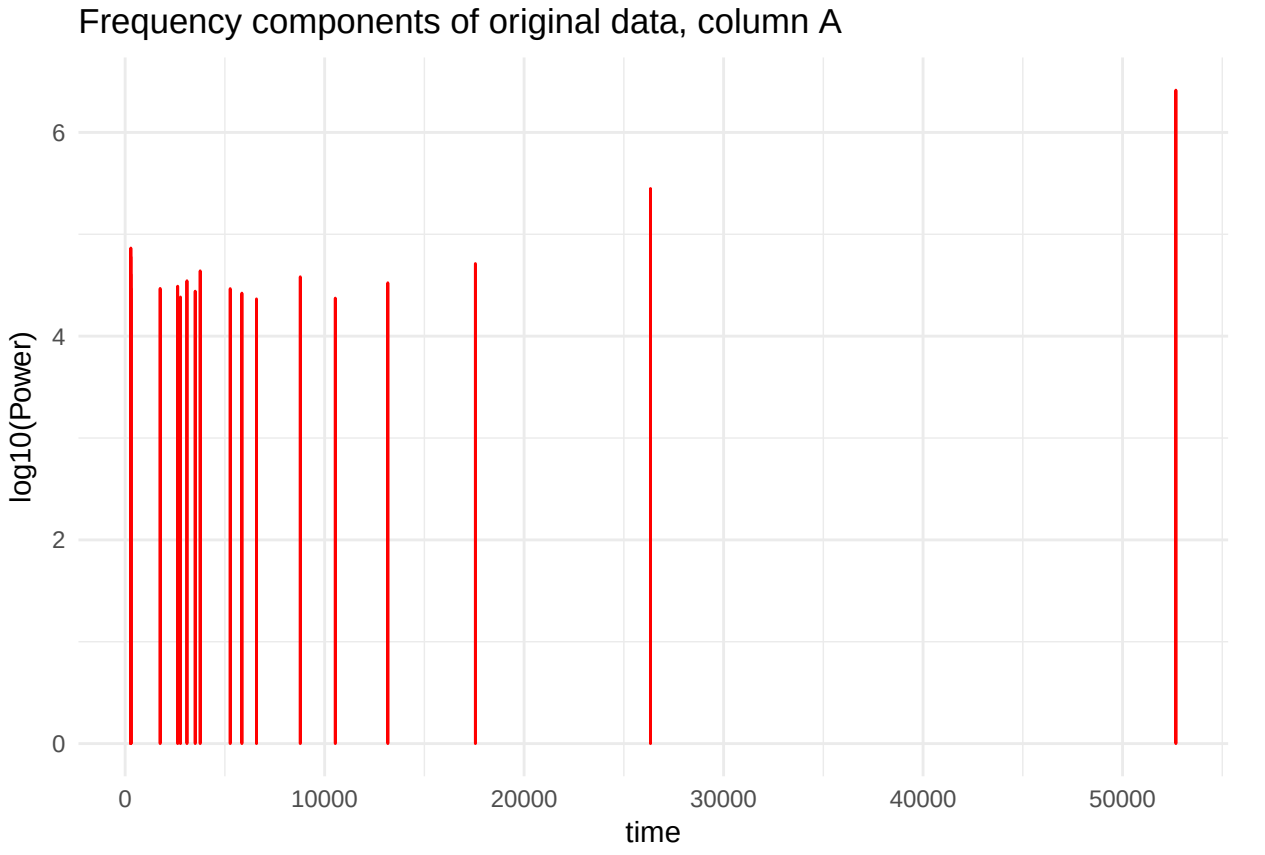
detrend the data

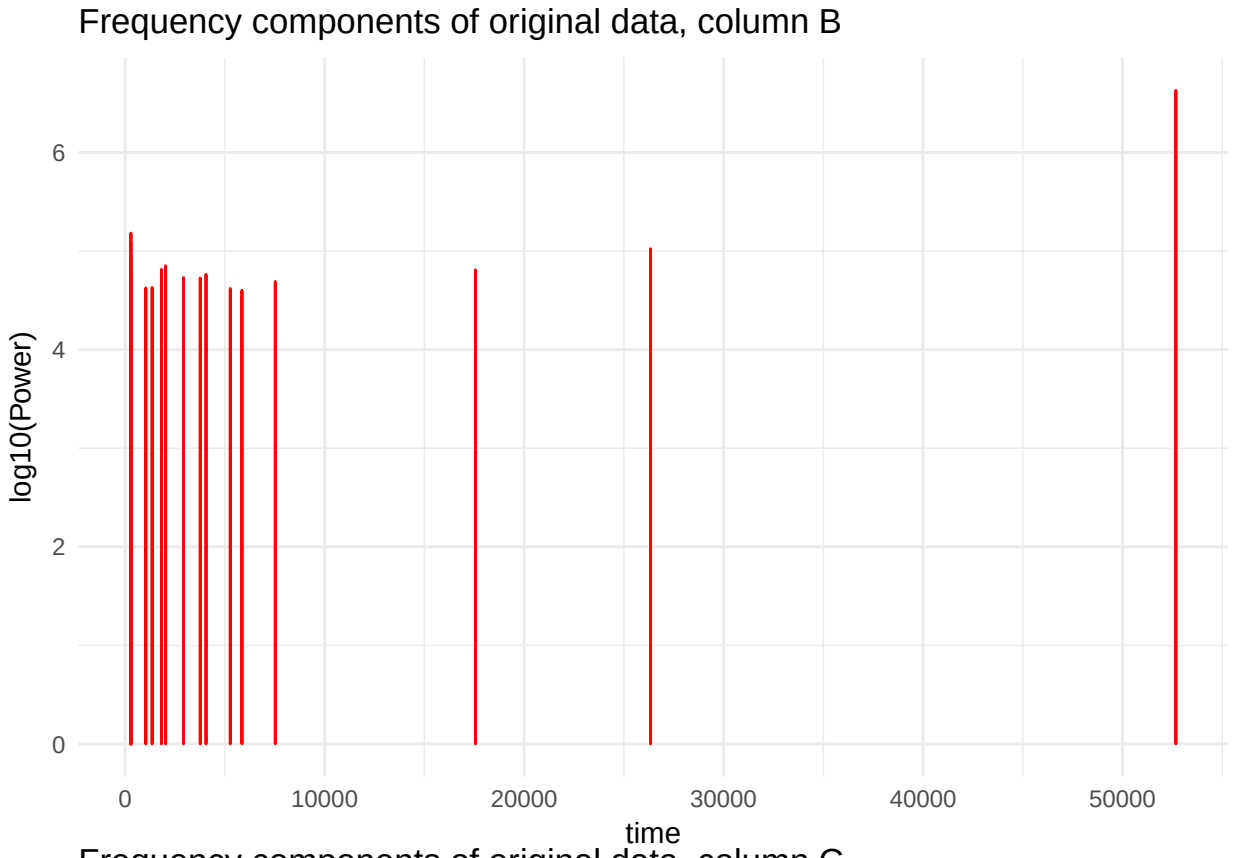
```
dat1_detrended <- detrend_dataset(inp=dat1_noid,times=1)
```

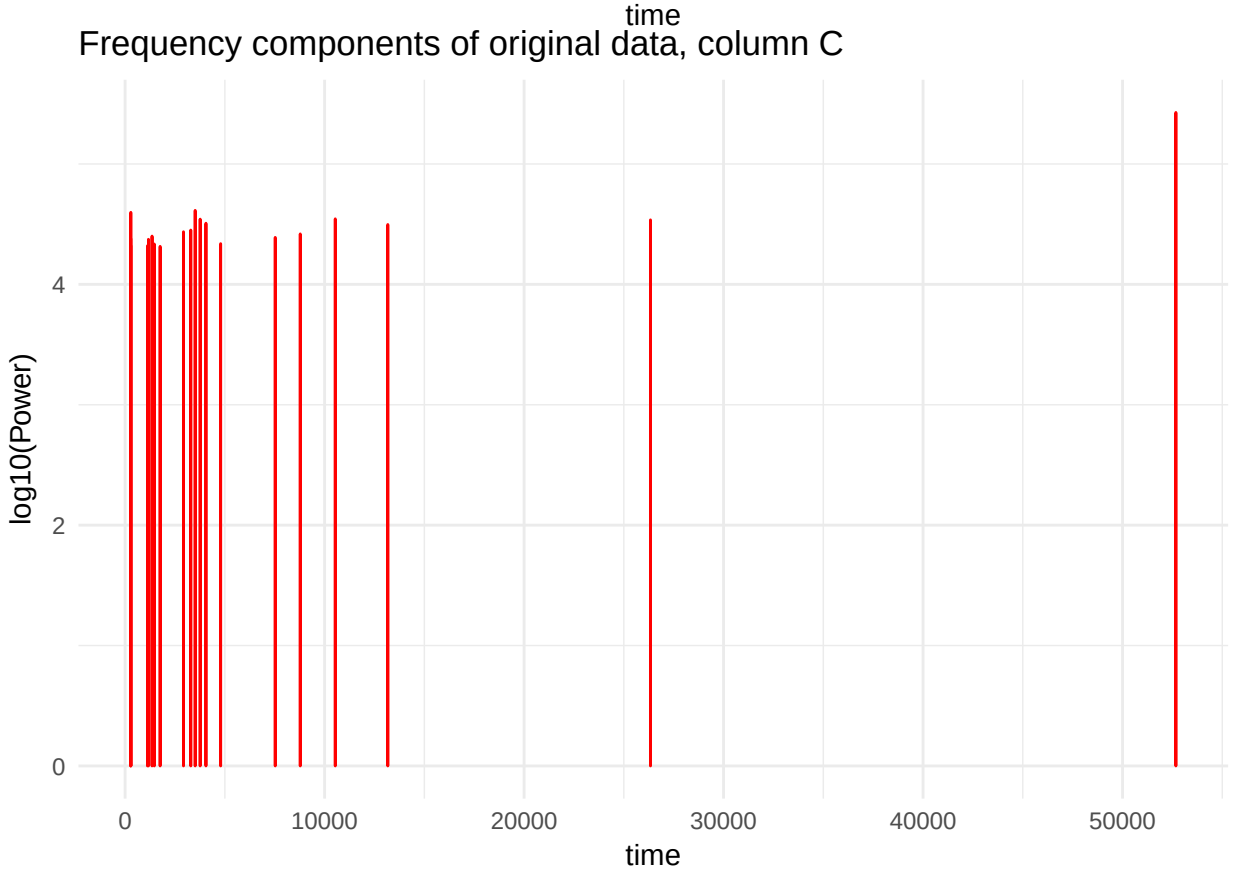
In order to investigate seasonality we will find the significant frequency components of each column of the data using the Fourier Transform.

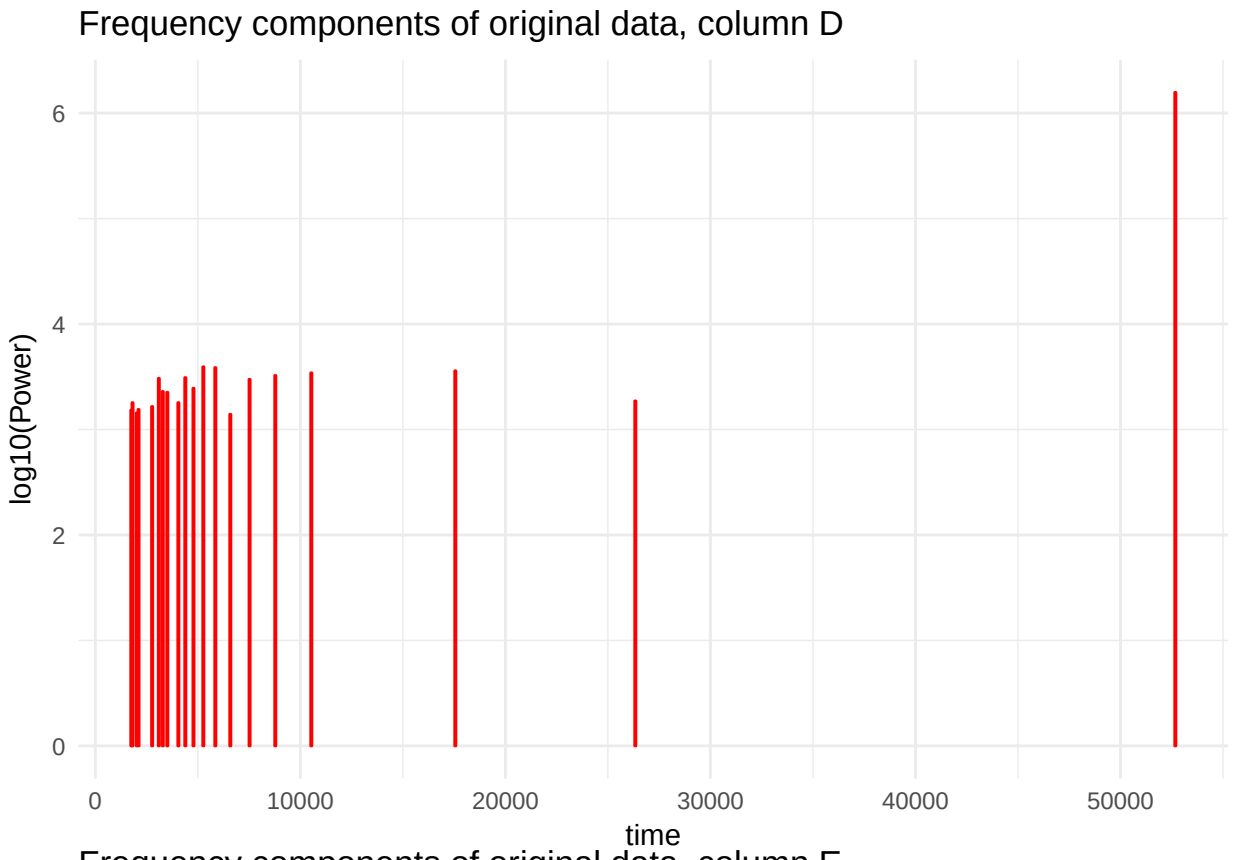
```
dat1_fft <- fourier_transform_analysis(inp=dat1_noid)
dat1_detrended_fft <- fourier_transform_analysis(inp=dat1_detrended)

library(ggplot2)
for(acol in names(dat1_fft)){
    ag <- ggplot(
        as.data.frame(
            tail(dat1_fft[[acol]],20)
        ),
        aes(x=time,y=log10power)
    )+geom_bar(stat="identity", color='red')+
    ggtitle(paste0("Frequency components of original data, column ", acol, sep=''))+
    xlab("time")+
    ylab("power")+
    scale_y_continuous(name='log10(Power)')+
    theme_minimal()
```

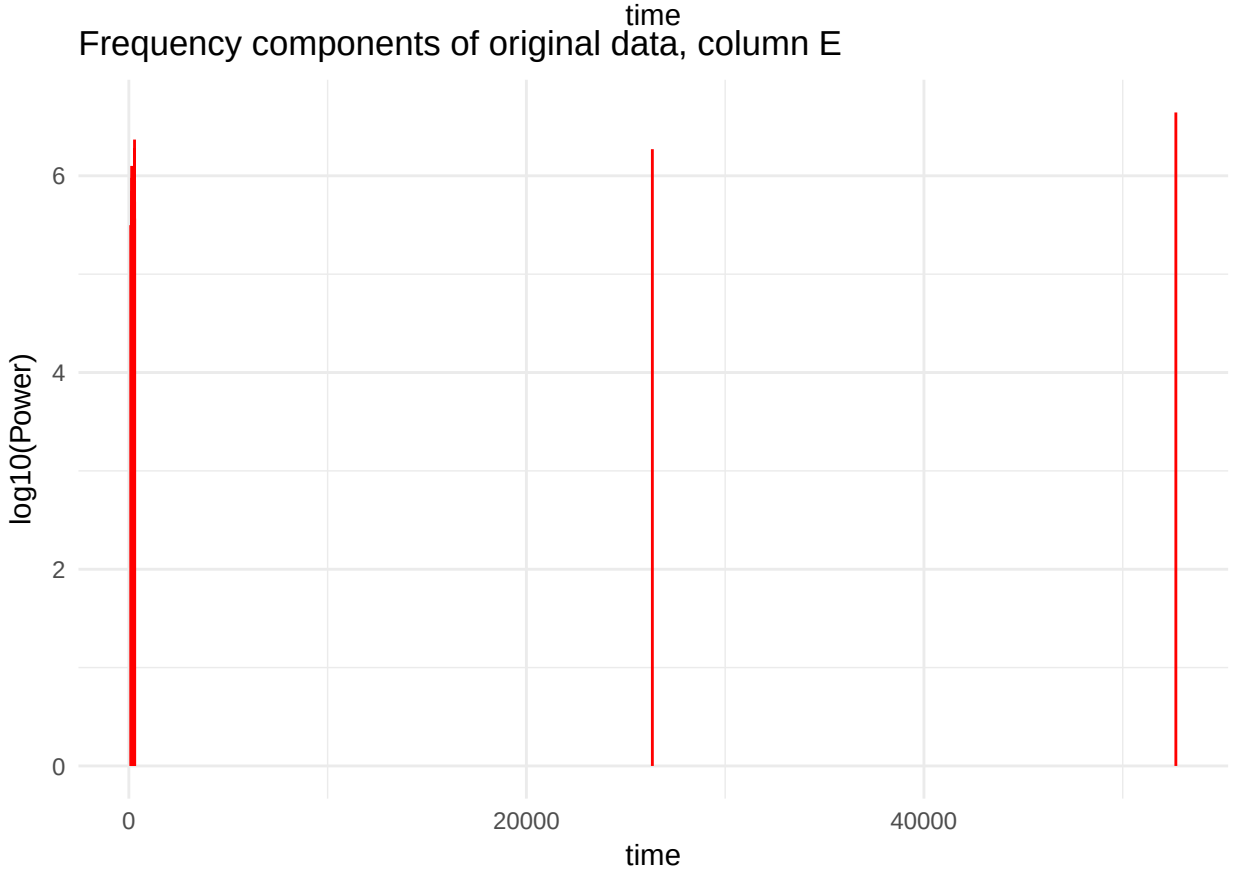
```
    print(ag)
}
```

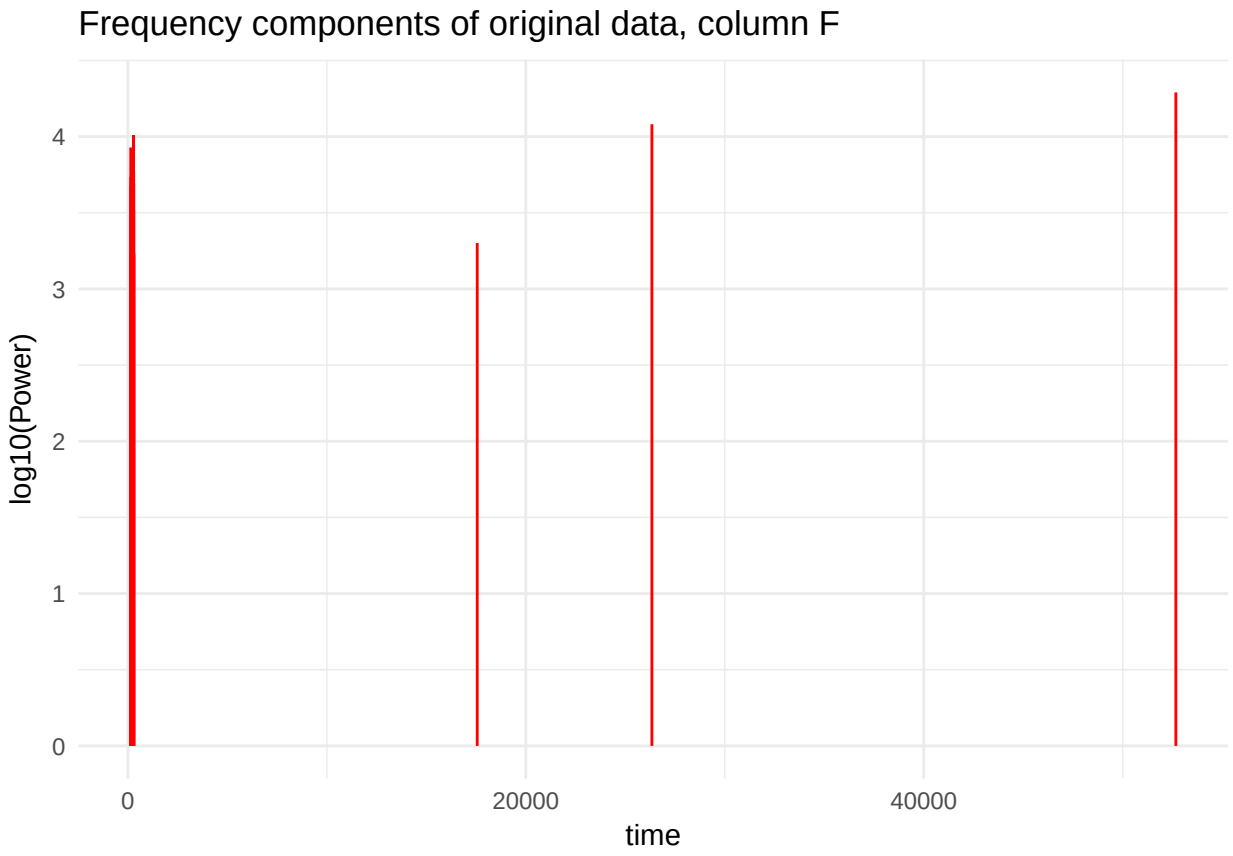

Frequency components of original data, column A
log10(Power)
0
2
4
6
0
10000
20000
30000
40000
50000
time


Frequency components of original data, column B
log10(Power)
0
2
4
6
0
10000
20000
30000
40000
50000
time

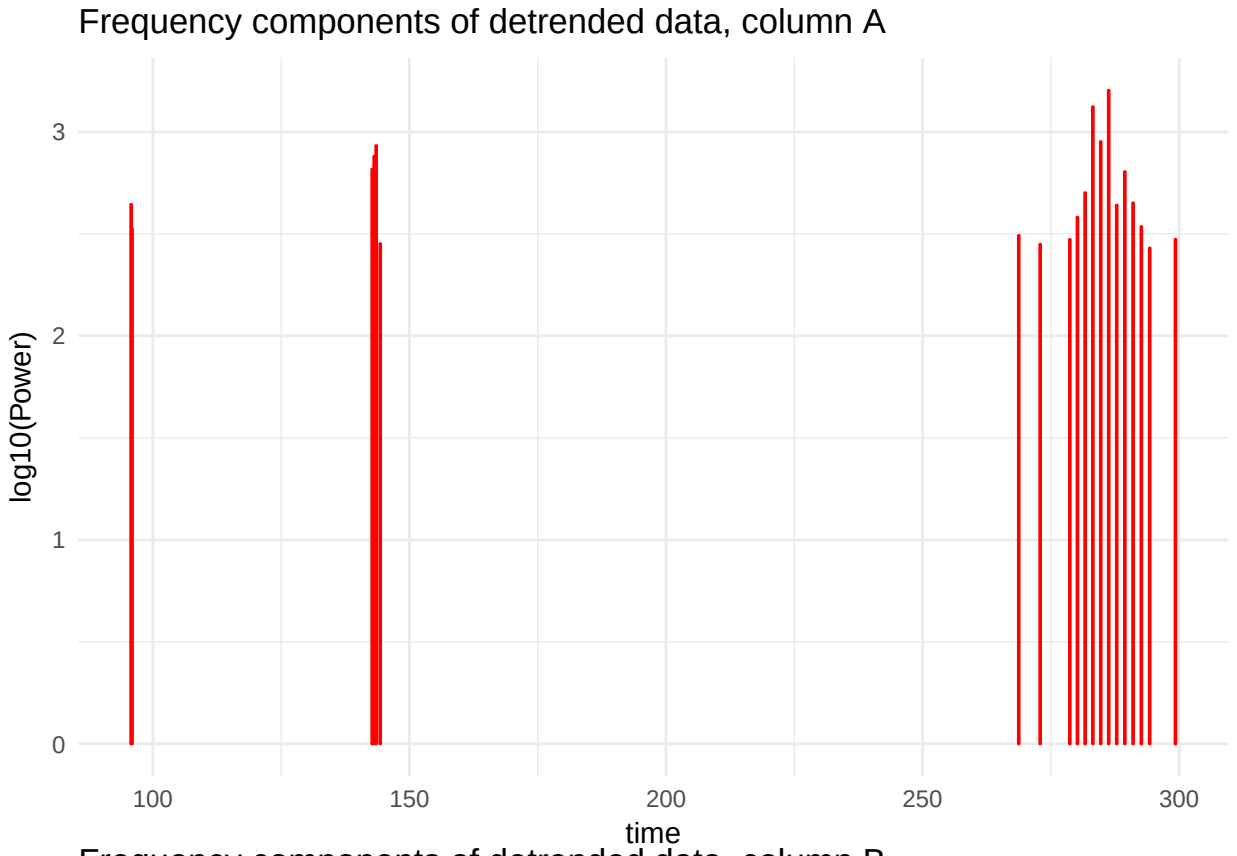

Frequency components of original data, column C
log10(Power)
0
2
4
0
10000
20000
30000
40000
50000
time

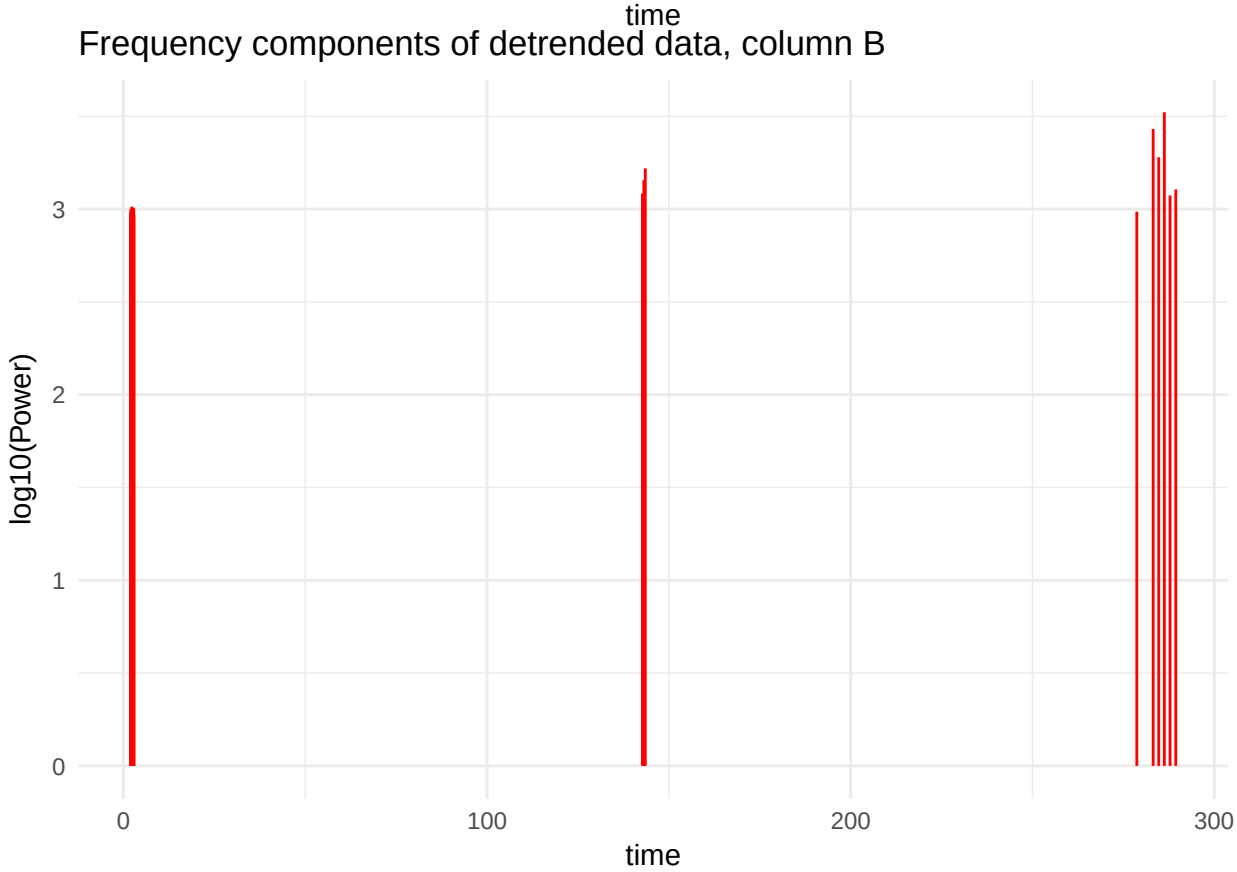

Frequency components of original data, column D
log10(Power)
0
2
4
6
0
10000
20000
30000
40000
50000
time

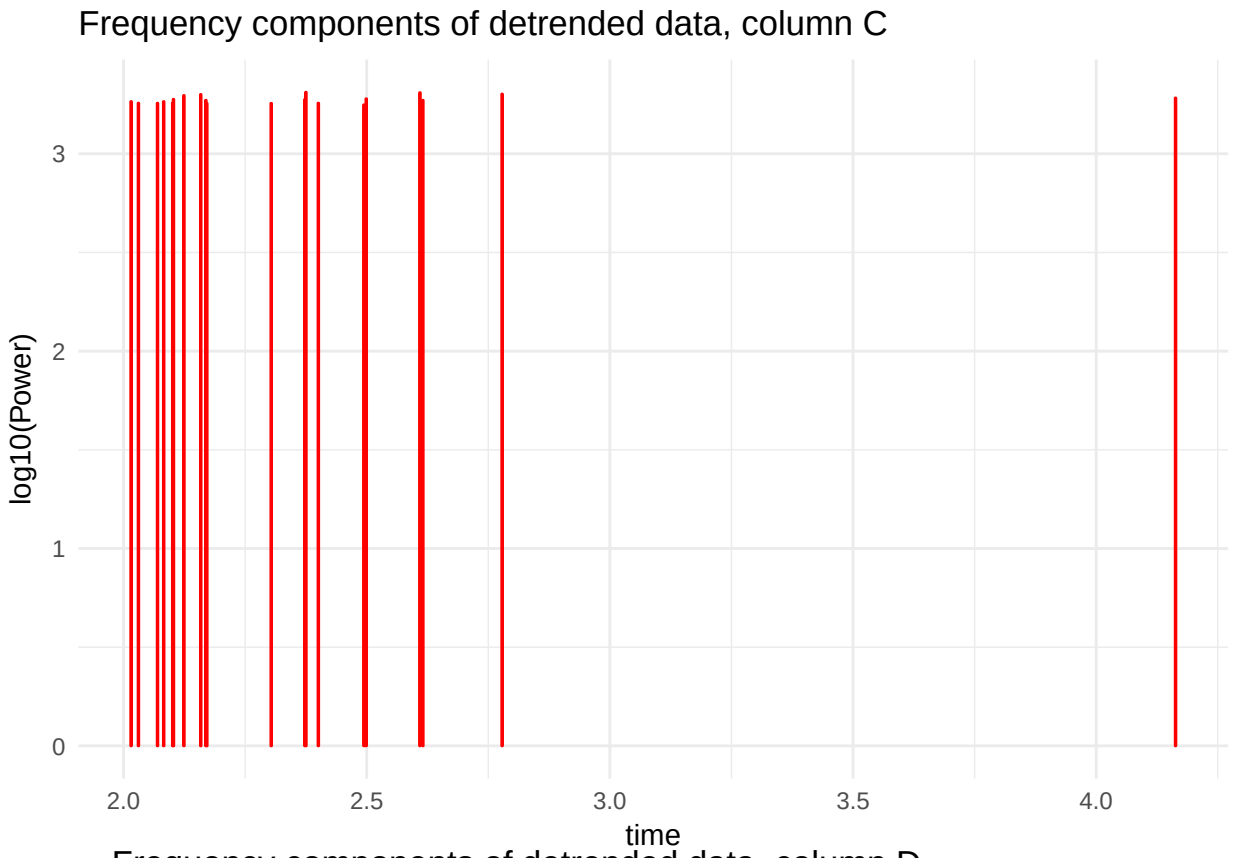

Frequency components of original data, column E
log10(Power)
0
2
4
6
0
20000
40000
time

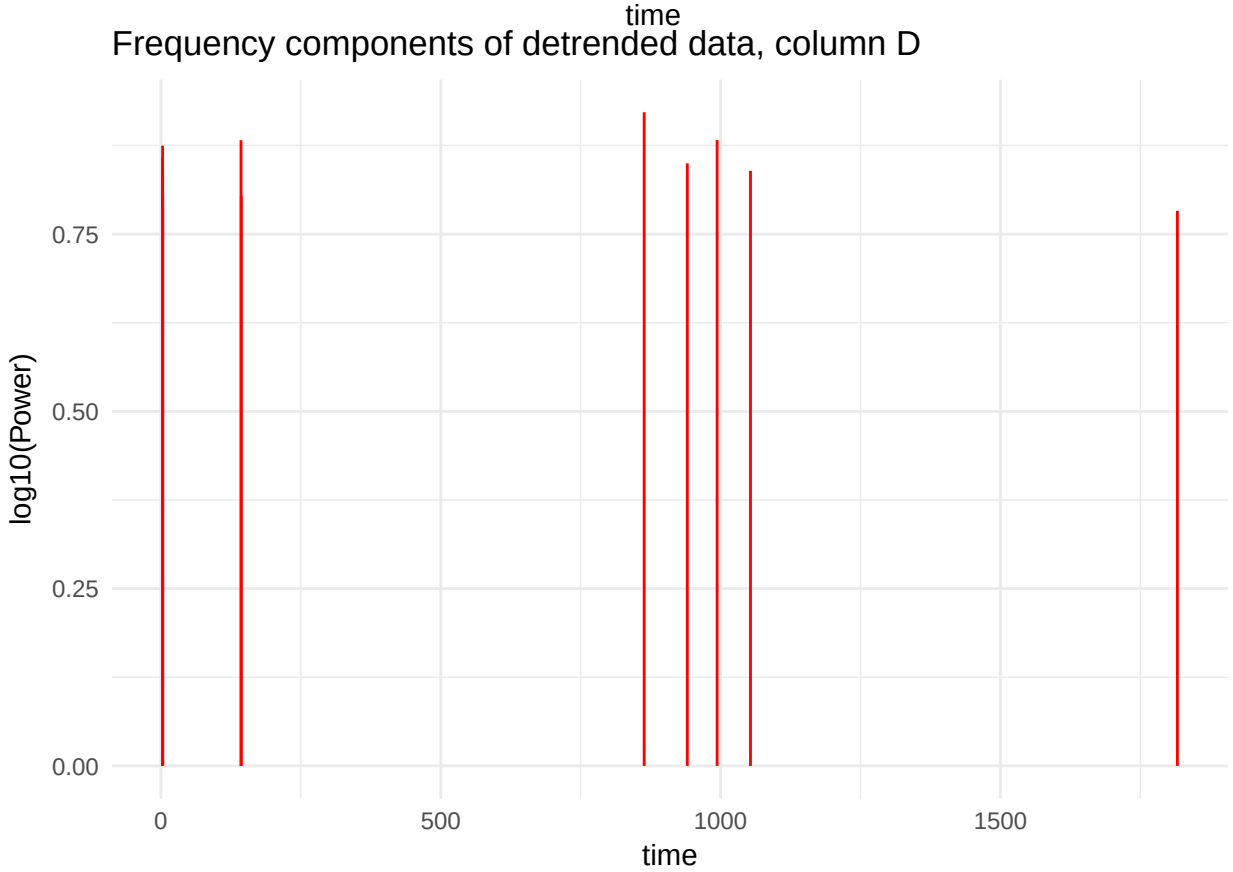


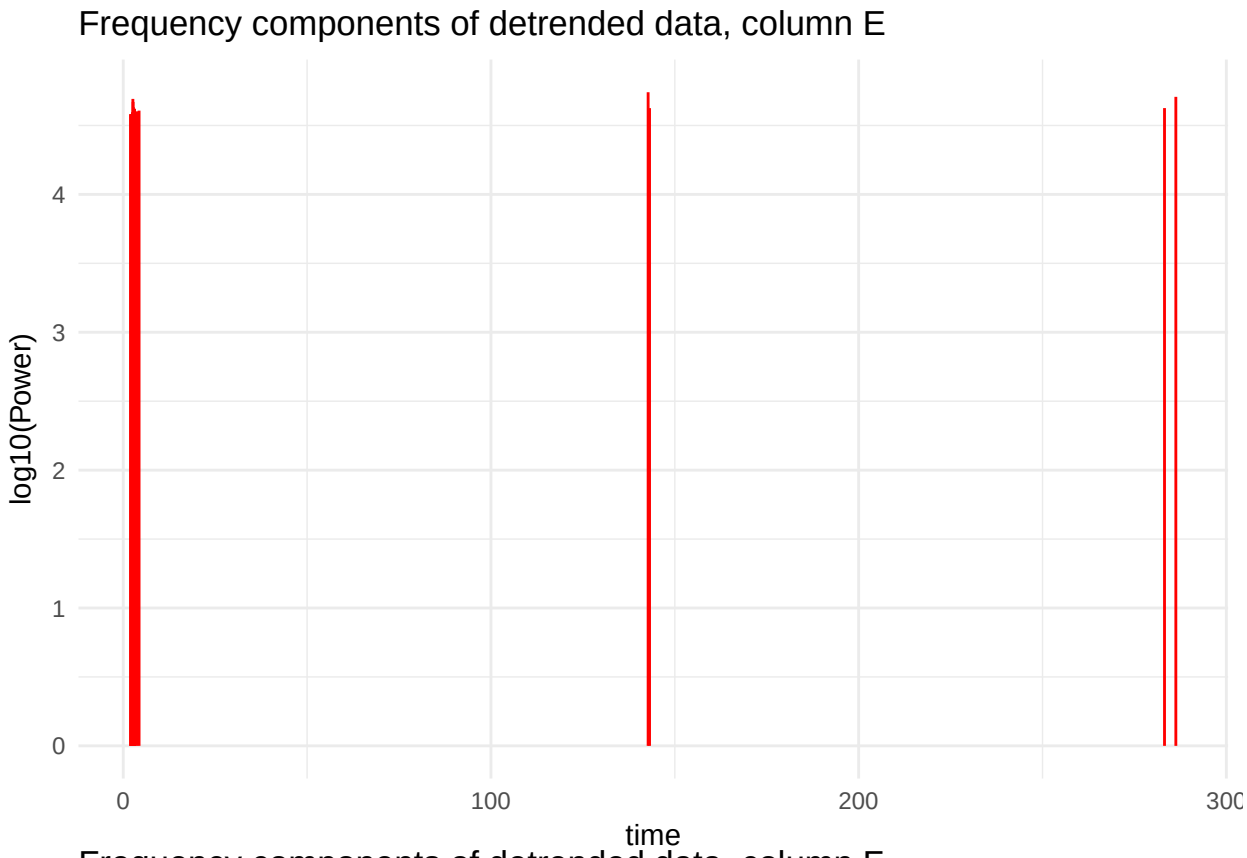

```
for(acol in names(dat1_detrended_fft)){
    ag <- ggplot(
        as.data.frame(
            tail(dat1_detrended_fft[[acol]],20)
        ),
        aes(x=time,y=log10power)
    )+geom_bar(stat="identity", color='red')+
    ggtitle(paste0("Frequency components of detrended data, column ", acol, sep=''))+
    xlab("time")+
    ylab("power")+
    scale_y_continuous(name='log10(Power)')+
    theme_minimal()
    print(ag)
}
```

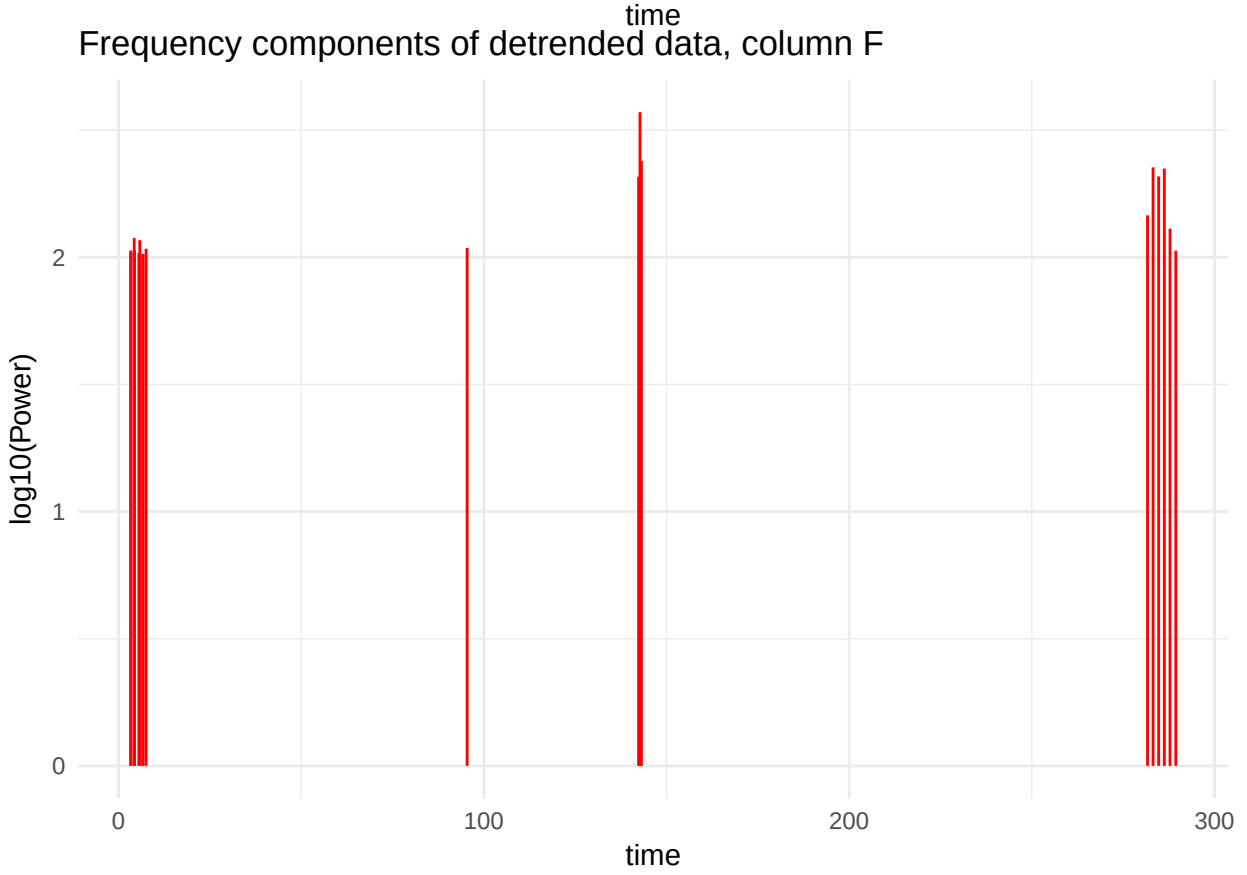

Frequency components of detrended data, column A
log10(Power)
0
1
2
3
100
150
200
250
300
time


Frequency components of detrended data, column B
log10(Power)
0
1
2
3
0
100
200
300
time


Frequency components of detrended data, column C
log10(Power)
3
2
1
0
2.0
2.5
3.0
3.5
4.0
time


Frequency components of detrended data, column D
log10(Power)
0.75
0.50
0.25
0.00
0
500
1000
1500
time





Before making any seasonality investigations, let's test whether data is Stationary, meaning that its properties

do not change over time. This is important because many algorithms assume Stationary data.

```
library(tseries)
for(acol in names(dat1_noid)){
    x <- dat1_noid[[acol]]
    suppressWarnings(adft <- adf.test(x, alternative='stationary'))
    apv = adft$p.value
    if( apv <= 0.05 ){
        cat("Original data, column ", acol, " : data IS stationary with a p-value of ", apv, ".\n", sep=
    } else {
        cat("Original data, column ", acol, " : data IS NOT stationary with a p-value of ", apv, ".\n",
    }
}
```

```
## Original data, column A : data IS stationary with a p-value of 0.01.
## Original data, column B : data IS stationary with a p-value of 0.01.
## Original data, column C : data IS stationary with a p-value of 0.01.
## Original data, column D : data IS stationary with a p-value of 0.01.
## Original data, column E : data IS stationary with a p-value of 0.01.
## Original data, column F : data IS stationary with a p-value of 0.01.
```

Because differencing is a common way to fix non-Stationary data, our detrended (that is differenced) data must be Stationary.

```
for(acol in names(dat1_detrended)){
    x <- dat1_detrended[[acol]]
    suppressWarnings(adft <- adf.test(x, alternative='stationary'))
    apv = adft$p.value
    if( apv <= 0.05 ){
        cat("Detrended data, column ", acol, " : data IS stationary with a p-value of ", apv, ".\n", se
    } else {
        cat("Detrended data, column ", acol, " : data IS NOT stationary with a p-value of ", apv, ".\n"
    }
}
```

```
## Detrended data, column A : data IS stationary with a p-value of 0.01.
## Detrended data, column B : data IS stationary with a p-value of 0.01.
## Detrended data, column C : data IS stationary with a p-value of 0.01.
## Detrended data, column D : data IS stationary with a p-value of 0.01.
## Detrended data, column E : data IS stationary with a p-value of 0.01.
## Detrended data, column F : data IS stationary with a p-value of 0.01.
```

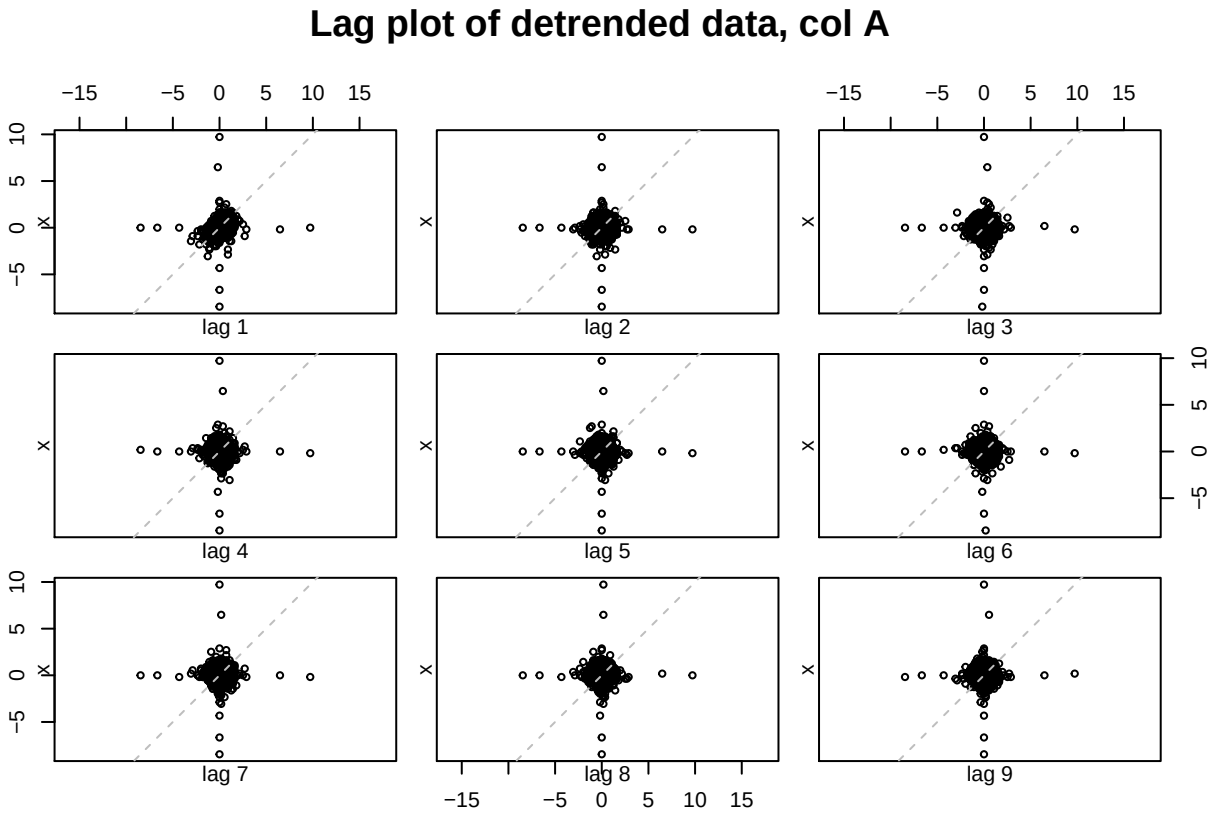
we will investigate whether the data is generated by an auto-regressive (AR) model meaning that data value at time point t(0) depends linearly on some previous data values (for example, t(-1) and t(-2)) plus a random term. In other words, data past values have an effect to current and future values. AR models have order which is equal to the number of previous data points affecting the present. AR(1): current data point depends only immediately previous data point. a first indication of AR comes from lag plots. That is, plot current data value against a value some time periods in the past, 1, 2, 3, etc.

```
print("lag plots of the detrended data")
```

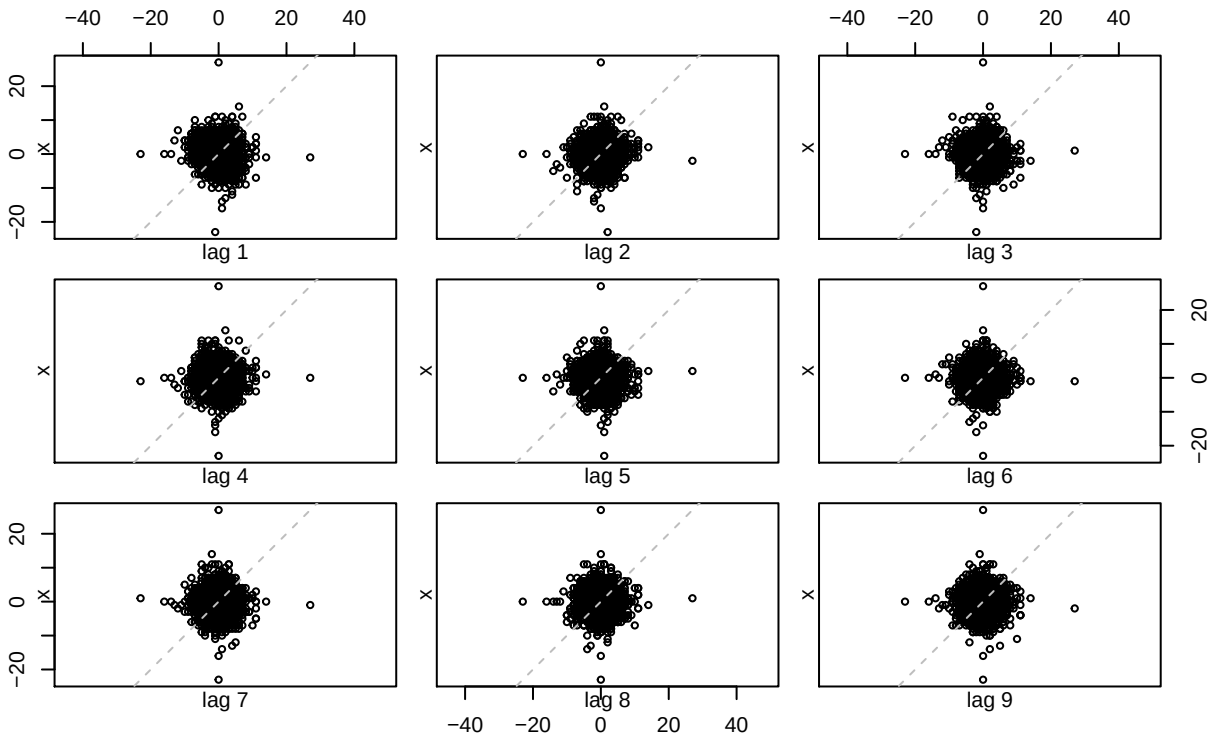
```
## [1] "lag plots of the detrended data"
```

```
for(acol in names(dat1_detrended)){
    x <- dat1_detrended[[acol]]
    lag.plot(
        x=x,
        lags=9,
```

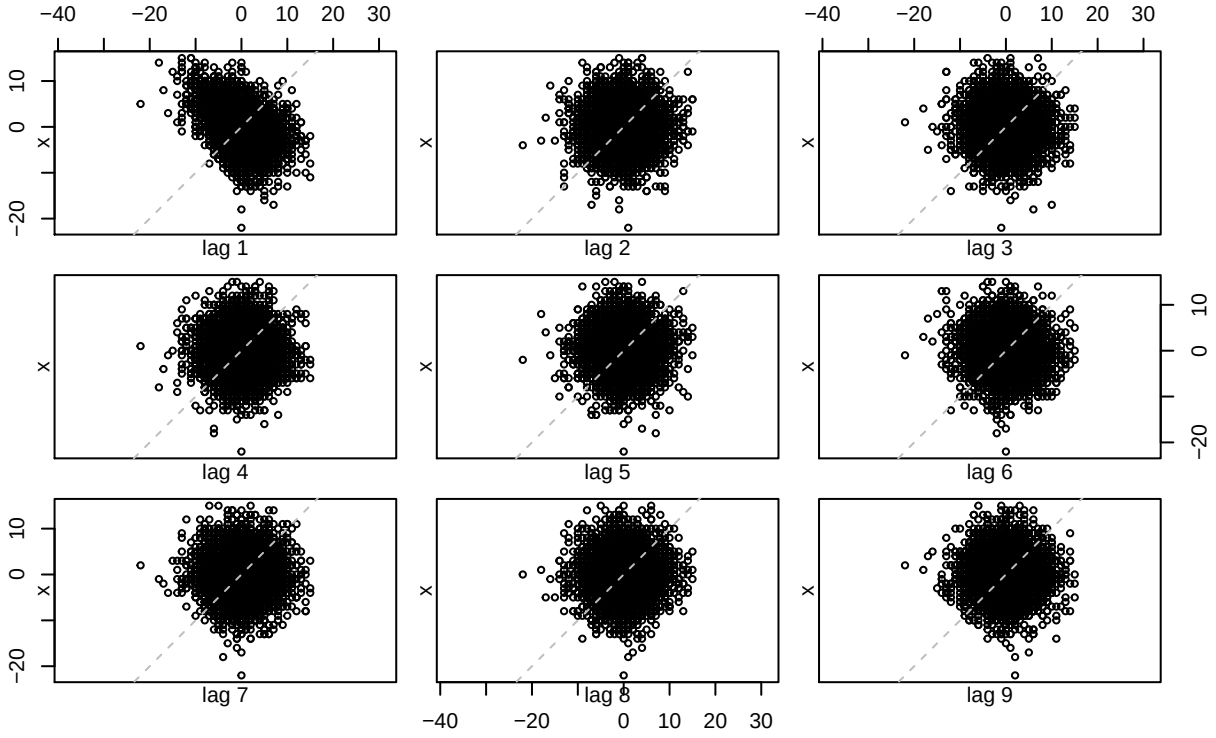
```
        main=paste0("Lag plot of detrended data, col ", acol, sep='')
    )
}
```

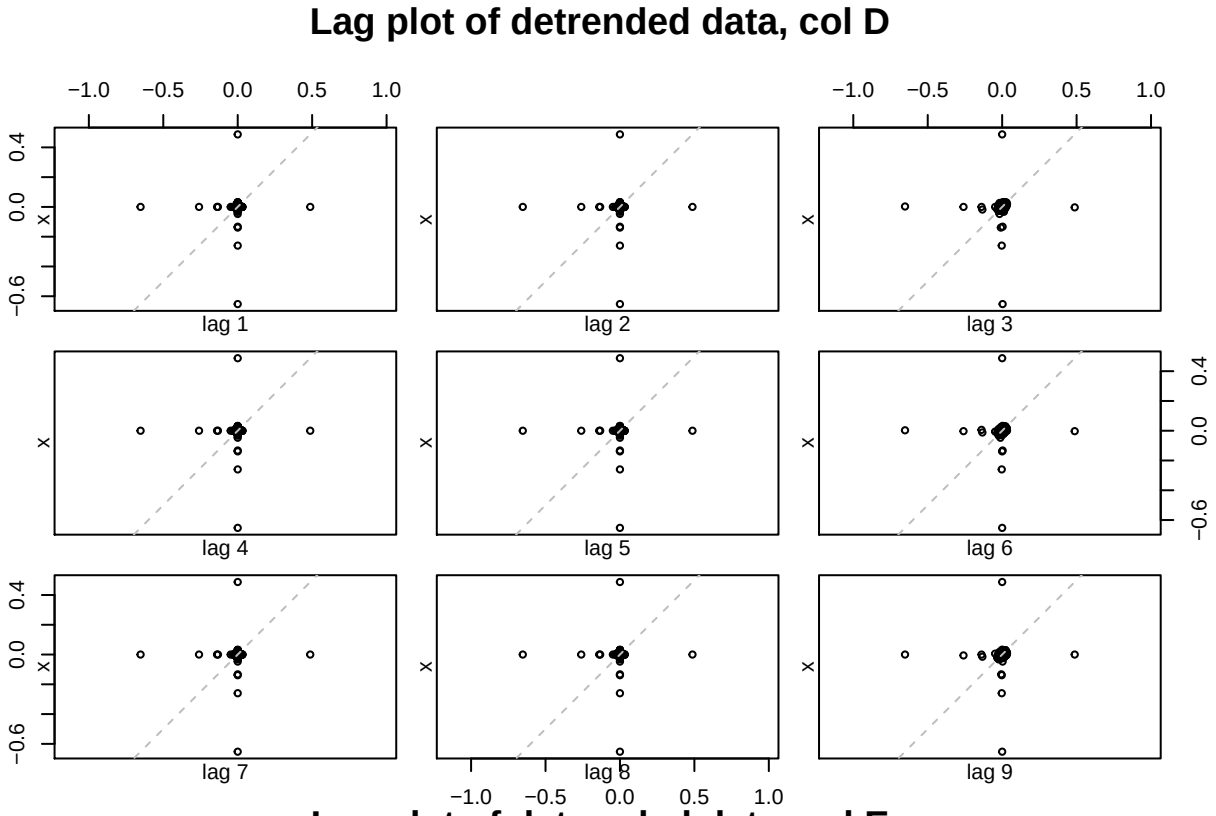


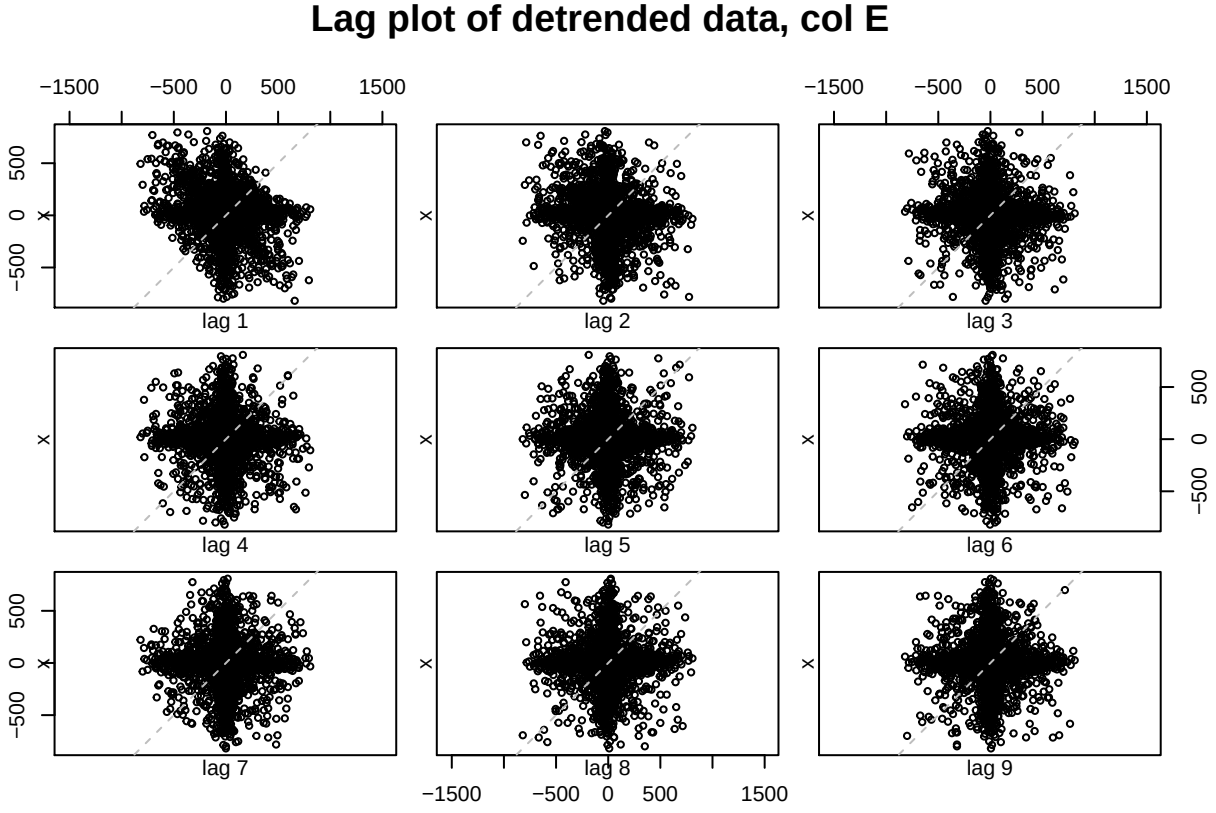
Lag plot of detrended data, col B

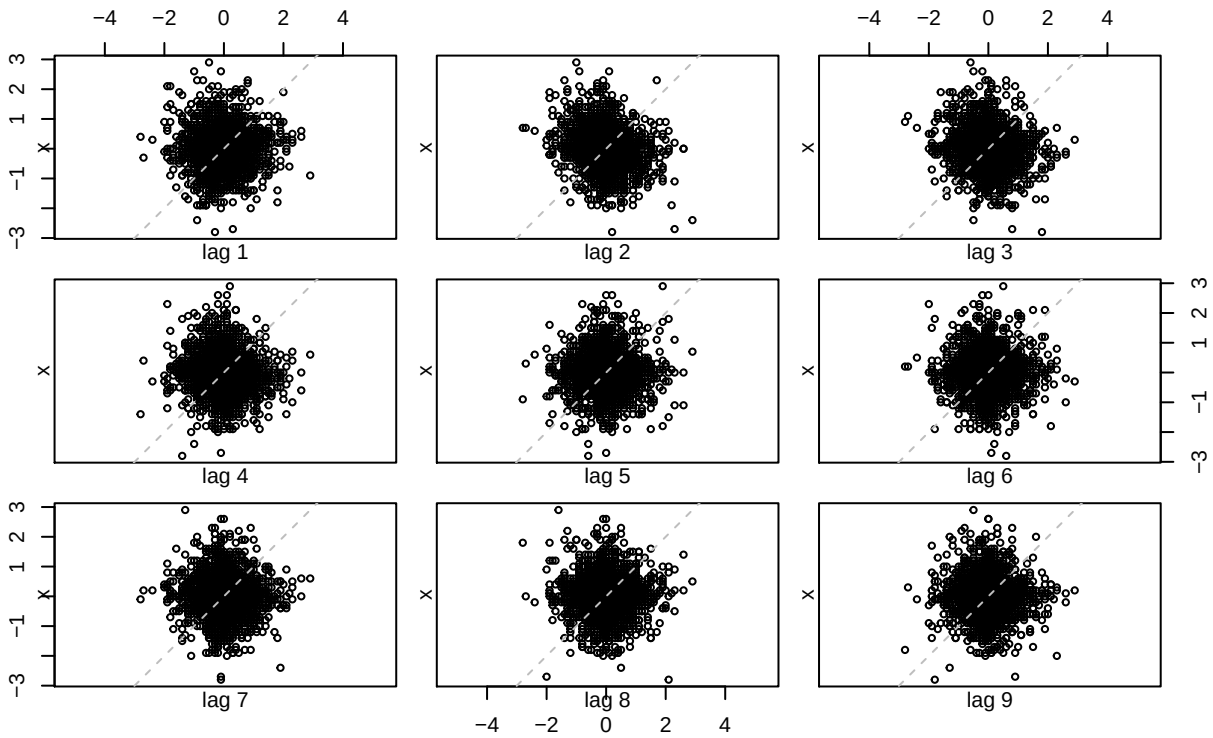


Lag plot of detrended data, col C




Lag plot of detrended data, col D
lag 1
lag 2
lag 3
lag 4
lag 5
lag 6
lag 7
lag 8
lag 9
x


Lag plot of detrended data, col E
lag 1
lag 2
lag 3
lag 4
lag 5
lag 6
lag 7
lag 8
lag 9
x

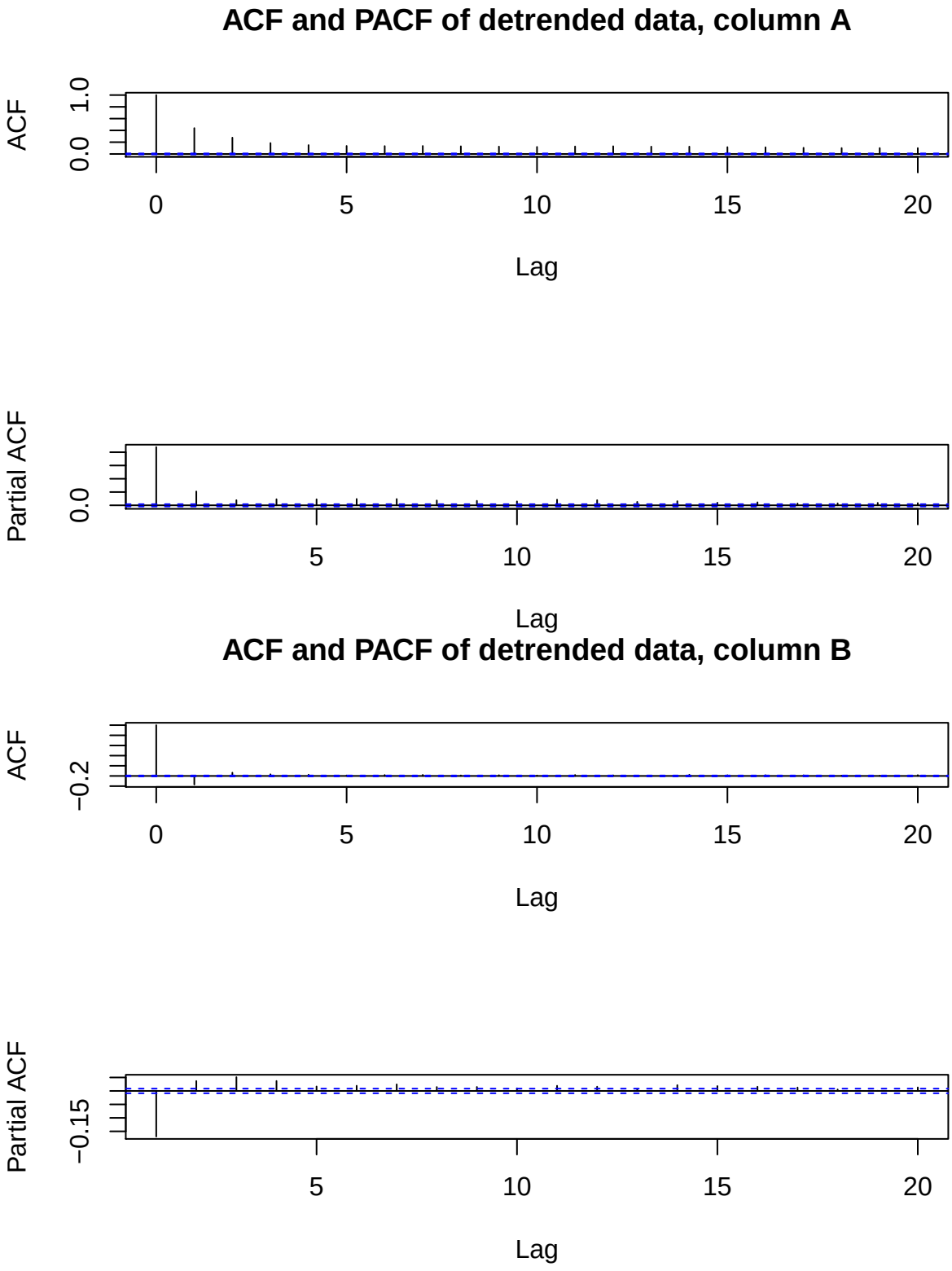


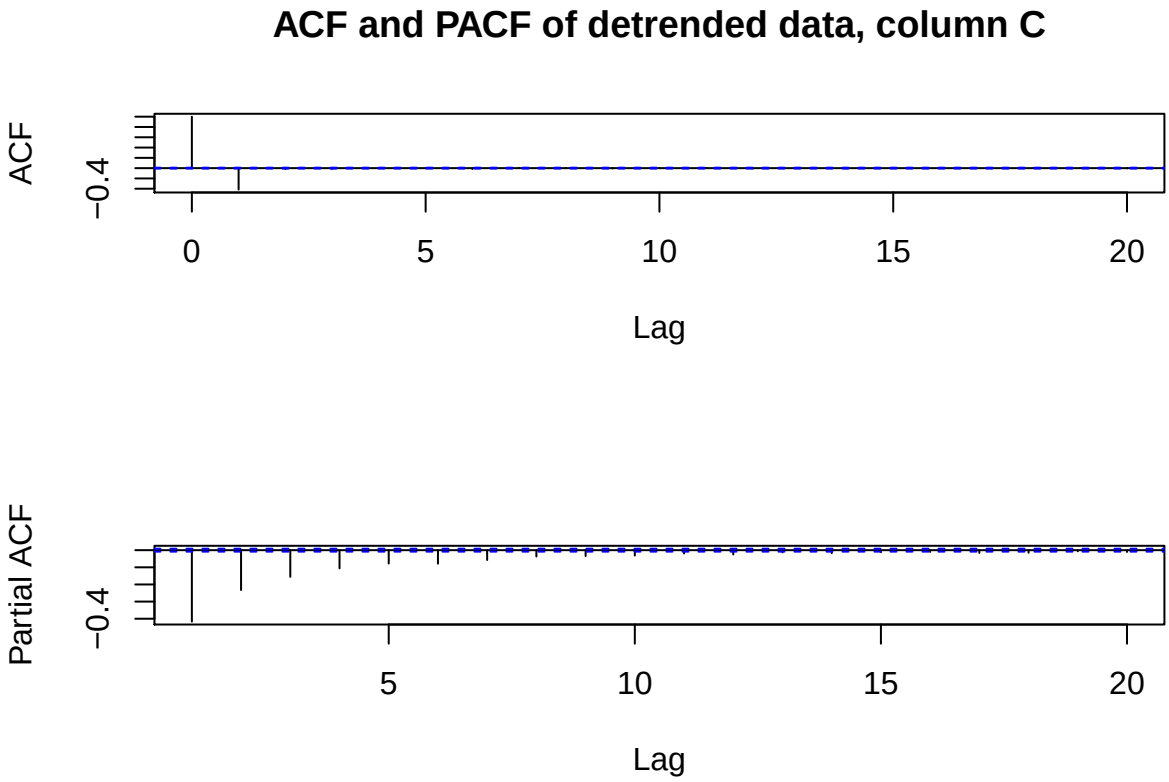
ACF is the correlation between points of our data lagged by some time period. E.g. for a time period equal to 1, we get the correlation between successive (in time) data points. The Partial ACF is the same as ACF but when all the shorter-term correlations are removed. If the PACF produces spikes which are statistical significant at time lag k then this is an indication that there is an autoregressive term in the data between two data points k time periods away. The more spikes, the more the dependencies on past values and the higher the order of the AR model. produce ACF and check the statistical significance of the spikes (over the dashed blue line) the Lag is measured in time-points as given in the data set (which we do not know neither we know if it is regular).

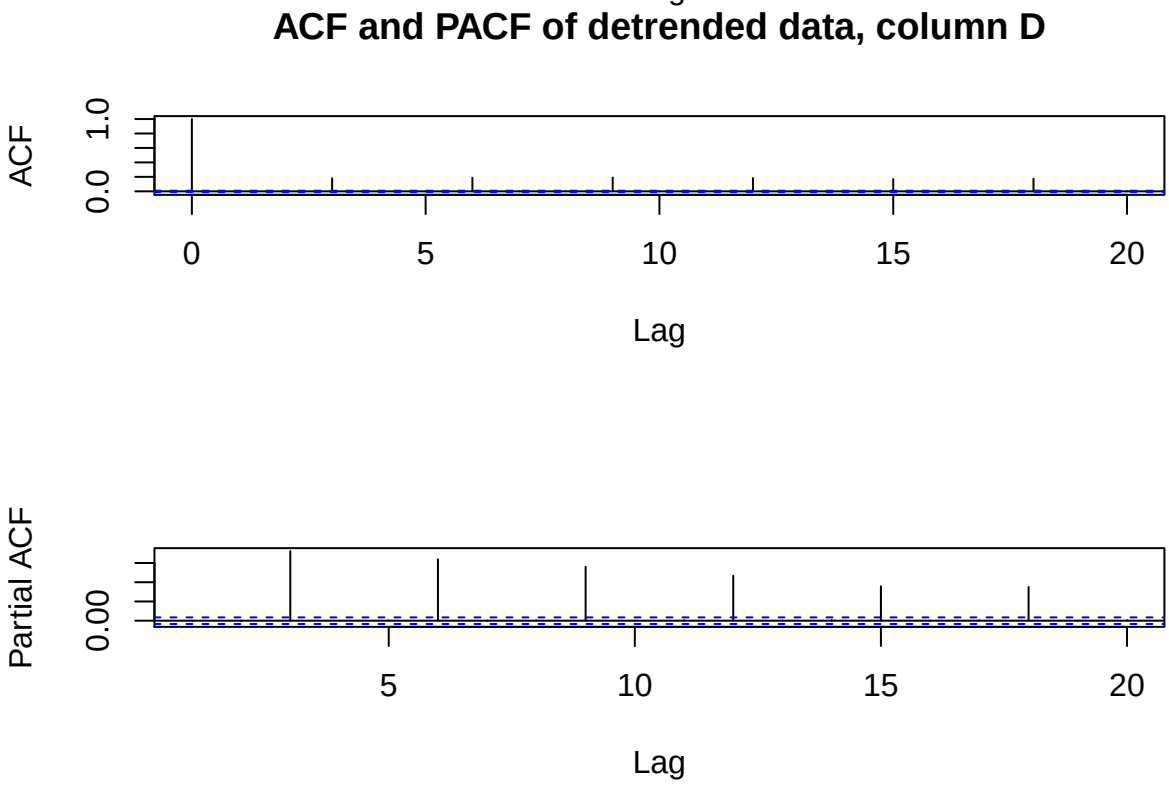
```
print("lag plots of the detrended data")
```

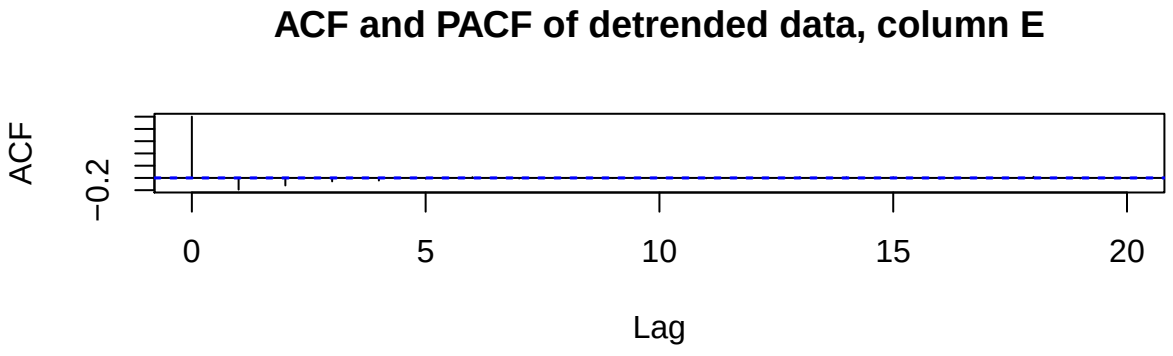
```
## [1] "lag plots of the detrended data"
```

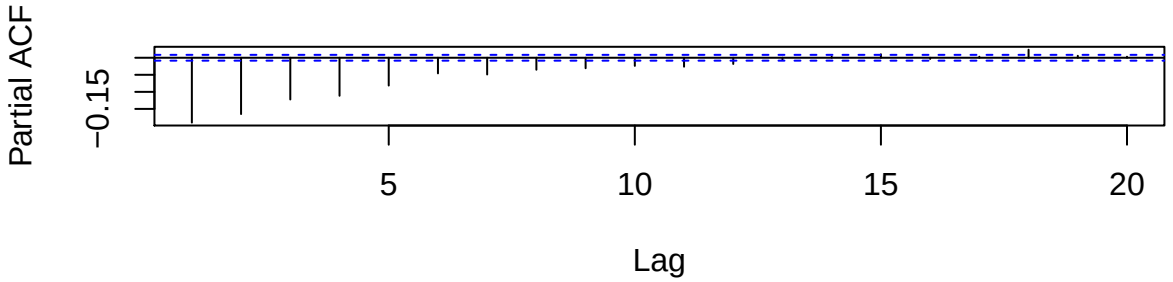
```
for(acol in names(dat1_detrended)){
    x <- dat1_detrended[[acol]]
    par(mfrow=c(2,1))
    acf(
        x=ts(x, freq=1),
        lag.max=20,
        main=paste0("ACF and PACF of detrended data, column ", acol, sep='')
    )
    pacf(
        x=ts(x, freq=1),
        lag.max=20,
        main=""
    )
}
```

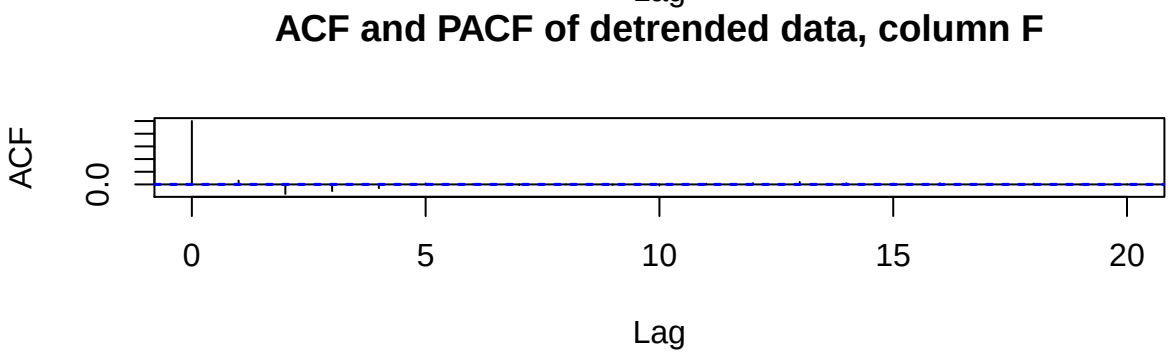

ACF and PACF of detrended data, column A
ACF
0.0
1.0
0
5
10
15
20
Lag
Partial ACF
0.0
5
10
15
20
Lag
ACF and PACF of detrended data, column B
ACF
−0.2
0
5
10
15
20
Lag
Partial ACF
−0.15
5
10
15
20
Lag

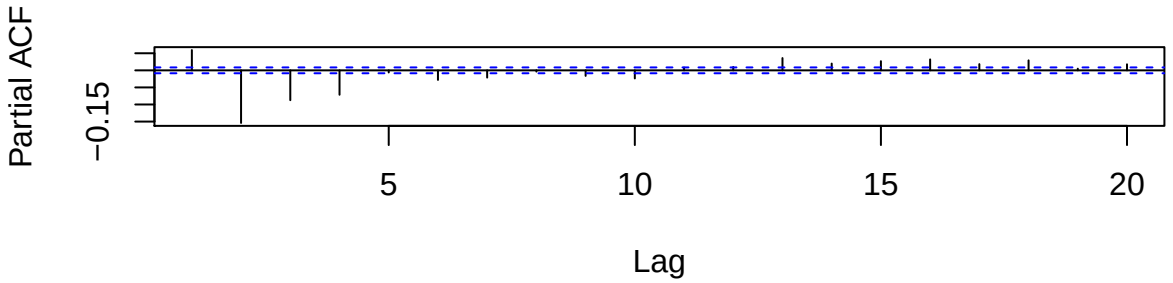

ACF and PACF of detrended data, column C
ACF
−0.4
0
5
10
15
20
Lag
Partial ACF
−0.4
5
10
15
20
Lag


ACF and PACF of detrended data, column D
ACF
0.0
1.0
0
5
10
15
20
Lag
Partial ACF
0.00
5
10
15
20
Lag









Fit our data to an ARIMA model (autoregressive integrated moving average). The model will allow us to forecast data if we need to. Or do simulations in order to assess our data.

```
library(forecast)
for(acol in names(dat1_detrended)){
```

```
    next #' but not right now...
    x <- dat1_detrended[[acol]]
    afit <- auto.arima(x)
}
```

Let's try standard timeseries decomposition into trend, seasonality and error components. Finally, we can use the ... prophet() it is extremely resource-hungry so here is one I did earlier for column 'A' of the original data: adat <- to_prophet_data_format(x=dat1_detrended[['A']])

```
adat <- readRDS('prophet/prophet_dat1_noid_A.adat.Rds')
head(adat)
```

```
##           ds    y
## 1 1873-12-02 36.5
## 2 1873-12-03 36.5
## 3 1873-12-04 36.5
## 4 1873-12-05 36.5
## 5 1873-12-06 36.5
## 6 1873-12-07 36.5
```

m <- prophet(df)

```
m <- readRDS('prophet/prophet_dat1_noid_A.Rds')
```

future <- make_future_dataframe(m, periods = 1)

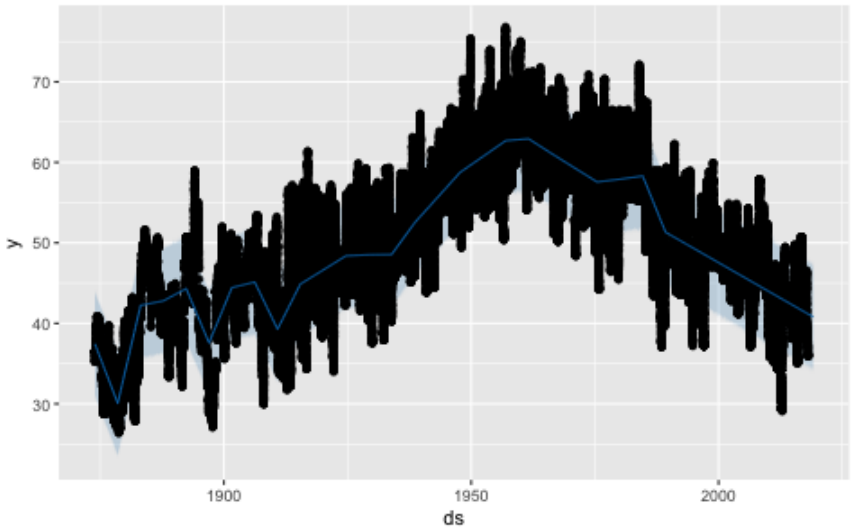
```
future <- readRDS('prophet/prophet_dat1_noid_A.future.Rds')
```

predict() will decompose the trend and seasonal components from the timeseries forecast <- predict(m, future)

```
forecast <- readRDS('prophet/prophet_dat1_noid_A.forecast.Rds')
```

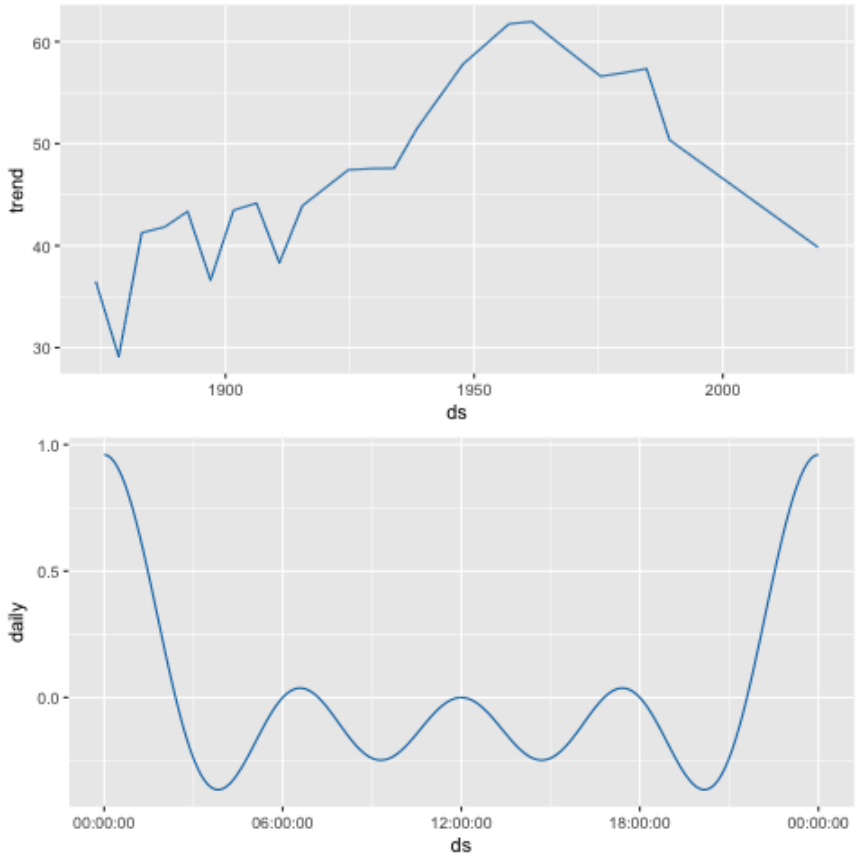
plot the forecast plot(m, forecast)

```
library(png)
img<-readPNG('prophet/prophet_dat1_noid_A.plot.png')
grid::grid.raster(img)
```



plot trend and seasonal components prophet_plot_components(m, forecast)

```
library(png)
img<-readPNG('prophet/prophet_dat1_noid_A.plot_components.png')
grid::grid.raster(img)
```


60
50
40
30
trend
1900
1950
2000
ds
1.0
0.5
0.0
daily
00:00:00
06:00:00
12:00:00
18:00:00
00:00:00
ds