Final Report

Andreas Hadjiprocopis

February, 2018

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

Section 1: 3rd party library dependencies: 1. FactoMineR 2. PerformanceAnalytics 3. Rmixmod 4. corrplot 5. factoextra 6. fitdistrplus 7. flexmix 8. forecast 9. ggplot2 10. ggpubr 11. mclust 12. nortest 13. png 14. reshape2 15. tseries

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

Section 2: Data preparation: filling or removing the NAs author: Andreas Hadjiprocopis date: February, 2018 output: pdf_document: latex_engine: pdflatex toc: true highlight: zenburn html_document: toc: true theme: united —

```
source('lib/IO.R');
source('lib/NA.R');

infile='original_data/dat1.csv'
outdir='cleaned_data';
outcleanfile=file.path(outdir, 'dat1.clean.csv')
outeliminateNAfile=file.path(outdir, 'dat1.eliminateNA.csv')
```

Read the original csv data file (dat1.csv)

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

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

```
if( is.null(dat1) ){
    cat("call to read_data() has failed for file '",infile,"'.\n", sep='')
    quit(status=1)
}
```

Remove all rows with at least one NA

```
results <- clean_dataset(
    inp=dat1,
    methods=c('remove_entire_row')
)
if( is.null(results) ){
    cat("call to clean_dataset() has failed for file '",infile,"'.\n", sep='')
    quit(status=1)
}
```

this is the clean data

```
clean_dat1 <- results$imputed
```

And save it

```
if( ! save_data(clean_dat1, outeliminateNAfile) ){
    cat("call to save_data() has failed for file '",outeliminateNAfile,"'.\n", sep='')
```

```
    quit(status=1)
}
```

```
## save_data(): data saved to file 'cleaned_data/dat1.eliminateNA.csv'.
```

Alternatively one can replace (fill) NAs with guesses. See notes at the end of this document. But we will not do it now because it takes quite some time. Uncomment this to try it out.

```
if( FALSE ){
results <- clean_dataset(
    inp=dat1,
    methods=c("mice", "kNN"),
    rng.seed=1234,
    # how many artificial NAs to introduce?
    random_NAs_to_introduce_in_each_column_percent=5/100,
    # number of repeats for the assessment
    repeats=16,
    # parallelise the procedure over so many CPU cores
    ncores=8
)
if( is.null(results) ){
    #cat("call to clean_dataset() has failed for file '",infile,"'.\n", sep='')
    quit(status=1)
}
}
```

this is the cleaned data as a list

```
clean_dat1 <- results$imputed
```

Statistics of the filling NAs procedure

```
stats_of_imputation <- results$stats
print(stats_of_imputation)
```

```
## NULL
```

finally, save the data

```
if( ! save_data(clean_dat1, outcleanfile) ){
    cat("call to save_data() has failed for file '",outcleanfile,"'.\n", sep='')
    quit(status=1)
}
```

```
## save_data(): data saved to file 'cleaned_data/dat1.clean.csv'.
```

The input data contains quite a few NAs I have created a few functions in lib/NA.R which either remove all rows with at least one NA or try to fill the NA with an appropriate value based on the other values in the row. Remove the NAs is straightforward but filling them out risks distorting the dataset. I have used two methods for filling: 1. kNN 2. mice I have created a function to assess which method works best. This is done as follows. 1. Remove all NAs from the dataset. 2. Create some random NAs in the clean dataset, say 5% of the total items. 3. Apply each method to filling the artificial NAs. 4. Calculate a metric of badness of fill by, say, simple root mean square of the difference between filled and actual values. 5. Report the method yielding the lowest value. 6. Repeat this process N times. Method 'kNN' has done consistently better. And so this is the method used. The main function is clean_dataset() which takes parameters to either eliminate rows with NAs or fill NAs. Additionally, input parameters control the number of repeats should a filling-NAs route was taken, the percentage of artificial NAs to introduce and the methods to assess, and the column names to process. A very important feature of the clean_dataset() function is that assessment, which is

quite time-consuming, can be *parallelised* over as many CPU cores as specified. A *parallelisation* option is important to exist in such methods and I always spend a bit more effort in implementing it, and considerably more in ... debugging it. But it is worthy. here is a dump of the NA.R library:

```
clean_dataset <- function(
    inp=NULL,
    methods=c("mice", "kNN"),
    columns_to_do=c("A","B","C","D","E","F"),
    rng.seed=as.numeric(Sys.time()),
    random_NAs_to_introduce_in_each_column_percent=5/100,
    repeats=5,
    ncores=1
){
    whoami=paste0(match.call()[[1]],'()',collapse='')
    if( methods %in% 'remove_entire_row' ){
        # asked to remove all rows with at least 1 NA
        # no guessing of what an NA could be is required
        dummy <- remove_all_rows_with_NA(inp=inp)
        if( is.null(dummy) ){
            cat(whoami, " : call to remove_all_rows_with_NA() has failed.\n");
            return(NULL)
        }
        ret=list()
        ret[['stats']] = NULL
        ret[['imputed']] = dummy
        return(ret)
    }

    # we are asked to guess NA values
    best <- assess_na_methods_repeatedly(
        inp=inp,
        methods=methods,
        columns_to_do=columns_to_do,
        rng.seed=rng.seed,
        random_NAs_to_introduce_in_each_column_percent=random_NAs_to_introduce_in_each_column_percent,
        repeats=repeats,
        ncores=ncores
    )
    if( is.null(best) ){
        cat("call to assess_na_methods_repeatedly() has failed.\n", sep='')
        return(NULL)
    }
    recom_method = best$recommended_method
    if( recom_method == 'none' ){ recom_method = best$best_method_wrt_mean }

    clean_data = NULL;
    if( recom_method == 'kNN' ){
        clean_data <- deal_with_na_using_kNN(
            inp=inp,
            columns_to_do=columns_to_do,
            rng.seed=rng.seed
        )
        if( is.null(clean_data) ){
            cat(whoami, " : call to deal_with_na_using_kNN() has failed.\n", sep='')
```

```
            return(NULL)
        }
    } else {
        clean_data <- deal_with_na_using_mice(
            inp=inp,
            columns_to_do=columns_to_do,
            rng.seed=rng.seed
        )
        if( is.null(clean_data) ){
            cat(whoami, " : call to deal_with_na_using_mice() has failed.\n", sep='')
            return(NULL)
        }
    }
    ret=list()
    ret[['stats']] = best
    ret[['imputed']] = clean_data
    return(ret)
}
remove_all_rows_with_NA <- function(
    inp=NULL
){
    # asked to remove all rows with at least 1 NA
    # no guessing of what an NA could be is required
    return(na.omit(inp))
}

assess_na_methods_repeatedly <- function(
    inp=NULL,
    methods=c("mice", "kNN"),
    columns_to_do=c("A","B","C","D","E","F"),
    rng.seed=as.numeric(Sys.time()),
    random_NAs_to_introduce_in_each_column_percent=5/100,
    repeats=10,
    ncores=1
){
    library(parallel)
    whoami=paste0(match.call()[[1]],'()',collapse='')

    cat(whoami, " : spawning ", repeats, " processes over ", ncores, " cores.\n", sep='')
    set.seed(rng.seed)
    seeds = sample.int(n=1000, size=repeats)
    num_methods = length(methods)
    # mclapply will return a list of results, one for each process spawn,
    # some processes might be failures, in this case the result will be of class 'try-error'
    # how does that work:
    # 1. all is done within system.time() in order to measure time taken
    # 2.
    stime <- system.time({
      res <-mclapply(
        X=seeds,
        function(aseed){
            assess_na_methods_once(
                inp,
```

```
            methods,
            columns_to_do,
            aseed,
            random_NAs_to_introduce_in_each_column_percent
        )
    },
    mc.cores=ncores
  ) # mclapply
}) # system.time
print(res) # debug
# remove all test results which are NULL (crash, failed etc.)
clean_res=list()
num_failed = 0; num_clean_res = 0
for(i in 1:repeats){
    if( is.null(res[[i]]) || (class(res[[i]]) == "try-error") ){
        cat(whoami, " : repeat #",i," has failed.\n", sep='');
        print(is.null(res[[i]]))
        num_failed = num_failed + 1
    } else {
        num_clean_res = num_clean_res+1
        clean_res[[num_clean_res]] = res[[i]]
    }
}
if( num_failed > 0 ){
    cat(whoami, " : ", num_failed, " repeats (of ", repeats, ") have failed.\n", sep='')
} else {
    cat(whoami, " : all ", repeats, " repeats have succeeded.\n", sep='')
}

stats=matrix(0, nrow=num_methods, ncol=2)
rownames(stats) <- methods
colnames(stats) <- c('wins', 'mean')
mean_assessment=0
print(clean_res)
for(i in 1:num_clean_res){
    print(clean_res[[i]])
    winning_method = clean_res[[i]][['best_method']]
    stats[winning_method,'wins'] = stats[winning_method,'wins']+1
    for(amethod in methods){
        mean_got = clean_res[[i]][['assessment']][[amethod]][['mean']]
        stats[amethod,'mean'] = stats[amethod,'mean']+mean_got/num_clean_res
    }
}
best_method_wrt_wins = names(which(stats[,'wins'] == max(stats[,'wins'])))
best_method_wrt_mean = names(which(stats[,'mean'] == min(stats[,'mean'])))
best_wins = stats[best_method_wrt_wins,'wins']
best_mean = stats[best_method_wrt_mean,'mean']
ret = list()
ret[['individual_results']] = ret
ret[['stats']] = stats
ret[['best_method_wrt_wins']] = best_method_wrt_wins
ret[['best_method_wrt_mean']] = best_method_wrt_mean
ret[['best_wins']] = best_wins
```

```
    ret[['best_mean']] = best_mean
    if( best_method_wrt_wins == best_method_wrt_mean ){
        touse = best_method_wrt_mean
    } else {
        #no consensus
        touse = 'none'
    }
    ret[['recommended_method']] = touse
    cat(whoami, " : after ", repeats, " repeats here is the overall assessment:\n", sep='')
    print(stats)
    cat("best method wrt to mean: ", best_method_wrt_mean, "\n", sep='')
    cat("best method wrt to counting best performance (wins): ", best_method_wrt_wins, "\n", sep='')
    cat("recommended method for NA imputation: ", touse, "\n", sep='')
    cat(whoami, " : finished ", repeats, " repeats in ", stime[3], " seconds.\n", sep='')
    return(ret)
}
assess_na_methods_once <- function(
    inp=NULL,
    methods=c("mice", "kNN"),
    columns_to_do=c("A","B","C","D","E","F"),
    rng.seed=as.numeric(Sys.time()),
    random_NAs_to_introduce_in_each_column_percent=5/100
){
    whoami=paste0(match.call()[[1]],'()',collapse='')

    # First remove all NAs from input
    cleaned_inp = inp[complete.cases(inp), ]

    nrows = nrow(cleaned_inp)
    ncols_to_do = length(columns_to_do)
    nNAs_per_column = round(nrows * random_NAs_to_introduce_in_each_column_percent)

    idx_of_NAs_in_columns = list();
    # then introduce our own NAs for each column
    # first get a random set of indices for each column:
    total_NAs = 0
    for(acol in columns_to_do){
        idx_of_NAs_in_columns[[acol]] = sample(1:nrows, nNAs_per_column)
    }
    cat(whoami, " : ", (nNAs_per_column*ncols_to_do), " NAs were added in total over all columns (conta

    # then set them to NA
    cleaned_inp_with_NAs = cleaned_inp
    for(acol in columns_to_do){
        idx_for_NA_for_this_col = idx_of_NAs_in_columns[[acol]]
        cleaned_inp_with_NAs[[acol]][idx_for_NA_for_this_col] <- NA
    }

    # now call each method for imputing our random but controlled NAs
    imputed = NULL
    best_mean = -1
    best_method = NULL
    assessment = list()
```

```
mean_assessment = list()
for(amethod in methods){
    # inputed will be a list with NAs completed by the methods
    if( amethod == "kNN" ){
        imputed = deal_with_na_using_kNN(
            inp=cleaned_inp_with_NAs,
            columns_to_do=columns_to_do
        )
        if( is.null(imputed) ){
            cat(whoami, " : call to deal_with_na_using_kNN() has failed.\n", sep='')
            return(NULL)
        }
    } else if( amethod == "mice" ){
        imputed = deal_with_na_using_mice(
            inp=cleaned_inp_with_NAs,
            columns_to_do=columns_to_do
        )
        if( is.null(imputed) ){
            cat(whoami, " : call to deal_with_na_using_mice() has failed.\n", sep='')
            return(NULL)
        }
    } else {
        cat(whoami, " : method '",amethod,"' is not known.\n", sep='')
        return(NULL)
    }
    # and do the assessment of this specific imputation
    assessment[[amethod]] = c()
    for(acolname in columns_to_do){
        assessment[[amethod]][acolname] = calculate_discrepancy(
            imputed[[acolname]],
            cleaned_inp[[acolname]],
            idx_of_NAs_in_columns[[acolname]]
        )
    }
    assessment[[amethod]]['mean'] = mean(assessment[[amethod]])
    if( is.null(best_method) || (assessment[[amethod]]['mean'] < best_mean) ){
        best_mean = assessment[[amethod]]['mean']
        best_method = amethod
    }
} # for methods
# we are looking for the lowest mean discrepancy over all columns
cat(whoami, " : results of assessment:\n", sep='')
for(amethod in methods){
    print(assessment[[amethod]])
    cat("mean for method '", amethod, "' is ", assessment[[amethod]]['mean'], "\n\n", sep='')
}
cat("--------------------------------\n", sep='')
cat(whoami, " : best method wrt mean assessment over all columns is \"", best_method, "\" with mean
ret = list()
ret[['best_method']] = best_method
ret[['best_mean']] = best_mean
ret[['assessment']] = assessment
ret[['seed']] = rng.seed
```

```
    return(ret)
}
calculate_discrepancy <- function(
    vector1=NULL,
    vector2=NULL,
    indices=NULL
){
    whoami=paste0(match.call()[[1]],'()',collapse='')
    if( is.null(indices) ){ indices=c(1:nrow(vector1)) }
    sum = 0
    for(i in indices){
        sum = sum + (vector1[i]-vector2[i])^2
    }
    return( sqrt(sum/length(indices)) )
}
deal_with_na_using_kNN <- function(
    inp=NULL,
    columns_to_do=c("A","B","C","D","E","F"),
    rng.seed=as.numeric(Sys.time())
){
    library(DMwR)
    whoami=paste0(match.call()[[1]],'()',collapse='')
    columns_to_ignore=setdiff(colnames(inp), columns_to_do)
    # remove those 'ignore' columns from input data before processing
    ignored_columns=list()
    for(acol in columns_to_ignore){
        # move the ignored column temporarily here
        ignored_columns[[acol]] <- inp[[acol]]
        # and erase it from the data to be processed
        inp[[acol]] <- NULL
        cat(whoami, " : column '",acol,"' will not have its NAs imputed, it is temporarily removed.\n",
    }

    cat(whoami, " : calling knnImputation, columns to consider: ", paste0(columns_to_do,collapse=','),
    # use k >= 3 for eliminating the "Error in rep(1, ncol(dist)) : invalid 'times' argument"
    # see https://stackoverflow.com/questions/36239695/error-with-knnimputer-from-the-dmwr-package-inva
    knnOutput <- knnImputation(
        data=inp[,columns_to_do],
        k=5
    )
    if( is.null(knnOutput) ){
        cat(whoami, " : call to knnImputation() has failed.\n", sep='')
        return(NULL)
    }
    # now add the ignored columns back to output
    for(acol in columns_to_ignore){
        knnOutput[[acol]] <- ignored_columns[[acol]]
    }
    cat(whoami, " : done.\n", sep='')
    return(knnOutput)
}
deal_with_na_using_mice <- function(
    inp=NULL,
```

```
    columns_to_do=c("A","B","C","D","E","F"),
    rng.seed=as.numeric(Sys.time())
){
    library(mice)
    whoami=paste0(match.call()[[1]],'()',collapse='')

    columns_to_ignore=setdiff(colnames(inp), columns_to_do)
    # remove those 'ignore' columns from input data before processing
    ignored_columns=list()
    for(acol in columns_to_ignore){
        # move the ignored column temporarily here
        ignored_columns[[acol]] <- inp[[acol]]
        # and erase it from the data to be processed
        inp[[acol]] <- NULL

        # we can leave the ignored column in and use
        # predM[, c(acol)]=0
        # to ignore it, but we remove it.
        cat(whoami, " : column '",acol,"' will not have its NAs imputed, it is temporarily removed.\n",
    }

    init = mice(inp, maxit=0)
    meth = init$method
    predM = init$predictorMatrix

    cat(whoami, " : calling mice ...\n", sep='')
    miceOutput <- mice(
        inp,
        method='pmm',
        predictorMatrix=predM,
        m=6,
        seed=rng.seed
    )
    if( is.null(miceOutput) ){
        cat(whoami, " : call to mice() has failed.\n", sep='')
        return(NULL)
    }
    cat(whoami, " : information from mice:\n", sep='')
    print(miceOutput)
    cat(whoami, " : doing the imputation ...\n", sep='')
    compl <- complete(miceOutput)
    if( is.null(compl) ){
        cat(whoami, " : call to mice() has failed.\n", sep='')
        return(NULL)
    }
    cat(whoami, " : done.\n", sep='')
    return(compl)
}
```

‘‘‘

Section 3: timeseries analysis : normality, q-q plots, distr estimation

```
set.seed(1234)
```

```
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']]
dat1_detrended_id <- list(c(dat1_id[['id']][2:length(dat1_id[['id']])]))
names(dat1_detrended_id) <- c('id')
# detrend the data
dat1_detrended <- detrend_dataset(inp=dat1_noid,times=1)
```

Check for normality: do data come from a Gaussian distribution process For the Anderson-Darling and Shapiro-Wilk normality test the p-value refers to the null hypothesis being that data DOES not come from a Gaussian distribution process (i.e. high p-value means 'a high probability that data comes from Gaussian distribution process') Data (detrended or not) do not seem to come from a Gaussian distribution process because the p-values of the test are very very small. First let's test a normally distributed sample just to see what kind of p-values to expect.

```
library(nortest)
print(ad.test(rnorm(n=1000, mean=5, sd=3)))
```

```
##
##  Anderson-Darling normality test
##
## data:  rnorm(n = 1000, mean = 5, sd = 3)
## A = 0.57857, p-value = 0.1323
```

```
print(ad.test(rnorm(n=1000, mean=5, sd=1)))
```

```
##
##  Anderson-Darling normality test
##
## data:  rnorm(n = 1000, mean = 5, sd = 1)
## A = 0.18795, p-value = 0.9025
```

```
print(shapiro.test(rnorm(n=1000, mean=5, sd=3)))
```

```
##
##  Shapiro-Wilk normality test
##
```

```
## data:  rnorm(n = 1000, mean = 5, sd = 3)
## W = 0.99866, p-value = 0.6587
```

```
print(shapiro.test(rnorm(n=1000, mean=5, sd=1)))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  rnorm(n = 1000, mean = 5, sd = 1)
## W = 0.99902, p-value = 0.8817
```

it looks like if p-value is not extremely small then we can assume it passes the test of normality.

```
for(acol in names(dat1_noid)){
    cat("original data, col", acol, "\n", sep='')
    x<-ad.test(dat1_noid[[acol]])
    print(x)
}
```

```
## original data, colA
##
##  Anderson-Darling normality test
##
## data:  dat1_noid[[acol]]
## A = 75.256, p-value < 2.2e-16
##
## original data, colB
##
##  Anderson-Darling normality test
##
## data:  dat1_noid[[acol]]
## A = 1147.2, p-value < 2.2e-16
##
## original data, colC
##
##  Anderson-Darling normality test
##
## data:  dat1_noid[[acol]]
## A = 2030.8, p-value < 2.2e-16
##
## original data, colD
##
##  Anderson-Darling normality test
##
## data:  dat1_noid[[acol]]
## A = 138.07, p-value < 2.2e-16
##
## original data, colE
##
##  Anderson-Darling normality test
##
## data:  dat1_noid[[acol]]
## A = 7884.2, p-value < 2.2e-16
##
## original data, colF
##
```

```
##  Anderson-Darling normality test
##
## data:  dat1_noid[[acol]]
## A = 10508, p-value < 2.2e-16
```

```
for(acol in names(dat1_detrended)){
    cat("detrended data, col", acol, "\n", sep='')
    x<-ad.test(dat1_detrended[[acol]])
    print(x)
}
```

```
## detrended data, colA
##
##  Anderson-Darling normality test
##
## data:  dat1_detrended[[acol]]
## A = 3369.7, p-value < 2.2e-16
##
## detrended data, colB
##
##  Anderson-Darling normality test
##
## data:  dat1_detrended[[acol]]
## A = 3424, p-value < 2.2e-16
##
## detrended data, colC
##
##  Anderson-Darling normality test
##
## data:  dat1_detrended[[acol]]
## A = 1491.1, p-value < 2.2e-16
##
## detrended data, colD
##
##  Anderson-Darling normality test
##
## data:  dat1_detrended[[acol]]
## A = 10395, p-value < 2.2e-16
##
## detrended data, colE
##
##  Anderson-Darling normality test
##
## data:  dat1_detrended[[acol]]
## A = 11831, p-value < 2.2e-16
##
## detrended data, colF
##
##  Anderson-Darling normality test
##
## data:  dat1_detrended[[acol]]
## A = 12326, p-value < 2.2e-16
```

Conclusion: data do not pass the Normality test. This means that non-parameteric statstical tools should be used for assessing the data. For example Pearson's require bivariate normality while Spearman's does not.

print summary statistics for original data:

```
for(acol in names(dat1_noid)){
    x <- dat1_noid[[acol]]
    cat("Original data, column ", acol, ", mean=", mean(x), ", sd=", sd(x), "\n", sep='')
}
```

```
## Original data, column A, mean=49.39303, sd=9.499579
## Original data, column B, mean=80.99835, sd=11.71468
## Original data, column C, mean=5.08511, sd=5.262529
## Original data, column D, mean=29.81647, sd=0.3481571
## Original data, column E, mean=83.66461, sd=159.3282
## Original data, column F, mean=0.3703302, sd=0.8356359
```

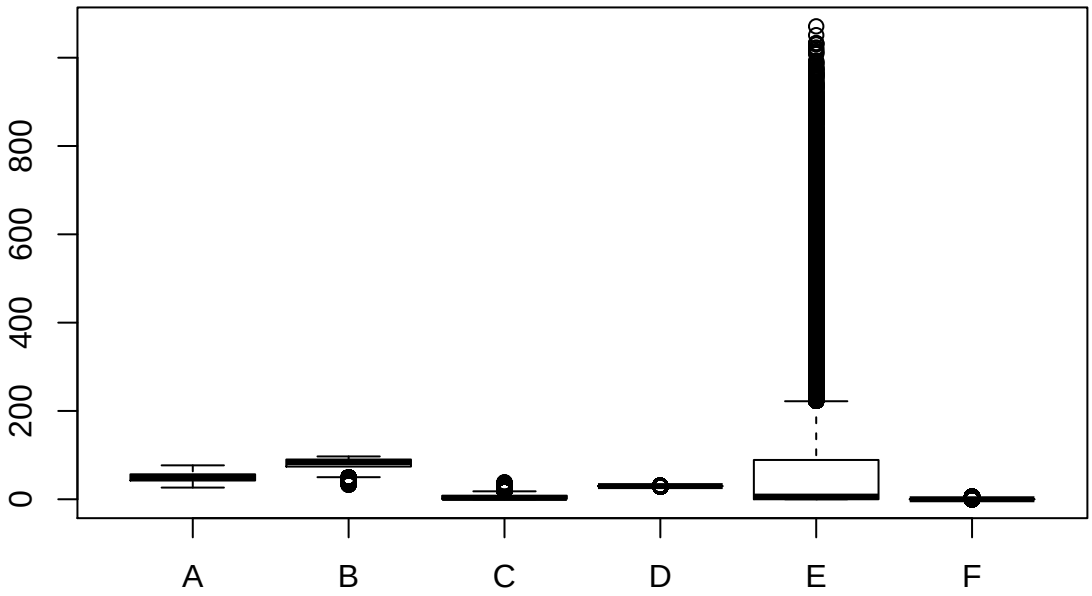
print summary statistics for detrended data:

```
for(acol in names(dat1_detrended)){
    x <- dat1_detrended[[acol]]
    cat("Detrended data, column ", acol, ", mean=", mean(x), ", sd=", sd(x), "\n", sep='')
}
```

```
## Detrended data, column A, mean=-1.025213e-05, sd=0.2437584
## Detrended data, column B, mean=0.0002468104, sd=1.291171
## Detrended data, column C, mean=0, sd=2.358147
## Detrended data, column D, mean=-2.018302e-06, sd=0.00484357
## Detrended data, column E, mean=0, sd=57.02373
## Detrended data, column F, mean=-9.7284e-21, sd=0.1546259
```

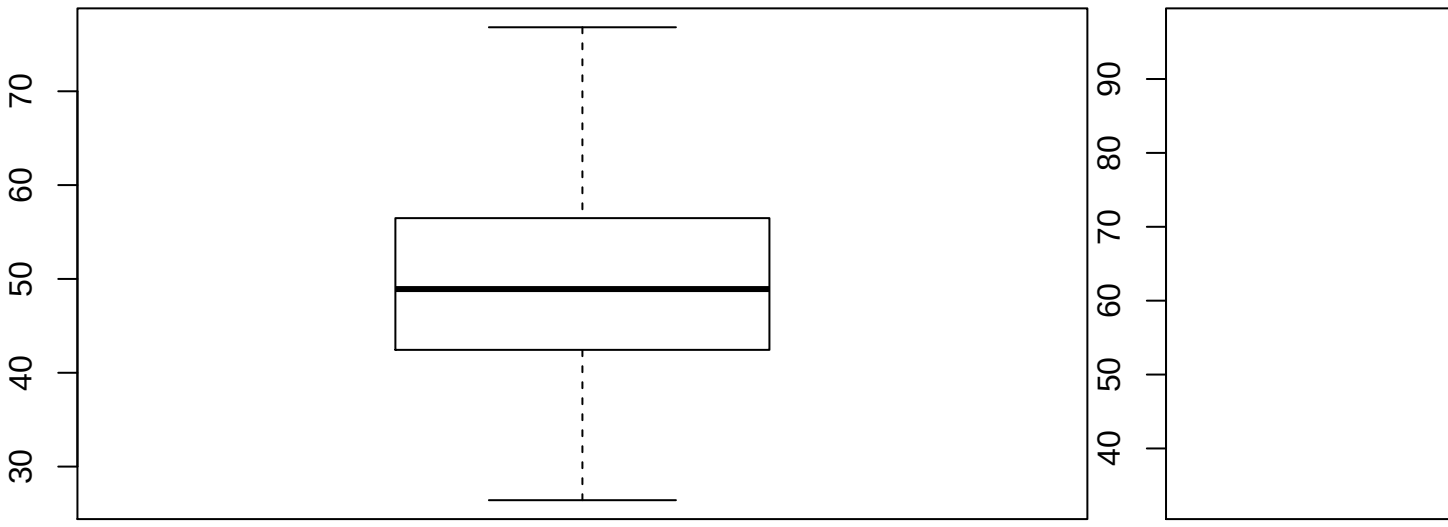
Boxplot of all of original data's columns in one page for comparison Note: a boxplot shows the minimum and maximum values in the whiskers The interquartile range (between lower and upper quartiles) is the rectangle's width. The median is the indicator line inside the rectangle.

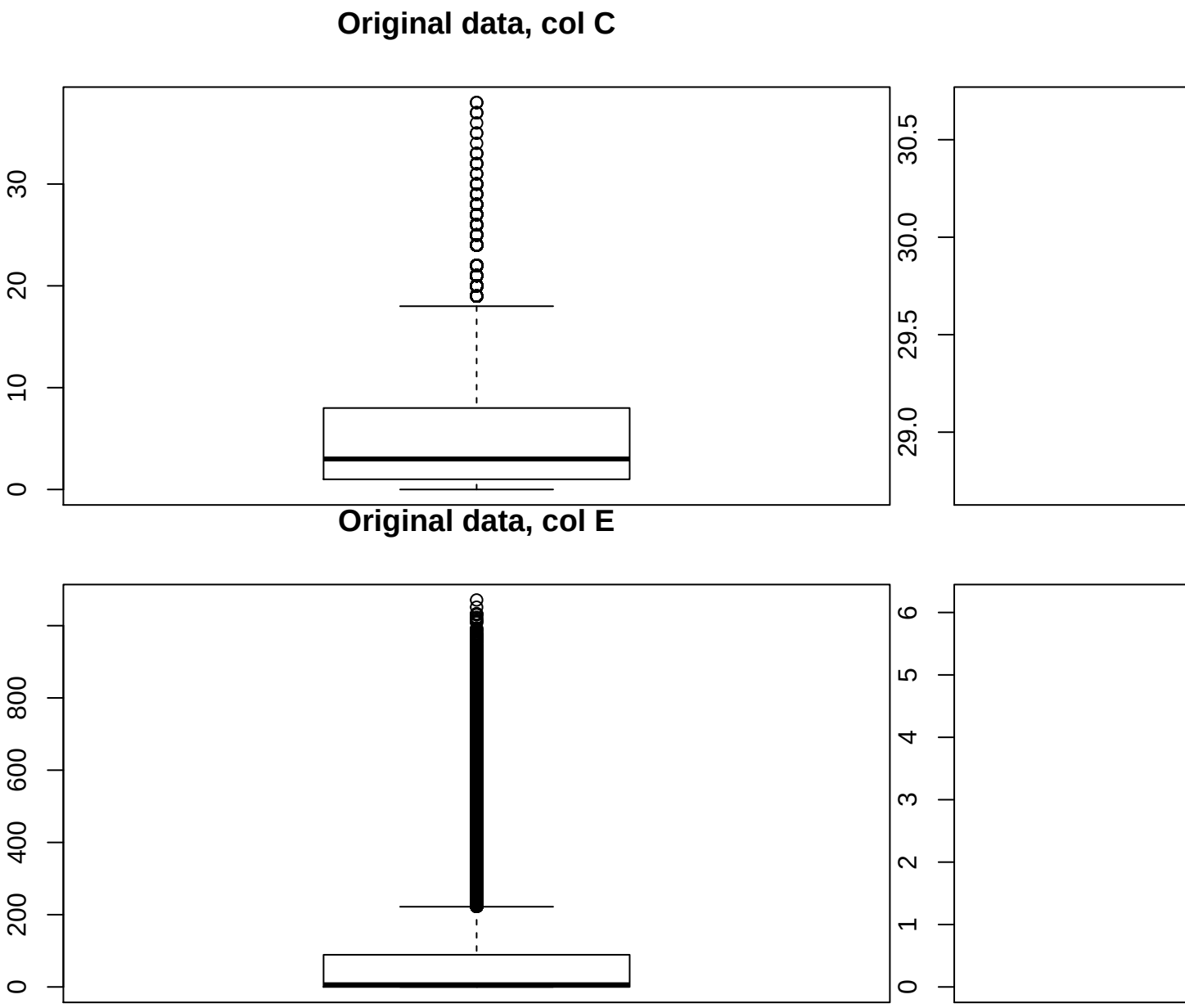
```
boxplot(
    dat1_noid,
    main=paste0('Original data boxplot', sep='')
)
```



boxplot of original data per column:

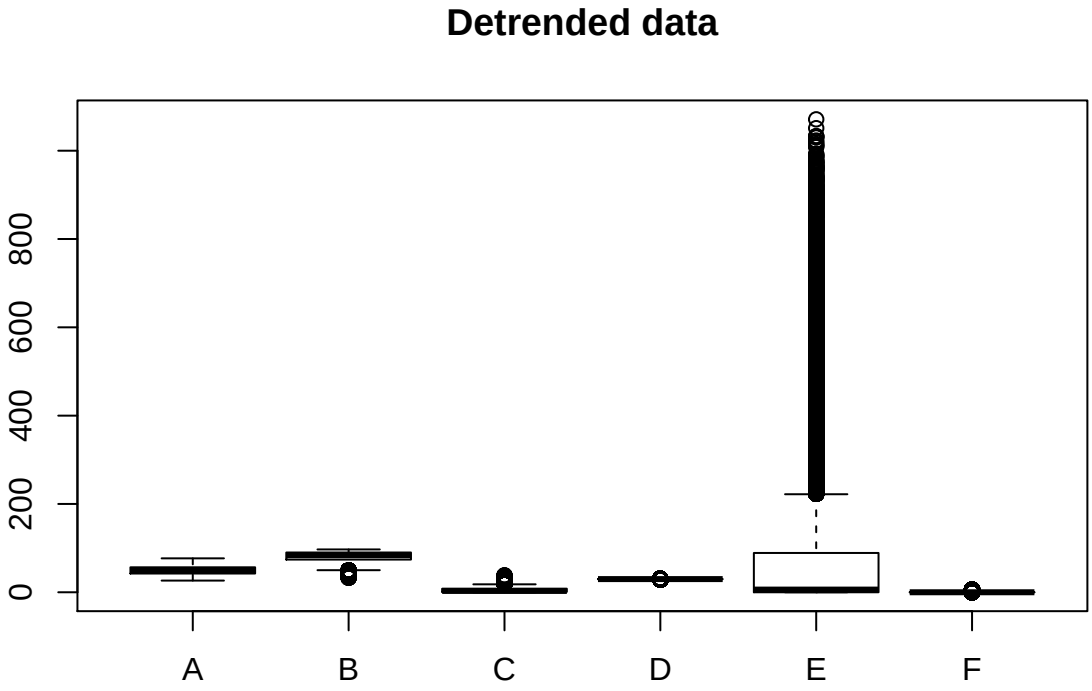
```
for(acol in names(dat1_noid)){
    x <- dat1_noid[[acol]]
    boxplot(
        x,
        main=paste0('Original data, col ', acol, sep='')
    )
}
```





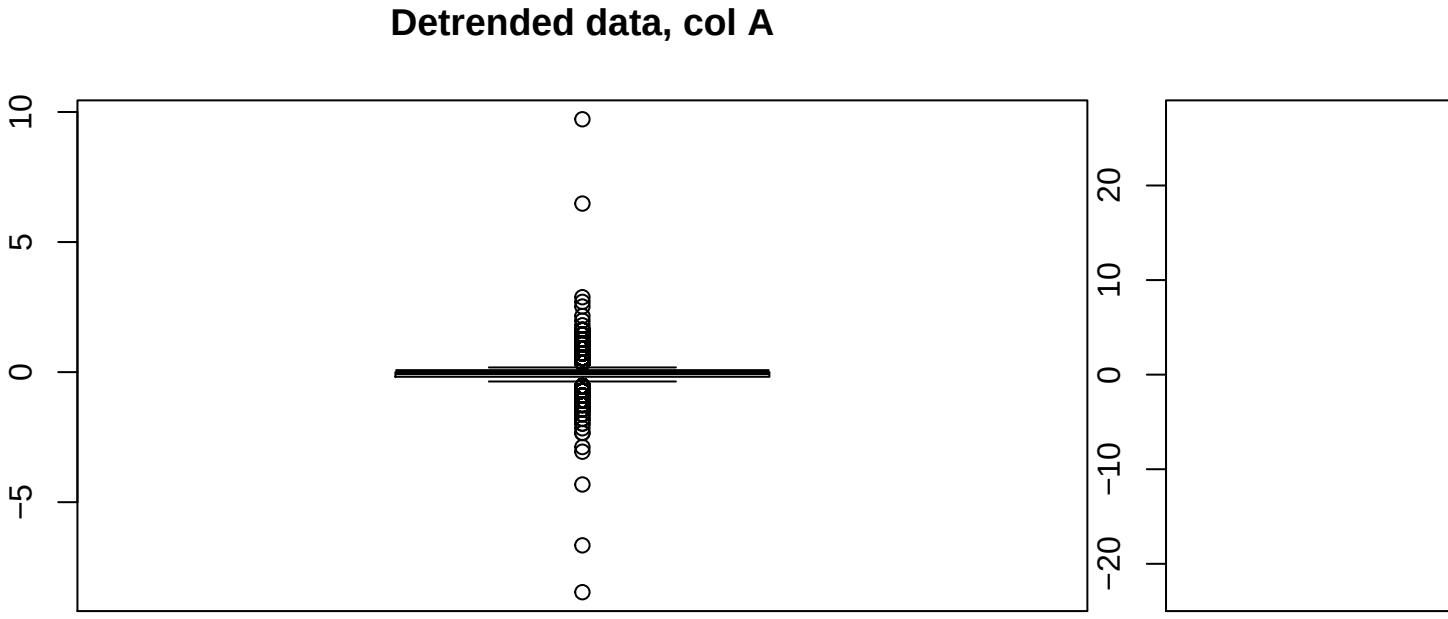
boxplot of all of detrended data's columns in one page for comparison

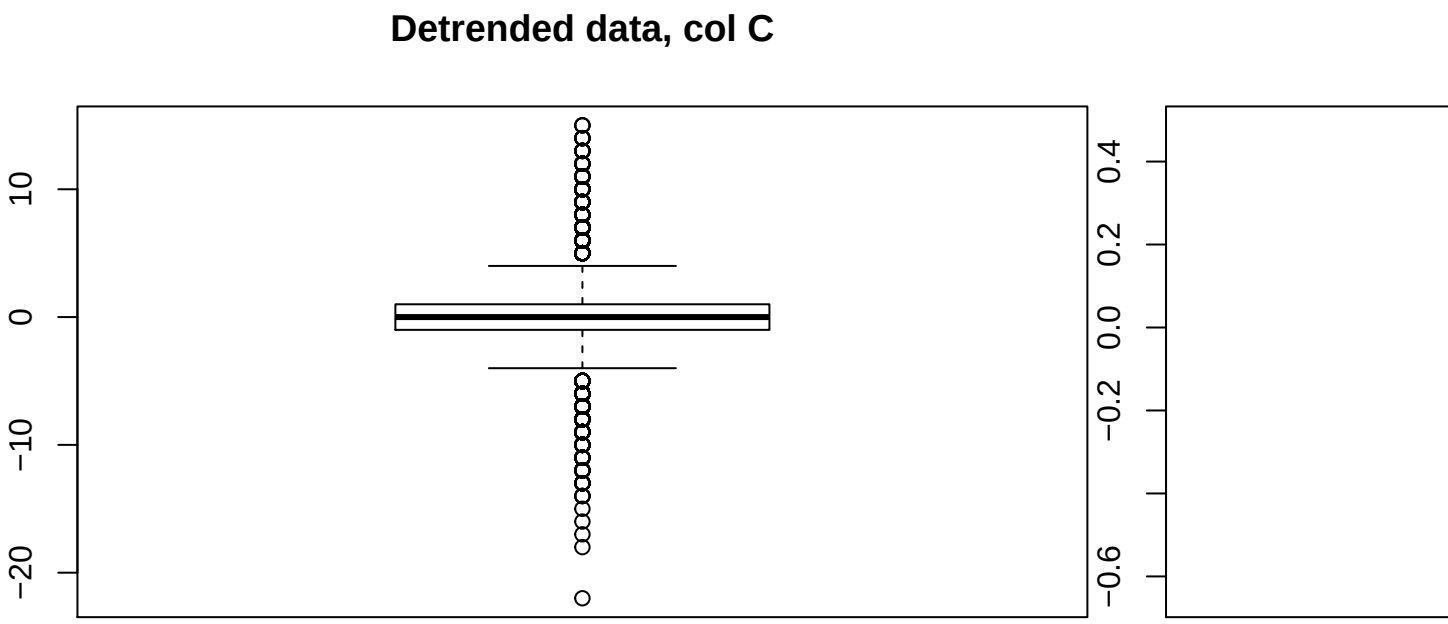
```
boxplot(
    dat1_noid,
    main=paste0('Detrended data', sep='')
)
```

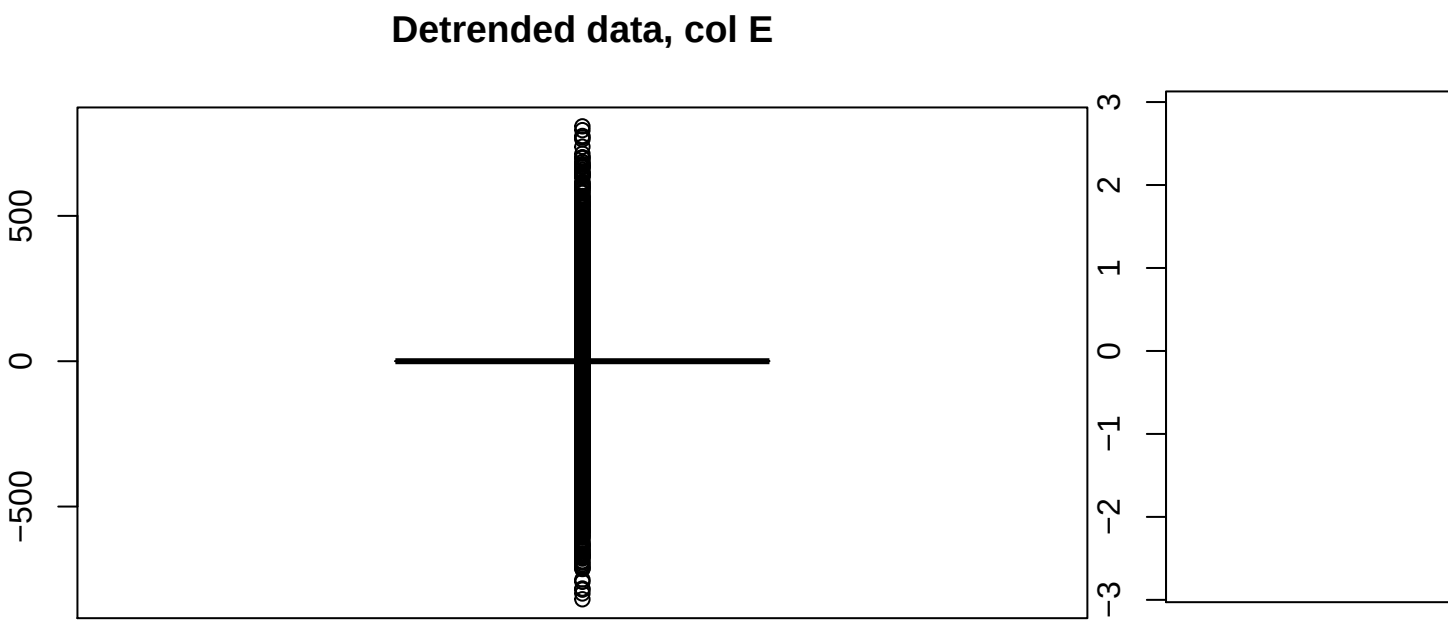


boxplot of detrended data per column:

```
for(acol in names(dat1_noid)){
    x <- dat1_detrended[[acol]]
    boxplot(
        x,
        main=paste0('Detrended data, col ', acol, sep='')
    )
}
```







Do a scatter plot of pairs of variables with a regression line added for the original data

```
library(ggpubr)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: magrittr
```

```
adf <- list2dataframe(dat1_noid)
cnames <- names(adf)
Ncnames <- length(cnames)
for(i in 1:Ncnames){
    acol1 = cnames[i]
    for(j in (i+1):Ncnames){
        acol2 = cnames[j]
        ggscatter(
```

```
            adf,
            x=acol1,
            y=acol2,
            add="reg.line",
            conf.int=TRUE,
            cor.coef=TRUE,
            cor.method = "spearman",
            title=paste0('Scatter plot of original data, columns ', acol1, ' and ', acol2,sep='')
        )
    }
}
```

Do a scatter plot of pairs of variables with a regression line added for the detrended data

```
library(ggpubr)
adf <- list2dataframe(dat1_detrended)
cnames <- names(adf)
Ncnames <- length(cnames)
for(i in 1:Ncnames){
    acol1 = cnames[i]
    for(j in (i+1):Ncnames){
        if( j > Ncnames ){ next }
        acol2 = cnames[j]
        ggscatter(
            adf,
            x=acol1,
            y=acol2,
            add="reg.line",
            conf.int=TRUE,
            cor.coef=TRUE,
            cor.method = "spearman",
            title=paste0('Scatter plot of detrended data, columns ', acol1, ' and ', acol2,sep='')
        )
    }
}
```

Print and plot (Spearman) correlation matrix of original data (pairwise). Correlation tells us whether two variables move together in the same or opposite direction (correlations of +1 and -1) or are their movement is not related at all (correlation of 0). Correlation is related to covariance in that it is essentially covariance normalised over the product of the standard deviations of the two variables.

```
cnames <- names(dat1_noid)
Ncnames <- length(cnames)
amatr <- matrix(unlist(dat1_noid), ncol=Ncnames, byrow=T)
colnames(amatr) <- cnames
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
cor_matrix <- cor(amatr, method='spearman')
cor_pvalues <- cor.mtest(amatr)
```

the correlation matrix of original columns set:

```
print("correlation matrix of original data:")
```

```
## [1] "correlation matrix of original data:"
```

```
print(cor_matrix)
```

```
##           A         B         C         D         E         F
## A 1.0000000 0.9953697 0.9926465 0.9903925 0.9882165 0.9860988
## B 0.9953697 1.0000000 0.9952715 0.9925822 0.9904287 0.9881424
## C 0.9926465 0.9952715 1.0000000 0.9955148 0.9928157 0.9904968
## D 0.9903925 0.9925822 0.9955148 1.0000000 0.9952246 0.9925382
## E 0.9882165 0.9904287 0.9928157 0.9952246 1.0000000 0.9953121
## F 0.9860988 0.9881424 0.9904968 0.9925382 0.9953121 1.0000000
```

statistical significance of the correlation matrix

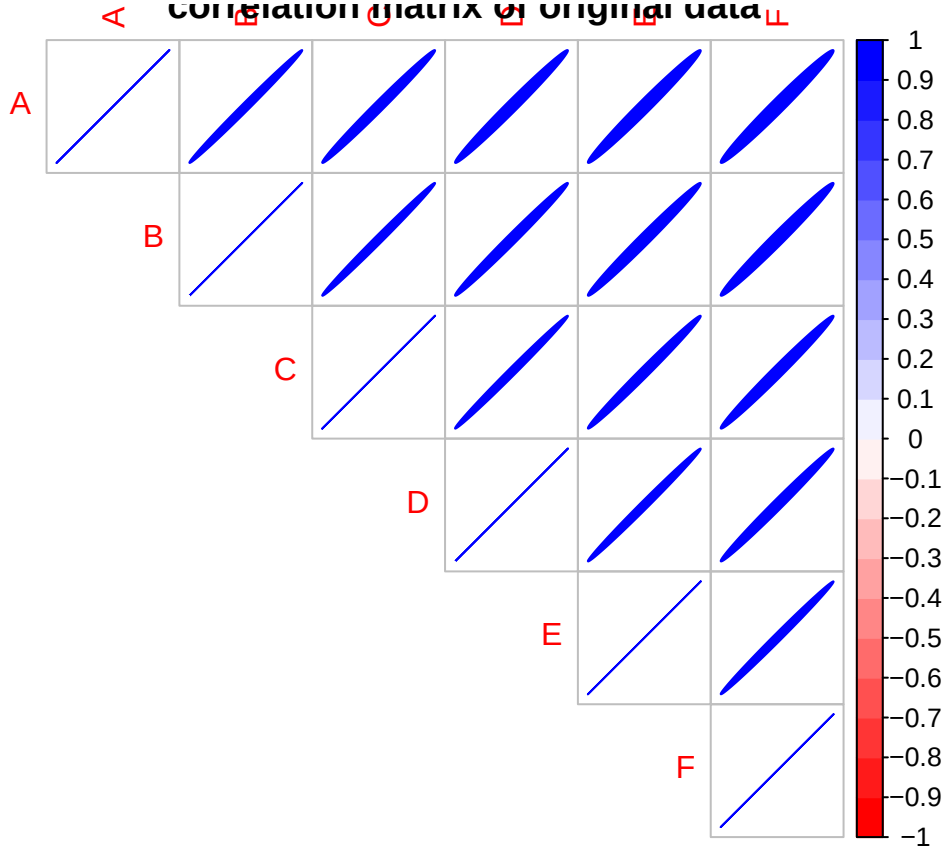
```
print("and its statistical significance:")
```

```
## [1] "and its statistical significance:"
```

```
colnames(cor_pvalues$p) <- cnames
rownames(cor_pvalues$p) <- cnames
print(cor_pvalues$p)
```

```
##   A B C D E F
## A 0 0 0 0 0 0
## B 0 0 0 0 0 0
## C 0 0 0 0 0 0
## D 0 0 0 0 0 0
## E 0 0 0 0 0 0
## F 0 0 0 0 0 0
```

```
acolor <- colorRampPalette(c("red", "white", "blue"))(20)
corrplot(
    cor_matrix,
    p.mat=cor_pvalues$p,
    method="ellipse",
    order="original",
    col=acolor,
    type='upper',
    sig.level=0.0,
    insig="p-value",
    title='correlation matrix of original data'
)
```



plot correlation matrix of detrended data (pairwise)

```
cnames <- names(dat1_detrended)
Ncnames <- length(cnames)
amatr <- matrix(unlist(dat1_detrended), ncol=Ncnames, byrow=T)
colnames(amatr) <- cnames
library(corrplot)
cor_matrix <- cor(amatr, method='spearman')
cor_pvalues <- cor.mtest(amatr)
```

the correlation matrix of detrended columns set:

```
print("correlation matrix of original data:")
```

```
## [1] "correlation matrix of original data:"
```

```
print(cor_matrix)
```

```
##            A          B          C          D          E          F
## A 1.00000000 0.01846806 0.08630014 0.10118207 0.07398528 0.06682168
## B 0.01846806 1.00000000 0.01899633 0.09033755 0.10640717 0.08220713
## C 0.08630014 0.01899633 1.00000000 0.02234015 0.09594644 0.10275620
## D 0.10118207 0.09033755 0.02234015 1.00000000 0.02221574 0.08849447
## E 0.07398528 0.10640717 0.09594644 0.02221574 1.00000000 0.01224202
## F 0.06682168 0.08220713 0.10275620 0.08849447 0.01224202 1.00000000
```

statistical significance of the correlation matrix

```
colnames(cor_pvalues$p) <- cnames
rownames(cor_pvalues$p) <- cnames
```

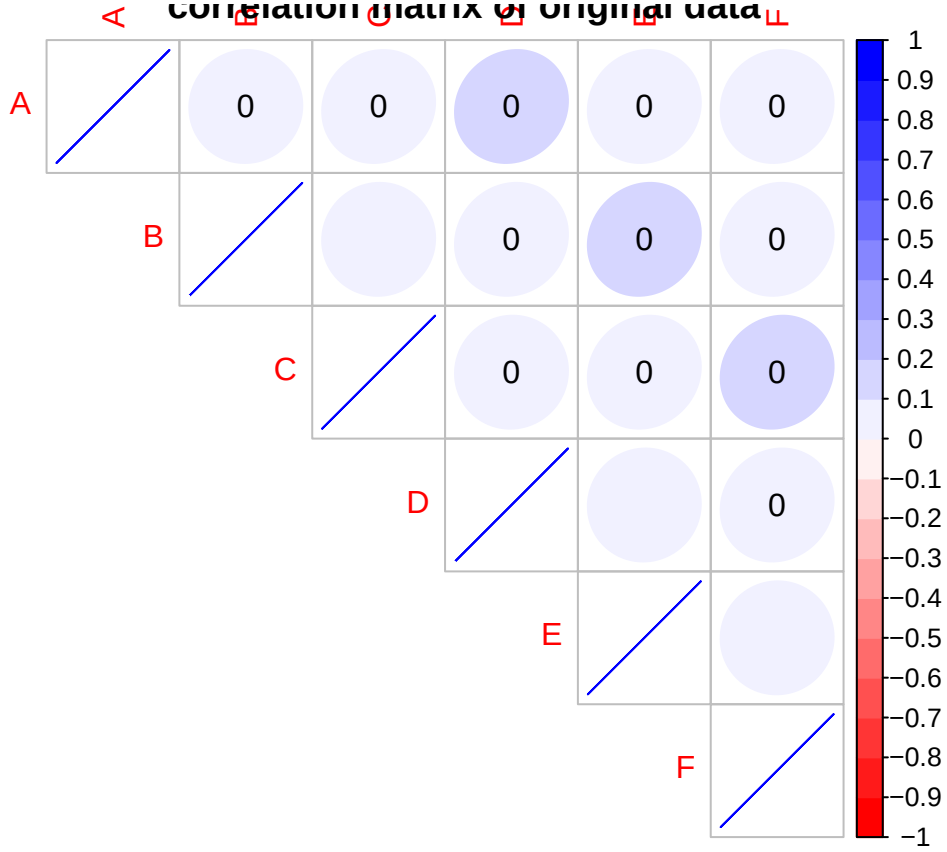
```r
print("and its statistical significance:")
```

```
## [1] "and its statistical significance:"
```

```r
print(cor_pvalues$p)
```

```
##                A             B             C             D             E
## A  0.000000e+00 1.662902e-283  4.154338e-79  1.760060e-34  6.462174e-94
## B 1.662902e-283  0.000000e+00  0.000000e+00 3.566732e-226  6.802040e-04
## C  4.154338e-79  0.000000e+00  0.000000e+00 4.933289e-299 1.153638e-198
## D  1.760060e-34 3.566732e-226 4.933289e-299  0.000000e+00  0.000000e+00
## E  6.462174e-94  6.802040e-04 1.153638e-198  0.000000e+00  0.000000e+00
## F  7.410103e-09  1.417765e-45  5.378597e-38  1.399490e-54  0.000000e+00
##              F
## A 7.410103e-09
## B 1.417765e-45
## C 5.378597e-38
## D 1.399490e-54
## E 0.000000e+00
## F 0.000000e+00
```

```r
acolor <- colorRampPalette(c("red", "white", "blue"))(20)
corrplot(
    cor_matrix,
    p.mat=cor_pvalues$p,
    method="ellipse",
    order="original",
    col=acolor,
    type='upper',
    sig.level=0.0,
    insig="p-value",
    title='correlation matrix of original data'
)
```



Correlation summary of original data:

```
library(PerformanceAnalytics)
```

```
## Loading required package: xts
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## Attaching package: 'PerformanceAnalytics'
## The following object is masked from 'package:graphics':
##
##     legend
```

```
chart.Correlation(dat1_noid, histogram=TRUE, pch=19)
```

```
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

```
## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

```
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
```

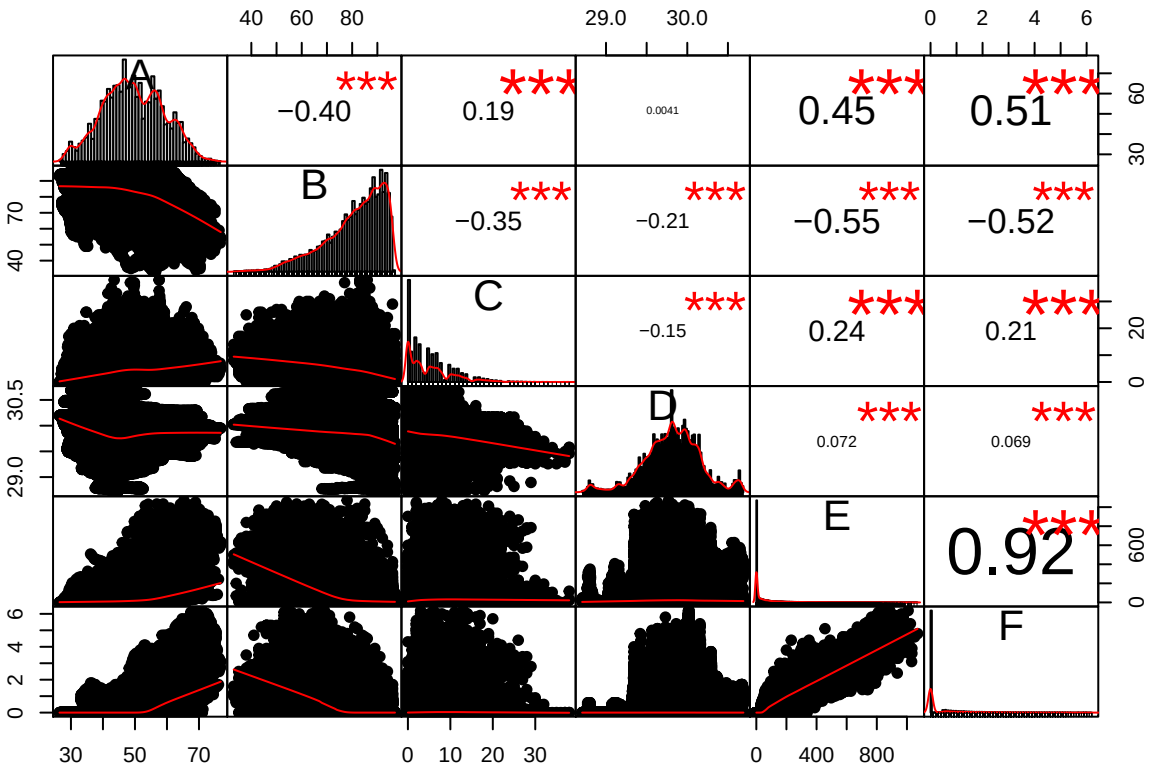
```
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
```

```
## Warning in plot.window(...): "method" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

```
## Warning in title(...): "method" is not a graphical parameter
```



Correlation summary of detrended data:

```
library(PerformanceAnalytics)
chart.Correlation(dat1_noid, histogram=TRUE, pch=19)
```

```
## Warning in plot.window(...): "method" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

```
## Warning in title(...): "method" is not a graphical parameter
```

```
## Warning in plot.window(...): "method" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

```
## Warning in title(...): "method" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
```

```
## Warning in plot.window(...): "method" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

```
## Warning in title(...): "method" is not a graphical parameter
```

```
## Warning in plot.window(...): "method" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

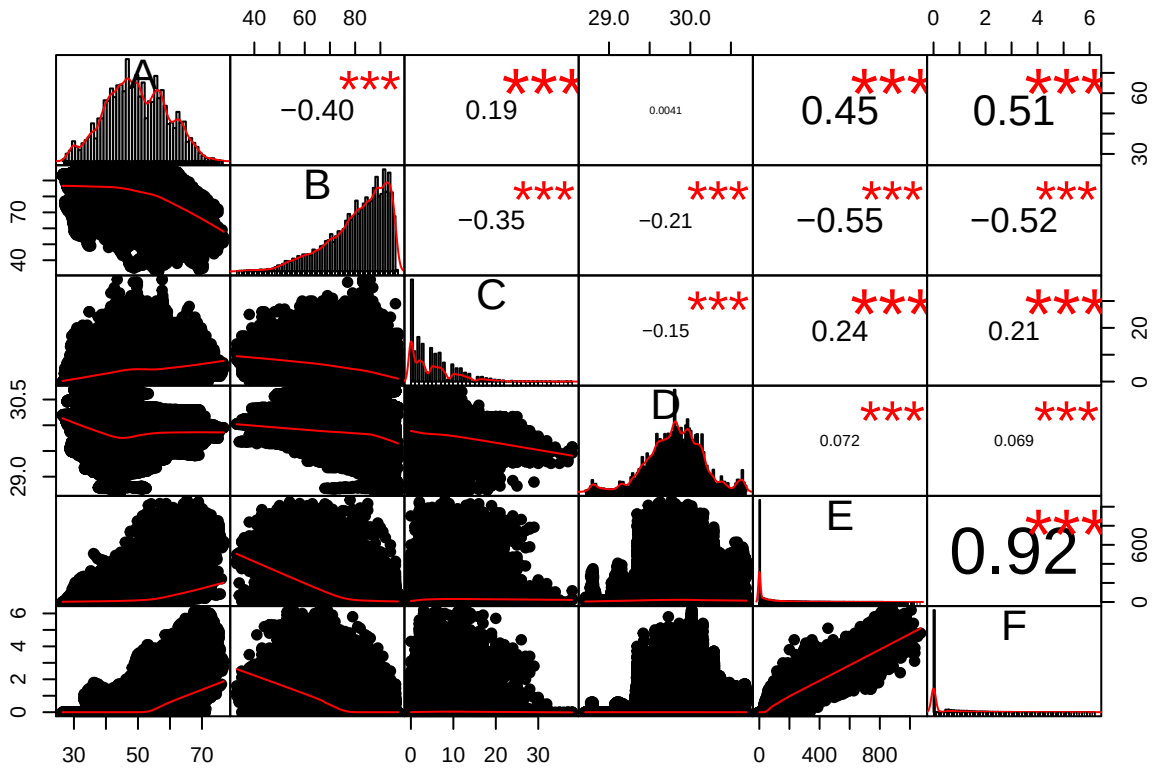
```
## Warning in title(...): "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
```

```
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
## Warning in plot.window(...): "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter
## Warning in title(...): "method" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "method" is
## not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "method" is not a
## graphical parameter

## Warning in plot.window(...): "method" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "method" is not a graphical parameter

## Warning in title(...): "method" is not a graphical parameter
```



plot covariance matrix of original data (pairwise)

```
cnames <- names(dat1_noid)
Ncnames <- length(cnames)
amatr <- matrix(unlist(dat1_noid), ncol=Ncnames, byrow=T)
colnames(amatr) <- cnames
cov_matrix <- cov(amatr)
```

the covariance matrix of original columns set:

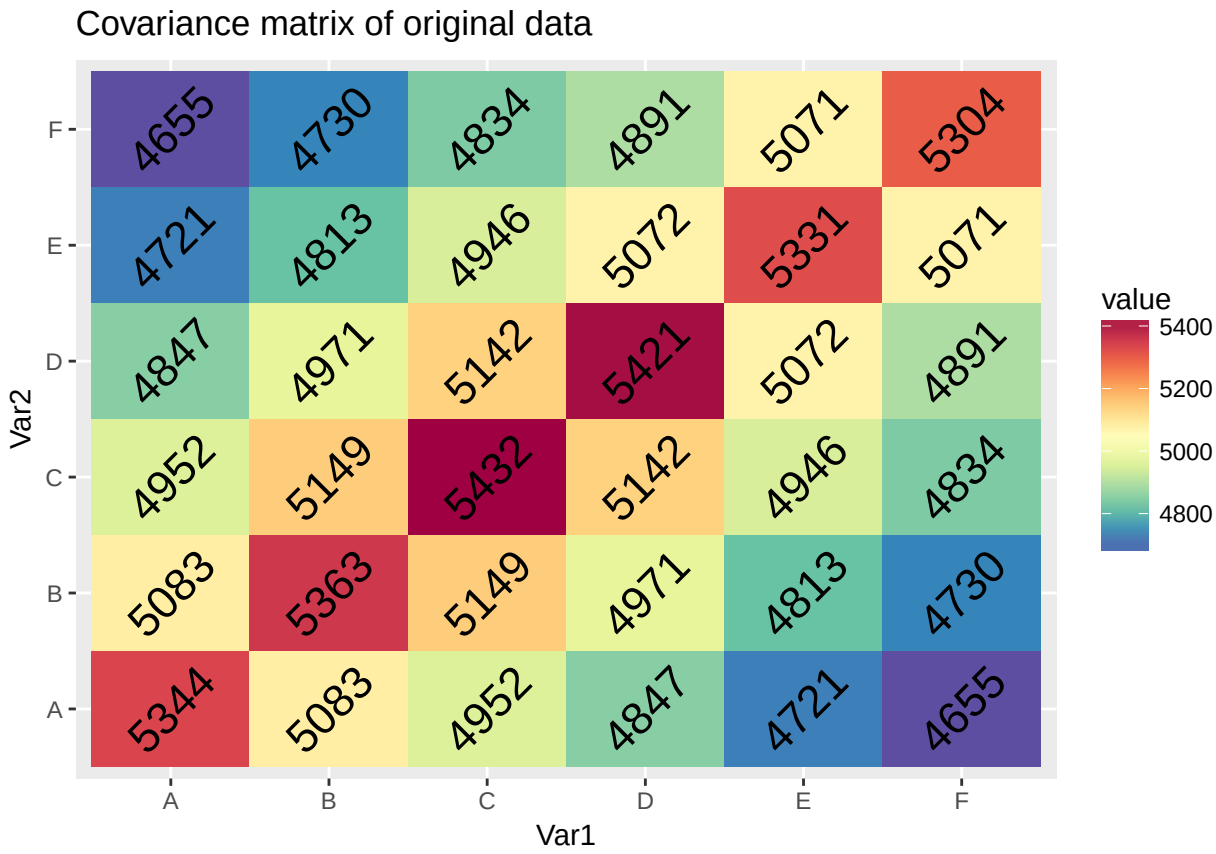
```
print("covariance matrix of original data:")
```

```
## [1] "covariance matrix of original data:"
```

```
print(cov_matrix)
```

```
##          A        B        C        D        E        F
## A 5344.077 5082.750 4952.304 4846.738 4720.588 4654.519
## B 5082.750 5362.658 5149.259 4971.203 4812.842 4729.721
## C 4952.304 5149.259 5432.363 5141.714 4946.124 4834.244
## D 4846.738 4971.203 5141.714 5421.468 5071.708 4890.835
## E 4720.588 4812.842 4946.124 5071.708 5331.006 5071.083
## F 4654.519 4729.721 4834.244 4890.835 5071.083 5304.439
```

```
p <- ggplot_heatmap(
    inp=cov_matrix,
    plot.title='Covariance matrix of original data'
)
print(p)
```



plot covariance matrix of detrended data (pairwise) Covariance is similar to variance. The latter applies to a single variable Covariance applies to two variables and is the sum of the product of each data-point's deviation from the mean

$$\sum_{i=1}^{N} \frac{(x_1 - \bar{(}x_1))(x_2 - \bar{(}x_2))}{N-1}$$

```
cnames <- names(dat1_detrended)
Ncnames <- length(cnames)
amatr <- matrix(unlist(dat1_detrended), ncol=Ncnames, byrow=T)
colnames(amatr) <- cnames
cov_matrix <- cov(amatr)
```

the covariance matrix of detrended columns set:

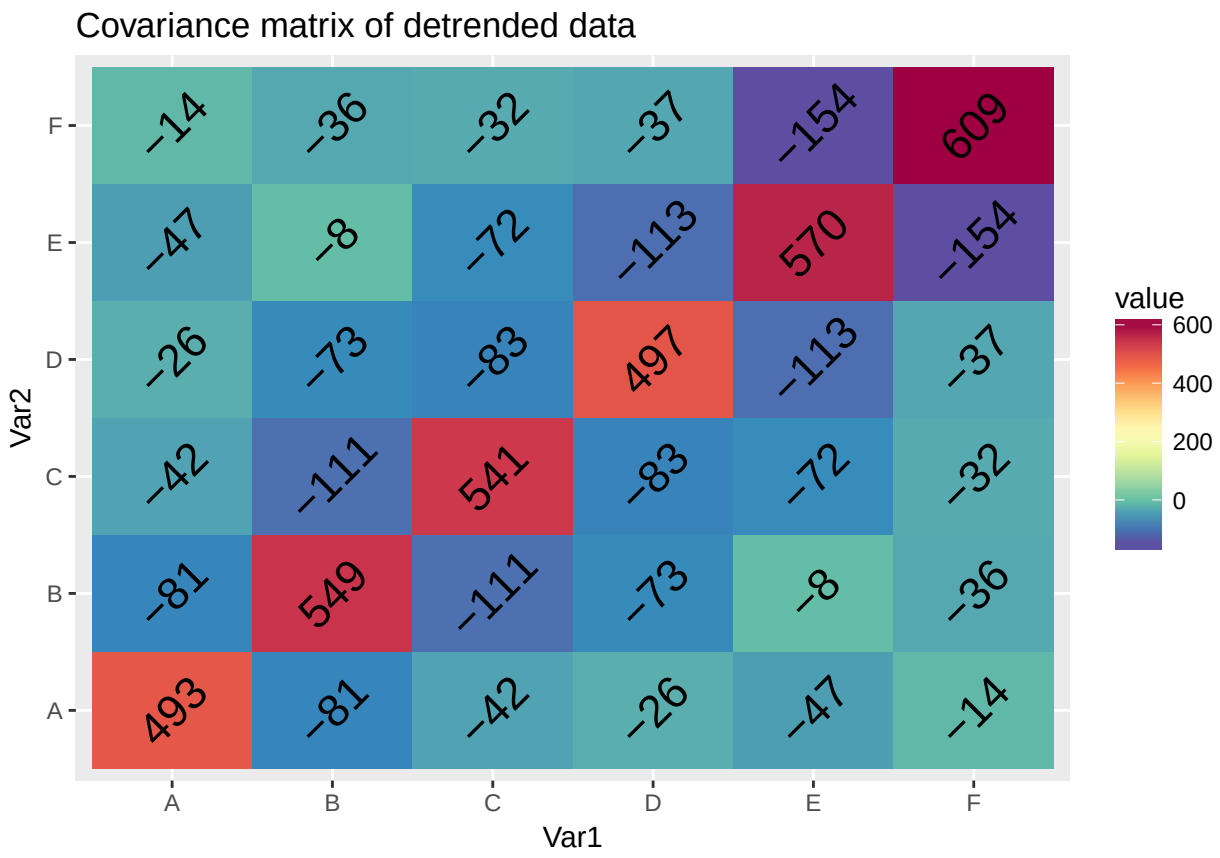
```
print("covariance matrix of detrended data:")
```

```
## [1] "covariance matrix of detrended data:"
```

```
print(cov_matrix)
```

```
##           A           B          C          D           E          F
## A 493.24296  -81.060207  -42.32725  -26.38814  -47.418538  -13.80354
## B -81.06021  548.650378 -111.09997  -72.66176   -8.280808  -35.65198
## C -42.32725 -111.099974  541.27997  -82.95156  -72.493255  -32.21031
## D -26.38814  -72.661759  -82.95156  496.51207 -112.577544  -37.23274
## E -47.41854   -8.280808  -72.49326 -112.57754  570.366279 -154.18776
## F -13.80354  -35.651978  -32.21031  -37.23274 -154.187763  608.91611
```

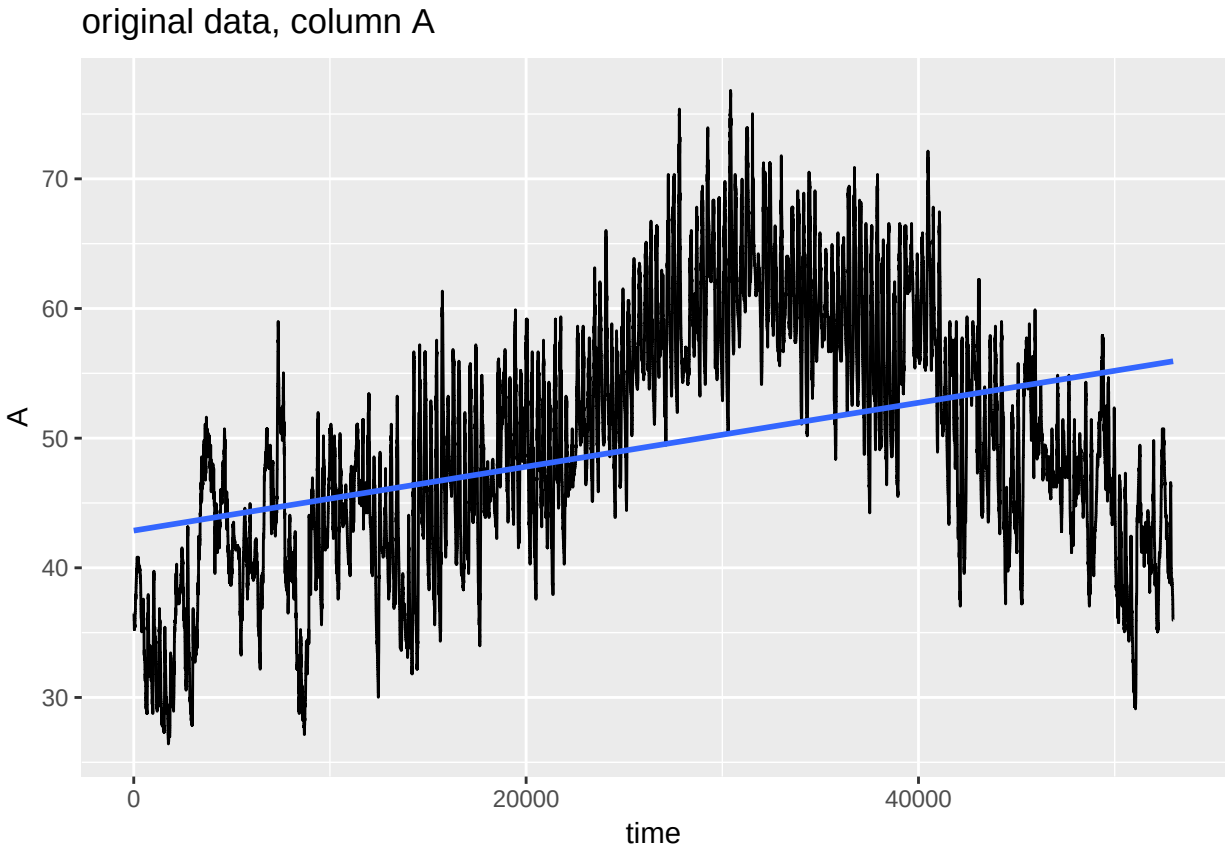
```
p <- ggplot_heatmap(
    inp=cov_matrix,
    plot.title='Covariance matrix of detrended data'
)
print(p)
```



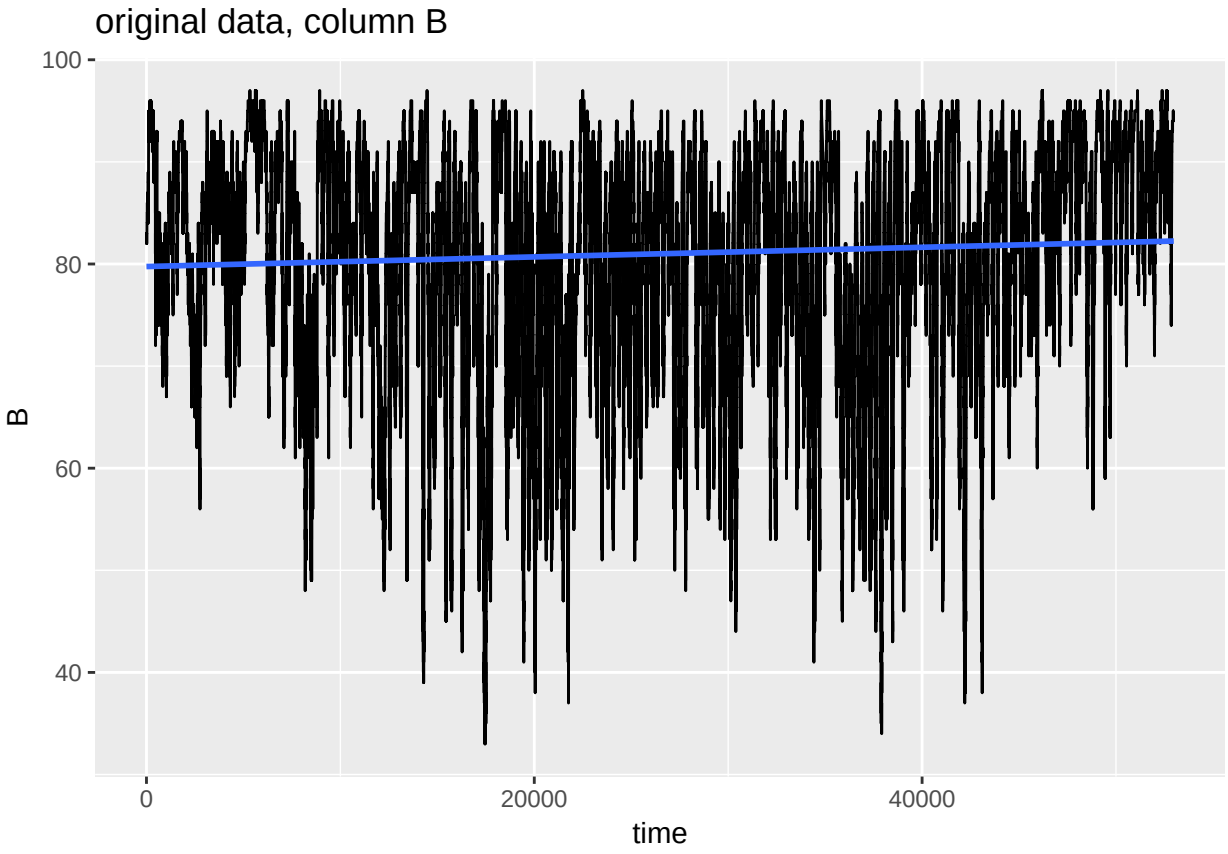
plot the original data

```
library(ggplot2)
library(reshape2)
adat = dat1_noid; adat[['id']] = dat1_id[['id']]
for(acol in names(dat1_noid)){
    atitle <- paste0("original data, column ", acol)
    cat("plot of ", atitle, "\n", sep='')
    print(
        ggplot(adat, aes_string(x='id',y=acol))+
        geom_line()+
        ggtitle(atitle)+
        xlab('time')+
        geom_smooth(method='gam')
    )
}
```

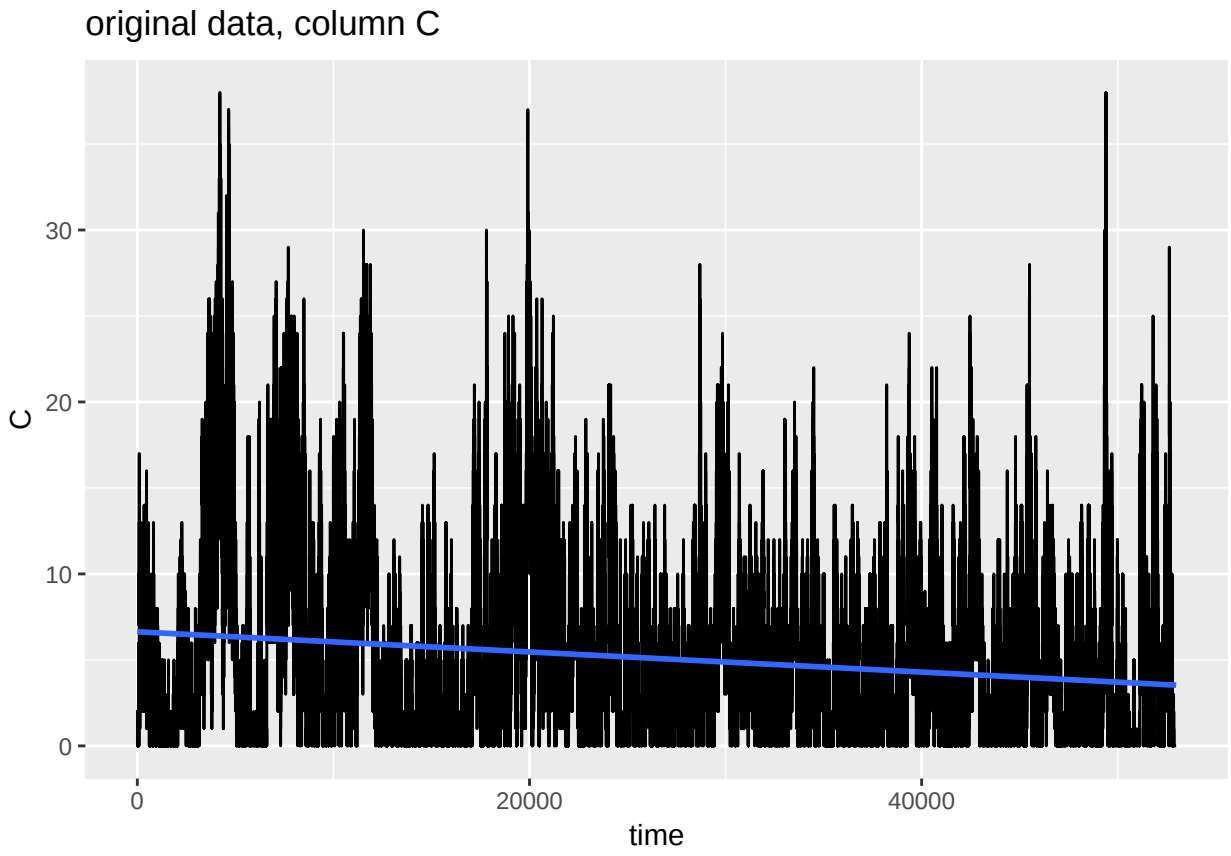
```
## plot of original data, column A
```



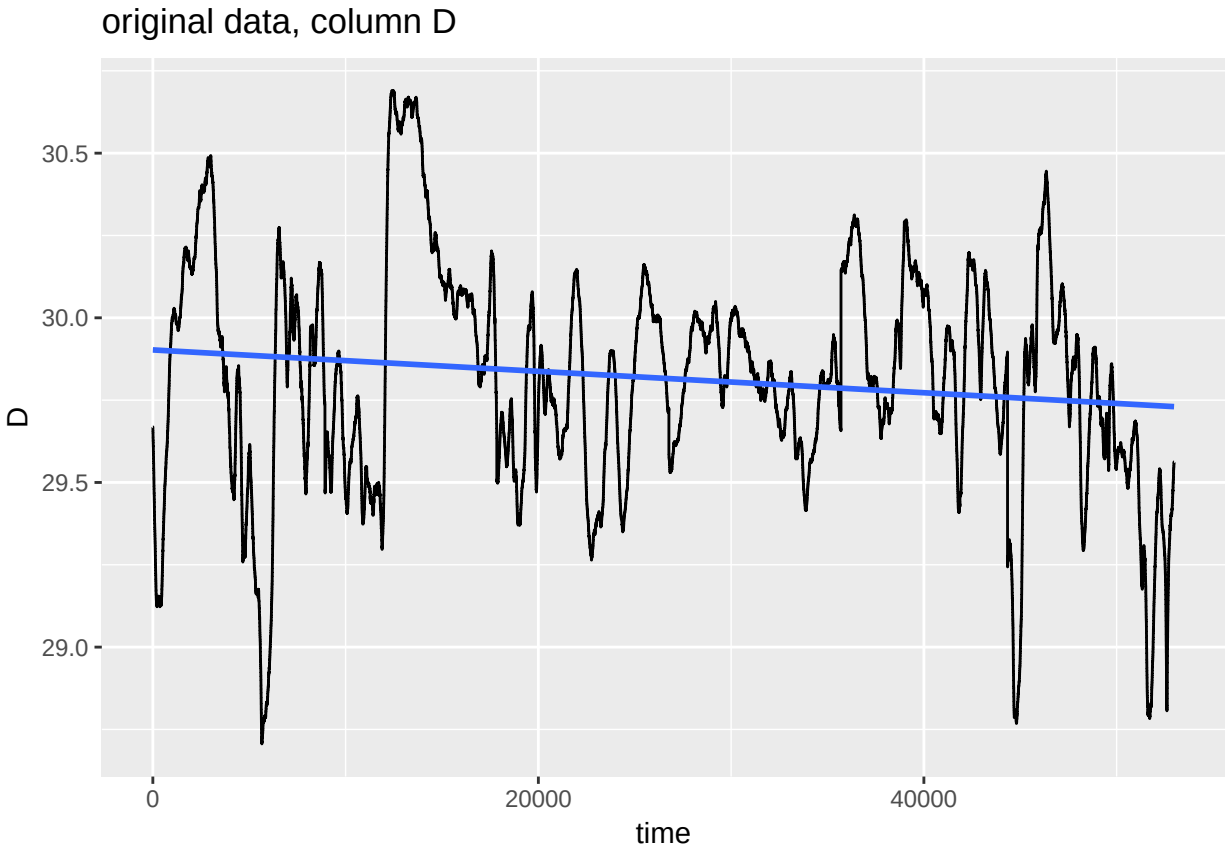

```
## plot of original data, column B
```



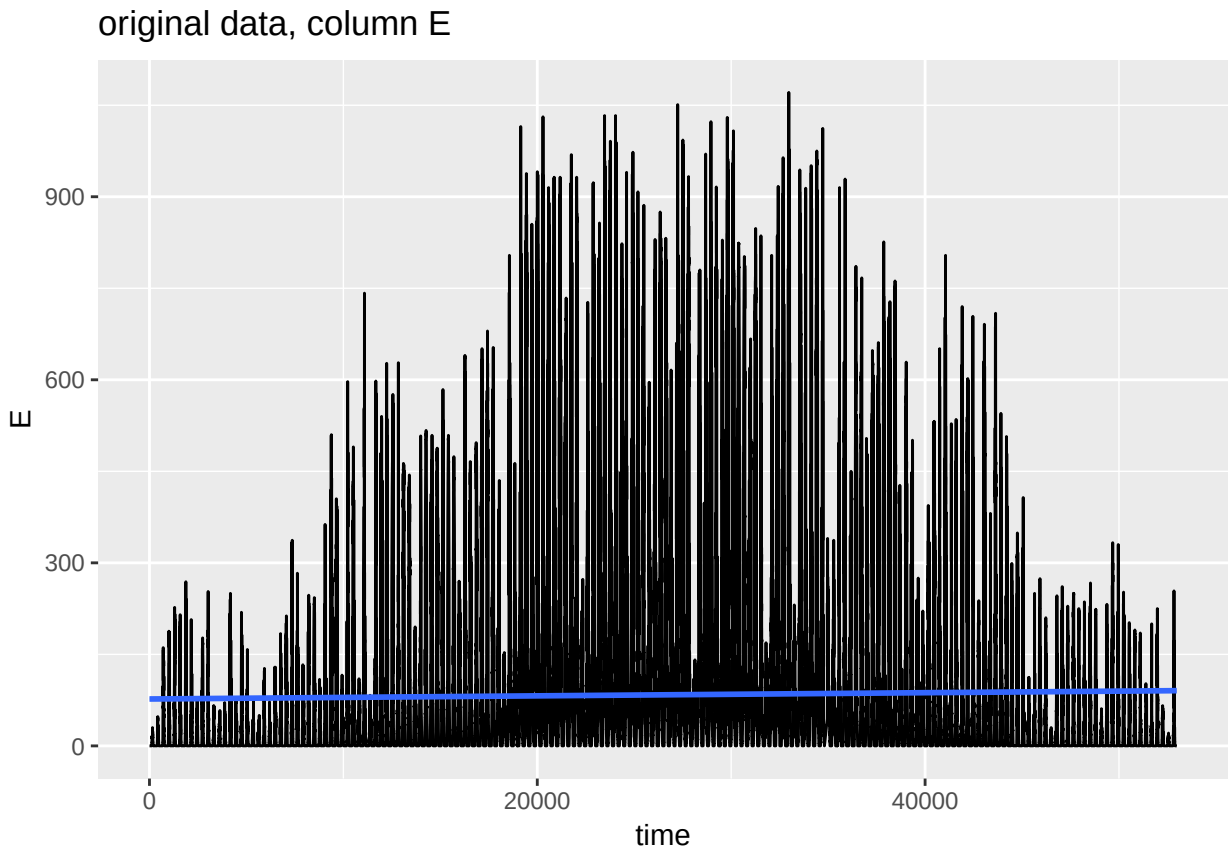

```
## plot of original data, column C
```



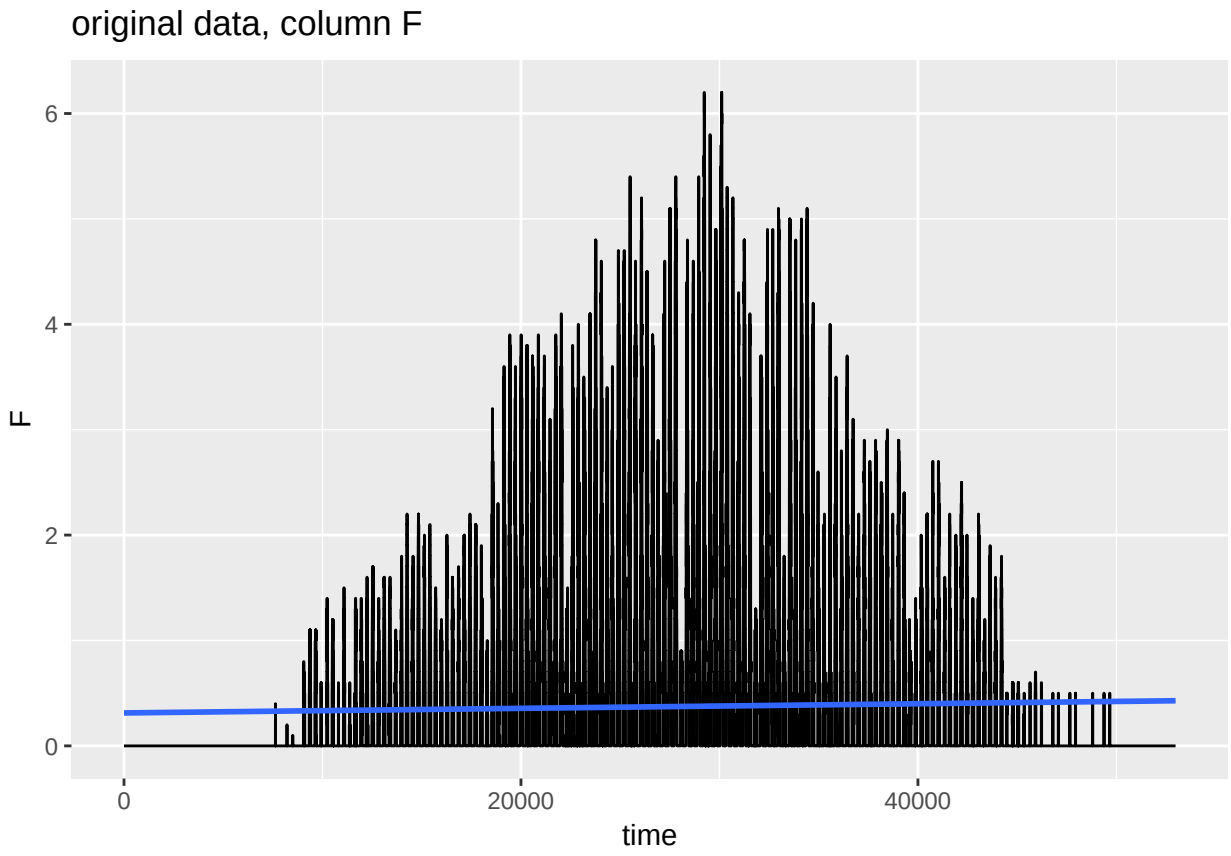

```
## plot of original data, column D
```




```
## plot of original data, column E
```



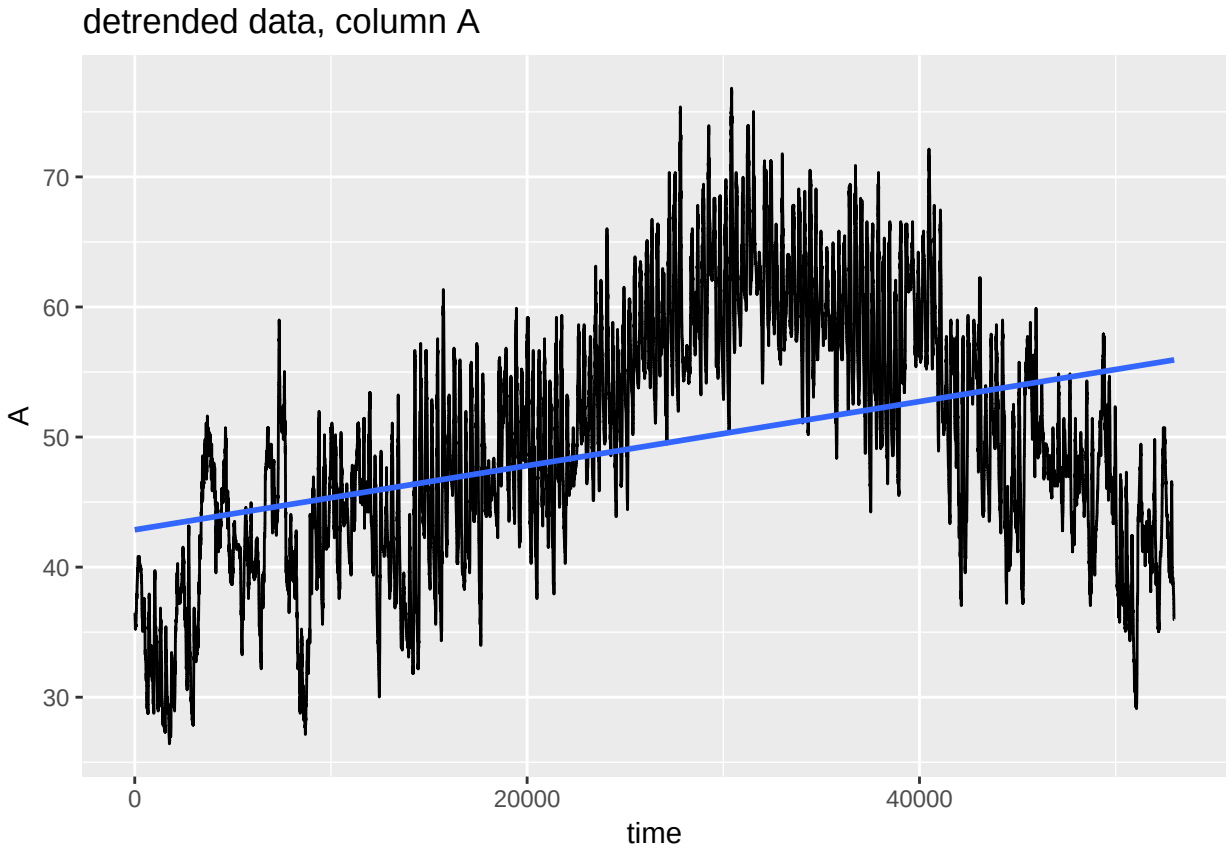

```
## plot of original data, column F
```



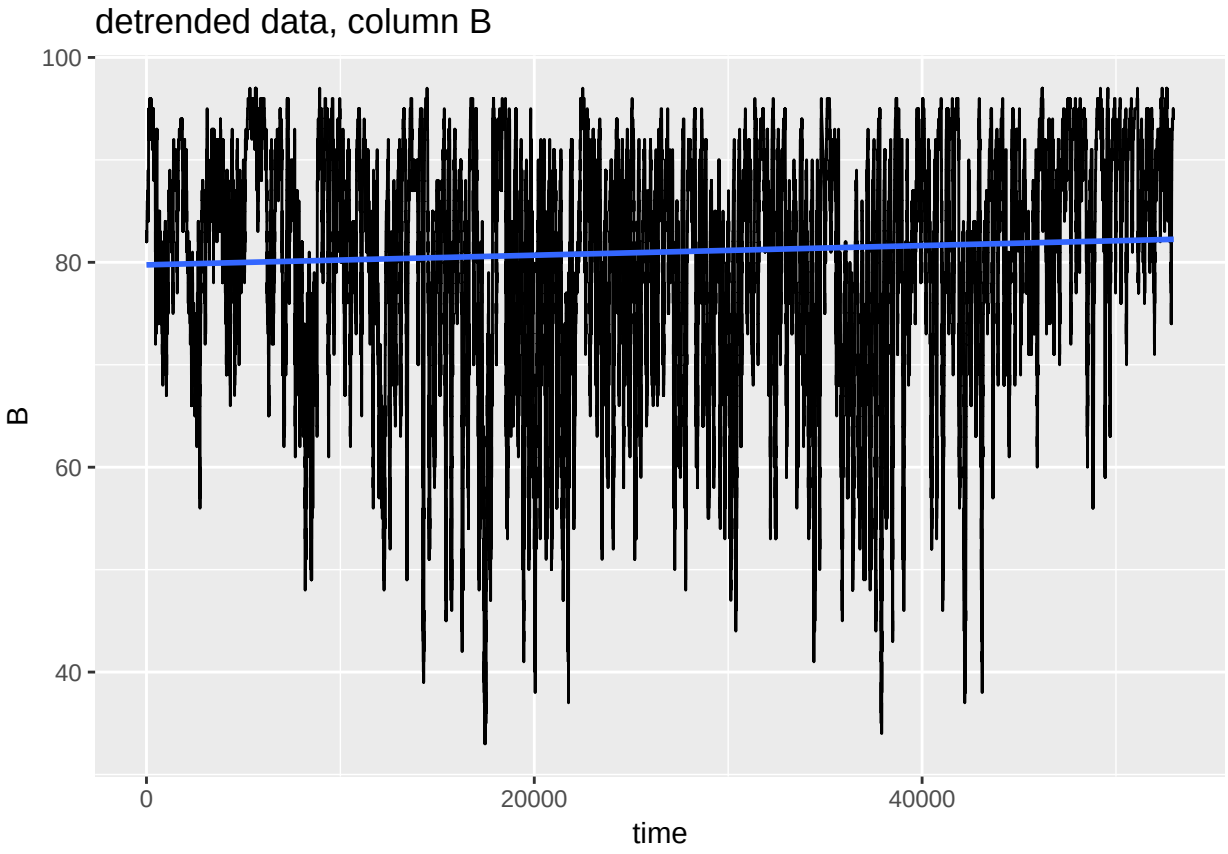
plot detrended data

```
adf <- data.frame(dat1_detrended); adf$id <- dat1_detrended_id[['id']]
for(acol in names(dat1_detrended)){
    atitle <- paste0("detrended data, column ", acol)
    cat("plot of ", atitle, "\n", sep='')
    print(
        ggplot(dat1, aes_string(x='id',y=acol))+
        geom_line()+
        ggtitle(atitle)+
        xlab('time')+
        geom_smooth(method='gam')
    )
}
```

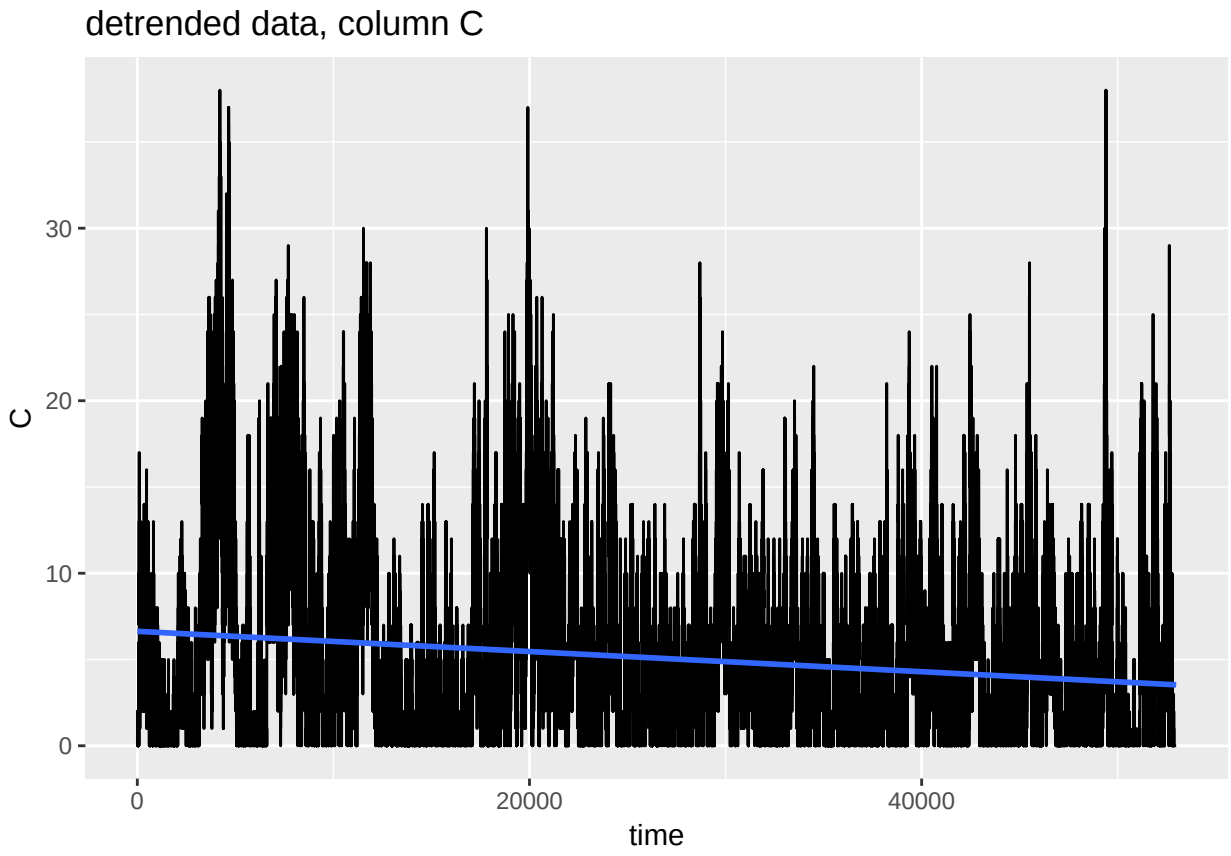
```
## plot of detrended data, column A
```



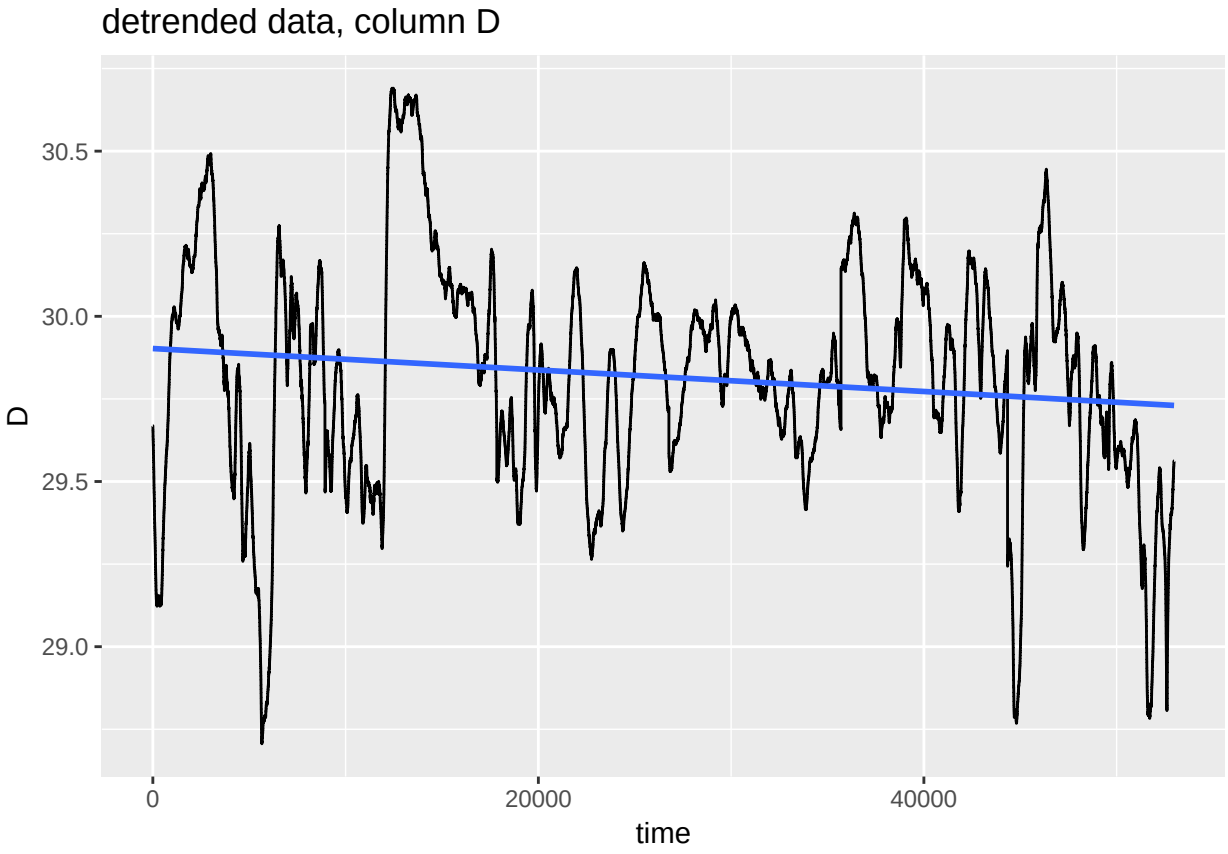

```
## plot of detrended data, column B
```



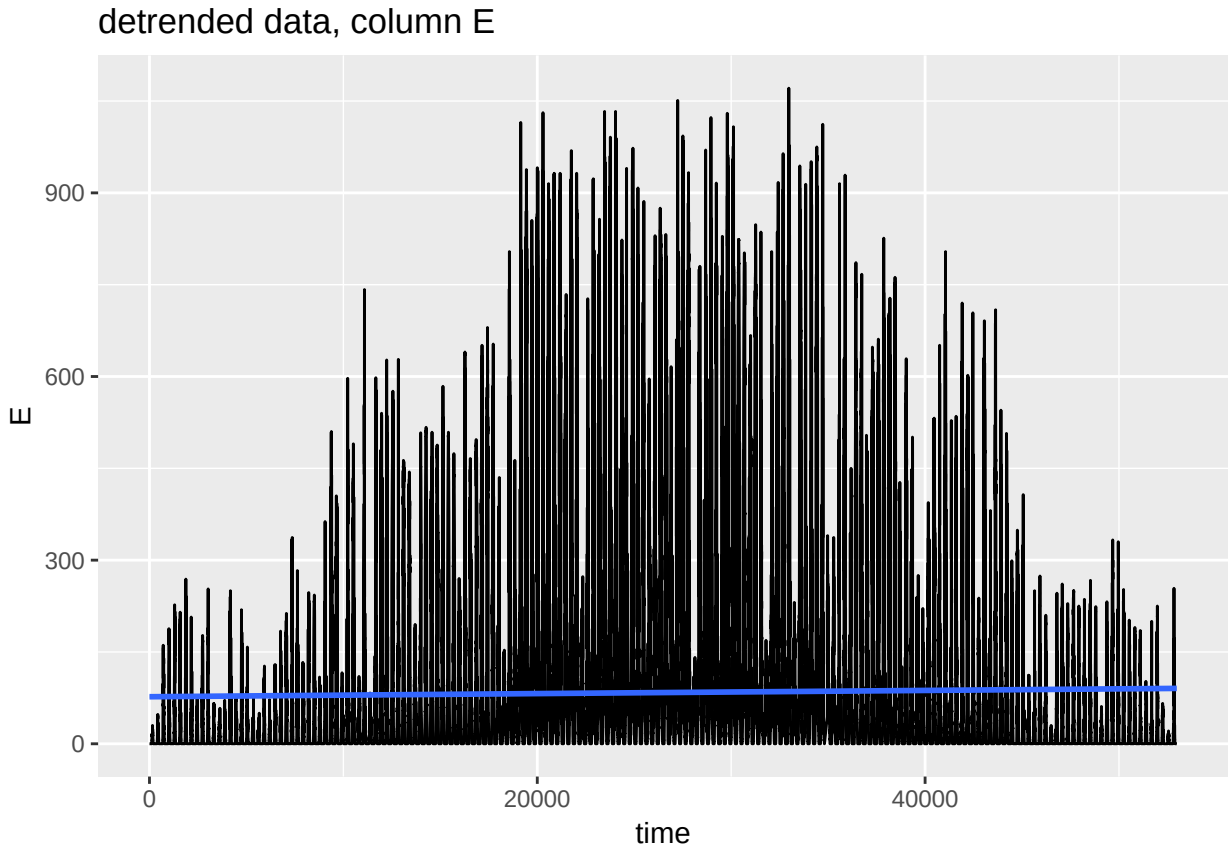

```
## plot of detrended data, column C
```



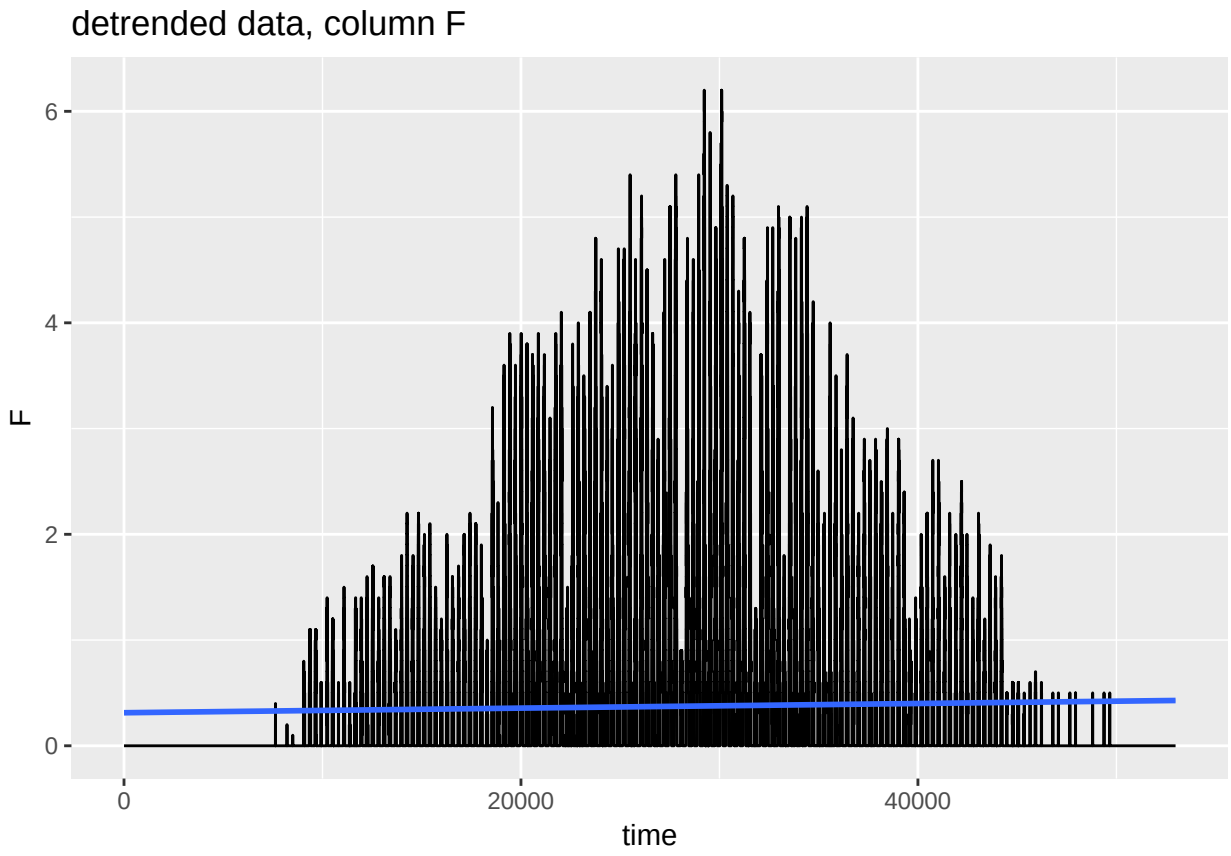

```
## plot of detrended data, column D
```




```
## plot of detrended data, column E
```




```
## plot of detrended data, column F
```



discretise the data to five levels dat1_discrete <- discretise(dat1_noid, levels=5) plot discrete data

```
if( FALSE ){
for(acol in names(dat1_discrete)){
    atitle <- paste0('Original data, discretised to 5 levels')
    adf <- data.frame(data=dat1_discrete[[acol]]$discretised)
    adf$id <- seq.int(nrow(adf))
    p <- ggplot(
        adf,
        aes_string(x='id',y='data')
    )+geom_line()+ggtitle(atitle)
    print(p)
}
}
```

discretise the detrended data dat1_detrended_discrete <- discretise(dat1_detrended, levels=5) plot detrended discrete data

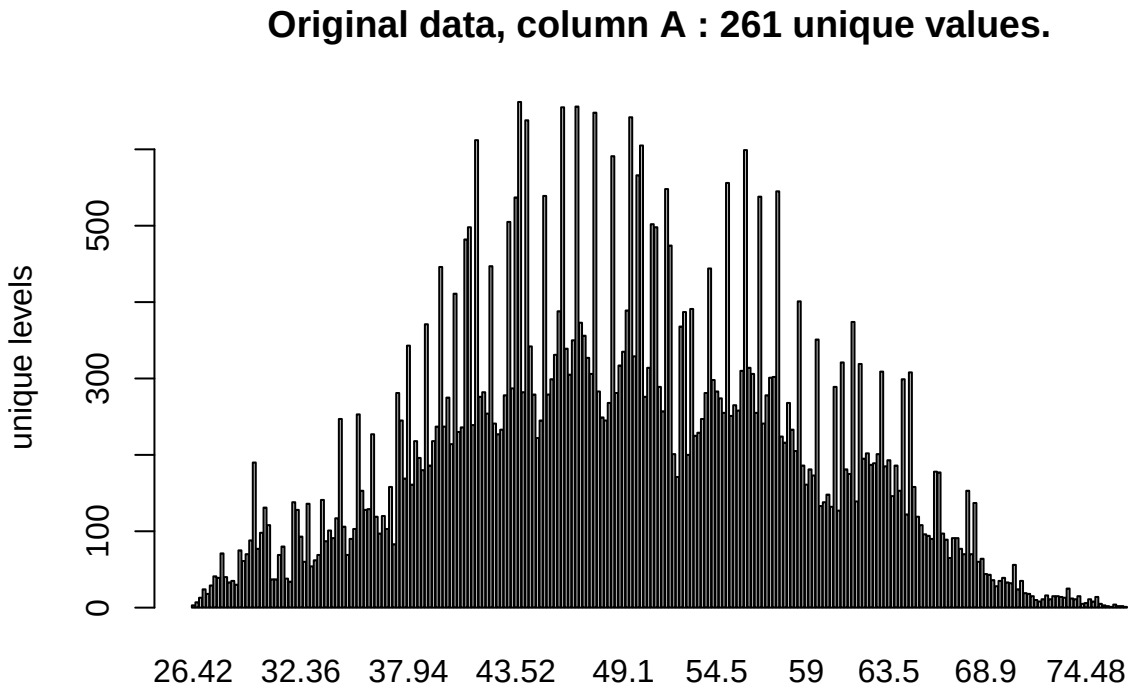
```
if( FALSE ){
for(acol in names(dat1_detrended_discrete)){
    atitle <- paste0('Detrended data, discretised to 5 levels')
    adf <- data.frame(data=dat1_detrended_discrete[[acol]]$discretised); adf$id <- seq.int(nrow(adf))
    p <- ggplot(
        adf,
        aes_string(x='id',y='data')
    )+geom_line()+ggtitle(atitle)
    print(p)
}
```

```
} # if( FALSE )
```

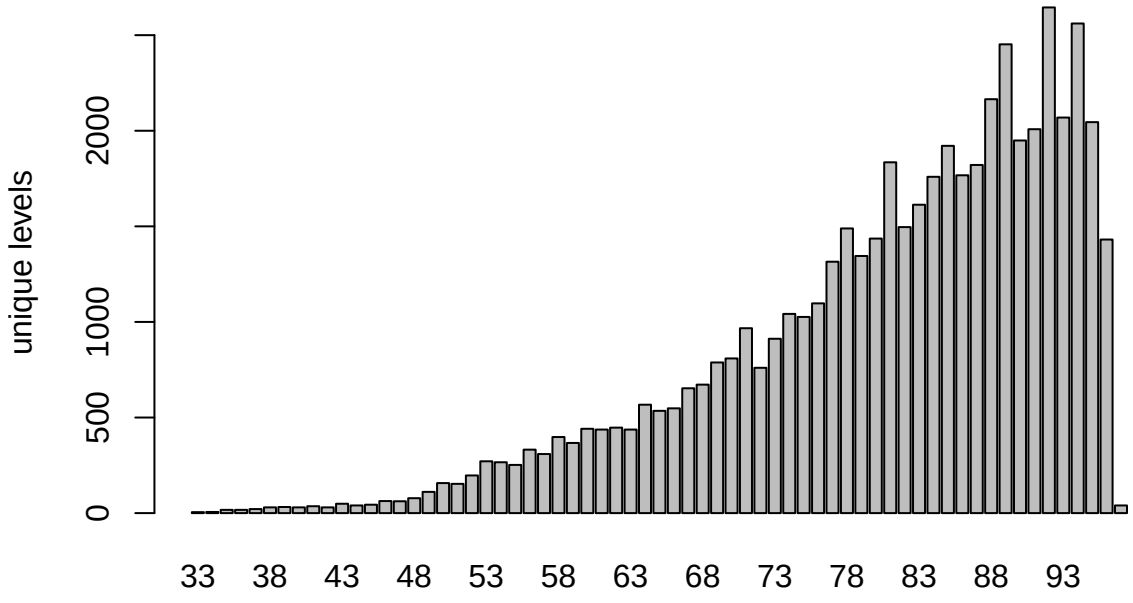
find unique values

```
uniq_vals <- unique_values(inp=dat1_noid)
histo_dat1 <- histo(inp=dat1_noid,unique_vals=uniq_vals)
for(acol in names(dat1_noid)){
    cat("found ", uniq_vals$num_unique_values[1,acol], " unique values in the original data, column ", a
    barplot(
        table(dat1_noid[[acol]]),
        main=paste0('Original data, column ', acol, ' : ', uniq_vals$num_unique_values[1,acol], ' unique
        ylab='unique levels'
    )
}
```

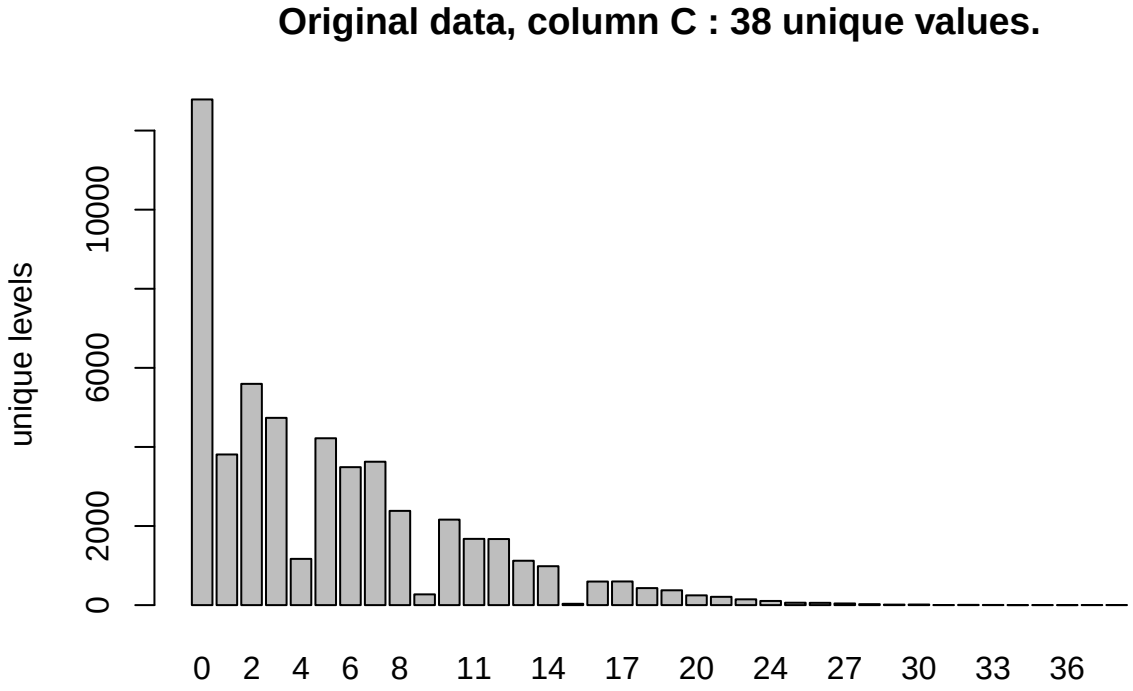
```
## found 261 unique values in the original data, column A
```



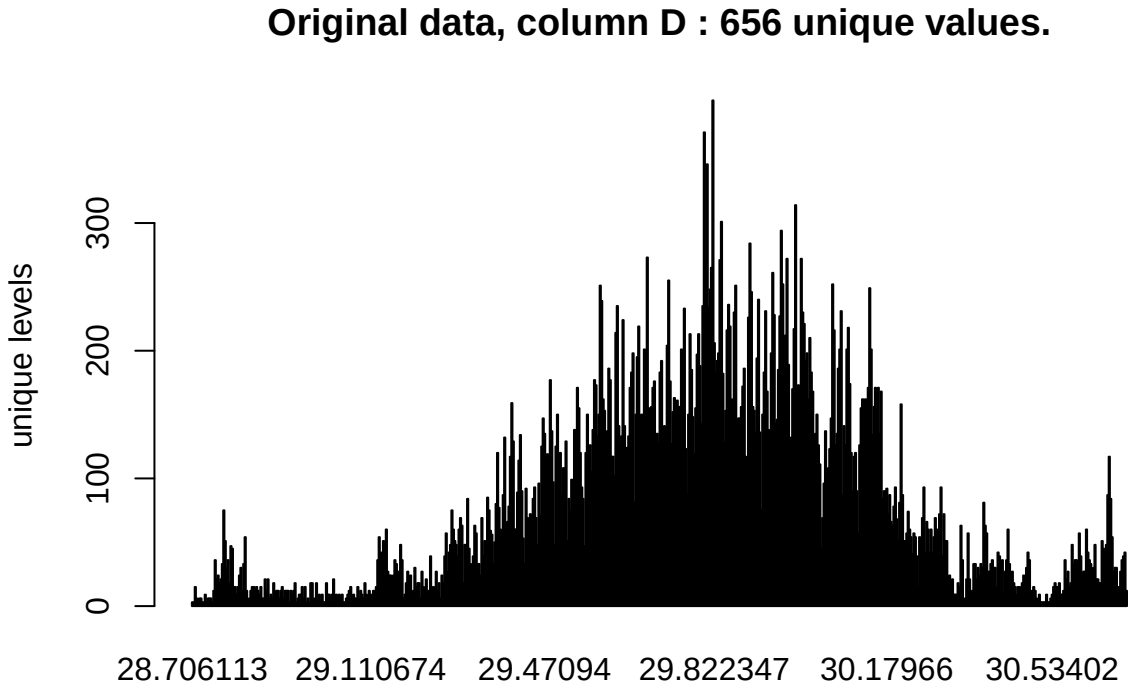

```
## found 65 unique values in the original data, column B
```



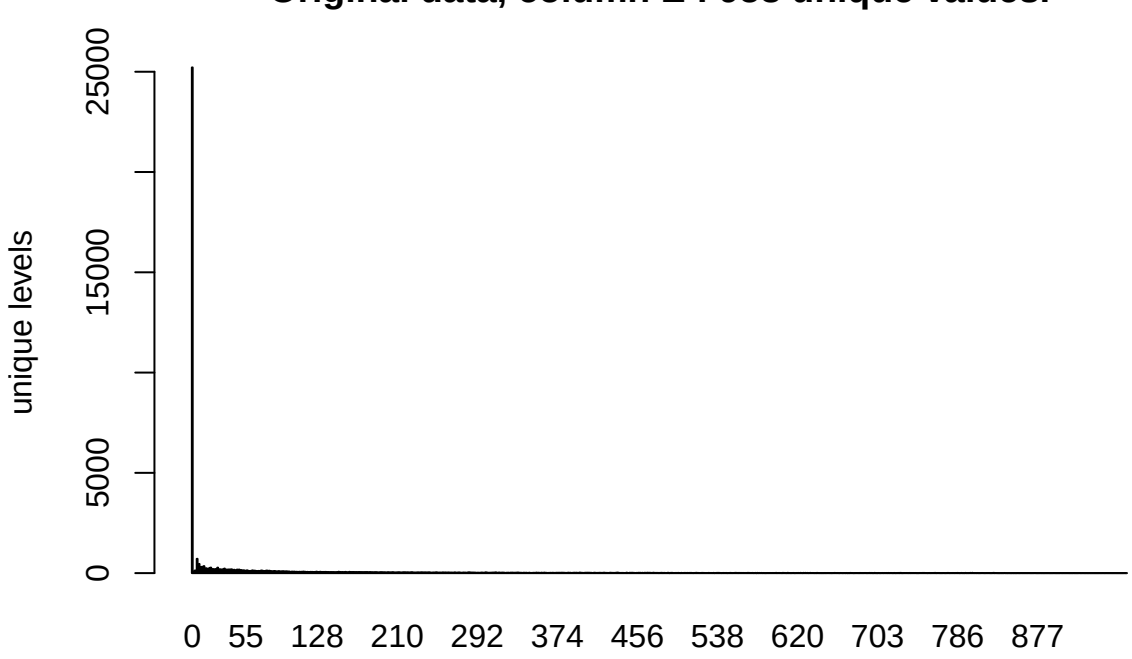

```
## found 38 unique values in the original data, column C
```



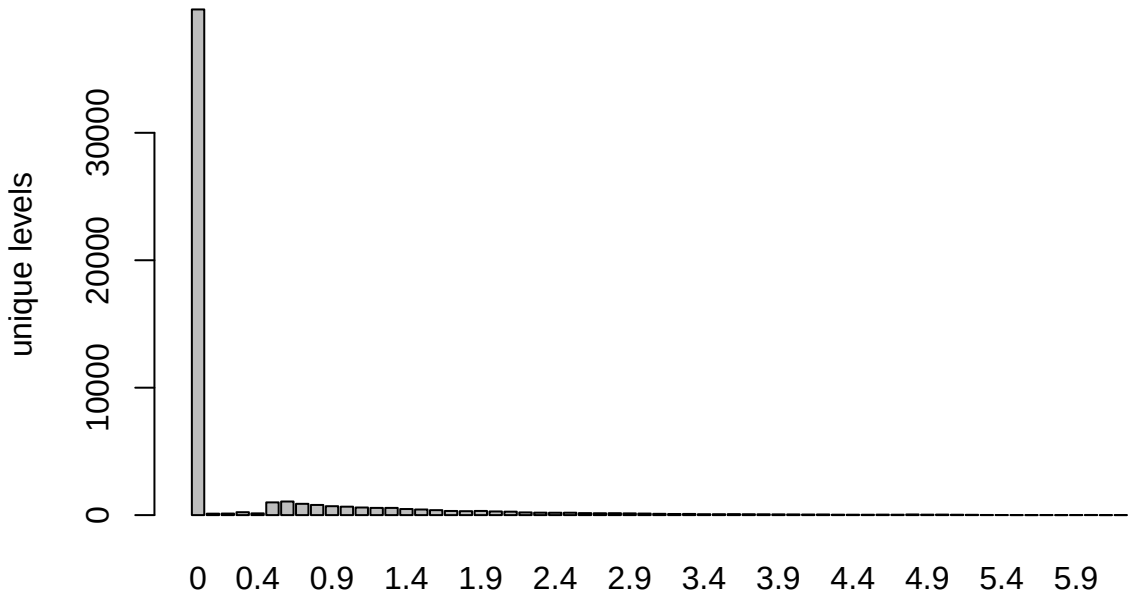

```
## found 656 unique values in the original data, column D
```




```
## found 958 unique values in the original data, column E
```




```
## found 63 unique values in the original data, column F
```

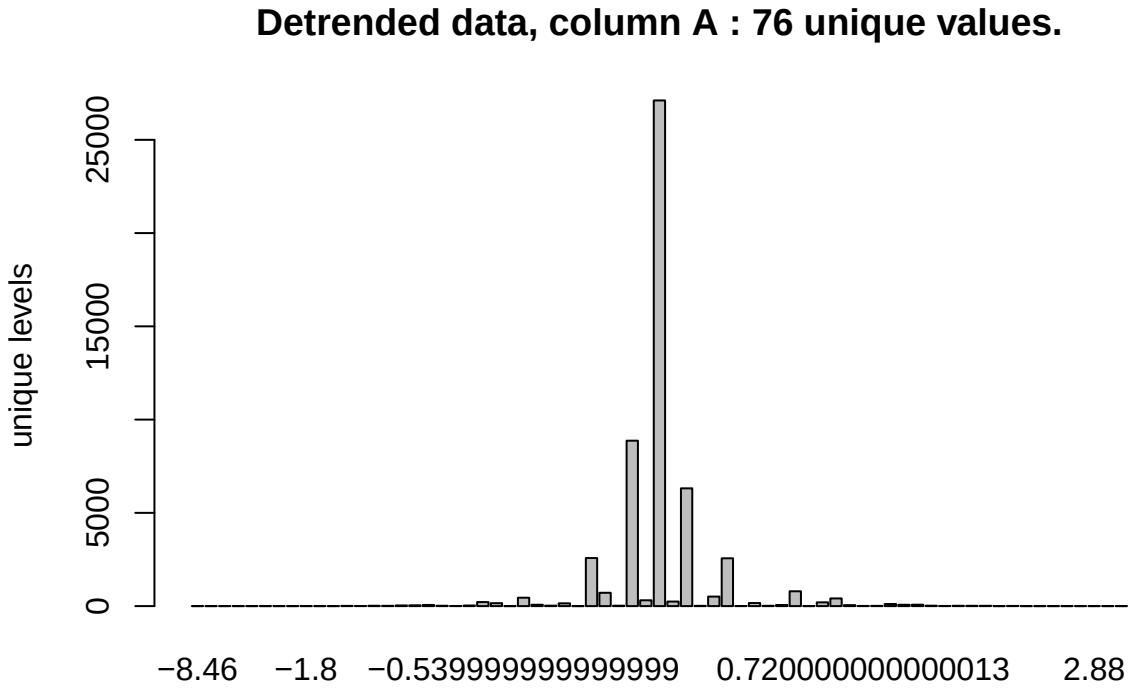



```
#print(histo_dat1)
```

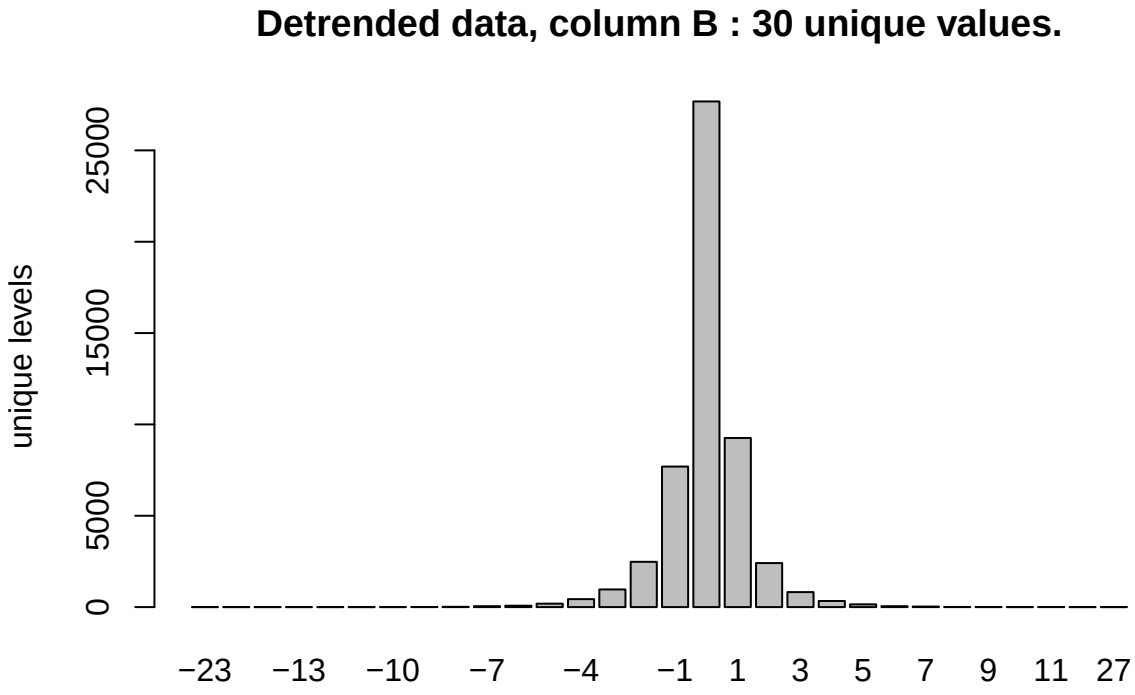
find unique values in the detrended data

```
uniq_vals_detrended <- unique_values(inp=dat1_detrended)
histo_dat1_detrended <- histo(inp=dat1_detrended,unique_vals=uniq_vals_detrended)
for(acol in names(dat1_detrended)){
    cat("found ", uniq_vals_detrended$num_unique_values[1,acol], " unique values in the detrended data,
    barplot(
        table(dat1_detrended[[acol]]),
        main=paste0('Detrended data, column ', acol, ' : ', uniq_vals_detrended$num_unique_values[1,acol
        ylab='unique levels'
    )
}
```

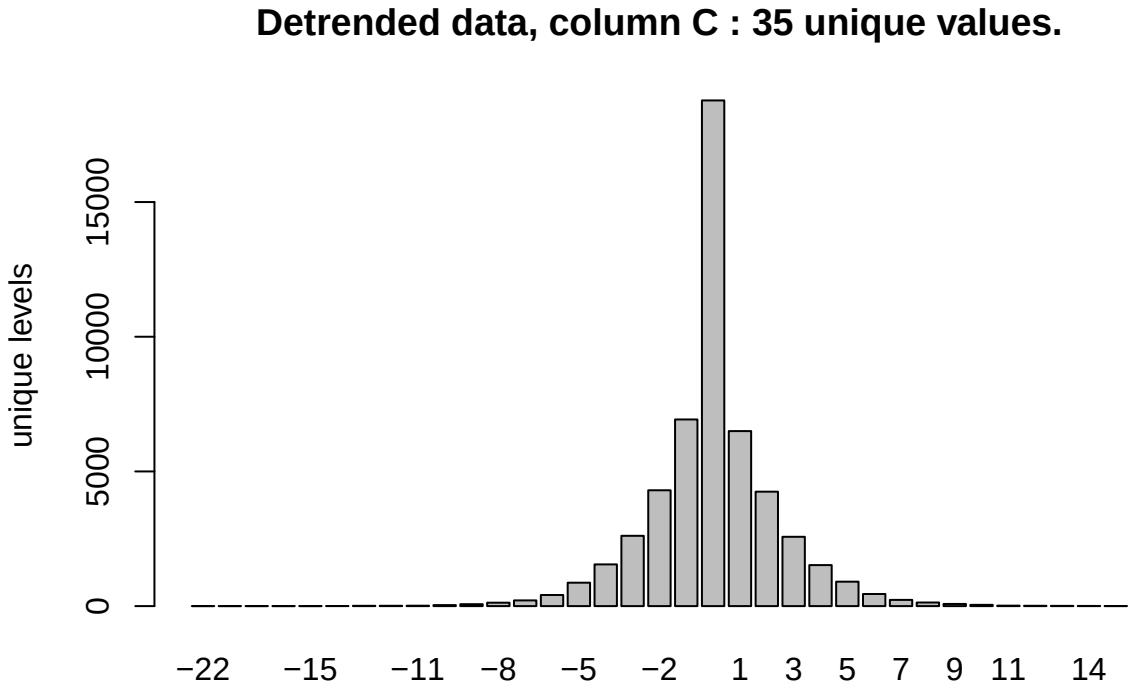
```
## found 76 unique values in the detrended data, column A
```



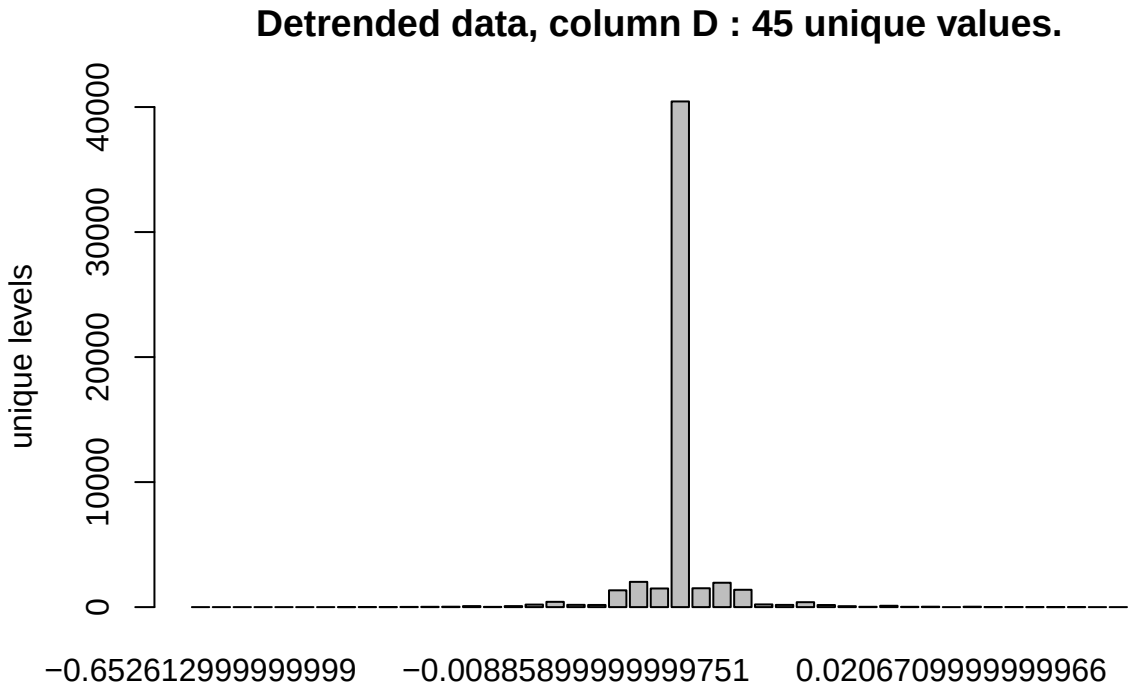

```
## found 30 unique values in the detrended data, column B
```



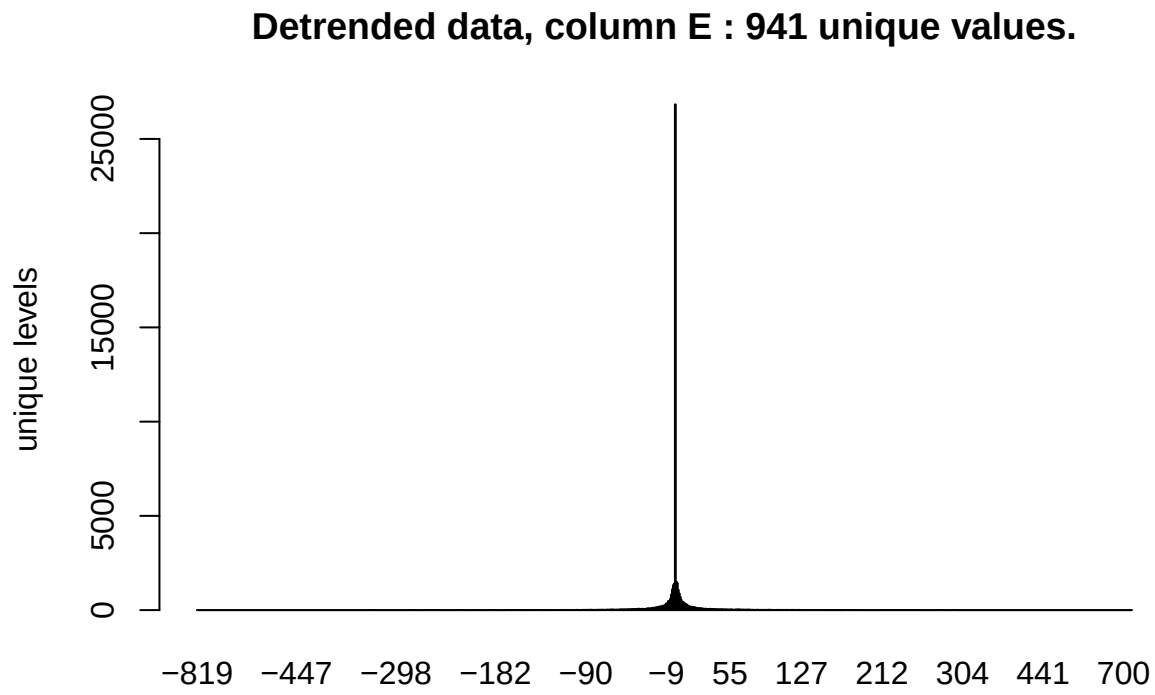

```
## found 35 unique values in the detrended data, column C
```



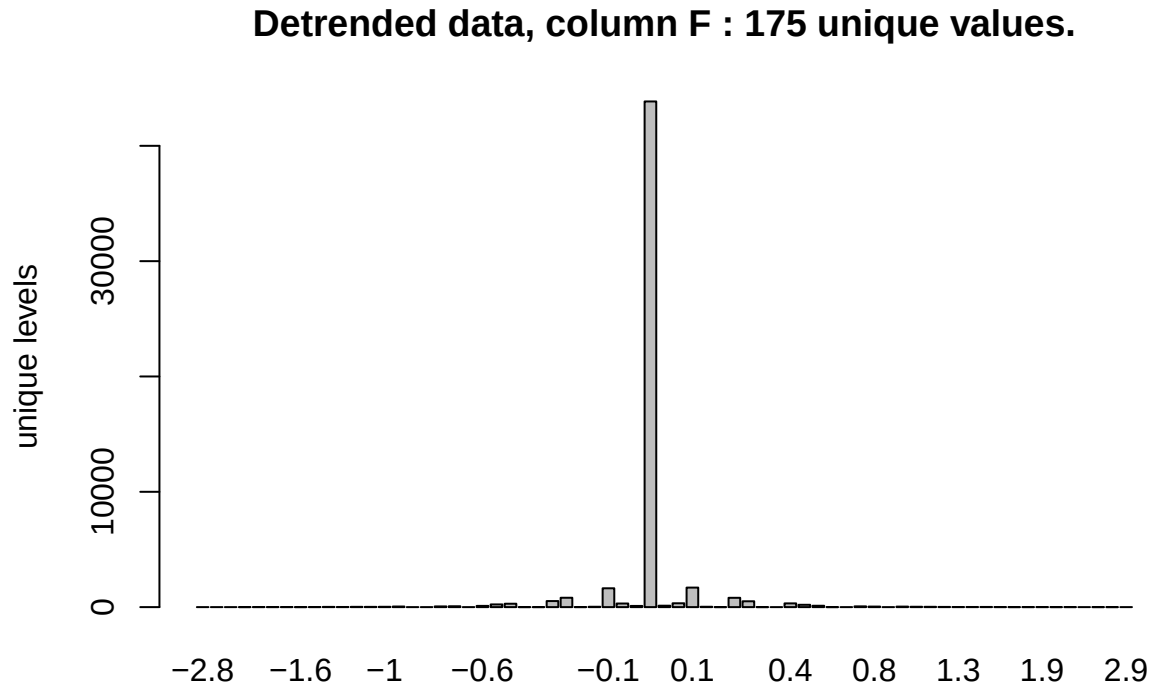

```
## found 45 unique values in the detrended data, column D
```




```
## found 941 unique values in the detrended data, column E
```




```
## found 175 unique values in the detrended data, column F
```

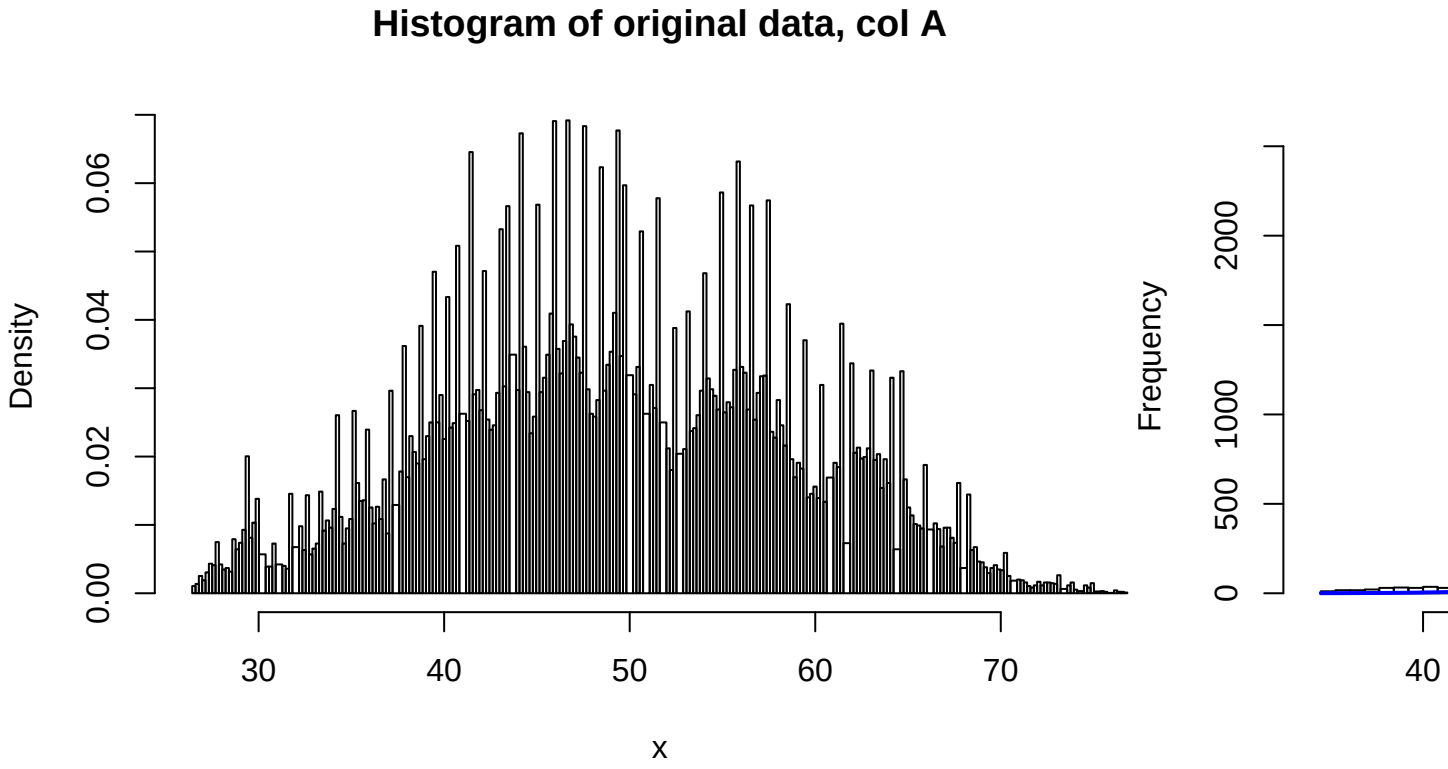


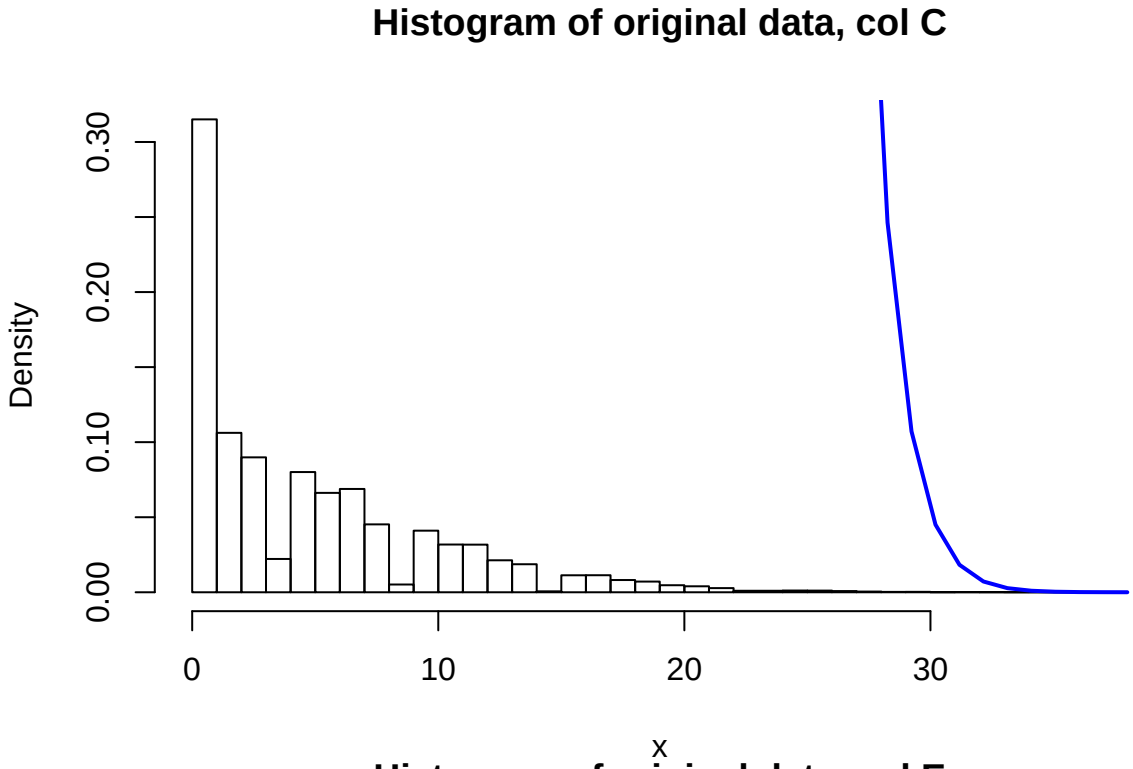

```
#print(histo_dat1_detrended)
```

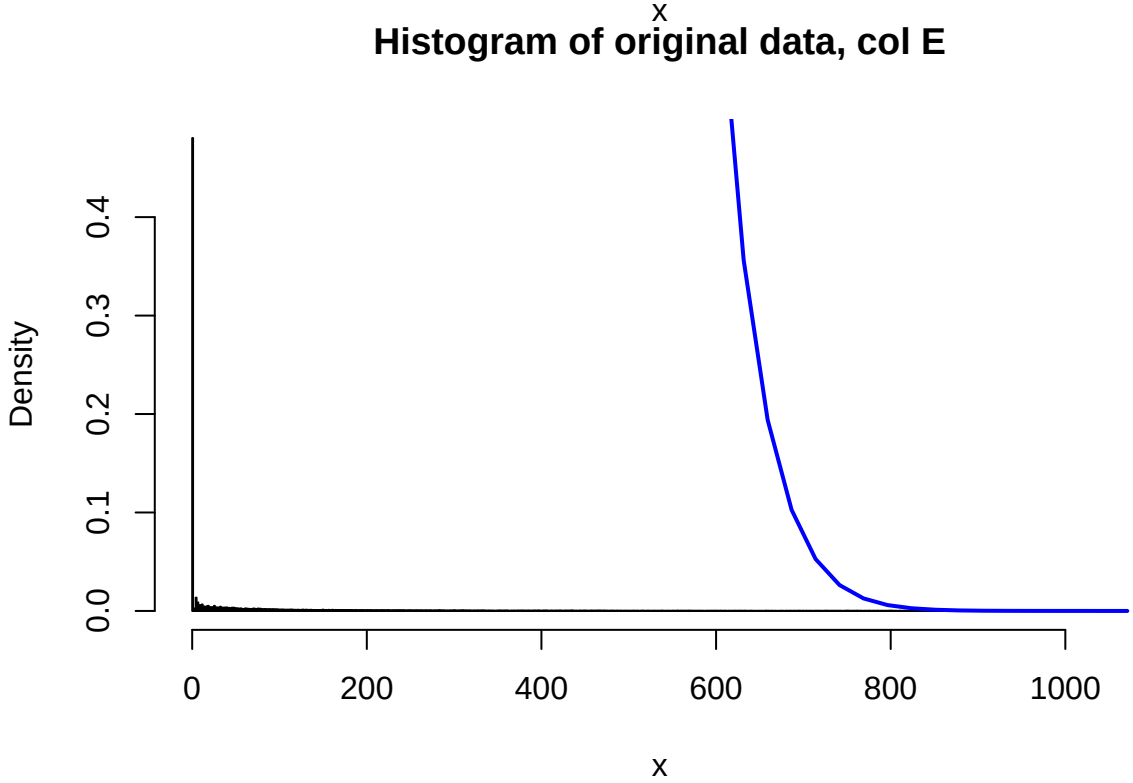
plot the histogram of each column

```
for(acol in names(dat1_noid)){
    x <- dat1_noid[[acol]]
    h <- hist(
        x=x,
        main=paste0("Histogram of original data, col ", acol, sep=''),
```

```
        breaks=uniq_vals$histograms[[acol]]$breaks
    )
    #' and plot a normal density (if it was coming from that), from https://www.statmethods.net/graphs/
    xfit<-seq(min(x),max(x),length=40)
    yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
    yfit <- yfit*diff(h$mids[1:2])*length(x)
    lines(xfit, yfit, col="blue", lwd=2)
}
```

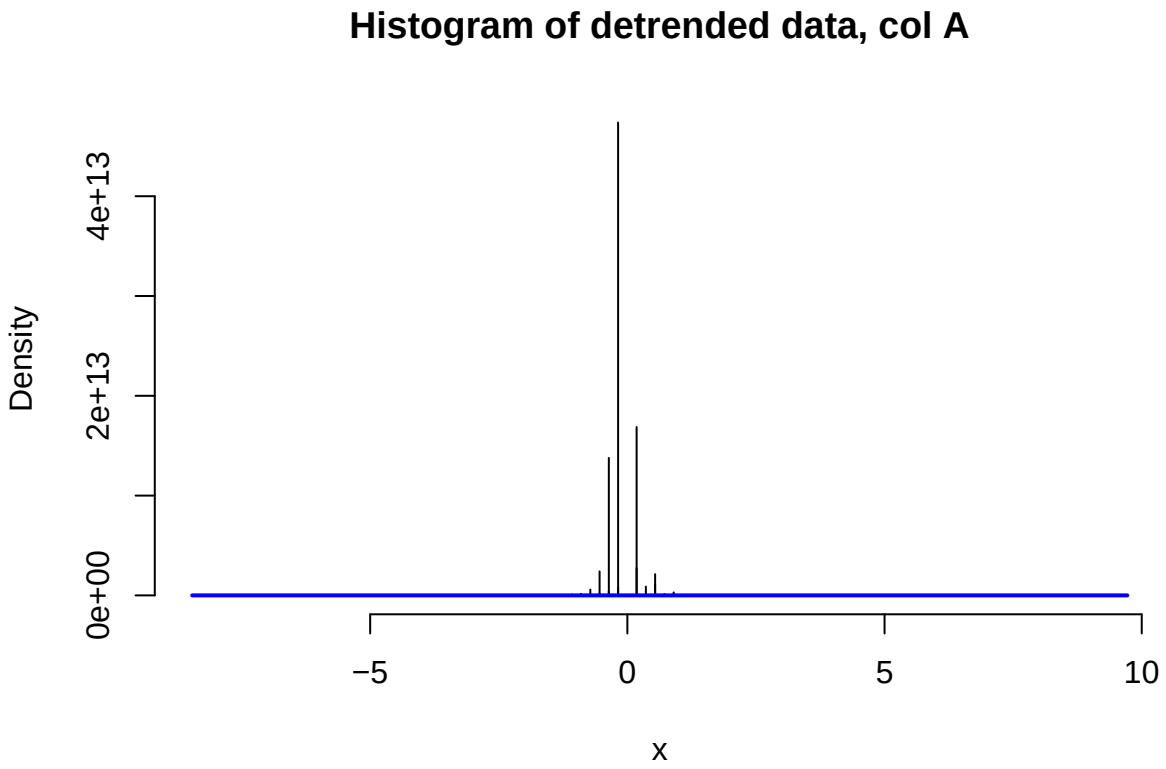


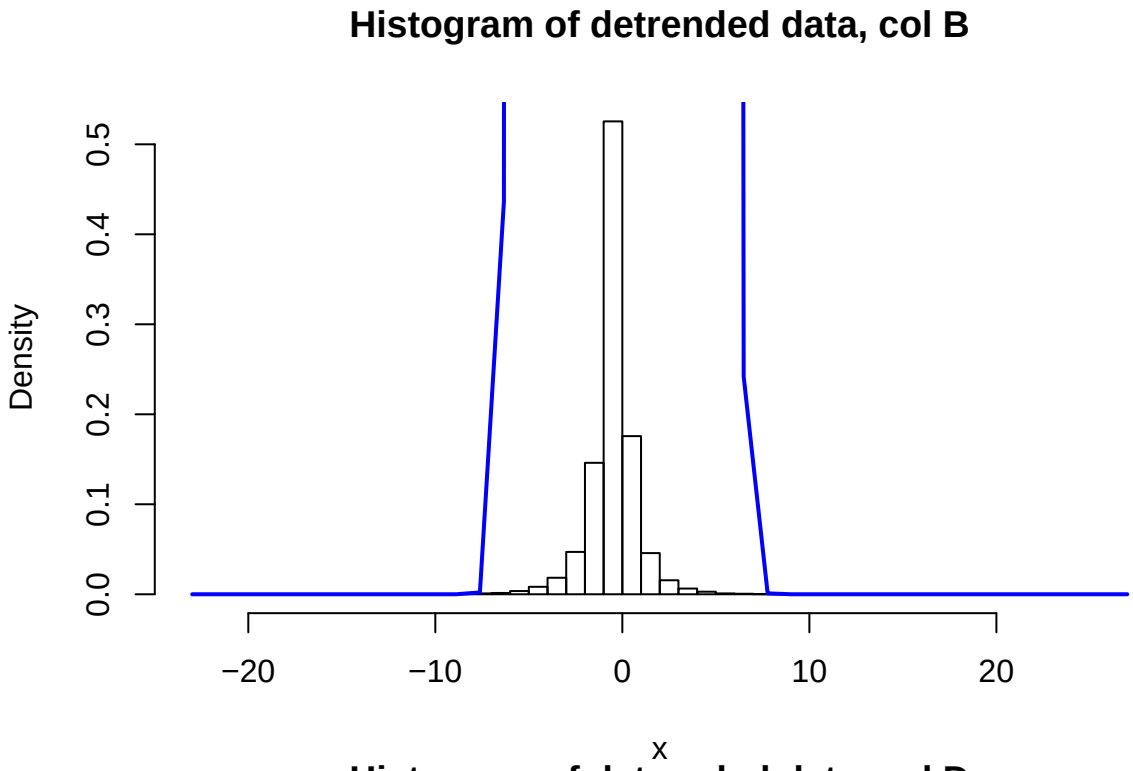


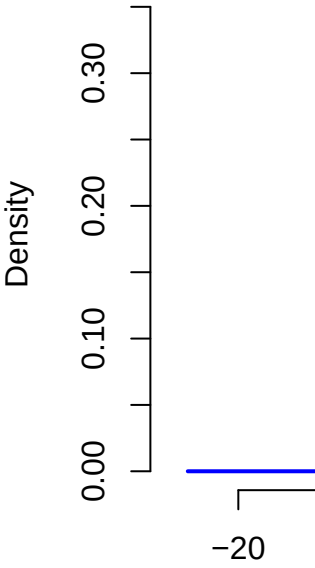


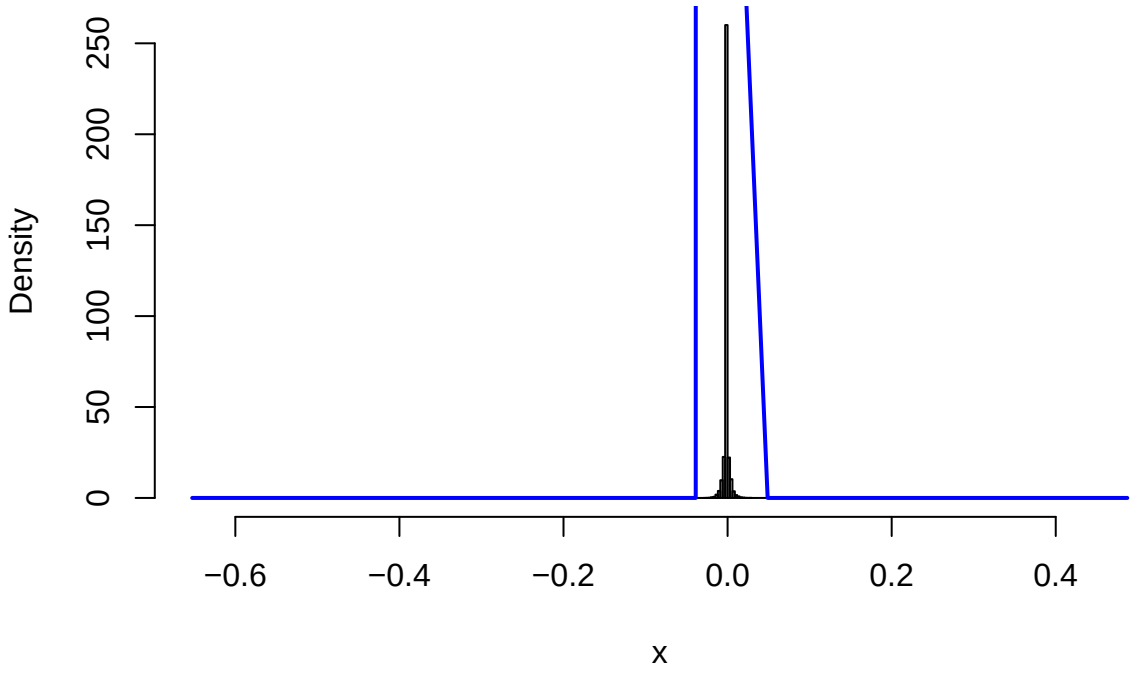

```
for(acol in names(dat1_detrended)){
    x <- dat1_detrended[[acol]]
    h <- hist(
        x=x,
        main=paste0("Histogram of detrended data, col ", acol, sep=''),
        breaks=uniq_vals_detrended$histograms[[acol]]$breaks
```

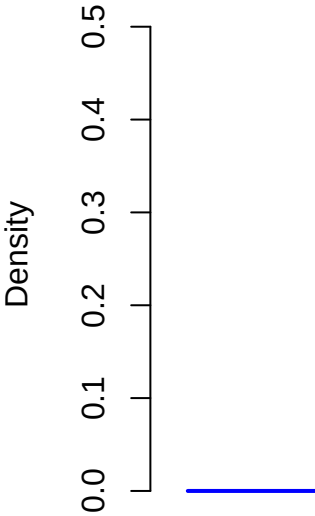
```
    )
    #' and plot a normal density (if it was coming from that), from https://www.statmethods.net/graphs/
    xfit<-seq(min(x),max(x),length=40)
    yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
    yfit <- yfit*diff(h$mids[1:2])*length(x)
    lines(xfit, yfit, col="blue", lwd=2)
}
```

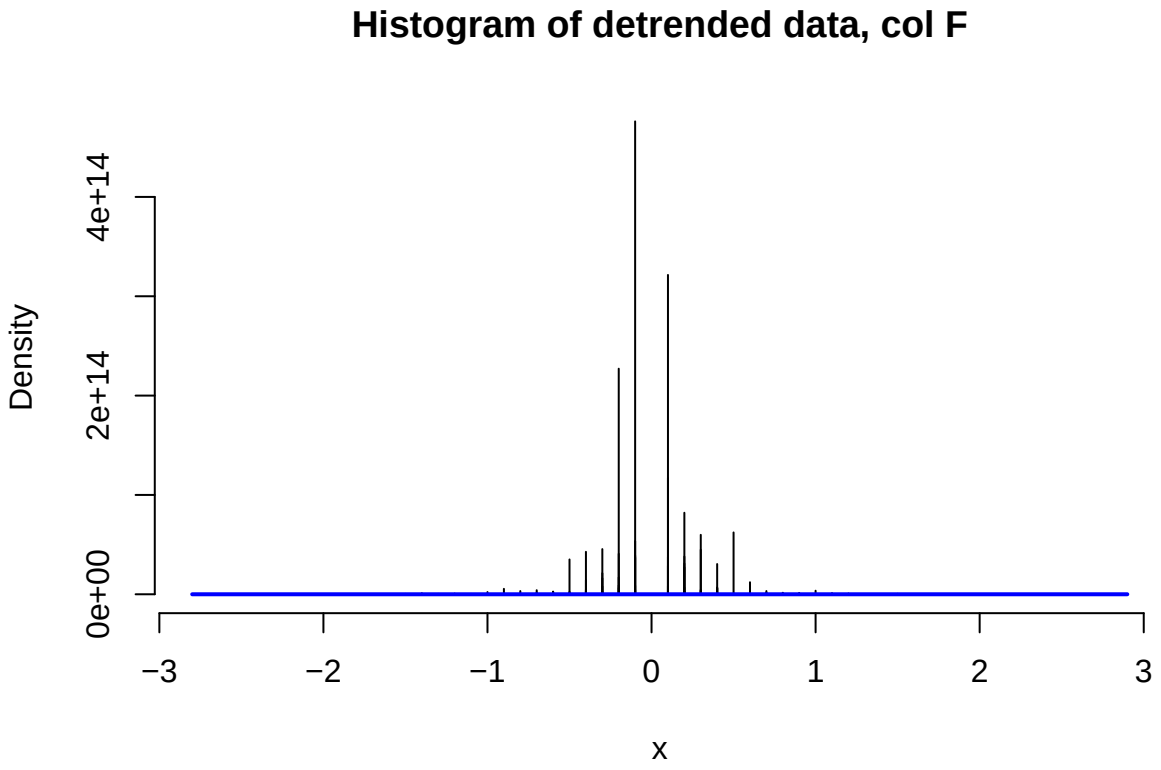



Histogram of detrended data, col B
Density
0.0
0.1
0.2
0.3
0.4
0.5
−20
−10
0
10
20
x


Density
0.00
0.10
0.20
0.30
−20

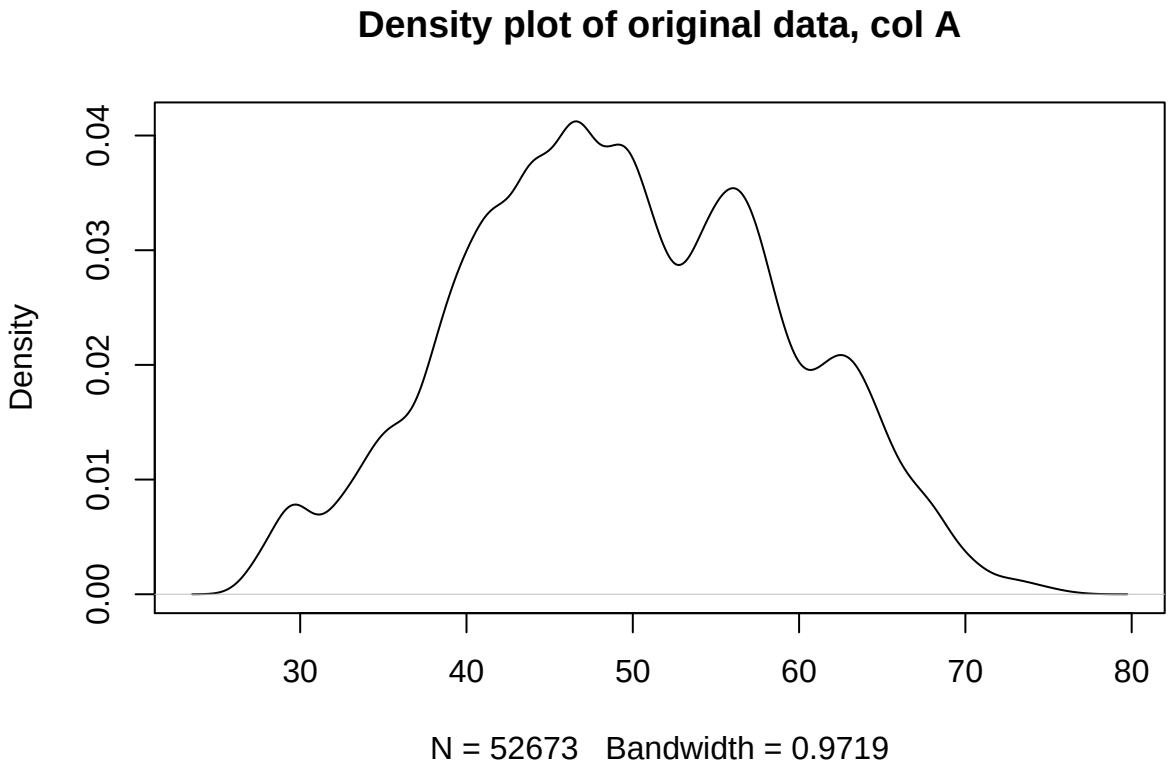

Histogram of detrended data, col D
Density
0
50
100
150
200
250
−0.6
−0.4
−0.2
0.0
0.2
0.4
x

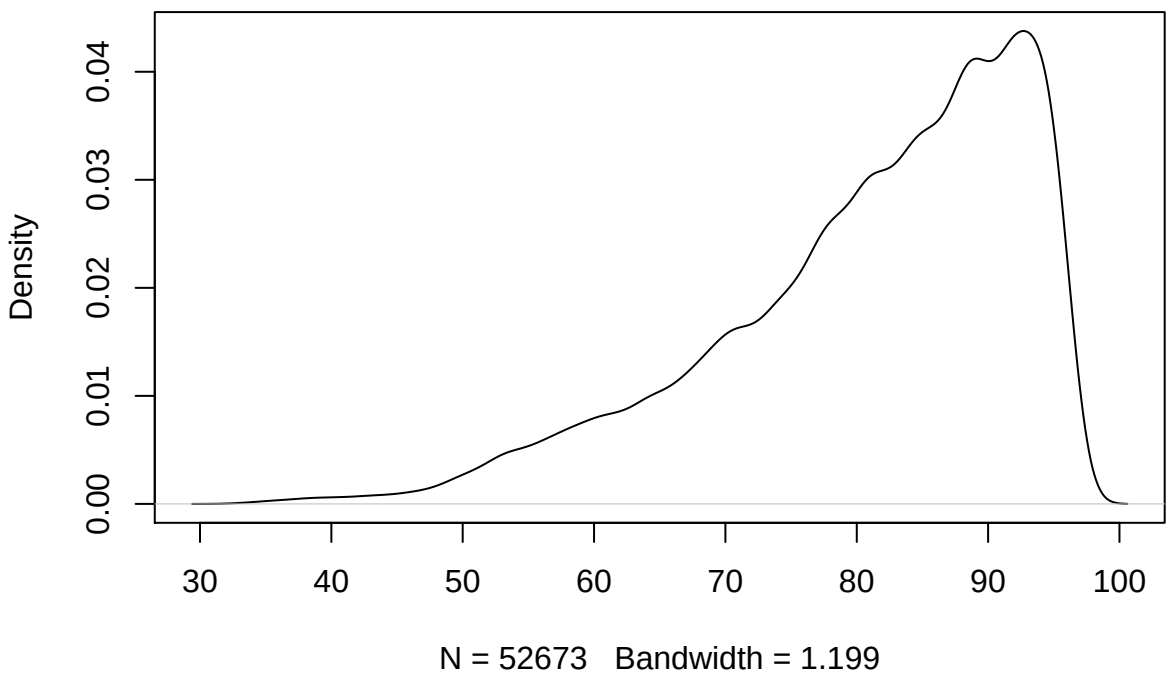

Density
0.0
0.1
0.2
0.3
0.4
0.5

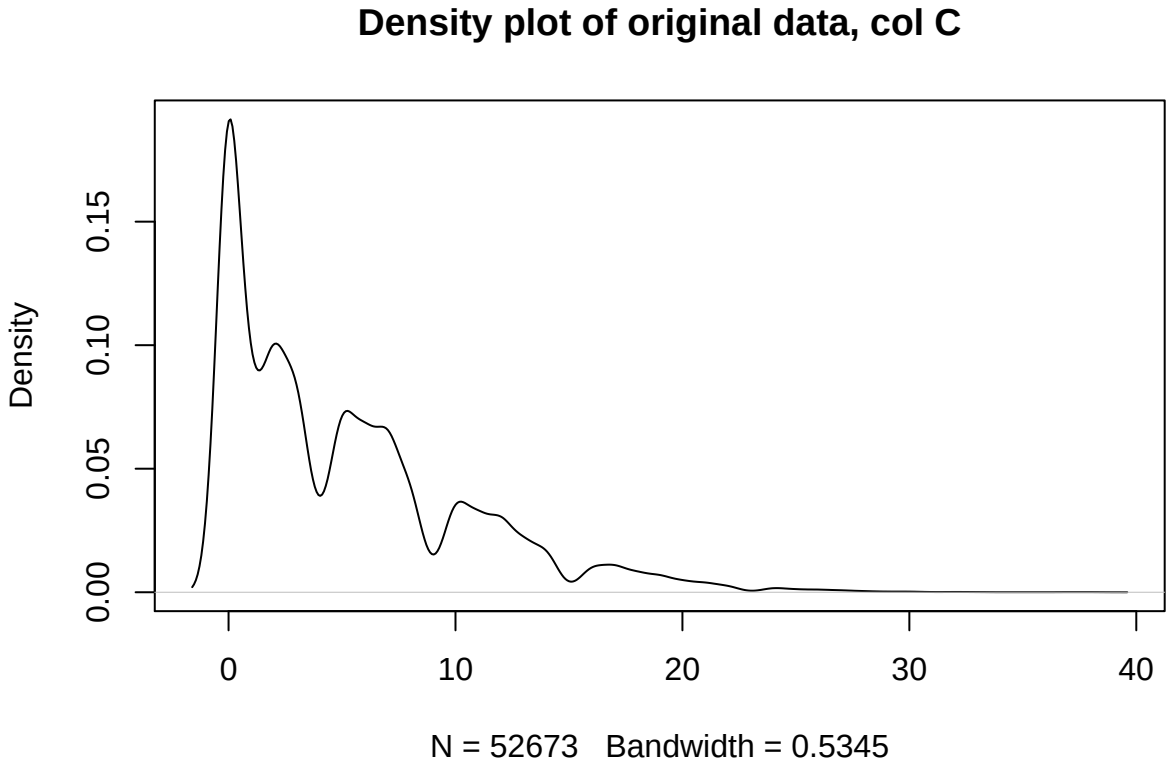


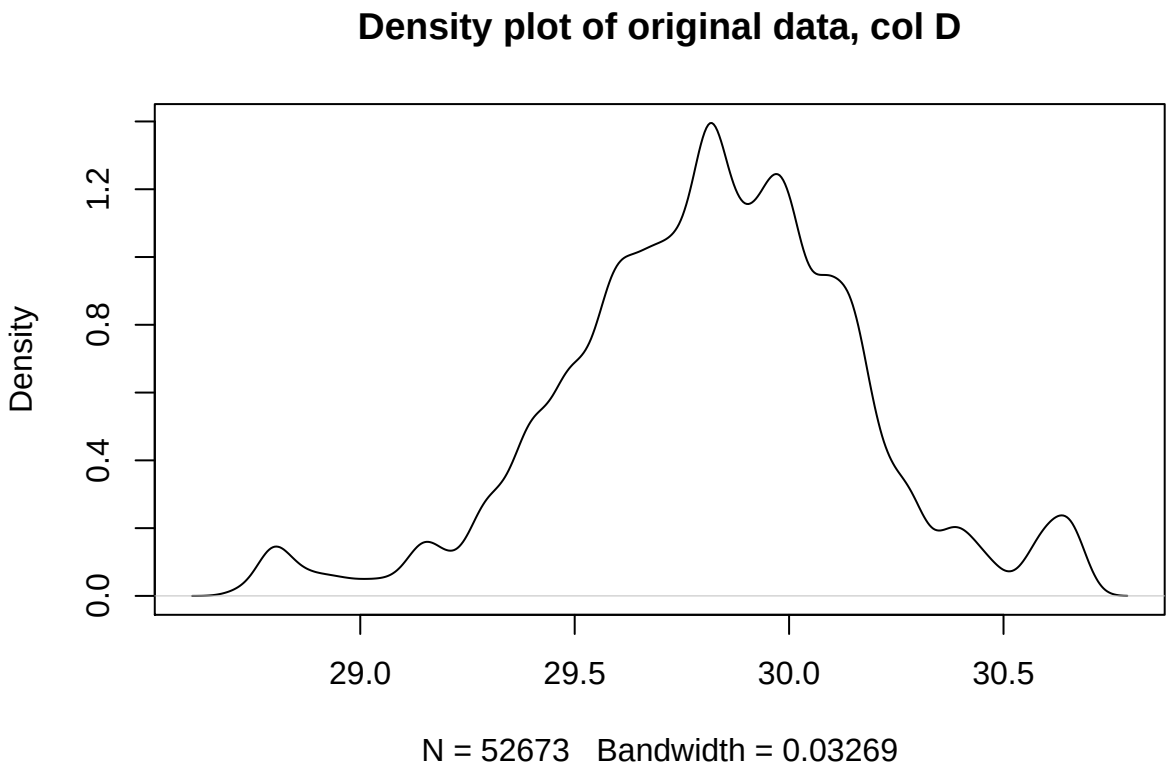
Density plots:

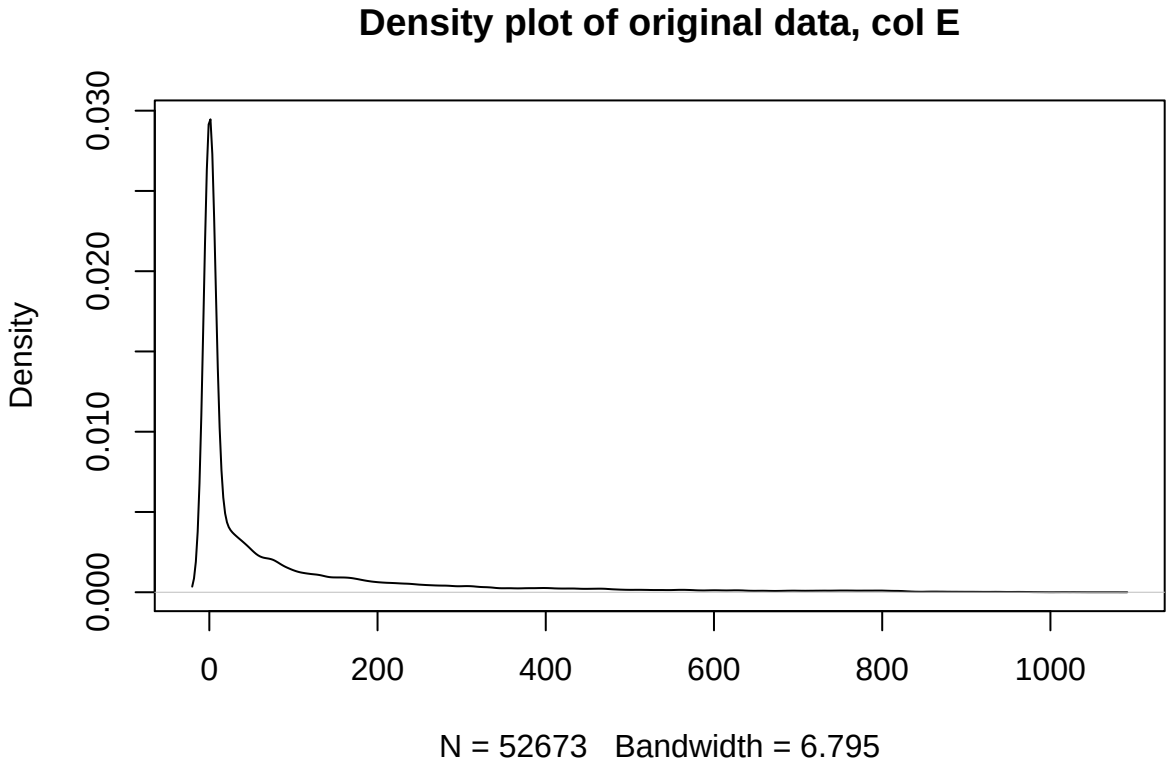
```
for(acol in names(dat1_noid)){
    plot(
        density(dat1_noid[[acol]]),
        main=paste0("Density plot of original data, col ", acol, sep='')
    )
}
```

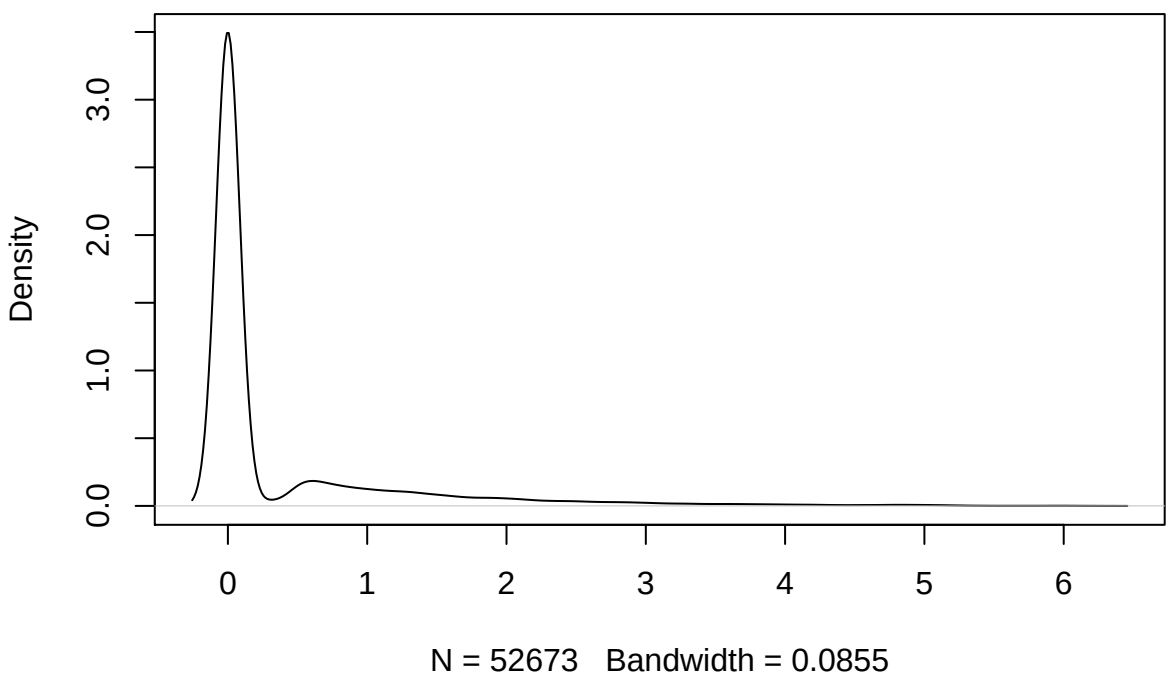

Density plot of original data, col A
Density
0.00
0.01
0.02
0.03
0.04
30
40
50
60
70
80
N = 52673 Bandwidth = 0.9719


Density plot of original data, col B
Density
0.00
0.01
0.02
0.03
0.04
30
40
50
60
70
80
90
100
N = 52673 Bandwidth = 1.199


Density plot of original data, col C
Density
0.00
0.05
0.10
0.15
0
10
20
30
40
N = 52673 Bandwidth = 0.5345

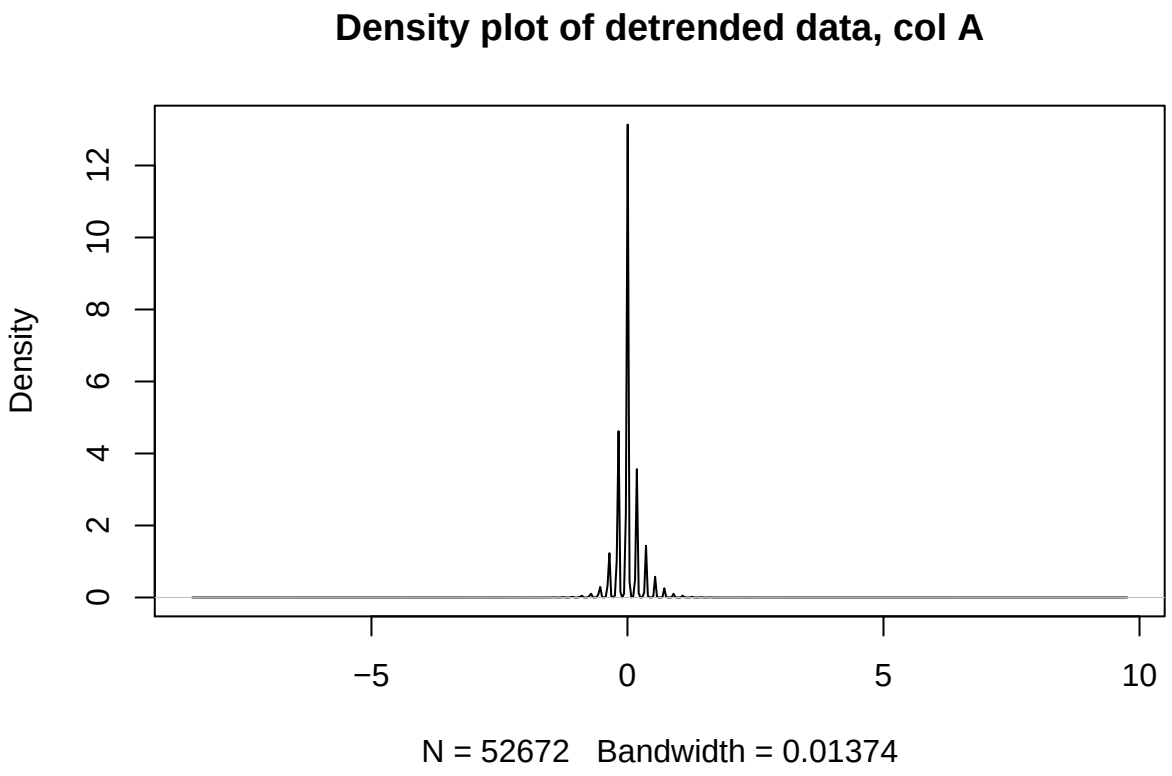

Density plot of original data, col D
Density
0.0
0.4
0.8
1.2
29.0
29.5
30.0
30.5
N = 52673 Bandwidth = 0.03269

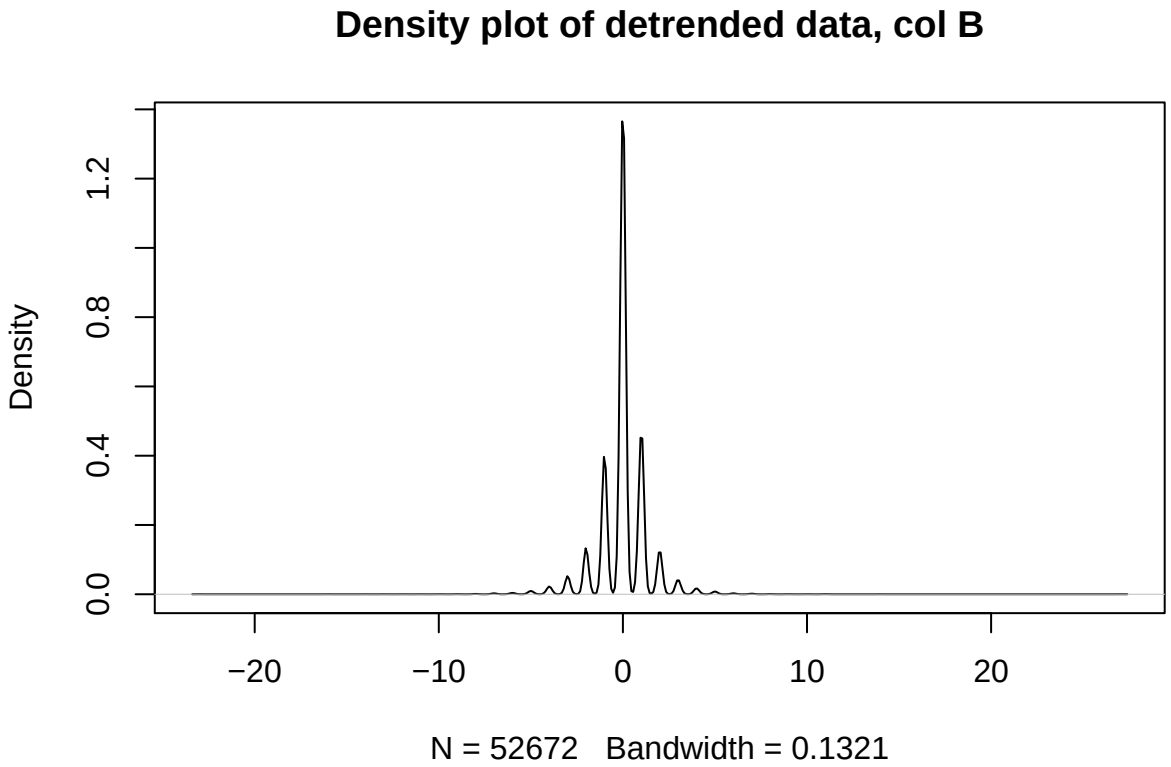


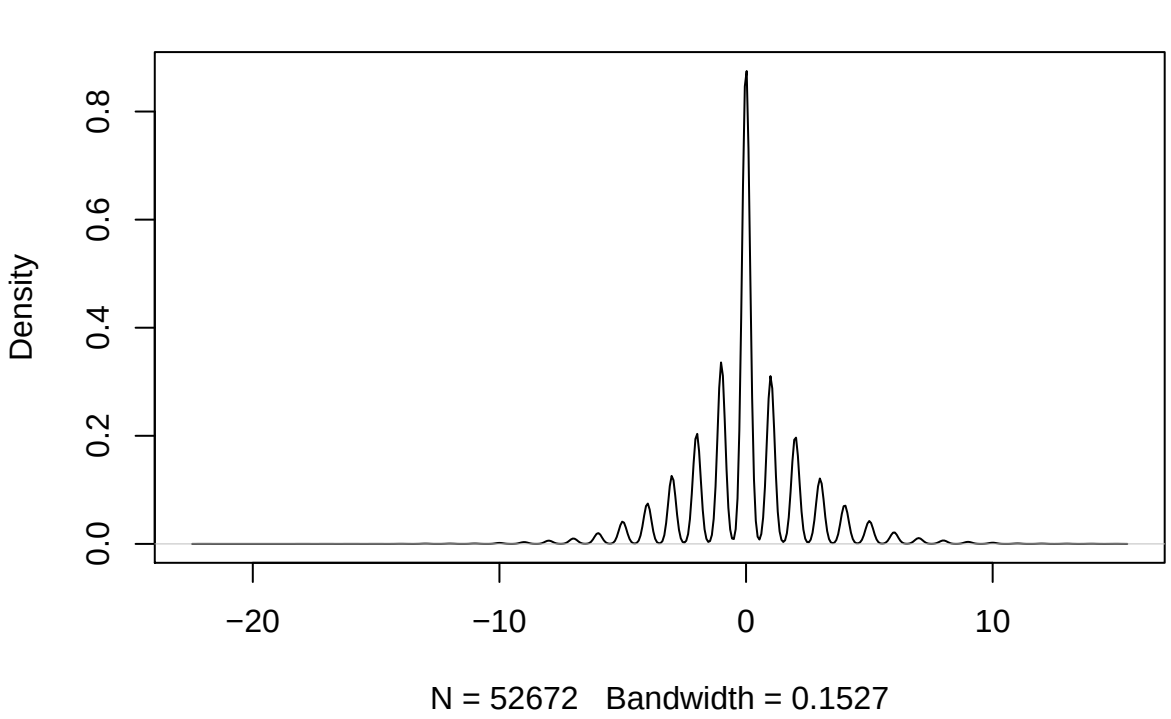


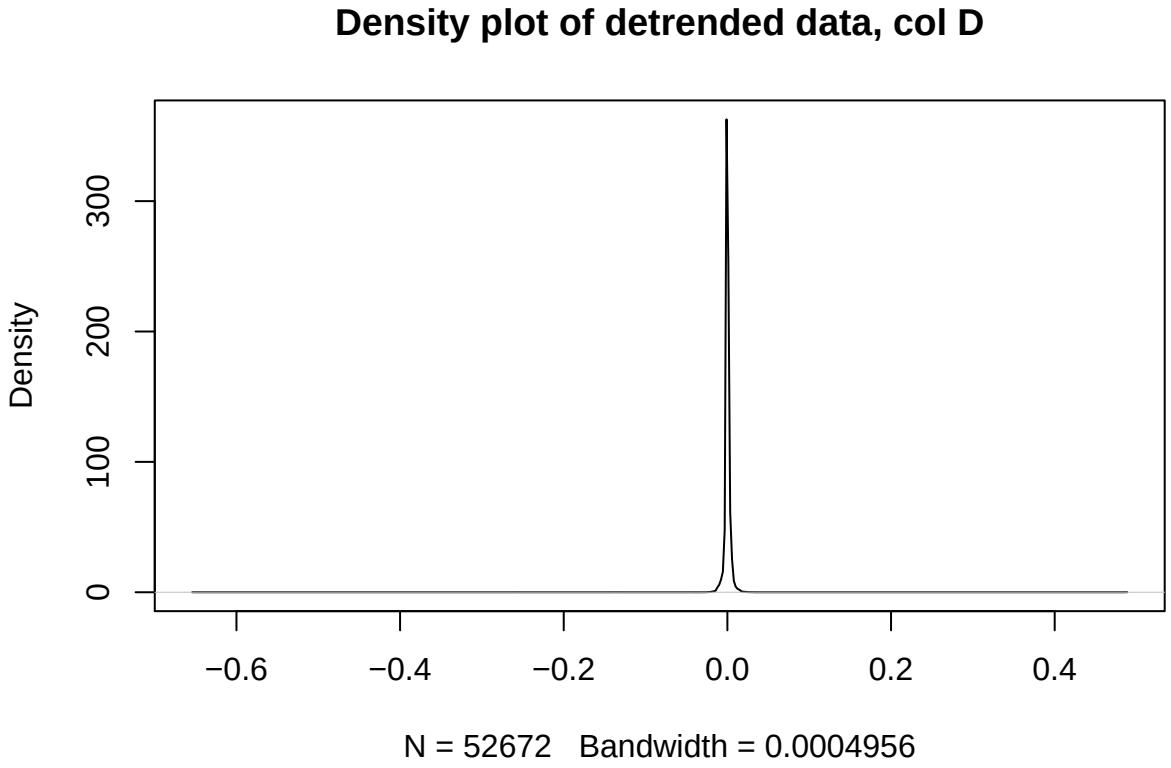
comments on the distribution of the generating process of the original data: A : looks like a mixture of Gaussians (maybe 4) B : a lognormal/weibull distrbution C : a mixture of lognormal/weibull distributions or 1 lognormal/weibull and some normal distributions D : a mixture of normal distributions E : a lognormal/weibull (shape ~1) F : a mixture of lognormal/weibull and 1 normal(?)

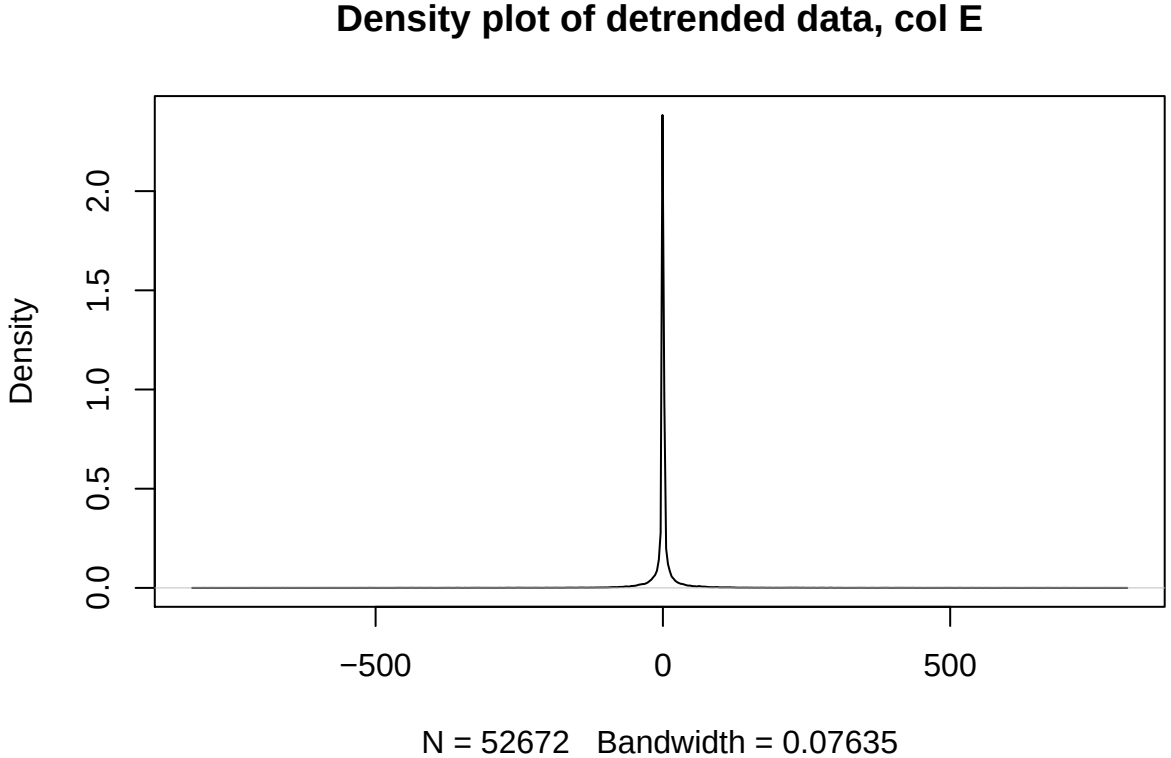
```r
for(acol in names(dat1_detrended)){
    plot(
        density(dat1_detrended[[acol]]),
        main=paste0("Density plot of detrended data, col ", acol, sep='')
    )
}
```

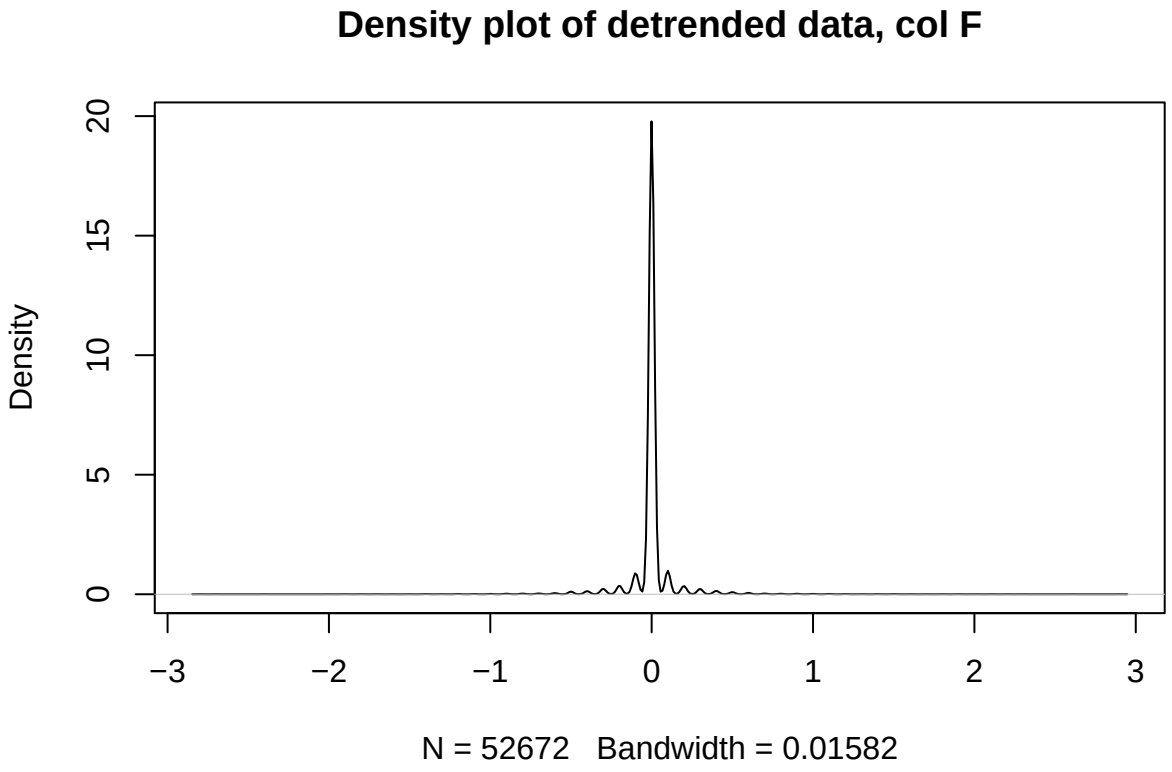



Density plot of detrended data, col B
Density
0.0
0.4
0.8
1.2
−20
−10
0
10
20
N = 52672 Bandwidth = 0.1321


Density plot of detrended data, col C
Density
0.0
0.2
0.4
0.6
0.8
−20
−10
0
10
N = 52672 Bandwidth = 0.1527


Density plot of detrended data, col D
Density
0
100
200
300
−0.6
−0.4
−0.2
0.0
0.2
0.4
N = 52672 Bandwidth = 0.0004956


Density plot of detrended data, col E
Density
0.0
0.5
1.0
1.5
2.0
−500
0
500
N = 52672 Bandwidth = 0.07635



comments on the distribution of the generating process of the original data: A,B,C,F : mixture of normal distributions D,E : pure normal distributions with very light tails, see below on fitting a distribution on them Let's also do a q-q plot (quantile-quantile plot) comparing with the Normal distribution. A q-q plot is a plot of the quantiles of test data coming e.g. from the Normal distribution against the corresponding quantiles of our data. 'the data in the x% quantile' means the first x% of the data sorted in ascending order. The more the plot coincides with the 45-degree line the stronger the assumption that our data derives from specific distribution is. Other distributions can be tested. note: flat regions indicate agreemenet, curved regions disagreement for example, flat in the middle but curved on the sides means that it probably comes from a Normal distribution but has heavy tails, more data at the tails than Normal.

```
library(ggpubr)
for(acol in names(dat1_noid)){
#   qqnorm( y=dat1_noid[[acol]],
#       main=paste0("Q-Q plot of original data, col ", acol, sep='')
#   ); qqline(dat1_noid[[acol]]);
    ggqqplot(
        dat1_noid[[acol]],
        main=paste0("Q-Q plot of original data, col ", acol, sep='')
    )
}
for(acol in names(dat1_detrended)){
#   qqnorm( y=dat1_detrended[[acol]],
#       main=paste0("Q-Q plot of detrended data, col ", acol, sep='')
#   ); qqline(dat1_detrended[[acol]])
    ggqqplot(
        dat1_detrended[[acol]],
        main=paste0("Q-Q plot of original data, col ", acol, sep='')
    )
}
```

One can also try to fit a distribution to our data for example, in the detrended column D (which appears to have a pure normal distribution)

```
library(fitdistrplus)
```

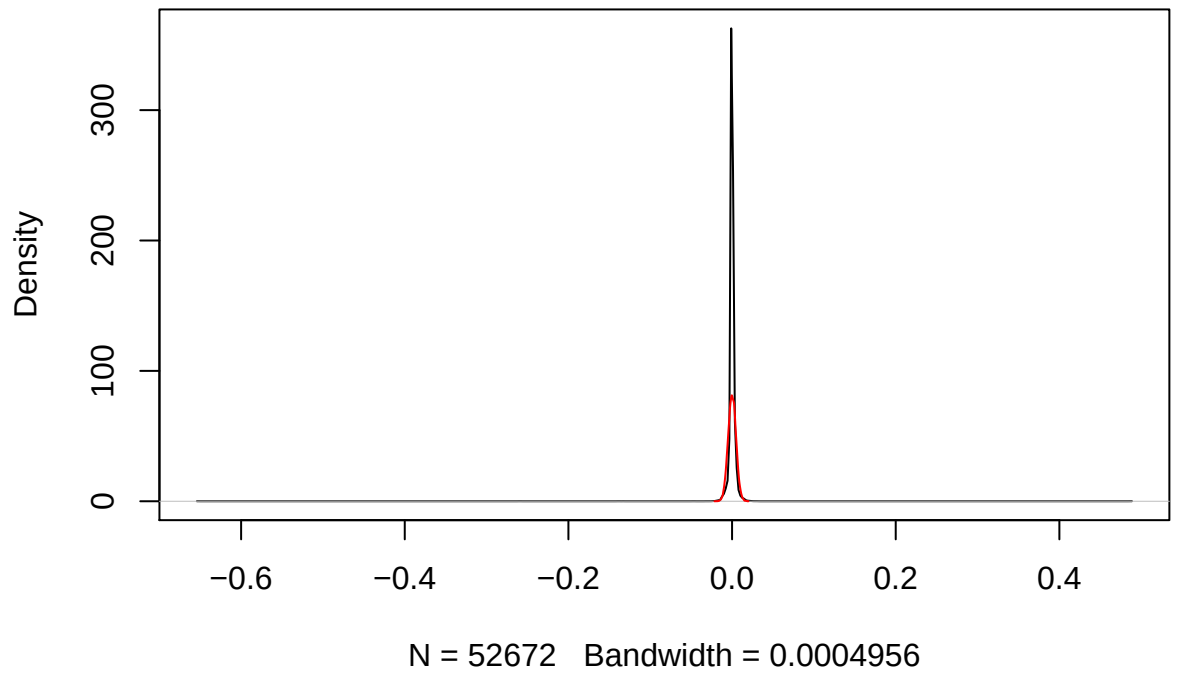
```
## Loading required package: MASS
```

```
## Loading required package: survival
```

```
acol='D'
x <- dat1_detrended[[acol]]
afit <- fitdistr(x, 'normal')
print(afit)
```

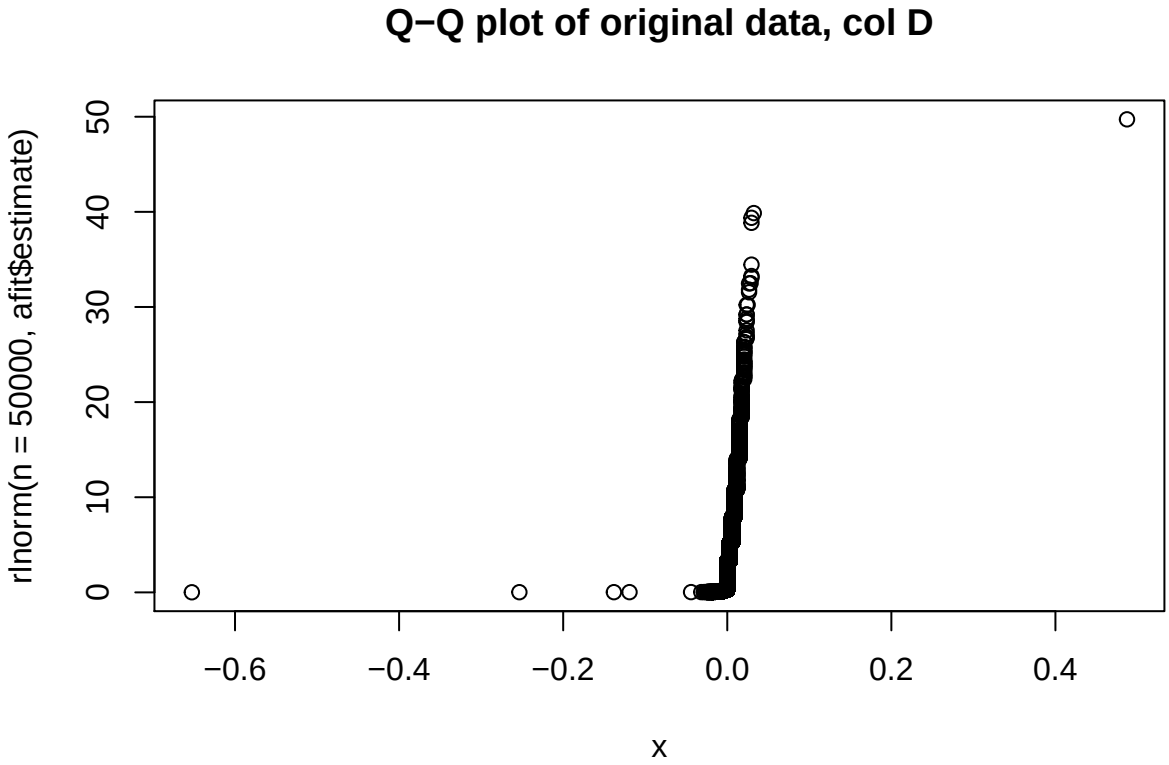
```
##        mean             sd
##   -2.018302e-06    4.843524e-03
##  ( 2.110433e-05) ( 1.492302e-05)
```

```
plot(
    density(x),
    main=paste0('Density plot of detrended data, col ', acol, ' superimposed with fitted Normal dist.',
)
lines(
    density(
        rnorm(
            n=50000,
            mean=afit$estimate['mean'],
            sd=afit$estimate['sd']
        )
    ),
    col='red',
    pch=22
)
```



the q-q plot

```
qqplot(
    x=x,
    y=rlnorm(n=50000, afit$estimate),
    main=paste0('Q-Q plot of original data, col ', acol, sep='')
)
```



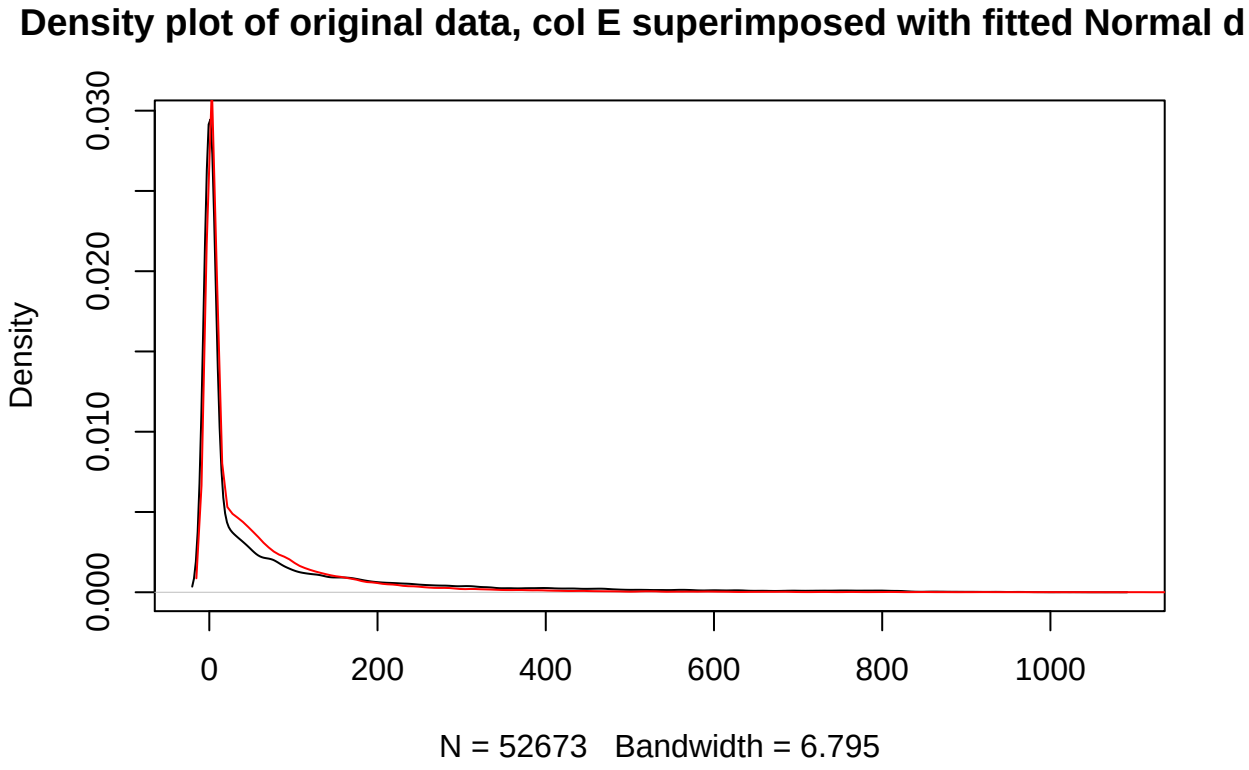
In this case (column D), we can see the peak of the data is much higher than that which comes from the fitted distribution Here is column E (original) which appears as coming from lognormal distribution

```
acol='E'
x <- dat1_noid[[acol]]+0.02 #' must be positive!
afit <- fitdistr(x, 'lognormal')
print(afit)
```

```
##     meanlog       sdlog
##   0.38862519   4.23427355
##  (0.01844951) (0.01304578)
```

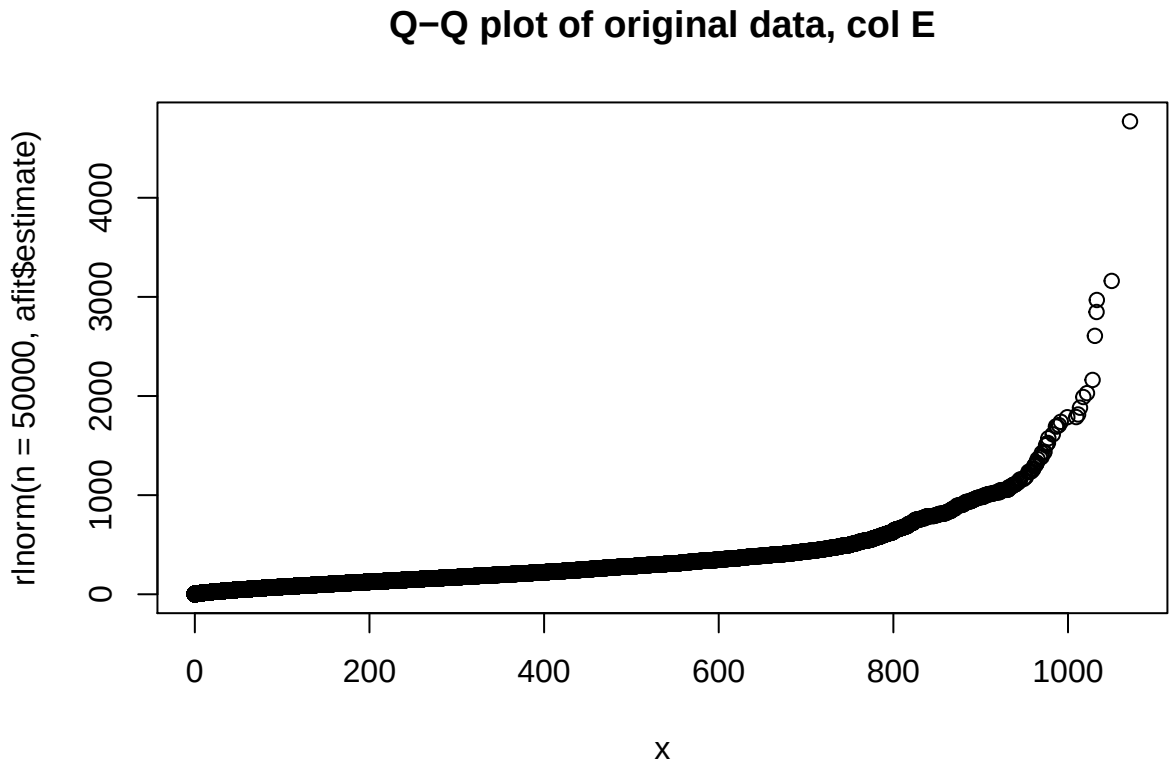
and plot

```
plot(
    density(x),
    main=paste0('Density plot of original data, col ', acol, ' superimposed with fitted Normal dist.', s
)
lines(density(rlnorm(n=50000, afit$estimate)), col='red', pch=22)
```



the q-q plot

```
qqplot(
    x=x,
    y=rlnorm(n=50000, afit$estimate),
    main=paste0('Q-Q plot of original data, col ', acol, sep='')
)
```

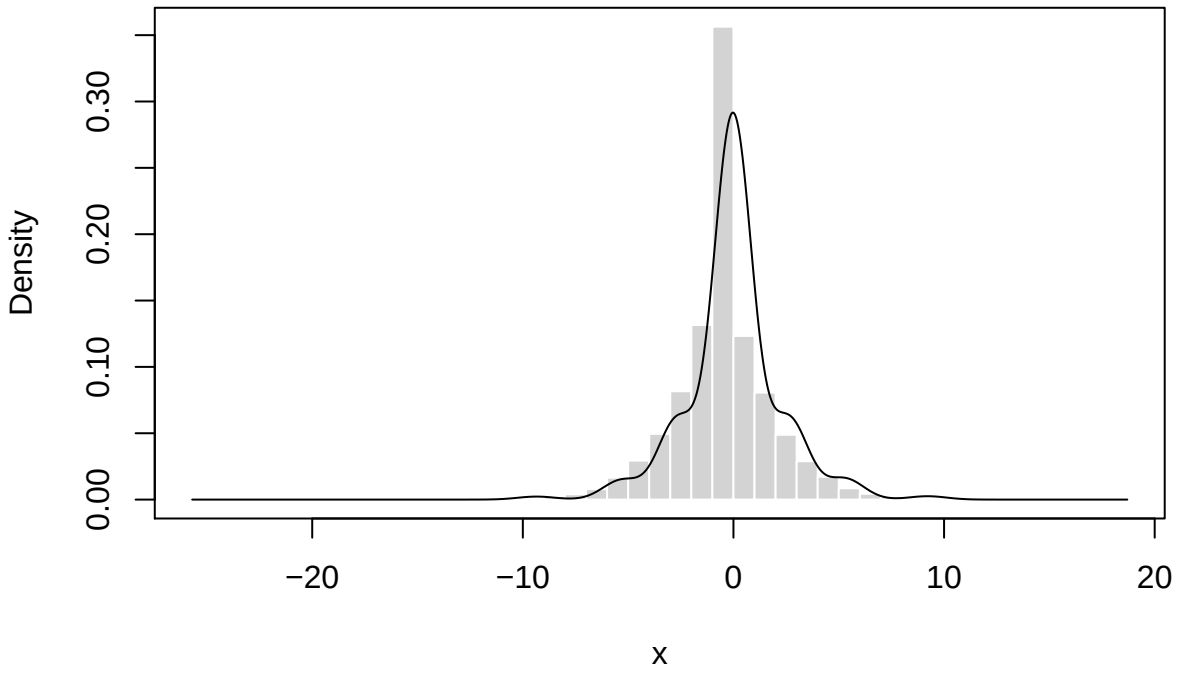


which is not bad except the tail is too heavy! For those columns that I think derive from mixture processes we can use a mixture model estimator

```
library(mclust)
```

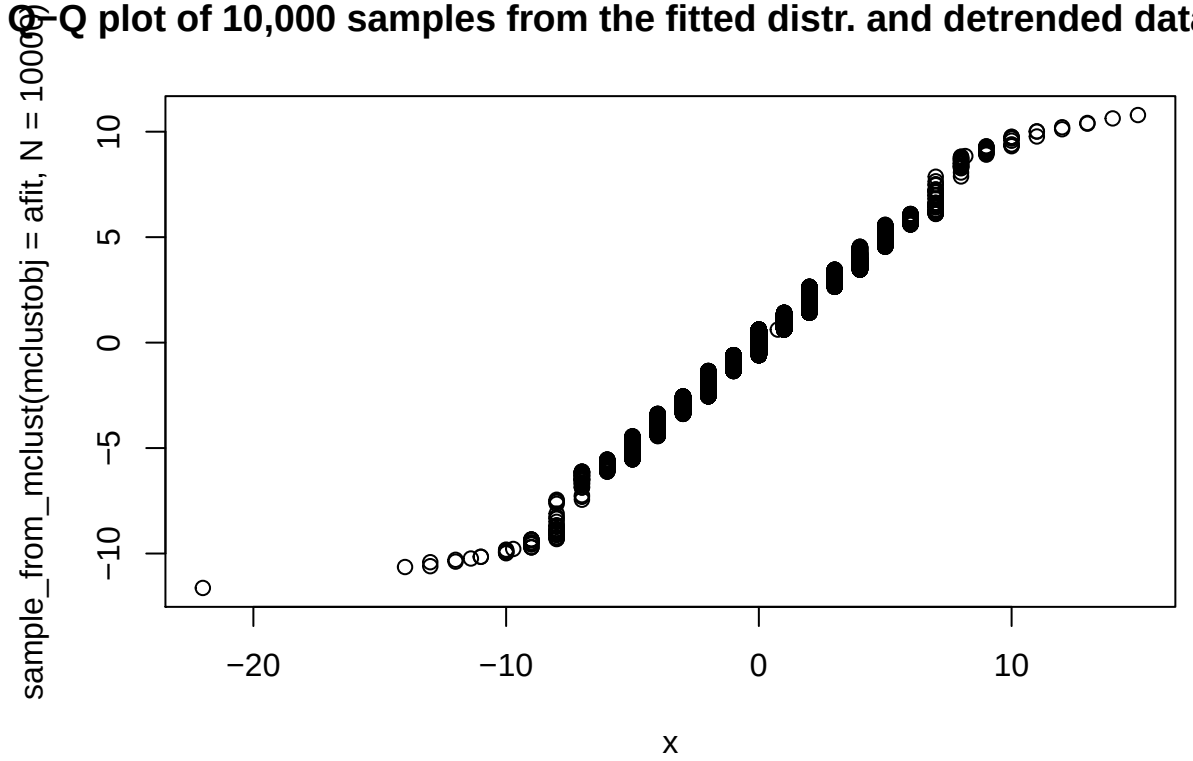
```
## Package 'mclust' version 5.4
## Type 'citation("mclust")' for citing this R package in publications.
```

```
acol='C'
x <- dat1_detrended[[acol]]
afit <- densityMclust(x)
plot(
    afit,
    what="density",
    data=x,
    breaks=50,
    main=paste0('Fitting a mixture of Gaussian distributions to detrended data, col, ', acol, sep='')
)
```



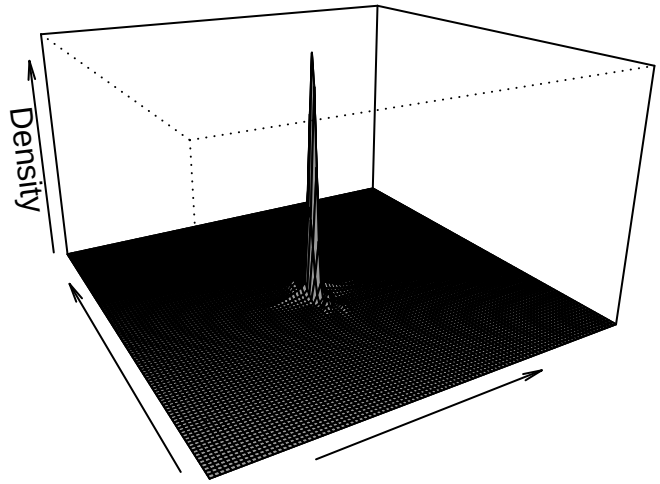
and here is a q-q plot of original and fitted distribution notice that drawing samples from mclust fit is handled by function sample_from_mclust() in library lib/MIXTURES.R

```
qqplot(
    x,
    sample_from_mclust(mclustobj=afit, N=10000),
    main=paste0('Q-Q plot of 10,000 samples from the fitted distr. and detrended data, col, ', acol, sep
)
```



Finally do a multi-variate fit using mclust. We used only 2 columns as it is better to visualise. All variables could have been used.

```
acol1='A'; acol2='B'
dat2 = cbind(
    dat1_detrended[[acol1]],
    dat1_detrended[[acol2]]
)
afit <- densityMclust(dat2)
plot(
    afit,
    what = "density",
    type = "persp",
    main=paste0('Multi-variate mixture fit for detrended data, cols ', acol1, ',', acol2, '.', sep='')
)
```



A mixture model for detrended column C with 6 Gaussians using package Rmixmod.

```
library(Rmixmod)
```

```
## Loading required package: Rcpp
```

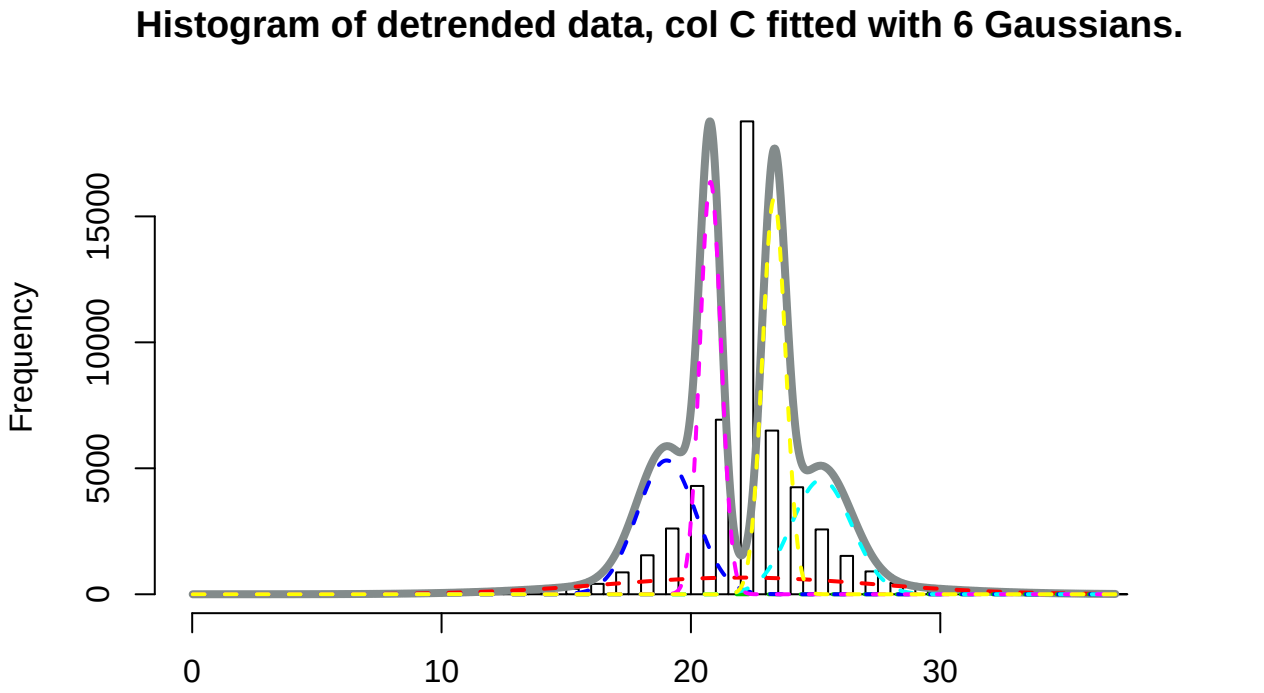
```
## Rmixmod version 2.1.1 loaded
## R package of mixmodLib version 3.2.2
##
## Condition of use
## ----------------
## Copyright (C)  MIXMOD Team - 2001-2013
##
## MIXMOD is publicly available under the GPL license (see www.gnu.org/copyleft/gpl.html)
## You can redistribute it and/or modify it under the terms of the GPL-3 license.
## Please understand that there may still be bugs and errors. Use it at your own risk.
## We take no responsibility for any errors or omissions in this package or for any misfortune that may
##
## Please report bugs at: http://www.mixmod.org/article.php3?id_article=23
##
## More information on : www.mixmod.org
```

```
acol='C'
nGaussians=6
x <- data.frame(data=(dat1_detrended[[acol]] - min(dat1_detrended[[acol]]) + 0.02))
```

```
afit <- mixmodCluster(x, nbCluster=nGaussians)
summary(afit)
```

```
## ****************************************************************
## * Number of samples    =  52672
## * Problem dimension    =  1
## ****************************************************************
## *       Number of cluster =  6
## *               Model Type =  Gaussian_pk_Lk_C
## *                Criterion =  BIC(-722301.6184)
## *               Parameters =  list by cluster
## *                   Cluster  1 :
##                           Proportion =  0.0766
##                                Means =  22.0132
##                            Variances =  28.2639
## *                   Cluster  2 :
##                           Proportion =  0.3564
##                                Means =  22.0200
##                            Variances =  0.0000
## *                   Cluster  3 :
##                           Proportion =  0.1362
##                                Means =  19.0239
##                            Variances =  1.3844
## *                   Cluster  4 :
##                           Proportion =  0.1202
##                                Means =  25.2334
##                            Variances =  1.4724
## *                   Cluster  5 :
##                           Proportion =  0.1513
##                                Means =  20.7855
##                            Variances =  0.1799
## *                   Cluster  6 :
##                           Proportion =  0.1593
##                                Means =  23.3310
##                            Variances =  0.2169
## *          Log-likelihood =  361243.2198
## ****************************************************************
```

```
histCluster(
    x=afit["bestResult"],
    data=x,
    breaks=100,
    # CHECK THIS:
    main=c(paste0('Histogram of detrended data, col ', acol, ' fitted with ', nGaussians, ' Gaussians.'
)
```



Here we estimate a mixture of *different* distribution families For example Gaussian and Gamma for original data, column C

```
library(flexmix)
```

```
## Loading required package: lattice
```

```
adat <- list()
acol='C'
adat[[acol]] = dat1_noid[[acol]]-min(dat1_noid[[acol]])+10
```

we will a mix of 3 Gaussians and 1 Gamma

```
mymodels <- list(
    FLXMRglm(family = "gaussian"),
    FLXMRglm(family = "gaussian"),
    FLXMRglm(family = "gaussian"),
    FLXMRglm(family = "gaussian")
)
```

use the data from column C (original) for k=6 components

```
st<- system.time(
    afit <- flexmix(
        C ~ 1,
        data=adat,
        k=6,
        model=mymodels
    )
)
cat("done, model estimated in ", st[2], " seconds.\n", sep='')
```

```
## done, model estimated in 2.981 seconds.
```

```
print(parameters(afit))
```

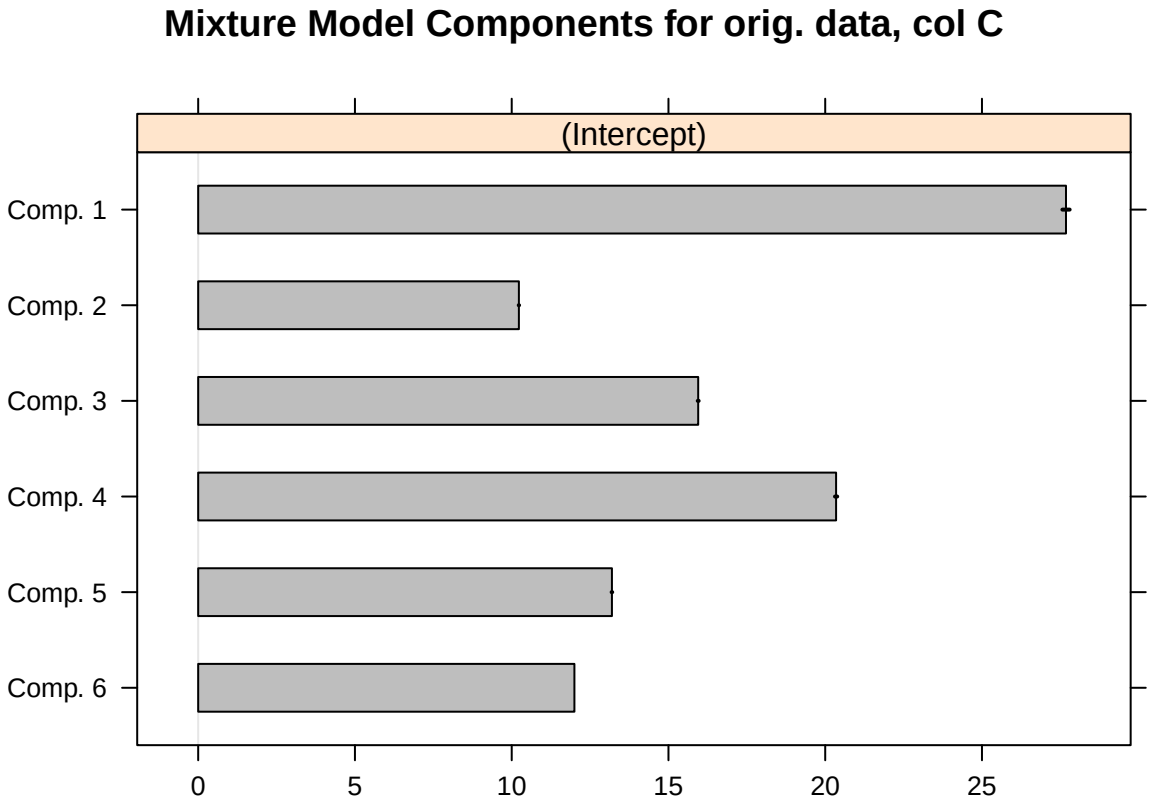
```
## [[1]]
##                      Comp.1     Comp.2     Comp.3    Comp.4     Comp.5
## coef.(Intercept) 27.680767 10.2295131 15.9485114 20.347395 13.1970149
## sigma             3.602226  0.4205237  0.8338333  1.704873  0.3977475
##                      Comp.6
## coef.(Intercept) 1.200000e+01
## sigma            1.198945e-12
##
## [[2]]
##                      Comp.1     Comp.2     Comp.3    Comp.4     Comp.5
## coef.(Intercept) 27.680767 10.2295131 15.9485114 20.347395 13.1970149
## sigma             3.602226  0.4205237  0.8338333  1.704873  0.3977475
##                      Comp.6
## coef.(Intercept) 1.200000e+01
## sigma            1.198945e-12
##
## [[3]]
##                      Comp.1     Comp.2     Comp.3    Comp.4     Comp.5
## coef.(Intercept) 27.680767 10.2295131 15.9485114 20.347395 13.1970149
## sigma             3.602226  0.4205237  0.8338333  1.704873  0.3977475
##                      Comp.6
## coef.(Intercept) 1.200000e+01
## sigma            1.198945e-12
##
## [[4]]
##                      Comp.1     Comp.2     Comp.3    Comp.4     Comp.5
## coef.(Intercept) 27.680767 10.2295131 15.9485114 20.347395 13.1970149
## sigma             3.602226  0.4205237  0.8338333  1.704873  0.3977475
##                      Comp.6
## coef.(Intercept) 1.200000e+01
## sigma            1.198945e-12
```

obtain additional information on the estimated model using refit()

```
arfit <- refit(afit)
```

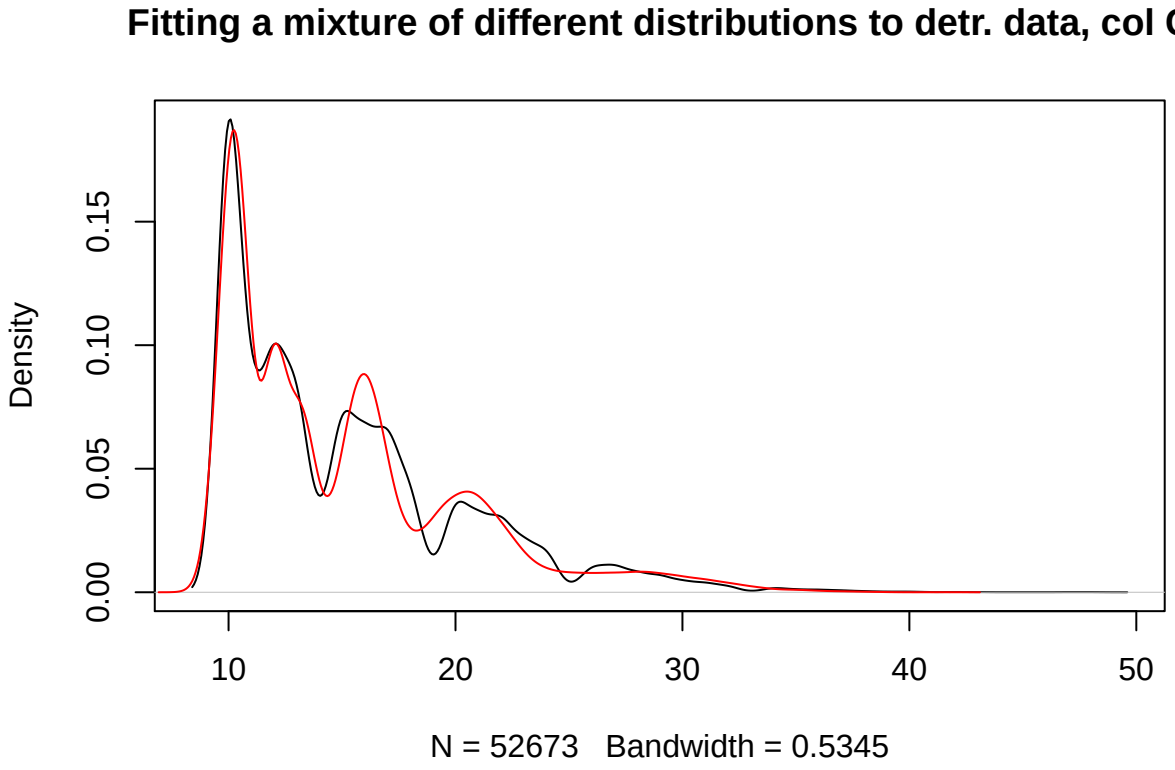
```
## Warning in sqrt(diag(z@vcov)[indices]): NaNs produced

## Warning in sqrt(diag(z@vcov)[indices]): NaNs produced

## Warning in sqrt(diag(z@vcov)[indices]): NaNs produced

## Warning in sqrt(diag(z@vcov)[indices]): NaNs produced
```

```
plot(
    arfit,
    bycluster=F,
    main=paste0('Mixture Model Components for orig. data, col ', acol, sep='')
)
```



compare densities, sample from our estimated model using rflexmix() :

```
sam <- rflexmix(afit)
plot(
    density(adat[[1]]),
    main=paste0('Fitting a mixture of different distributions to detr. data, col ', acol, sep='')
)
lines(density(sam$y[[1]]), col='red', pch=22)
```



not a bad fit at all!

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

Section 4: timeseries analysis : seasonality etc.

```
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'.
```

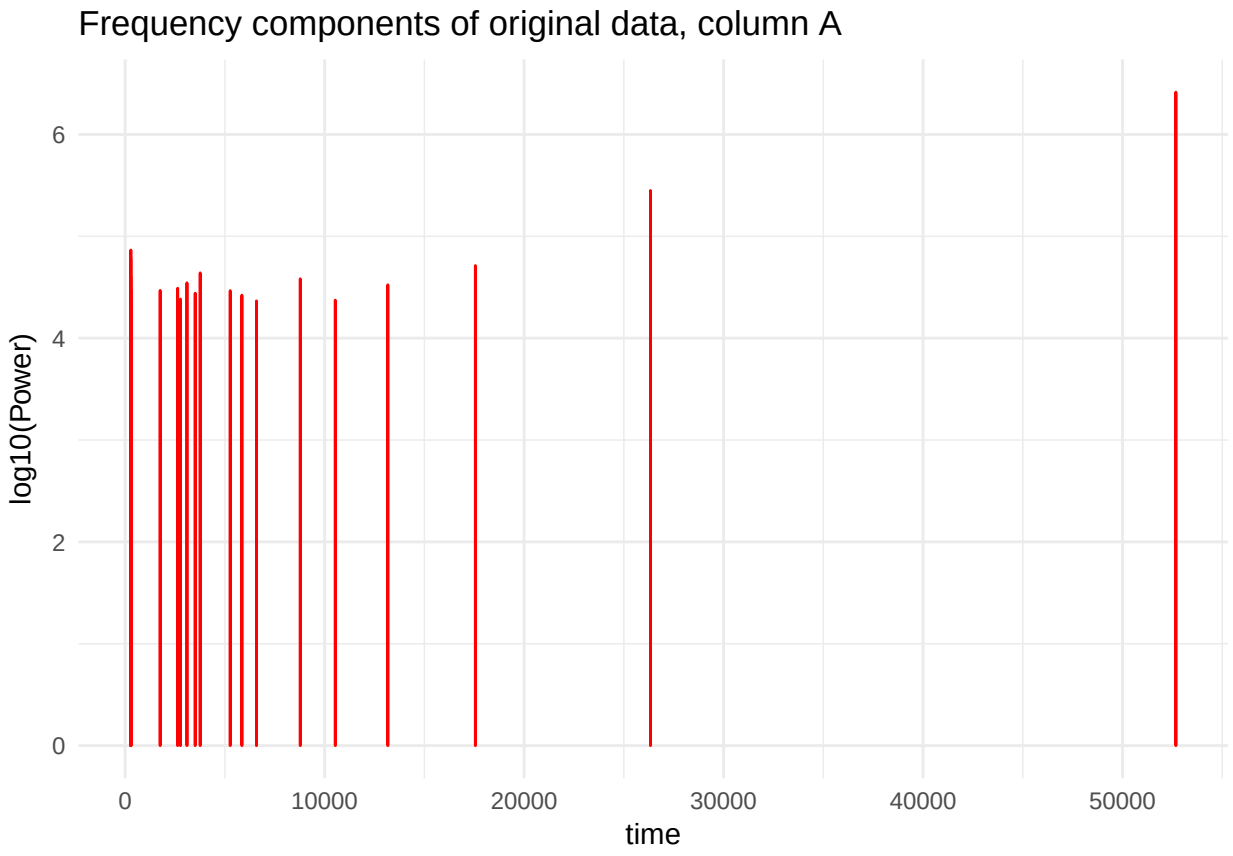
```
if( is.null(dat1) ){
    cat("call to read_data() has failed for file '",infile,"'.\n", sep='')
    quit(status=1)
}
dummy <- remove_columns(inp=dat1, colnames_to_remove=c('id'))
dat1_noid <- dummy[['retained']]
dat1_id <- dummy[['removed']]
```

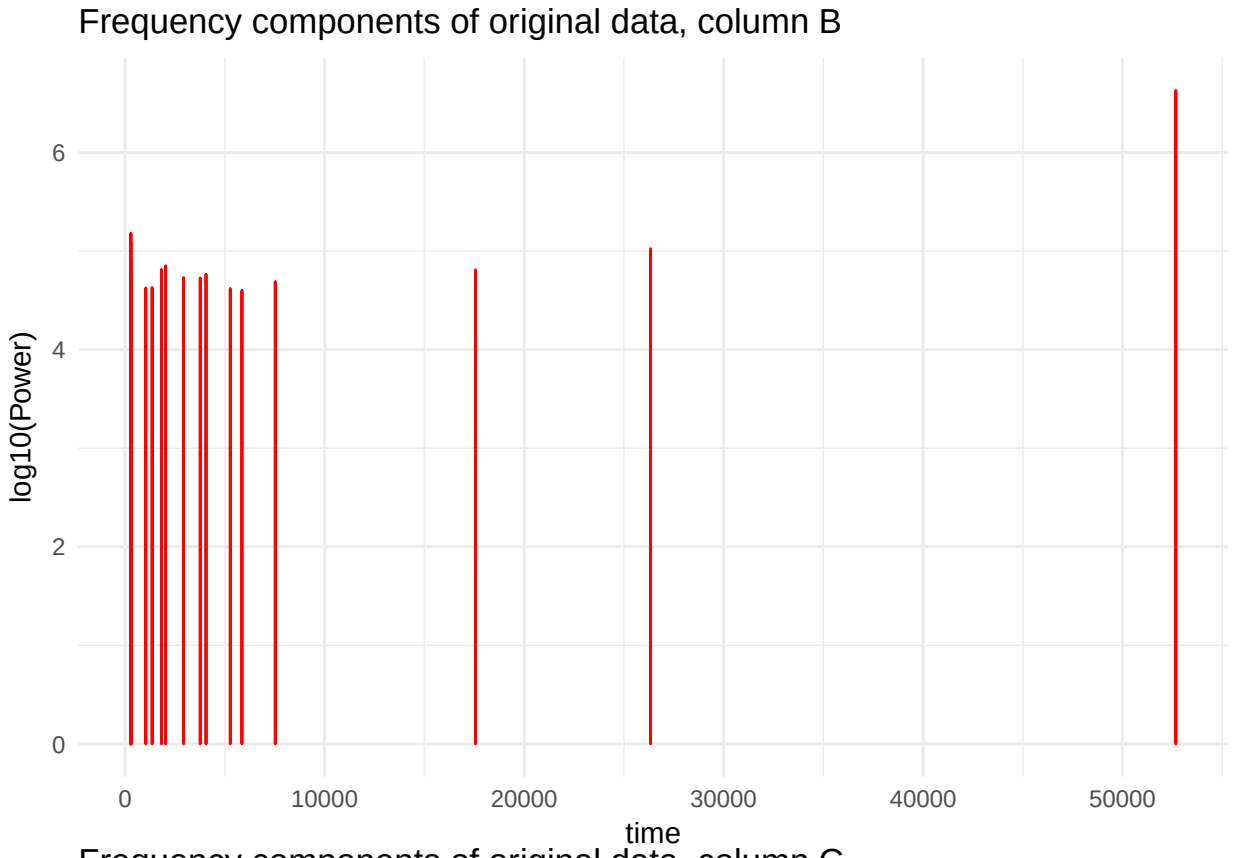
detrend the data

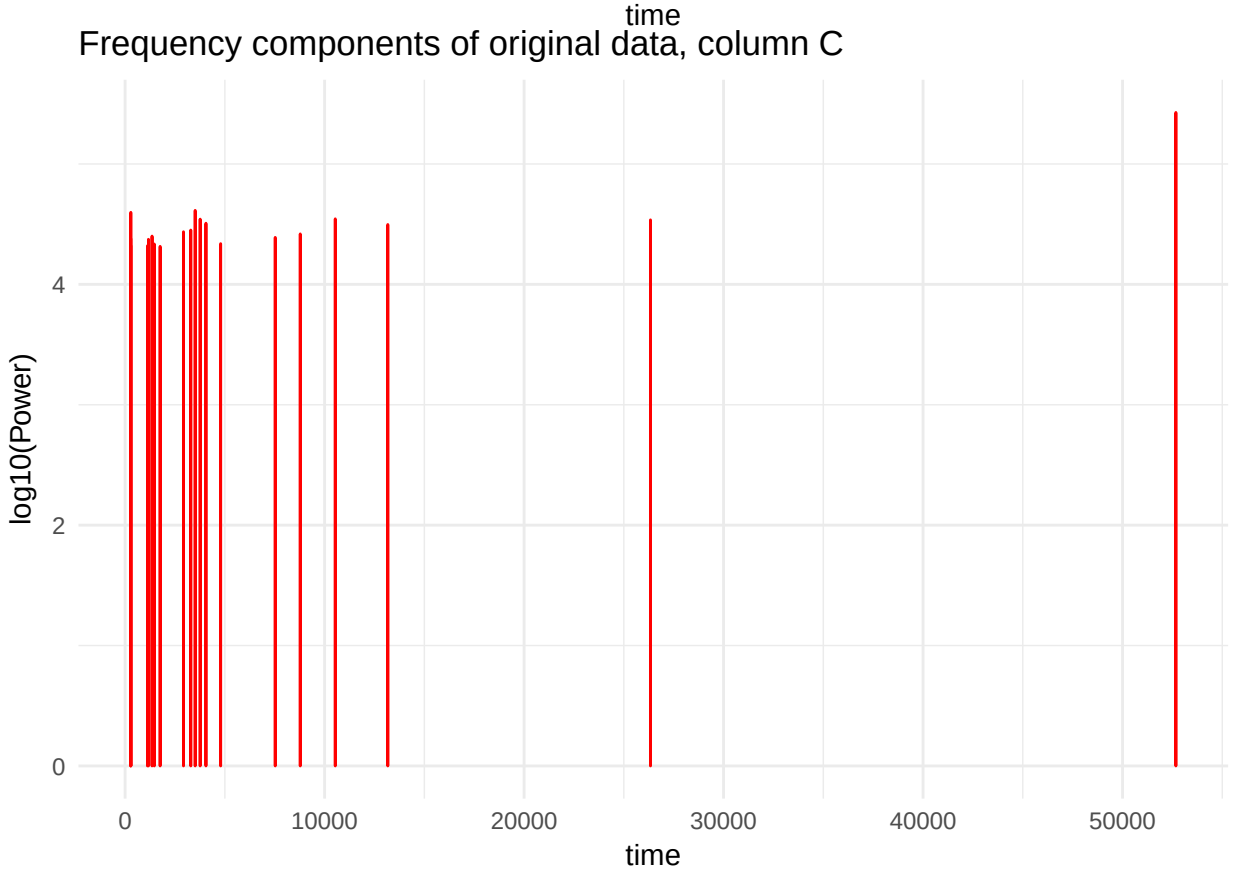
```
dat1_detrended <- detrend_dataset(inp=dat1_noid,times=1)
```

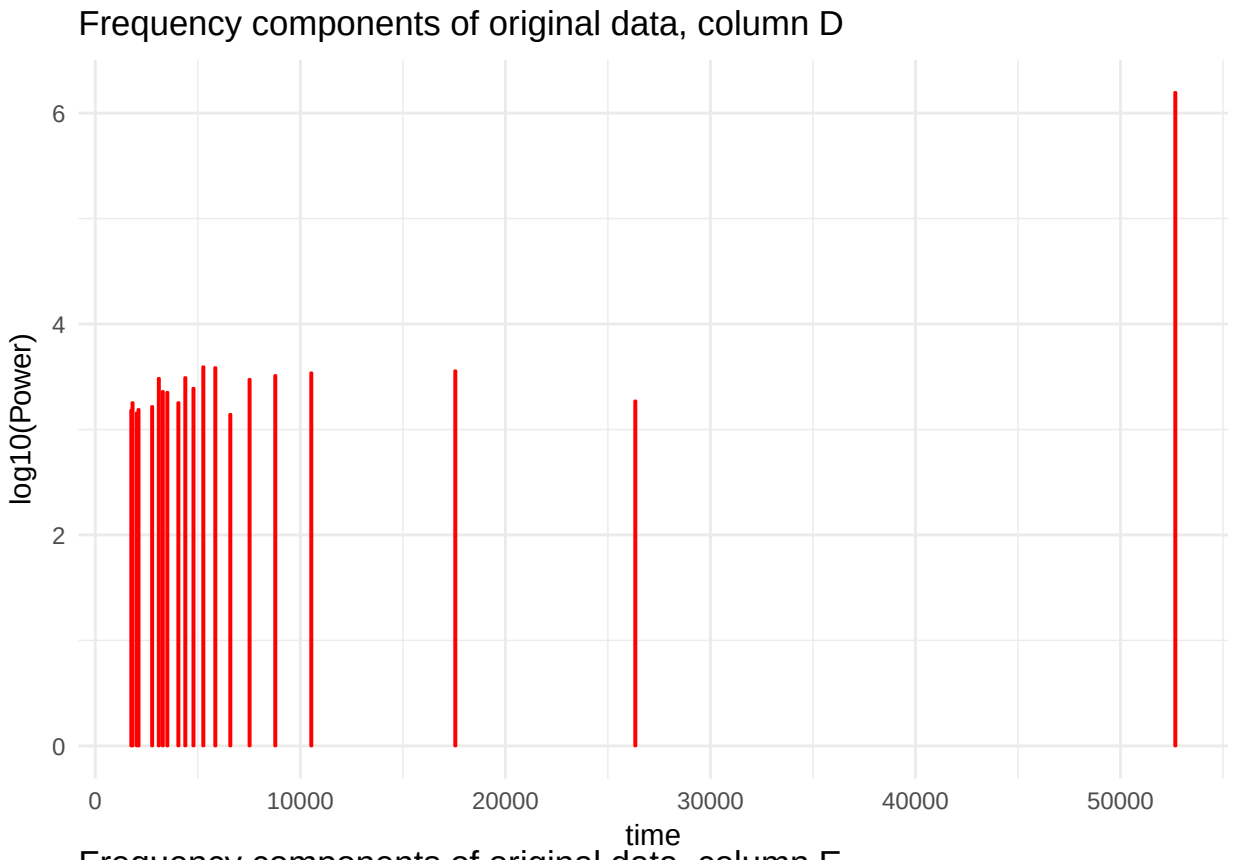
In order to investigate seasonality we will find the significant frequency components of each column of the data using the Fourier Transform.

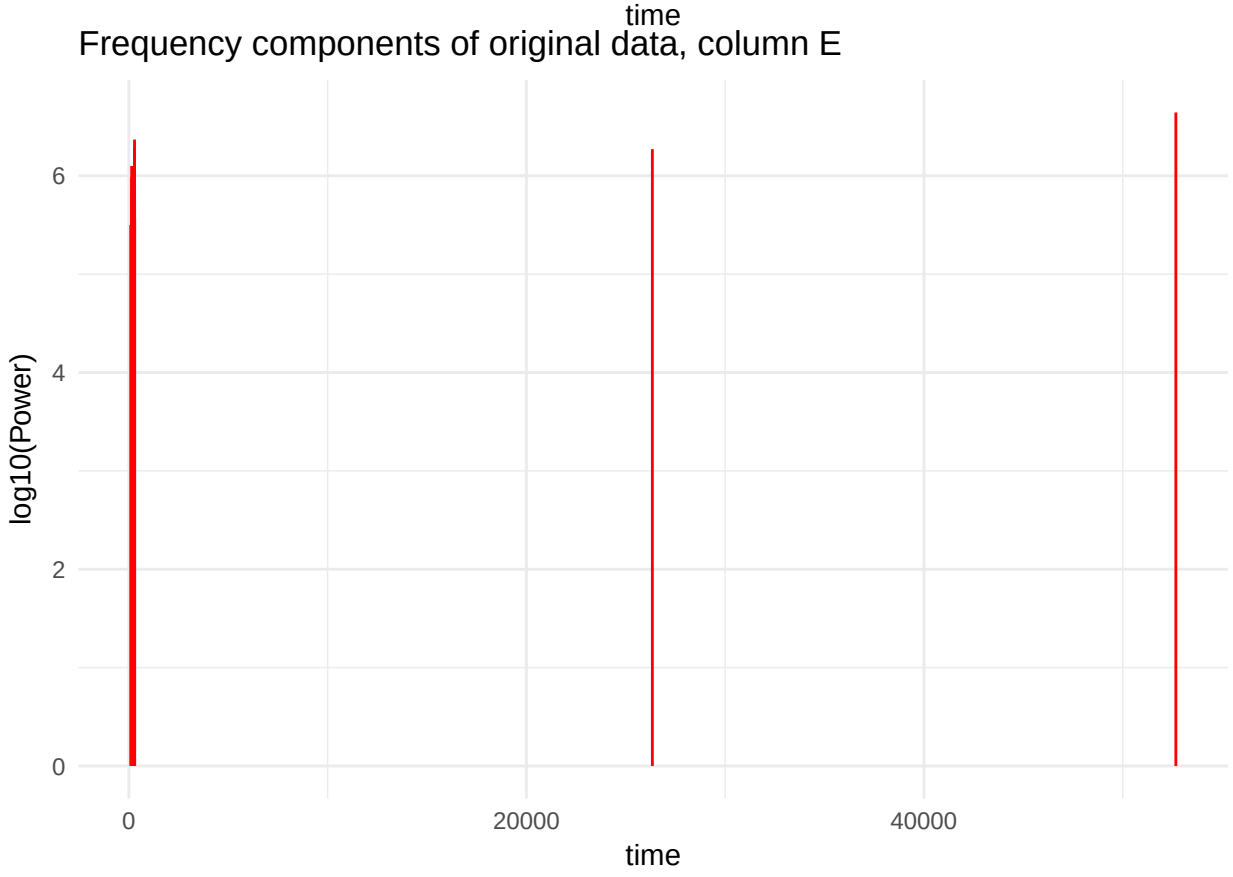
```
dat1_fft <- fourier_transform_analysis(inp=dat1_noid)
dat1_detrended_fft <- fourier_transform_analysis(inp=dat1_detrended)

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)
}
```

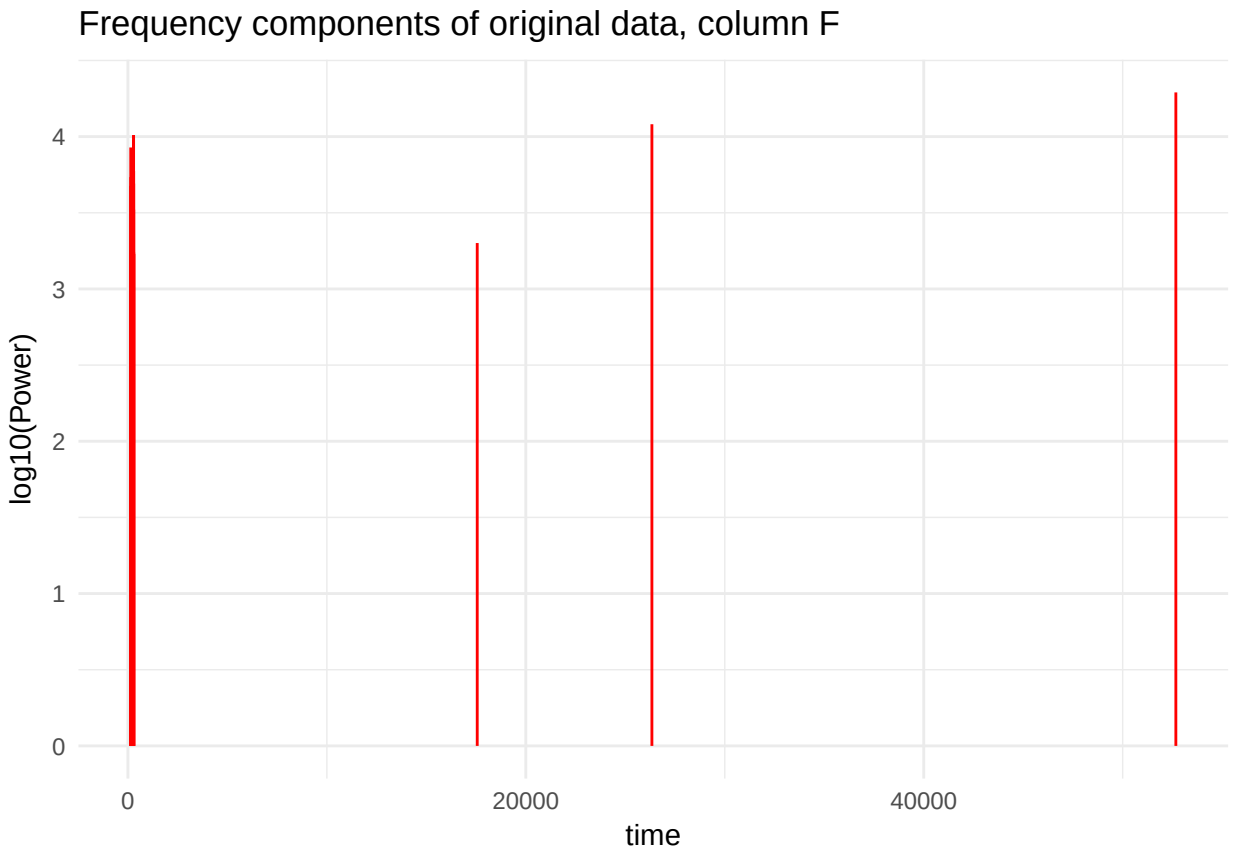



Frequency components of original data, column B
log10(Power)
0
2
4
6
0
10000
20000
30000
40000
50000
time

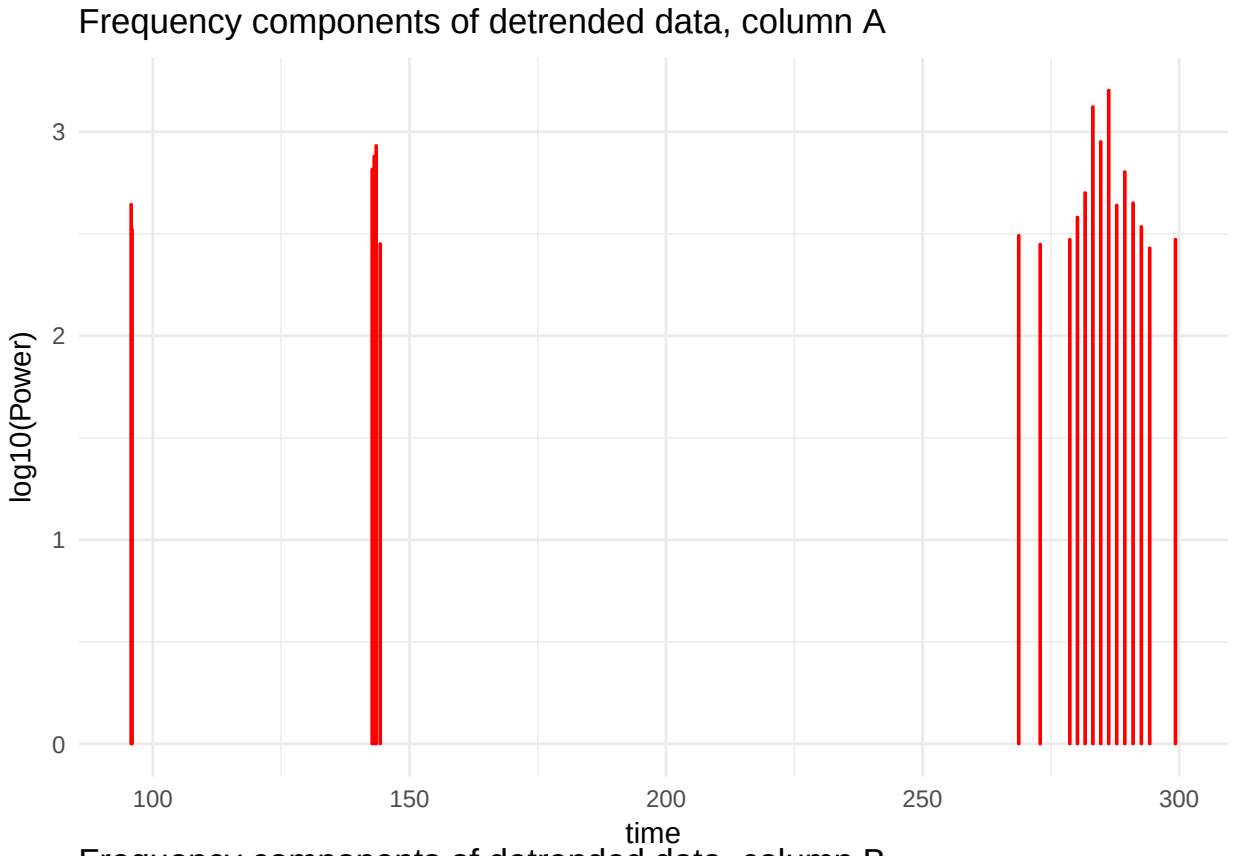

Frequency components of original data, column C
log10(Power)
0
2
4
0
10000
20000
30000
40000
50000
time

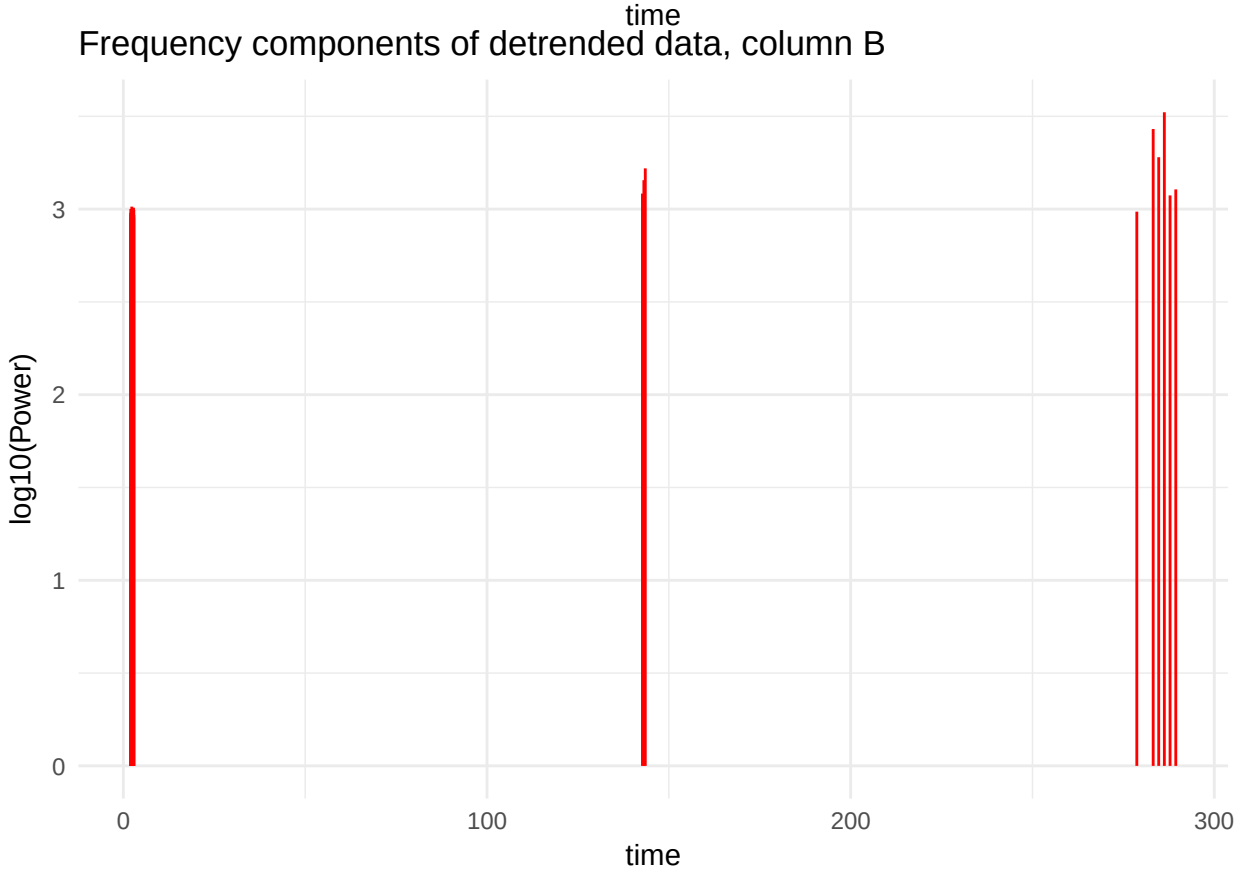

Frequency components of original data, column D
log10(Power)
0
2
4
6
0
10000
20000
30000
40000
50000
time

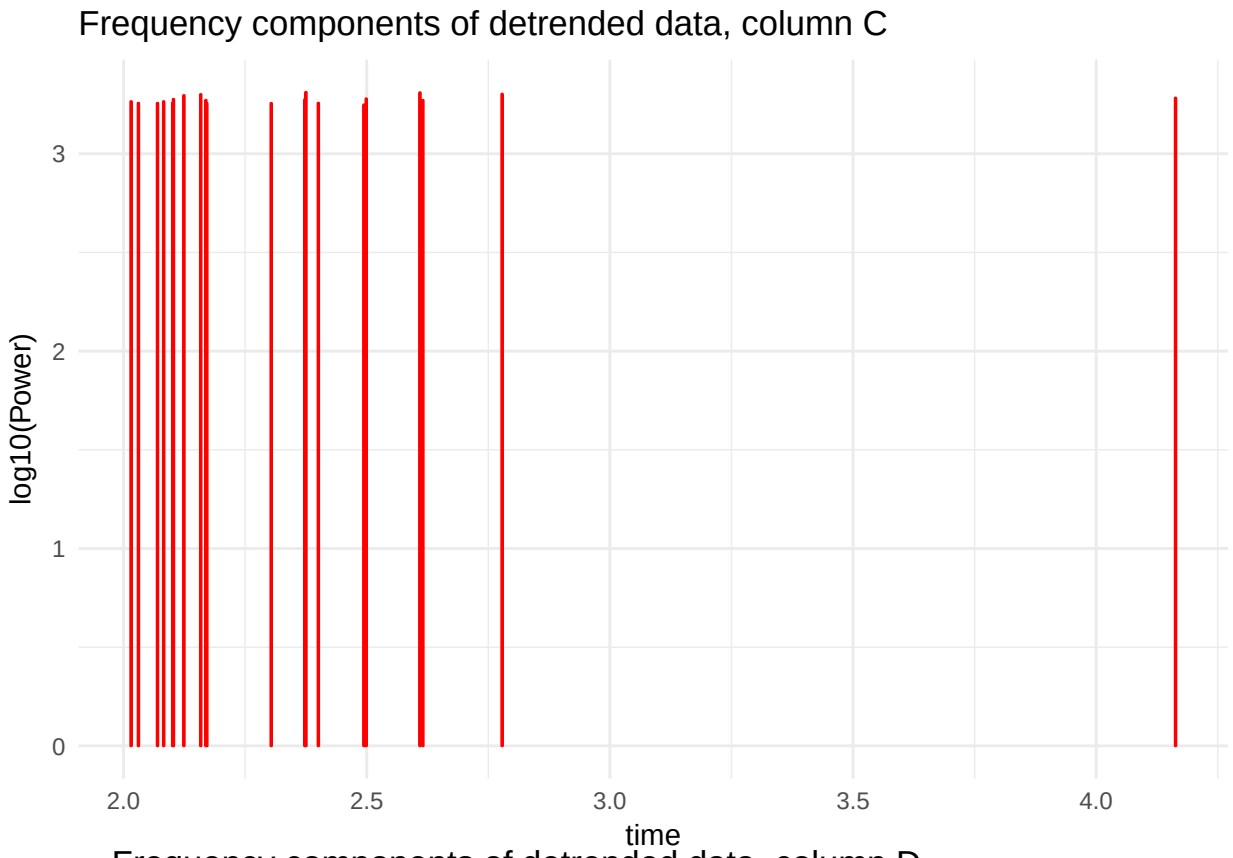

Frequency components of original data, column E
log10(Power)
0
2
4
6
0
20000
40000
time

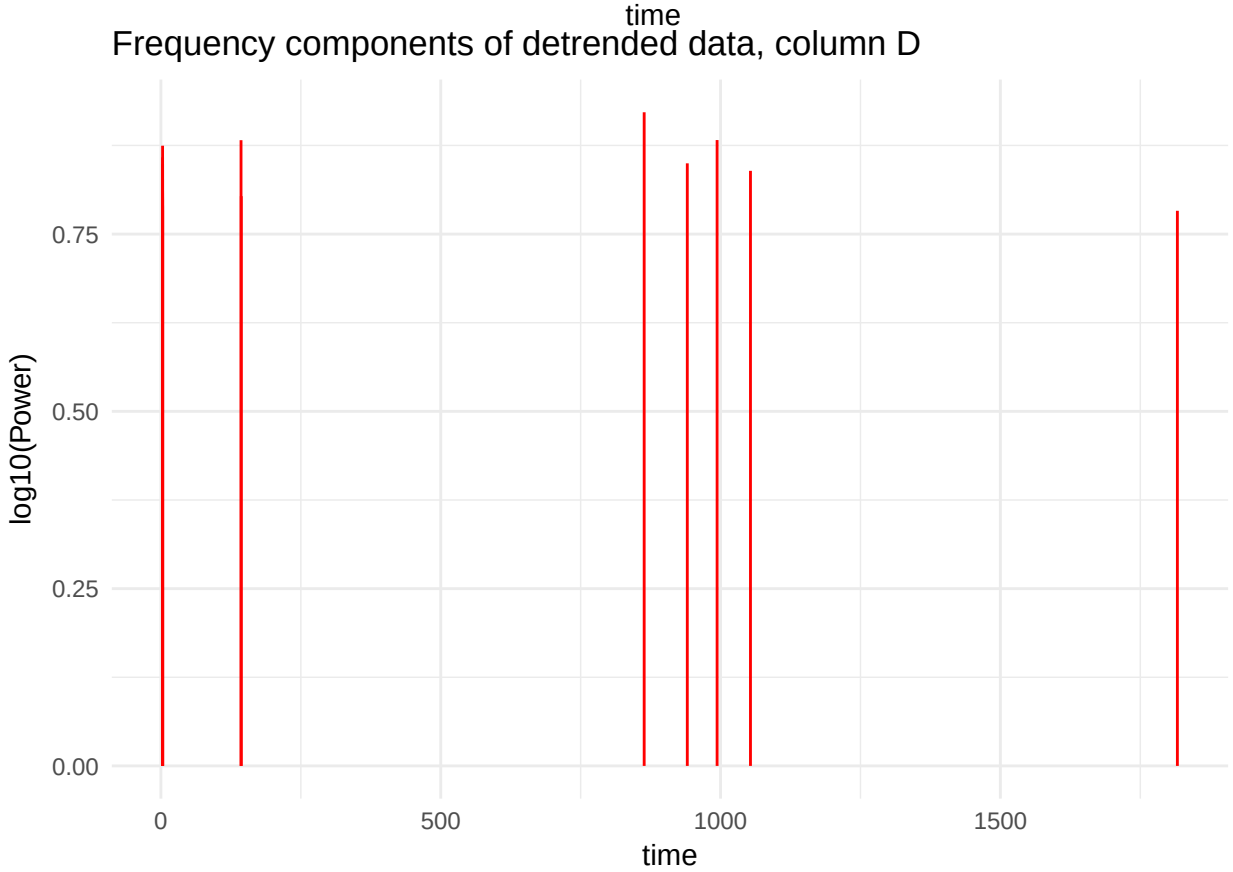


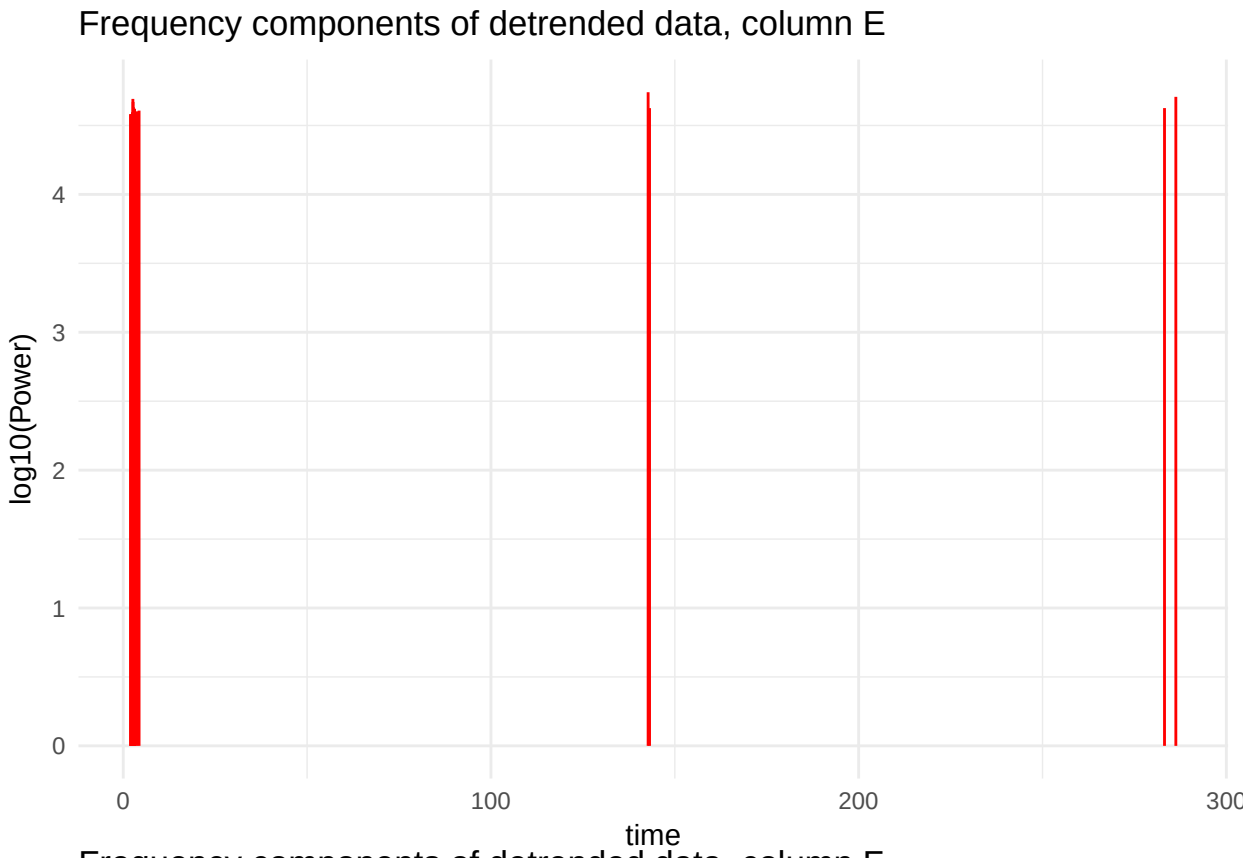

```
for(acol in names(dat1_detrended_fft)){
    ag <- ggplot(
        as.data.frame(
            tail(dat1_detrended_fft[[acol]],20)
        ),
        aes(x=time,y=log10power)
    )+geom_bar(stat="identity", color='red')+
    ggtitle(paste0("Frequency components of detrended data, column ", acol, sep=''))+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

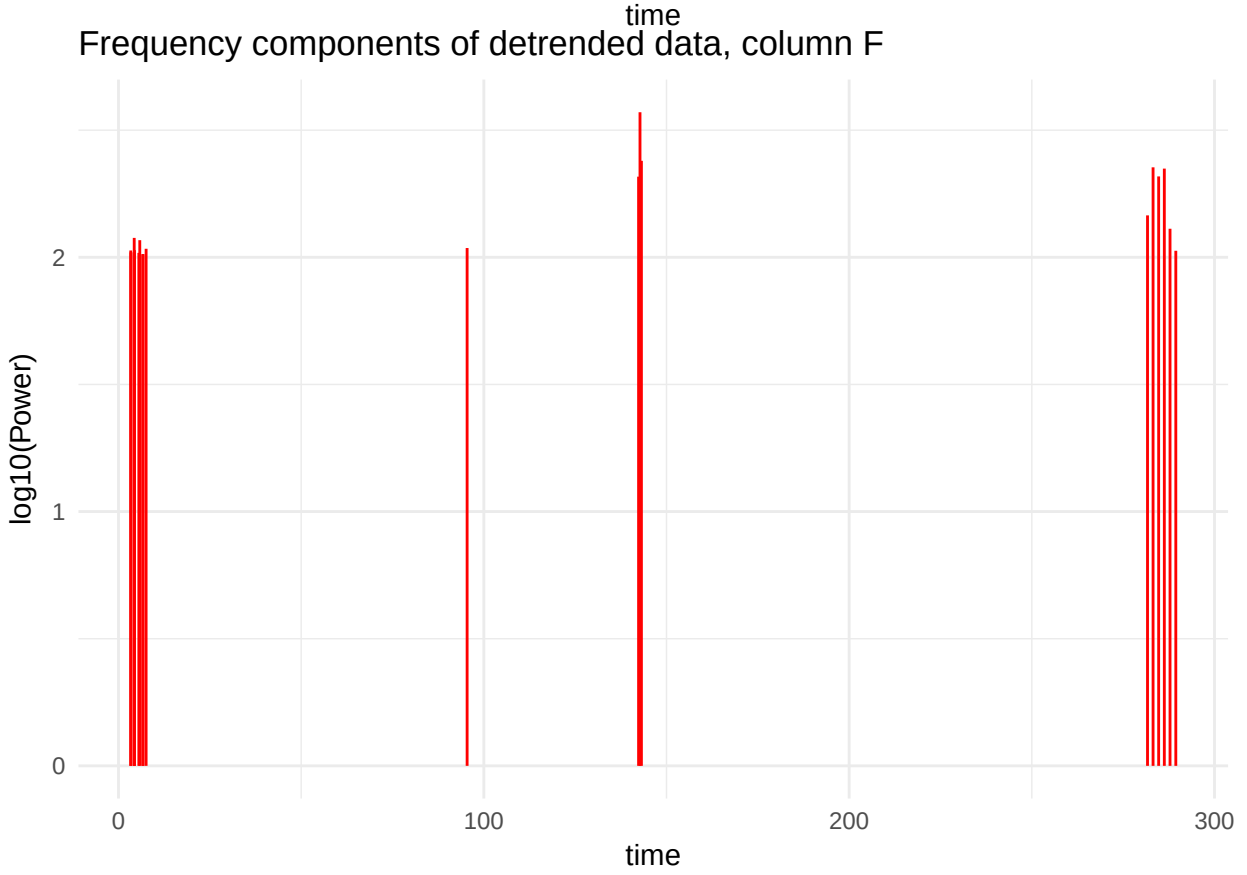

Frequency components of detrended data, column A
log10(Power)
0
1
2
3
100
150
200
250
300
time


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.
```

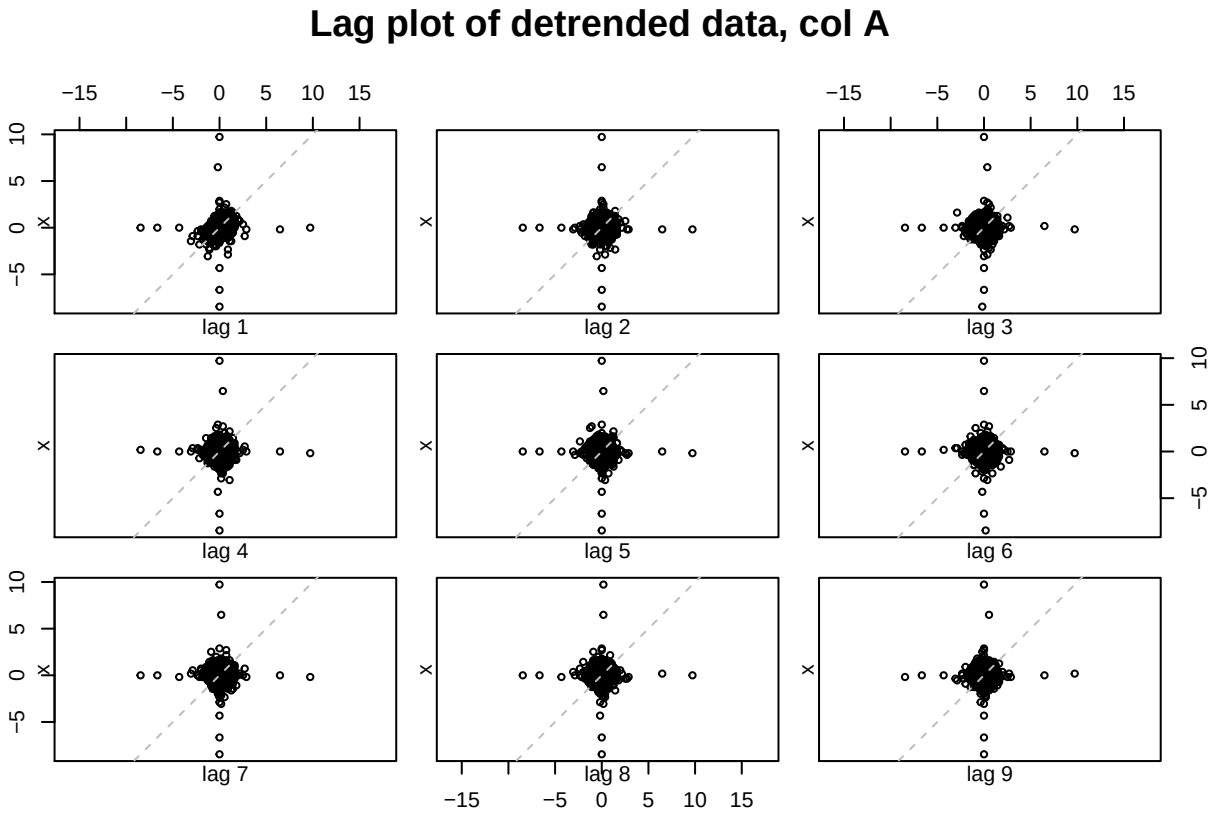
we will investigate whether the data is generated by an auto-regressive (AR) model meaning that data value at time point t(0) depends linearly on some previous data values (for example, t(-1) and t(-2)) plus a random term. In other words, data past values have an effect to current and future values. 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")
```

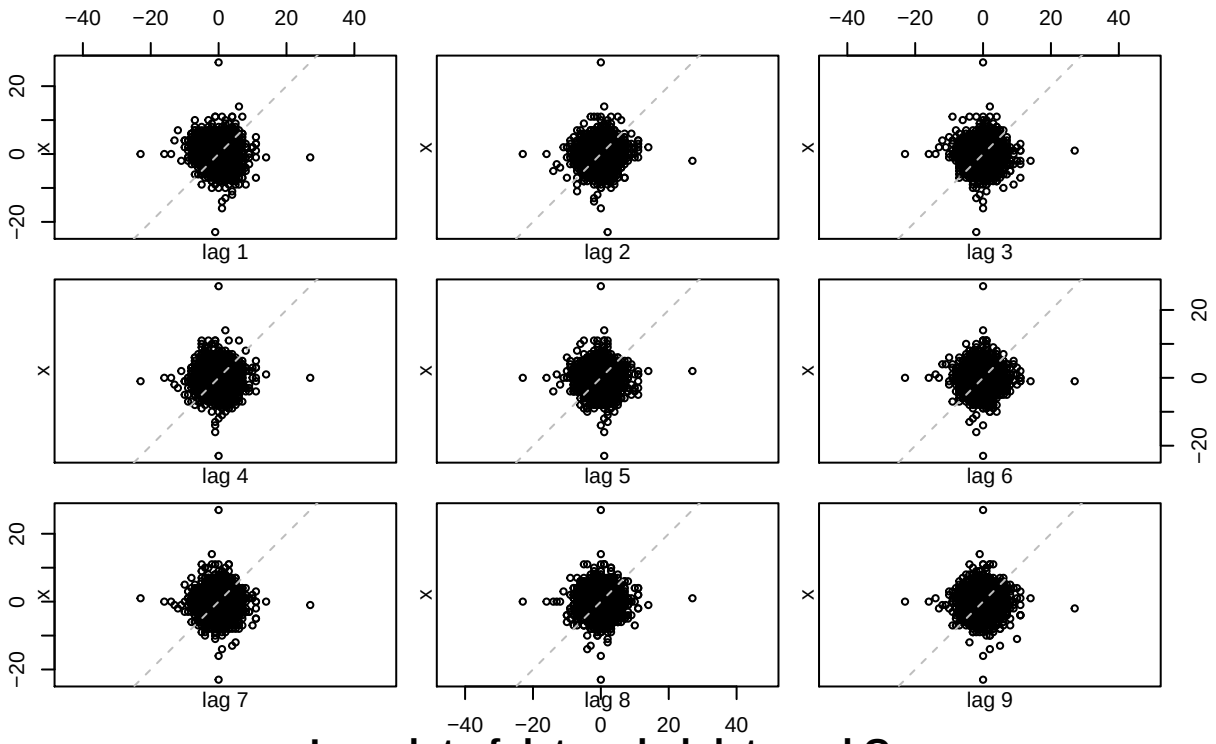
```
## [1] "lag plots of the detrended data"
```

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

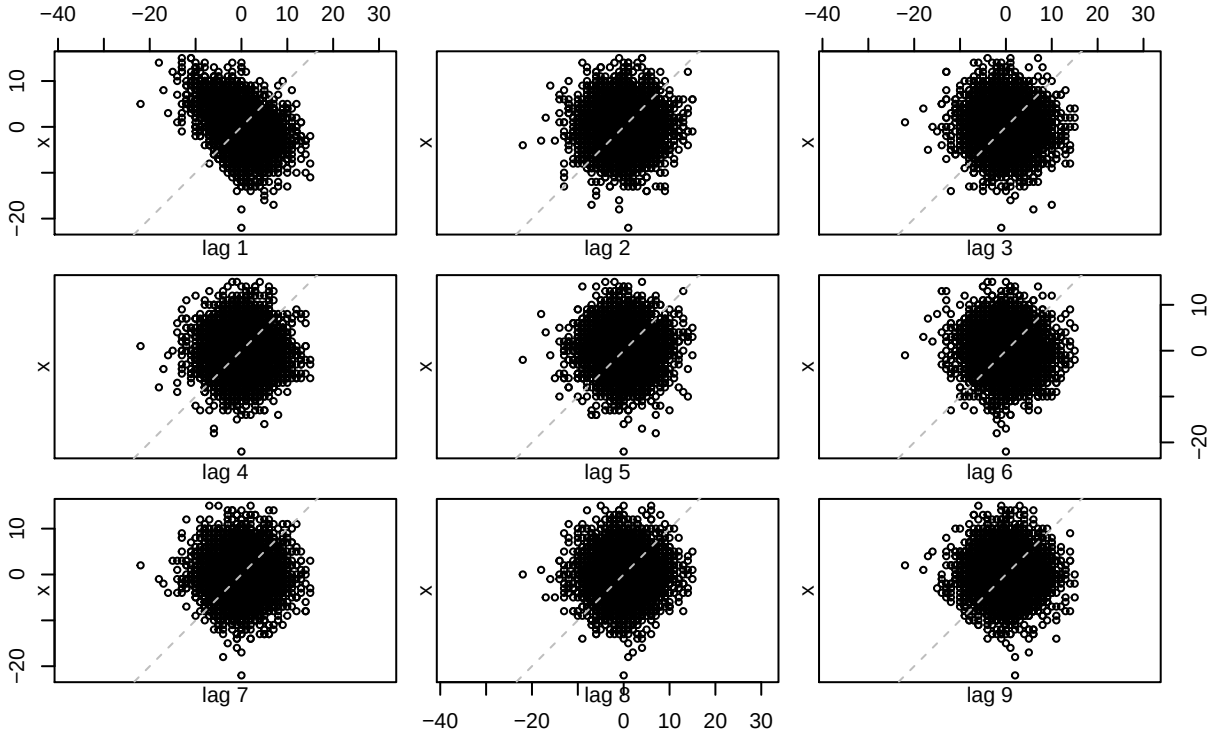
```
        main=paste0("Lag plot of detrended data, col ", acol, sep='')
    )
}
```



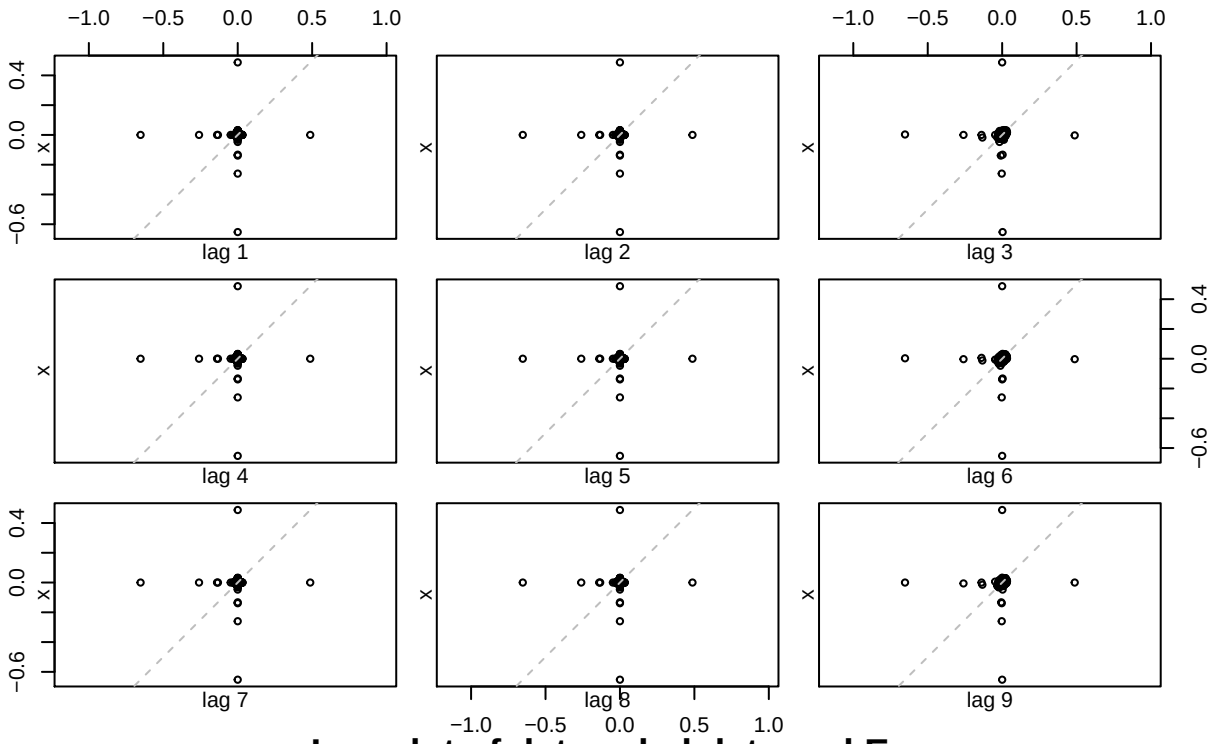
Lag plot of detrended data, col B



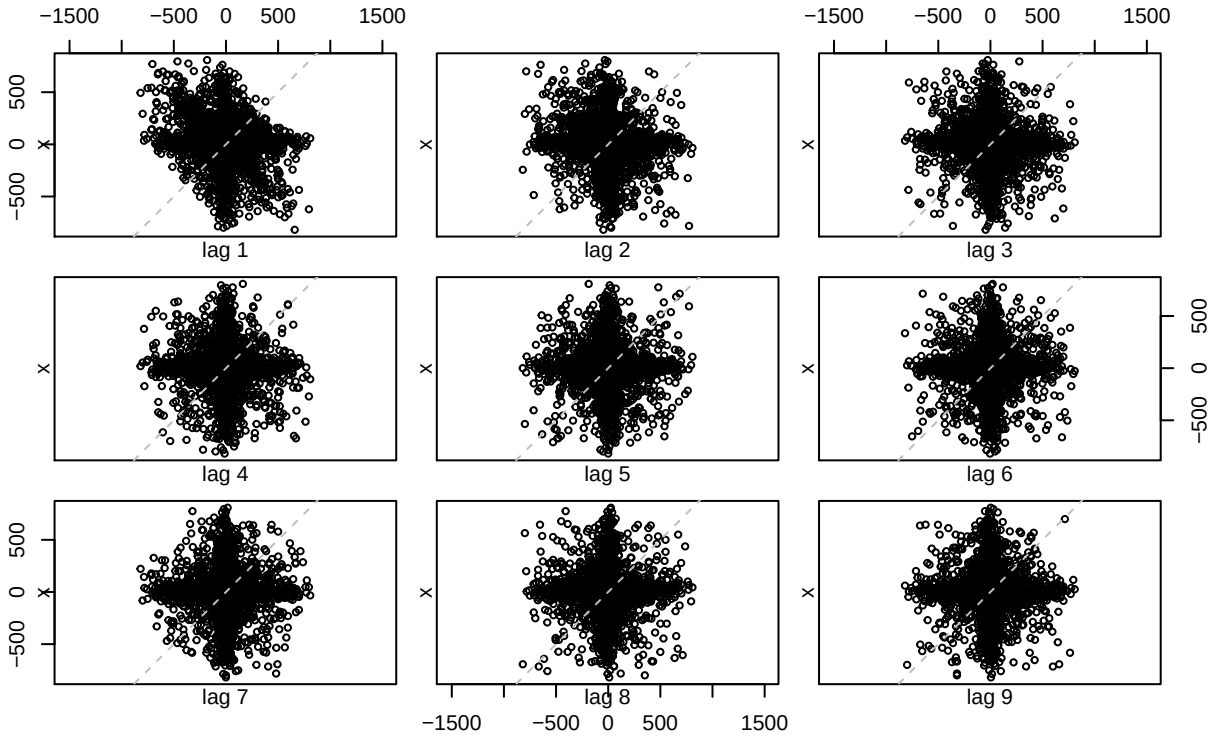
Lag plot of detrended data, col C

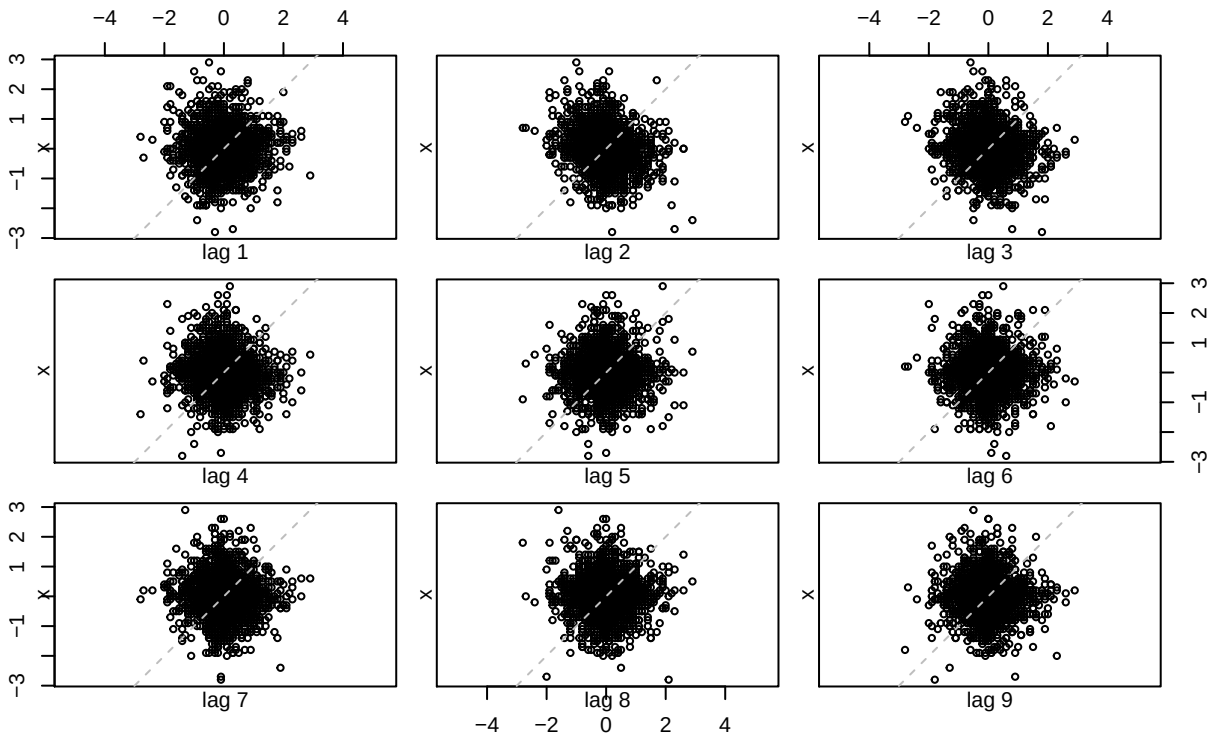


Lag plot of detrended data, col D



Lag plot of detrended data, col E



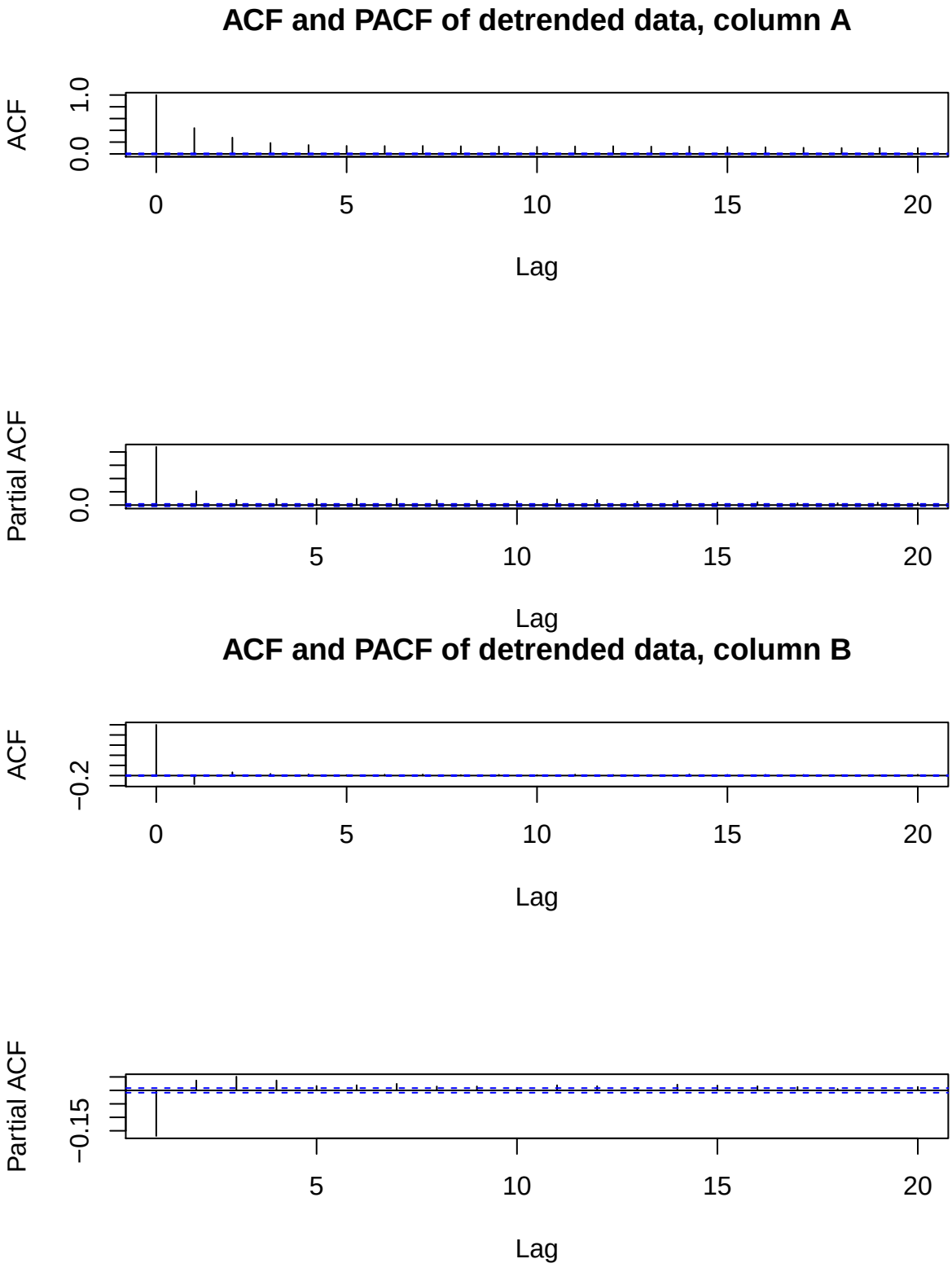


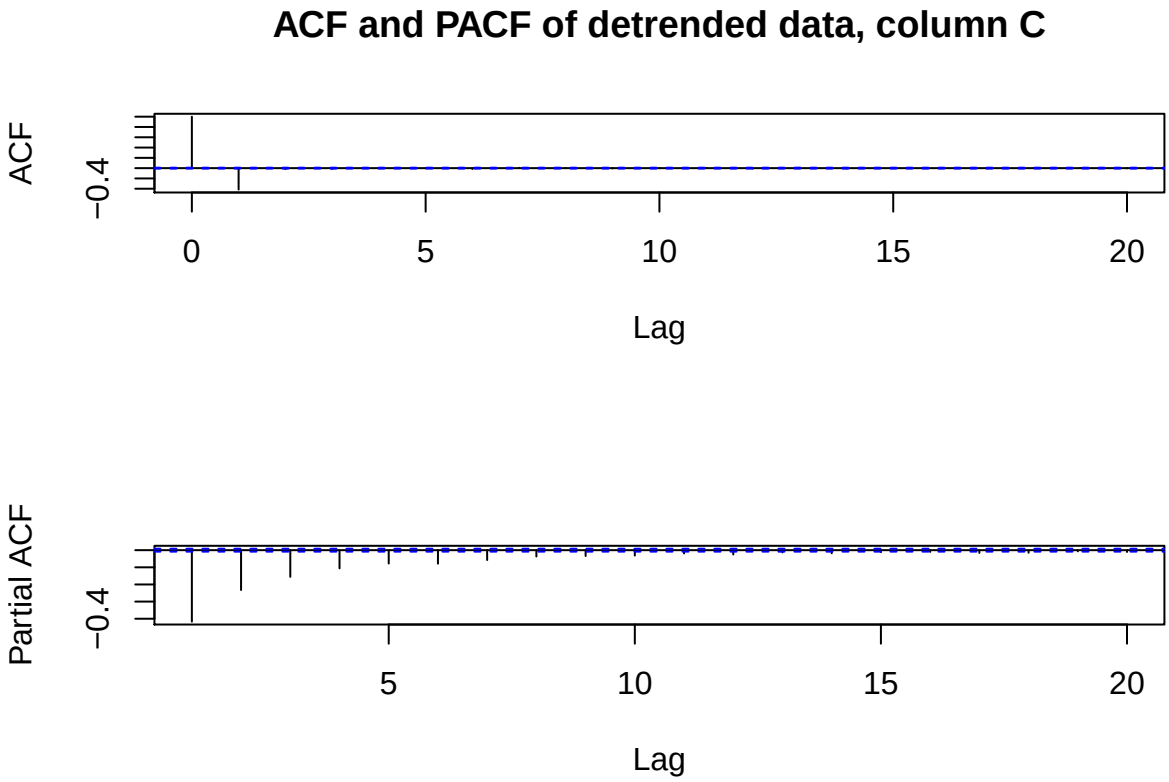
ACF is the correlation between points of our data lagged by some time period. E.g. for a time period equal to 1, we get the correlation between successive (in time) data points. The Partial ACF is the same as ACF but when all the shorter-term correlations are removed. If the PACF produces spikes which 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).

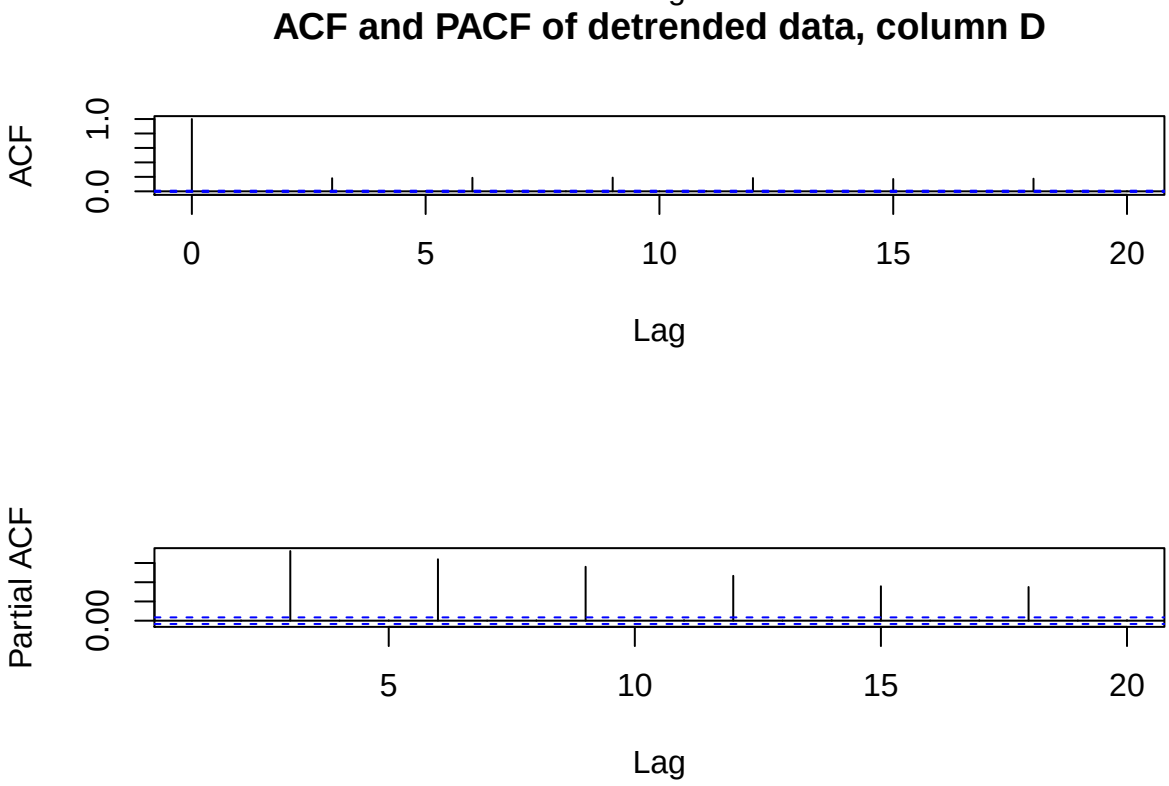
```
print("lag plots of the detrended data")
```

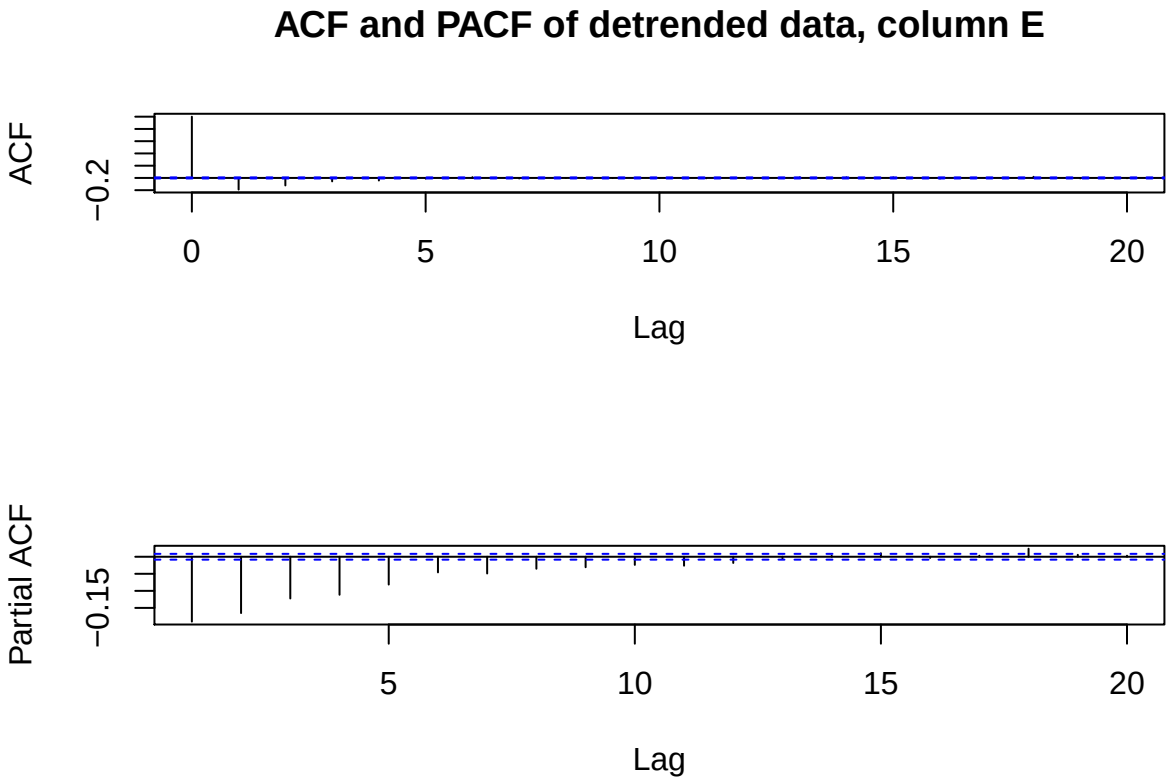
```
## [1] "lag plots of the detrended data"
```

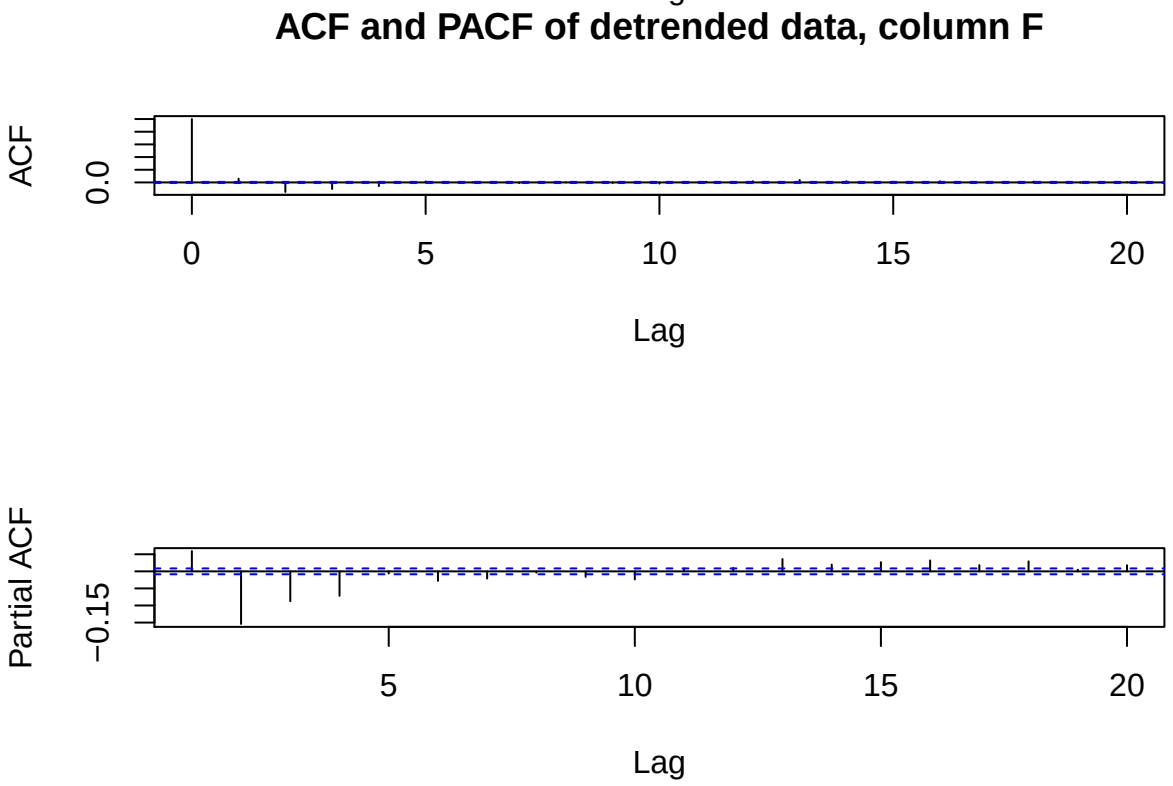
```
for(acol in names(dat1_detrended)){
    x <- dat1_detrended[[acol]]
    par(mfrow=c(2,1))
    acf(
        x=ts(x, freq=1),
        lag.max=20,
        main=paste0("ACF and PACF of detrended data, column ", acol, sep='')
    )
    pacf(
        x=ts(x, freq=1),
        lag.max=20,
        main=""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


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)
```

```
##
```

```
## Attaching package: 'forecast'
```

```
## The following object is masked from 'package:ggpubr':
##
##     gghistogram
```

```
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)

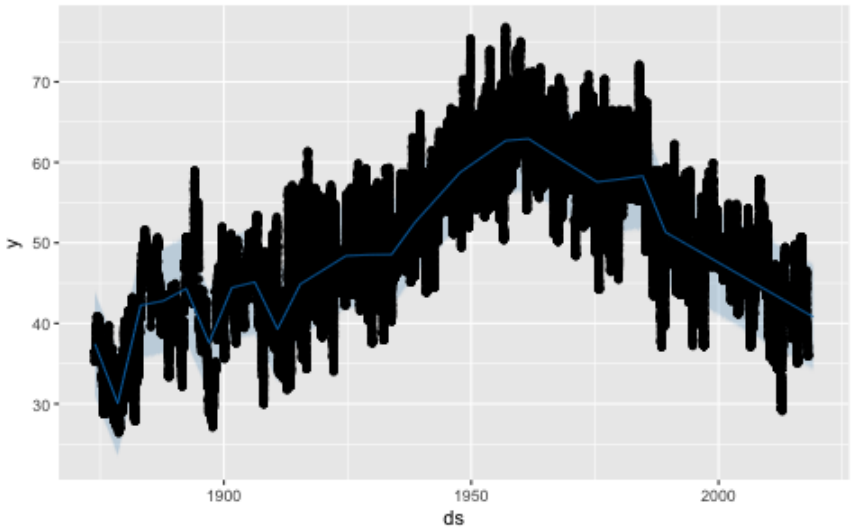
```
future <- readRDS('prophet/prophet_dat1_noid_A.future.Rds')
```

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

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

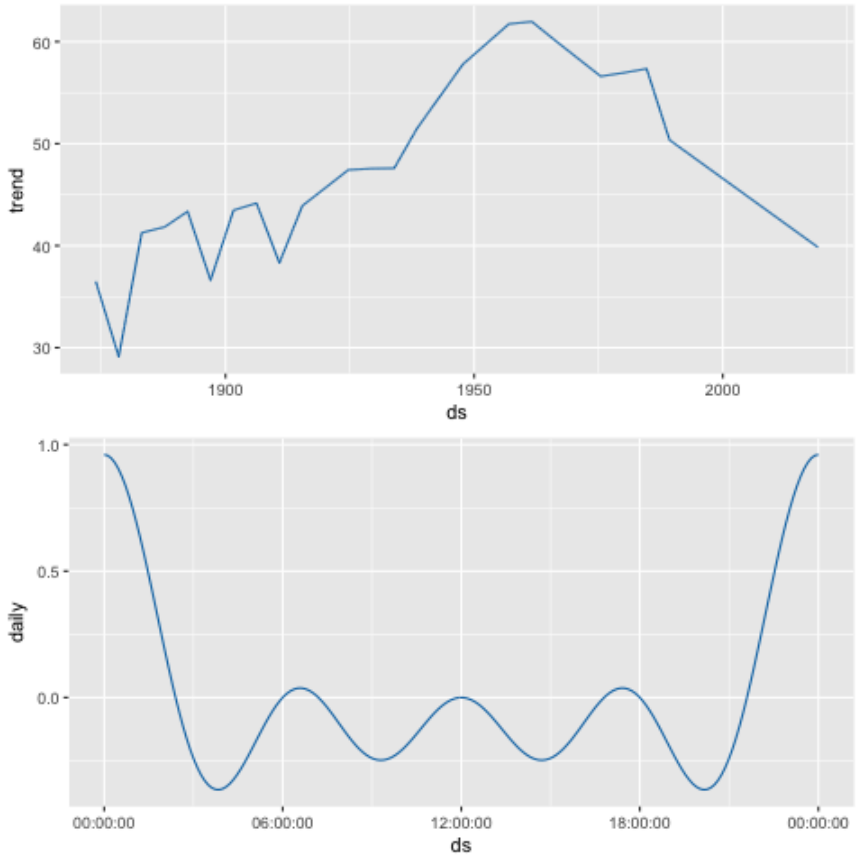
plot the forecast plot(m, forecast)

```
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)
```




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

Section 5: PCA analysis

```
set.seed(1234)

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']]
dat1_detrended_id <- list(c(dat1_id[['id']][2:length(dat1_id[['id']])]))
names(dat1_detrended_id) <- c('id')
# detrend the data
dat1_detrended <- detrend_dataset(inp=dat1_noid,times=1)
```

PCA In preparing this report I have followed the route of http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/112-pca-principal-component-analysis-essentials/

```
library(FactoMineR)
library(factoextra)
```

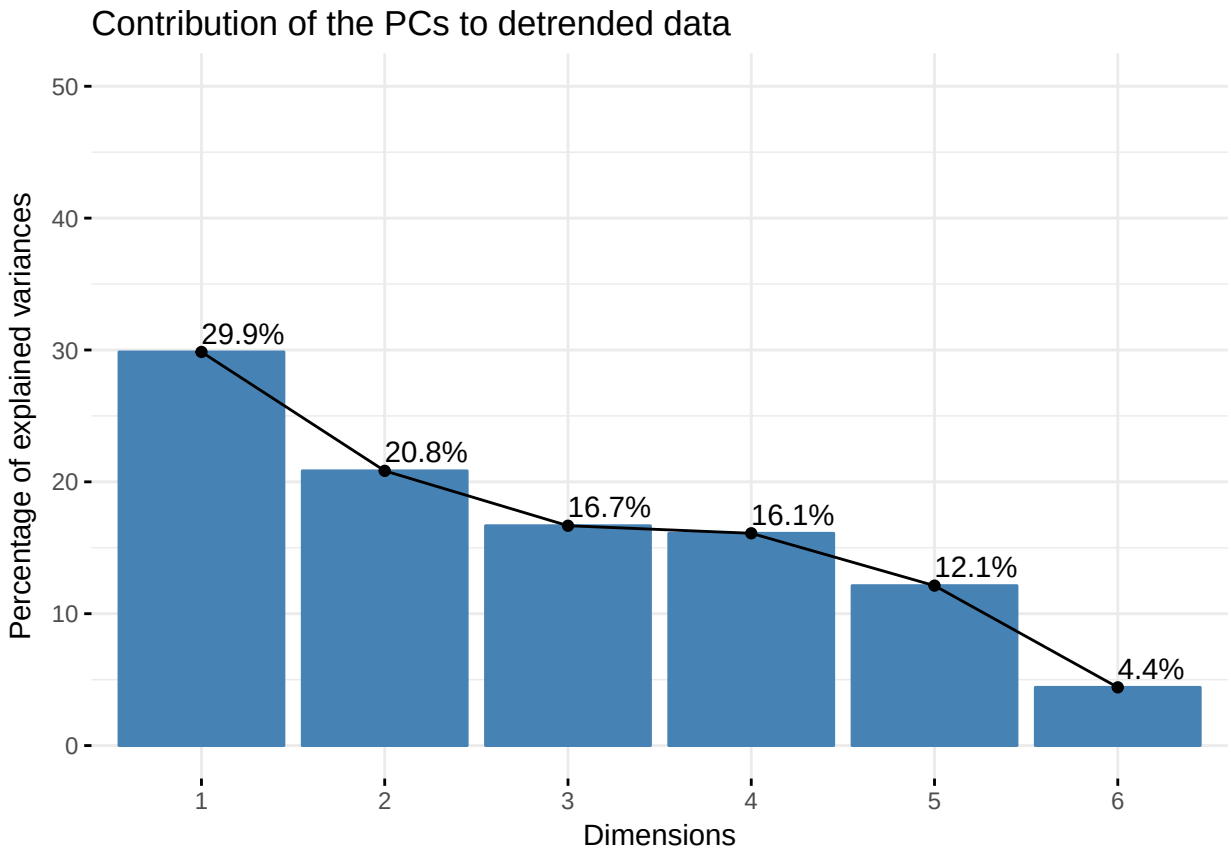
```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```
adf <- list2dataframe(dat1_detrended)
ncols=ncol(adf)
pcaobj <- PCA(
    adf,
    scale.unit=T, # unit variance, zero mean
    graph=F,
    ncp=ncols
)
print(get_eigenvalue(pcaobj))
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  1.7910283        29.850472                    29.85047
## Dim.2  1.2500565        20.834275                    50.68475
## Dim.3  1.0003289        16.672148                    67.35690
## Dim.4  0.9659387        16.098978                    83.45587
## Dim.5  0.7277402        12.129003                    95.58488
## Dim.6  0.2649074         4.415123                   100.00000
```

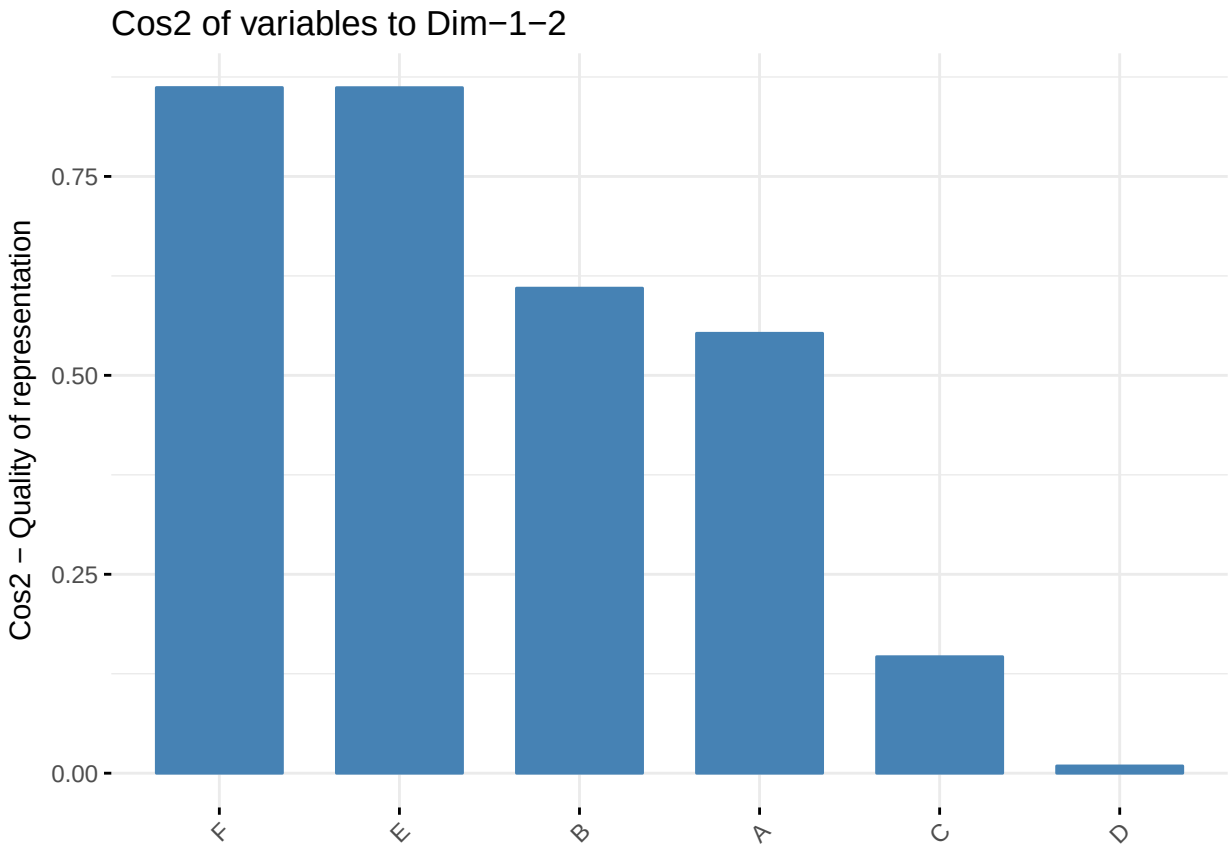
it does not look like there are redundant variables perhaps with the exception of the last component which only contributes 4% to the total variance. If we had to reduce the dimensionality of the original data I would recommend keeping the first 4 components. visualisng the contribution of each component to the variance of the data:

```
fviz_eig(pcaobj, addlabels = TRUE, ylim = c(0, 50), main='Contribution of the PCs to detrended data')
```

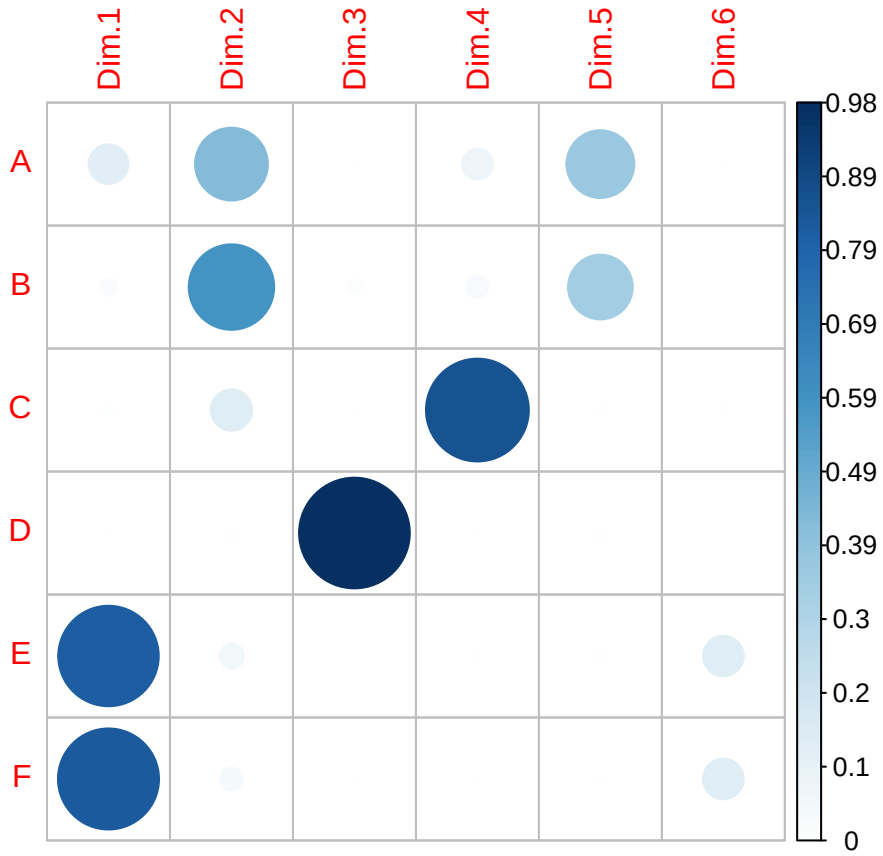


A plot of how is each variable represented by each principal component:

```
library("corrplot")
fviz_cos2(pcaobj, choice = "var", axes = 1:2)
```

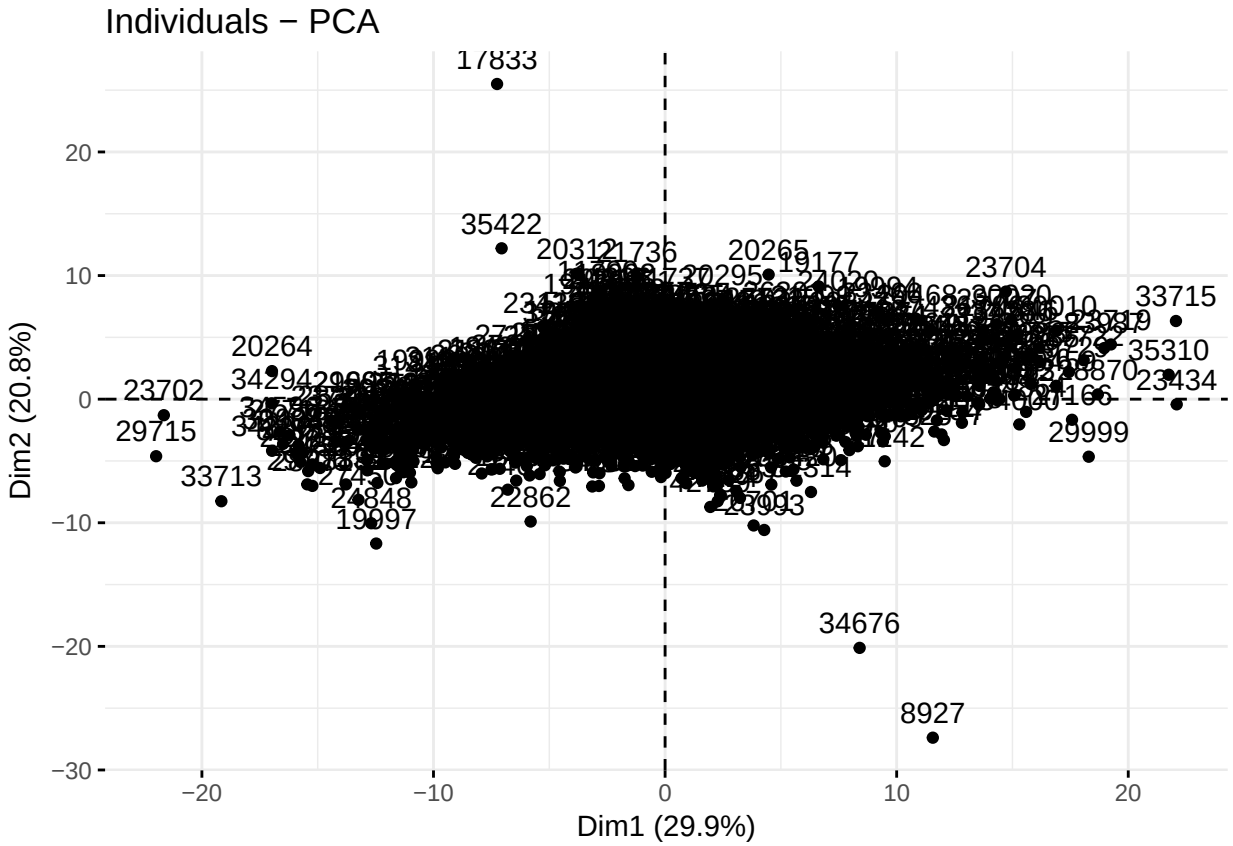



```
var <- get_pca_var(pcaobj)
corrplot(var$cos2, is.corr=FALSE)
```



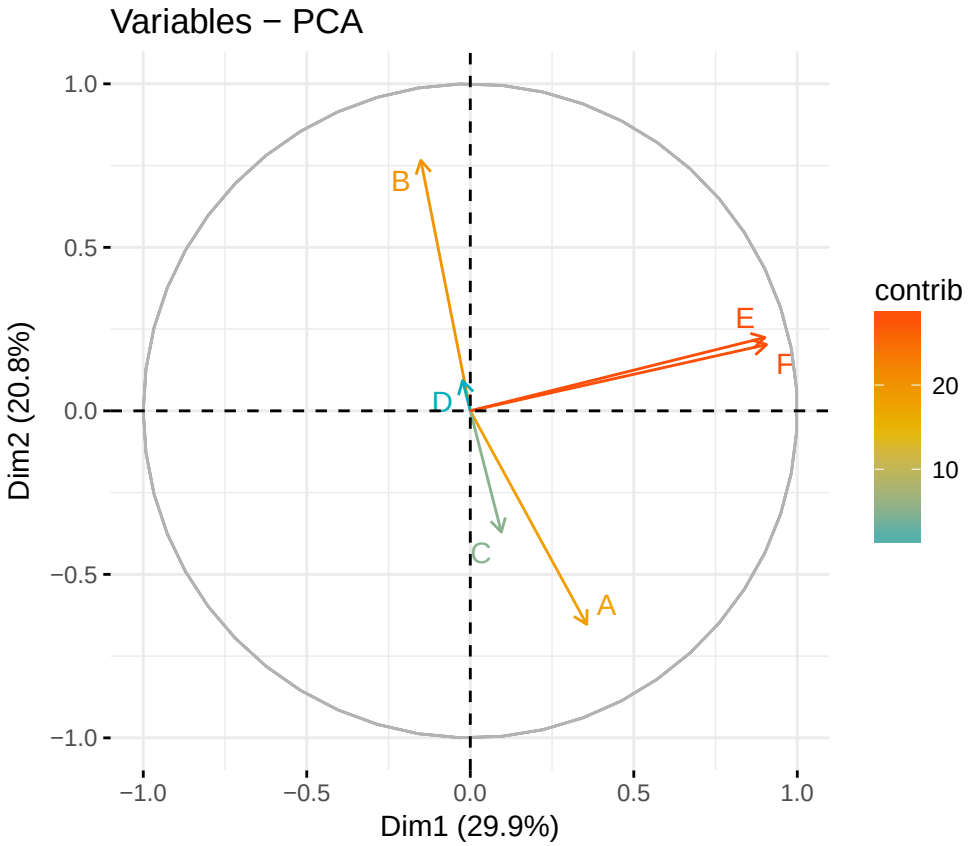
visualisation: a plot of the first 2 components.

```
fviz_pca_ind(pcaobj)
```



variables with shorter vectors are less important for the first components variables with aligned vectors are positively correlated.

```
fviz_pca_var(
    pcaobj, col.var = "contrib",
    gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
    repel = TRUE
)
```

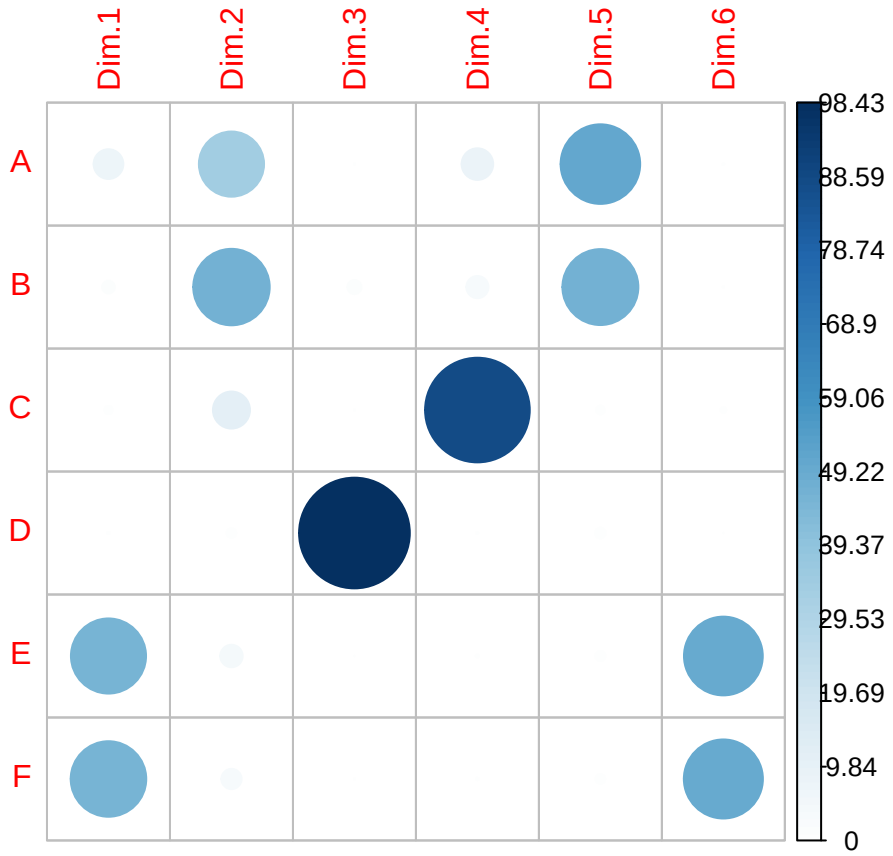


contribution of the variables to the variance of the data (i.e. importance of the 'real' variables):

```
print(var$contrib)
```

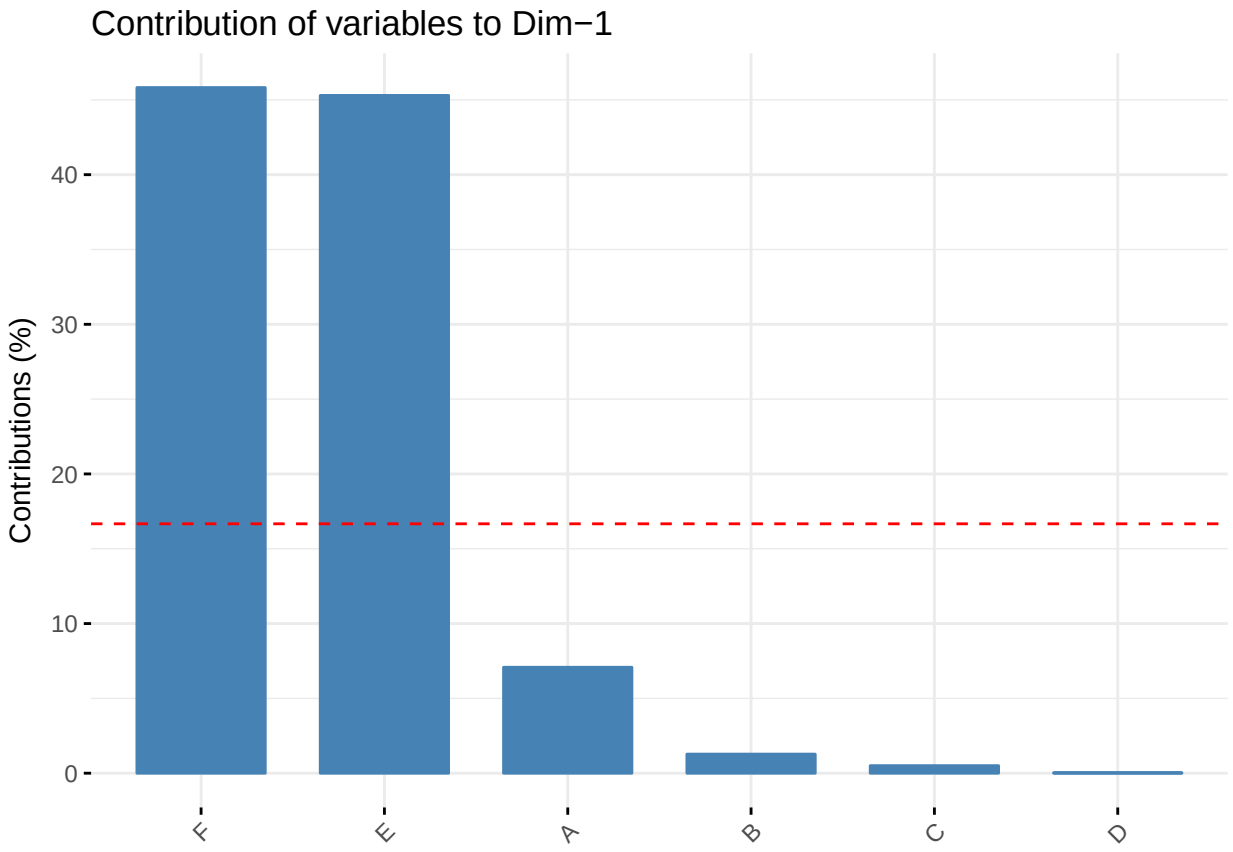
```
##         Dim.1      Dim.2        Dim.3       Dim.4      Dim.5        Dim.6
## A  7.07733359 34.0843255 6.375678e-03  7.99481606 50.8195298 1.761936e-02
## B  1.27466851 46.9385566 1.554029e+00  3.91886295 46.3069450 6.938345e-03
## C  0.50277472 10.9883708 6.305069e-03 87.95297911  0.5495663 3.945954e-06
## D  0.03097688  0.6833524 9.843002e+01  0.02230487  0.8331156 2.323091e-04
## E 45.29450517  4.0236214 3.761742e-04  0.07399062  0.7958687 4.981164e+01
## F 45.81974114  3.2817734 2.896544e-03  0.03704639  0.6949745 5.016357e+01
```

```
corrplot(var$contrib, is.corr=FALSE)
```



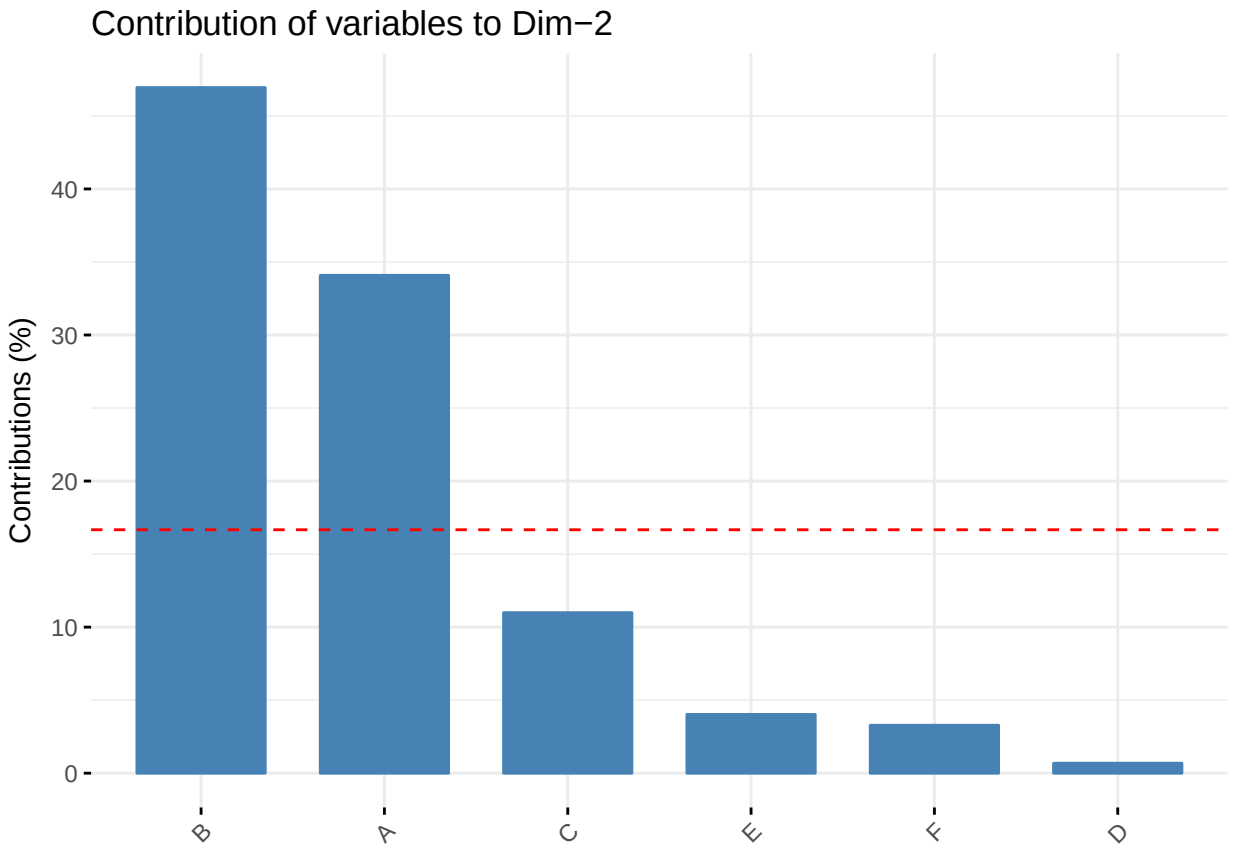
Contributions of variables to PC1

```
fviz_contrib(pcaobj, choice = "var", axes = 1, top = 10)
```



Contributions of variables to PC2

```
fviz_contrib(pcaobj, choice = "var", axes = 2, top = 10)
```




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

Section 6: Questions to be asked 1. What shall I do with the outliers? 2. What shall I do with the NAs? 3. Do you want to find redundant variables or all variables must be present for processing?