Tree learnign methods

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
library(tree)
library(ggplot2)
```

1 Exploratory data analysis

```r
data <- read.csv(params$myfile)
```

The data set contains date on 5631 compound where a solubility screen has been preformed where all compounds were classified as eather soluble of insoluble. Fro each compound 72 contineous variables were recorded.

A short exploratory data analysis is preformed to become familiar with the dateset.

```r
summary(data)
```

```
##        x1               x2               x3              x4
##  Min.   : 265.5   Min.   : 219.6   Min.   :1.208   Min.   :1.069
##  1st Qu.: 754.2   1st Qu.: 518.9   1st Qu.:1.447   1st Qu.:1.465
##  Median : 880.9   Median : 598.0   Median :1.469   Median :1.561
##  Mean   : 880.6   Mean   : 596.4   Mean   :1.472   Mean   :1.551
##  3rd Qu.:1000.1   3rd Qu.: 674.9   3rd Qu.:1.493   3rd Qu.:1.646
##  Max.   :1903.1   Max.   :1275.7   Max.   :1.703   Max.   :2.251
##
##        x5               x6               x7               x8
##  Min.   : 657.5   Min.   : 248.9   Min.   :  76.0   Min.   : 17.0
##  1st Qu.:1374.1   1st Qu.: 855.4   1st Qu.: 459.1   1st Qu.:175.8
##  Median :1560.4   Median :1000.9   Median : 575.8   Median :229.9
##  Mean   :1543.7   Mean   : 998.5   Mean   : 570.1   Mean   :230.9
##  3rd Qu.:1729.9   3rd Qu.:1149.1   3rd Qu.: 677.1   3rd Qu.:282.1
```

```
## Max.   :3375.8   Max.   :2329.4   Max.   :1392.8   Max.   :541.8
##
##       x9              x10              x11              x12
## Min.   :  0.00   Min.   :  0.00   Min.   : 0.00   Min.   : 0.00
## 1st Qu.: 84.12   1st Qu.: 41.50   1st Qu.:15.25   1st Qu.: 2.50
## Median :115.00   Median : 58.75   Median :22.50   Median : 5.00
## Mean   :116.37   Mean   : 59.70   Mean   :24.23   Mean   : 6.27
## 3rd Qu.:145.81   3rd Qu.: 75.38   3rd Qu.:30.56   3rd Qu.: 8.25
## Max.   :318.38   Max.   :188.88   Max.   :98.00   Max.   :36.00
##
##      x13              x14              x15              x16
## Min.   :0.00100   Min.   :0.0010   Min.   :0.0020   Min.   :0.0040
## 1st Qu.:0.03400   1st Qu.:0.0600   1st Qu.:0.0980   1st Qu.:0.1850
## Median :0.05200   Median :0.0910   Median :0.1500   Median :0.2830
## Mean   :0.06219   Mean   :0.1059   Mean   :0.1765   Mean   :0.3303
## 3rd Qu.:0.08100   3rd Qu.:0.1350   3rd Qu.:0.2250   3rd Qu.:0.4310
## Max.   :0.28600   Max.   :0.4800   Max.   :0.8800   Max.   :1.4670
##
##      x17              x18              x19              x20
## Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
## 1st Qu.:0.2720   1st Qu.:0.3480   1st Qu.:0.4530   1st Qu.:0.5660
## Median :0.4140   Median :0.5210   Median :0.6860   Median :0.8800
## Mean   :0.4777   Mean   :0.5951   Mean   :0.7531   Mean   :0.9349
## 3rd Qu.:0.6325   3rd Qu.:0.7890   3rd Qu.:0.9850   3rd Qu.:1.2180
## Max.   :1.9010   Max.   :2.2390   Max.   :2.5560   Max.   :3.2330
##
##      x21             x22             x23              x24
## Min.   :1.860   Min.   :0.695   Min.   :0.1760   Min.   :0.0270
## 1st Qu.:2.430   1st Qu.:1.452   1st Qu.:0.7870   1st Qu.:0.3060
## Median :2.610   Median :1.697   Median :0.9680   Median :0.3850
## Mean   :2.611   Mean   :1.699   Mean   :0.9705   Mean   :0.3927
## 3rd Qu.:2.777   3rd Qu.:1.927   3rd Qu.:1.1435   3rd Qu.:0.4710
## Max.   :4.060   Max.   :3.239   Max.   :2.1090   Max.   :1.1060
##
##      x25              x26              x27               x28
## Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
## 1st Qu.:0.1470   1st Qu.:0.0730   1st Qu.:0.02800   1st Qu.:0.00500
## Median :0.1910   Median :0.0980   Median :0.03700   Median :0.00800
## Mean   :0.1977   Mean   :0.1011   Mean   :0.04068   Mean   :0.01036
## 3rd Qu.:0.2430   3rd Qu.:0.1240   3rd Qu.:0.05000   3rd Qu.:0.01300
## Max.   :0.6660   Max.   :0.4010   Max.   :0.21100   Max.   :0.08500
##
##      x29              x30              x31              x32
## Min.   : 2.678   Min.   : 1.931   Min.   : 1.543   Min.   : 0.707
## 1st Qu.: 6.819   1st Qu.: 6.521   1st Qu.: 6.246   1st Qu.: 0.866
## Median : 7.219   Median : 6.882   Median : 6.569   Median : 1.500
## Mean   : 7.527   Mean   : 7.052   Mean   : 6.650   Mean   : 3.210
## 3rd Qu.: 8.033   3rd Qu.: 7.387   3rd Qu.: 6.968   3rd Qu.: 4.717
## Max.   :12.946   Max.   :11.980   Max.   :11.319   Max.   :17.292
##
##      x33              x34              x35              x36
## Min.   : 0.707   Min.   : 0.707   Min.   : 15.62   Min.   :  1.875
## 1st Qu.: 1.500   1st Qu.: 1.118   1st Qu.:212.50   1st Qu.:115.375
## Median : 3.162   Median : 2.121   Median :265.25   Median :144.000
```

```
##  Mean   : 4.548   Mean   : 3.855   Mean   :270.73   Mean   :146.687
##  3rd Qu.: 7.159   3rd Qu.: 6.164   3rd Qu.:326.25   3rd Qu.:176.250
##  Max.   :21.059   Max.   :21.651   Max.   :761.62   Max.   :379.500
##
##       x37              x38              x39              x40
##  Min.   :  1.125   Min.   :  0.375   Min.   :  0.125   Min.   :  0.00
##  1st Qu.: 64.500   1st Qu.: 40.625   1st Qu.: 25.750   1st Qu.: 19.38
##  Median : 83.250   Median : 54.000   Median : 36.250   Median : 28.12
##  Mean   : 83.455   Mean   : 54.082   Mean   : 36.732   Mean   : 28.59
##  3rd Qu.:101.125   3rd Qu.: 67.375   3rd Qu.: 47.125   3rd Qu.: 37.00
##  Max.   :212.875   Max.   :159.375   Max.   :136.625   Max.   :114.50
##
##       x41             x42              x43              x44
##  Min.   :  0.00   Min.   : 0.000   Min.   : 0.454   Min.   :  0.616
##  1st Qu.: 13.62   1st Qu.: 9.875   1st Qu.:13.755   1st Qu.: 20.369
##  Median : 20.88   Median :15.500   Median :21.045   Median : 29.984
##  Mean   : 20.96   Mean   :15.468   Mean   :22.580   Mean   : 32.402
##  3rd Qu.: 27.50   3rd Qu.:20.625   3rd Qu.:29.450   3rd Qu.: 41.964
##  Max.   :101.50   Max.   :89.625   Max.   :85.059   Max.   :115.804
##
##       x45              x46              x47              x48
##  Min.   :  0.543   Min.   :  0.91   Min.   :  0.751   Min.   :  0.00
##  1st Qu.: 25.117   1st Qu.: 30.68   1st Qu.: 37.080   1st Qu.: 42.18
##  Median : 35.880   Median : 44.48   Median : 54.345   Median : 61.22
##  Mean   : 39.327   Mean   : 49.33   Mean   : 60.700   Mean   : 68.64
##  3rd Qu.: 50.449   3rd Qu.: 64.01   3rd Qu.: 78.532   3rd Qu.: 88.62
##  Max.   :159.264   Max.   :210.14   Max.   :253.770   Max.   :275.60
##
##       x49             x50              x51               x52
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.0000   Min.   :  0.000
##  1st Qu.: 47.45   1st Qu.: 53.99   1st Qu.:  0.9605   1st Qu.:  0.737
##  Median : 70.16   Median : 79.66   Median :  1.4160   Median :  1.089
##  Mean   : 78.58   Mean   : 89.11   Mean   :  1.6947   Mean   :  1.416
##  3rd Qu.:101.12   3rd Qu.:114.34   3rd Qu.:  1.9990   3rd Qu.:  1.639
##  Max.   :341.64   Max.   :371.88   Max.   :171.1110   Max.   :216.500
##
##       x53            x54               x55              x56
##  Min.   :11.61   Min.   :  0.007   Min.   : 484.6   Min.   : 113.6
##  1st Qu.:17.37   1st Qu.:  1.139   1st Qu.:1092.6   1st Qu.: 420.5
##  Median :18.81   Median :  2.172   Median :1250.9   Median : 511.5
##  Mean   :18.84   Mean   :  3.372   Mean   :1358.5   Mean   : 541.4
##  3rd Qu.:20.25   3rd Qu.:  3.720   3rd Qu.:1417.2   3rd Qu.: 624.0
##  Max.   :27.25   Max.   :346.750   Max.   :5949.9   Max.   :1766.2
##
##       x57             x58              x59               x60
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.000   Min.   :  0.000
##  1st Qu.: 90.38   1st Qu.: 15.62   1st Qu.:  5.875   1st Qu.:  1.625
##  Median :135.00   Median : 31.00   Median : 11.500   Median :  3.250
##  Mean   :156.38   Mean   : 41.60   Mean   : 16.799   Mean   :  5.492
##  3rd Qu.:202.75   3rd Qu.: 59.12   3rd Qu.: 24.500   3rd Qu.:  8.000
##  Max.   :709.12   Max.   :265.50   Max.   :155.250   Max.   :122.750
##
##       x61               x62                x63                x64
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :-3252.8   Min.   :-344.9
```

```
##  1st Qu.:  0.0000   1st Qu.: 0.0000   1st Qu.:  243.7   1st Qu.: 328.6
##  Median :  0.2500   Median : 0.0000   Median :  322.1   Median : 457.0
##  Mean   :  0.8435   Mean   : 0.1091   Mean   :  185.2   Mean   : 457.2
##  3rd Qu.:  1.1250   3rd Qu.: 0.0000   3rd Qu.:  391.1   3rd Qu.: 596.5
##  Max.   :100.8750   Max.   :79.3750   Max.   :  727.6   Max.   :1200.4
##
##       x65               x66               x67               x68
##  Min.   : -10.25   Min.   : 10.38   Min.   :  0.00   Min.   :-10.75
##  1st Qu.: 305.31   1st Qu.:134.00   1st Qu.: 67.75   1st Qu.: 36.50
##  Median : 406.00   Median :185.88   Median : 98.00   Median : 53.00
##  Mean   : 413.76   Mean   :189.30   Mean   : 99.57   Mean   : 54.21
##  3rd Qu.: 521.94   3rd Qu.:239.94   3rd Qu.:128.50   3rd Qu.: 69.62
##  Max.   :1069.50   Max.   :511.12   Max.   :291.62   Max.   :161.88
##
##       x69              x70                x71                 x72
##  Min.   :-55.38   Min.   :-61.375   Min.   :  9.897   Min.   : 103.1
##  1st Qu.: 14.62   1st Qu.:  2.500   1st Qu.: 37.275   1st Qu.: 369.2
##  Median : 21.75   Median :  4.875   Median : 43.828   Median : 431.5
##  Mean   : 23.39   Mean   :  6.161   Mean   : 43.870   Mean   : 427.6
##  3rd Qu.: 29.62   3rd Qu.:  8.125   3rd Qu.: 50.443   3rd Qu.: 490.5
##  Max.   : 90.38   Max.   : 34.875   Max.   :100.170   Max.   :1056.7
##                                     NA's   :787
##        y
##  Min.   :-1.0000
##  1st Qu.:-1.0000
##  Median :-1.0000
##  Mean   :-0.2406
##  3rd Qu.: 1.0000
##  Max.   : 1.0000
##
```

In the summary one can see that the only variable that has missing values is $x71$ with a percentage of 13.97%. To preform analysis one can not have missing values hence we decided to substitute the NA values with the column mean in stead of for example deleting the entire colum or all rows with missin values.

```r
library(zoo)
```

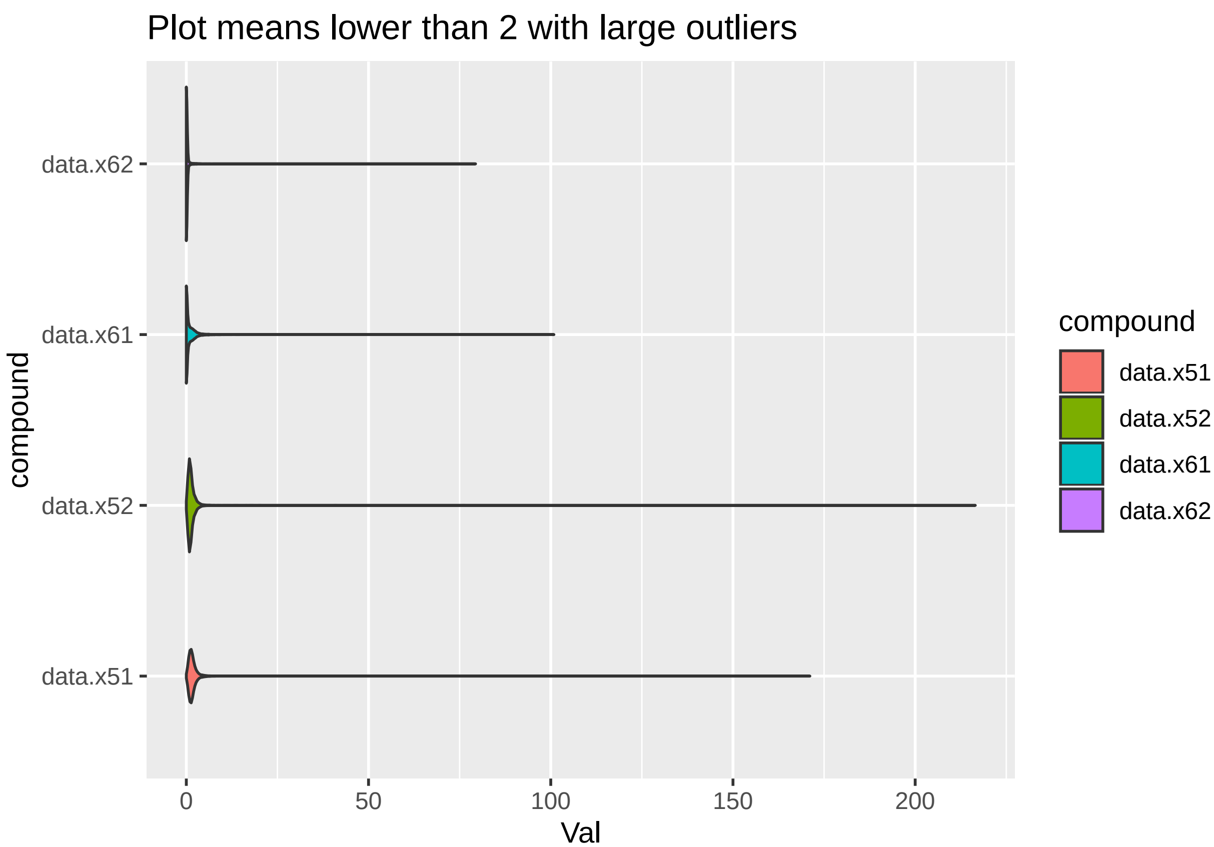
```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
data <- na.aggregate(data) # set the average of the column on NA values
```

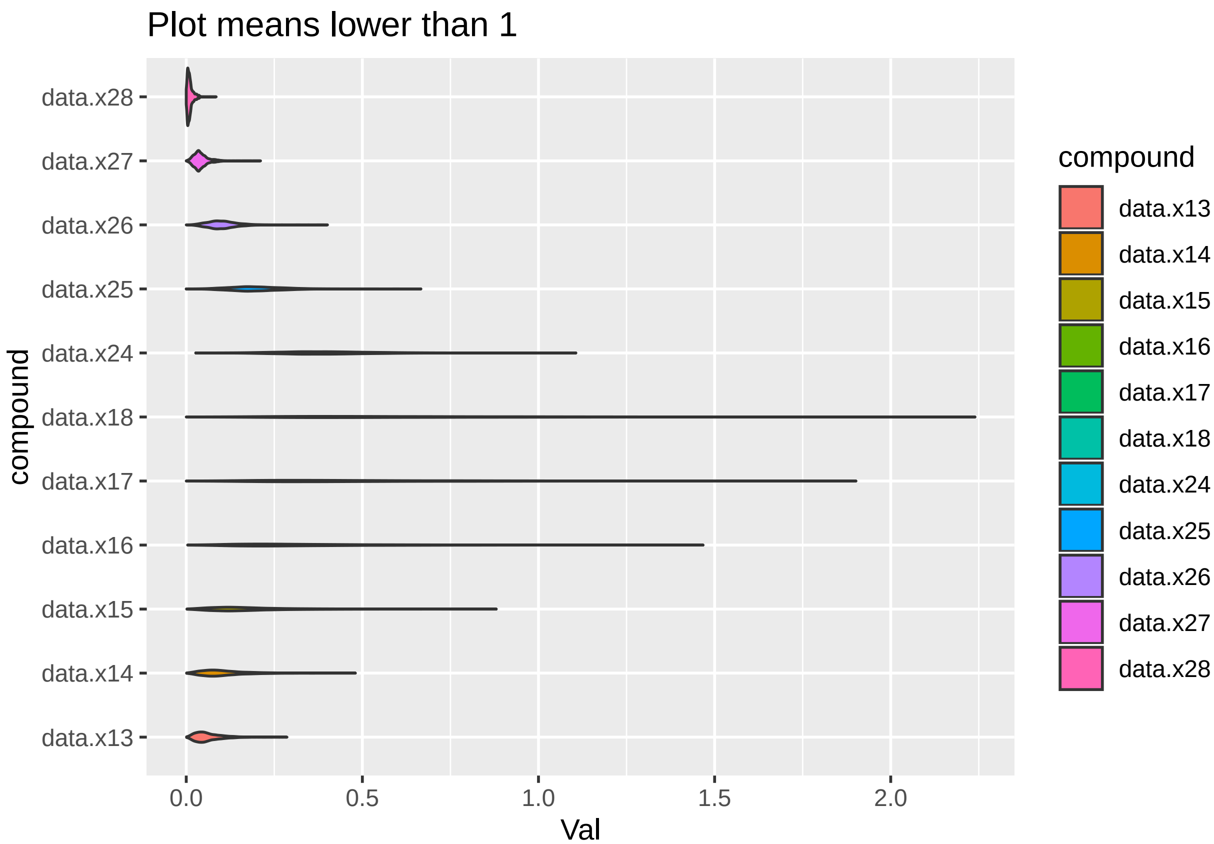
To check the variability of the data and how they are distributed we made violin plots. As we have a lot of variables the data is split with regards to the column mean in order to make the observation easier.

```r
library(tidyr)
library(ggplot2)
library(dplyr)

subset0 <- data.frame(data$x51, data$x52, data$x61, data$x62) # mean less than 2 with high outliers

subset1 <- data.frame(data$x13, data$x14, data$x15, data$x16, data$x17, data$x18, data$x24, data$x25, da
```

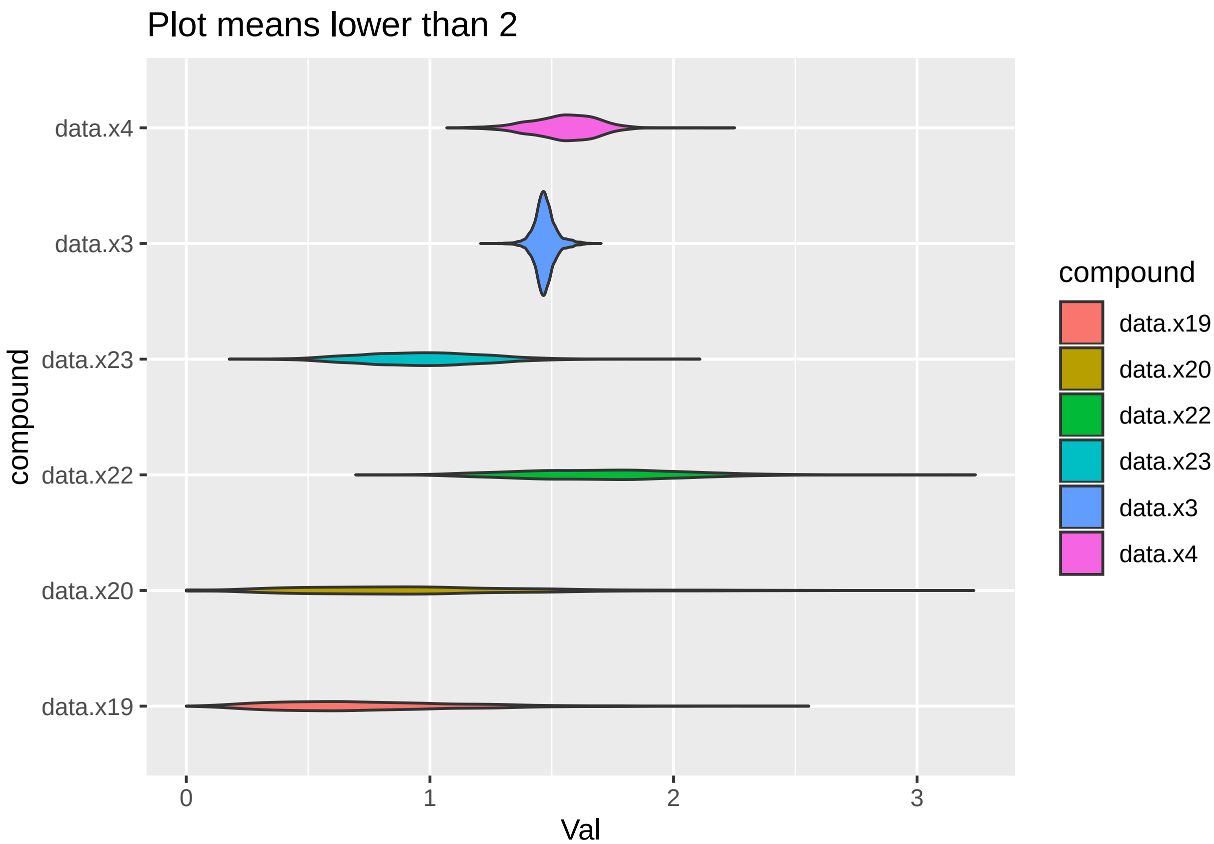
```r
subset1.1 <- data.frame(data$x3, data$x4, data$x19, data$x20, data$x22, data$x23) # mean less than 2

subset2 <- data.frame(data$x12, data$x21, data$x29, data$x30, data$x31, data$x32, data$x33, data$x34, da

subset3 <- data.frame(data$x11, data$x39, data$x40, data$x41, data$x42, data$x43, data$x44, data$x45, da

subset4 <-data.frame(data$x10, data$x37, data$x38, data$x47, data$x48, data$x49, data$x50, data$x68) # 

subset5 <- data.frame(data$x1, data$x2, data$x7, data$x8, data$x9, data$x35, data$x36, data$x56, data$x5

subset6 <- data.frame(data$x5, data$x6, data$x55)  # mean higher than 1000

subset0 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means lower than 2 with large outliers")
```



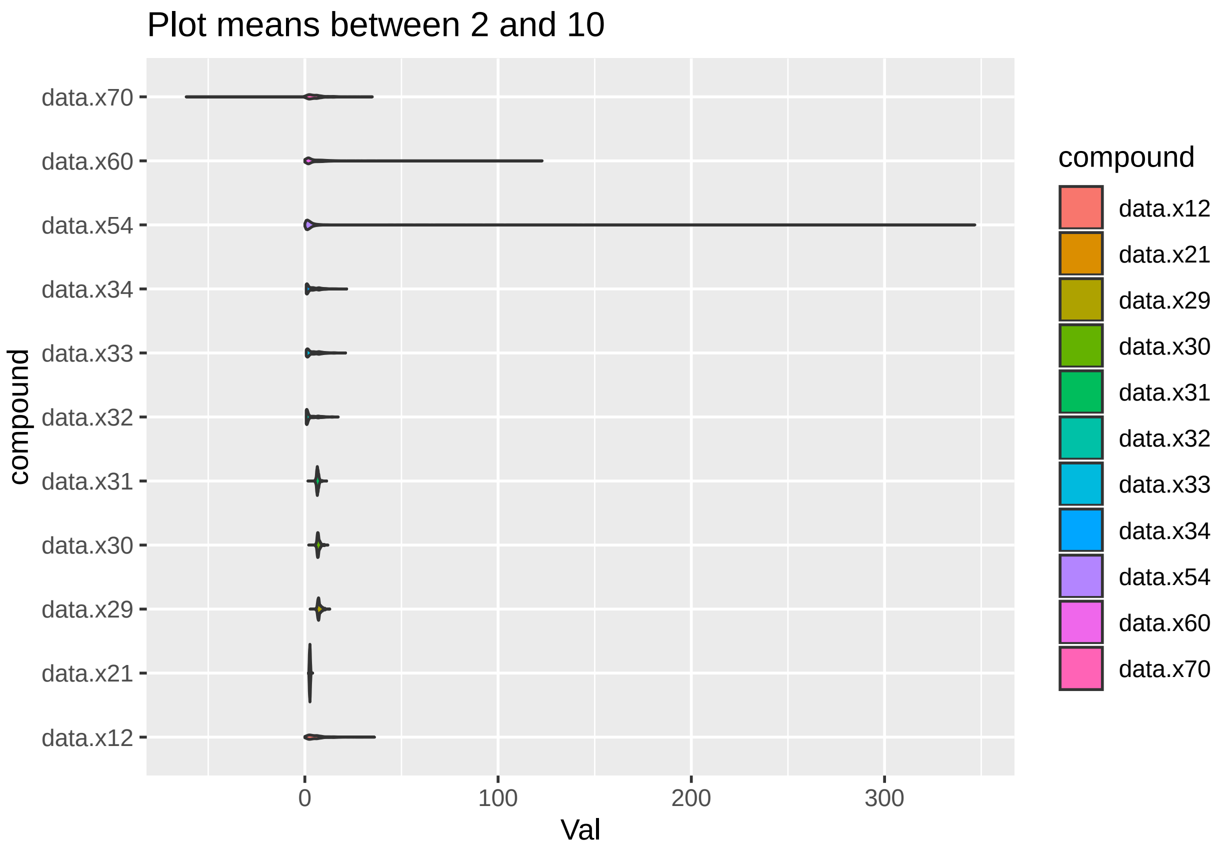

```r
subset1 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means lower than 1")
```



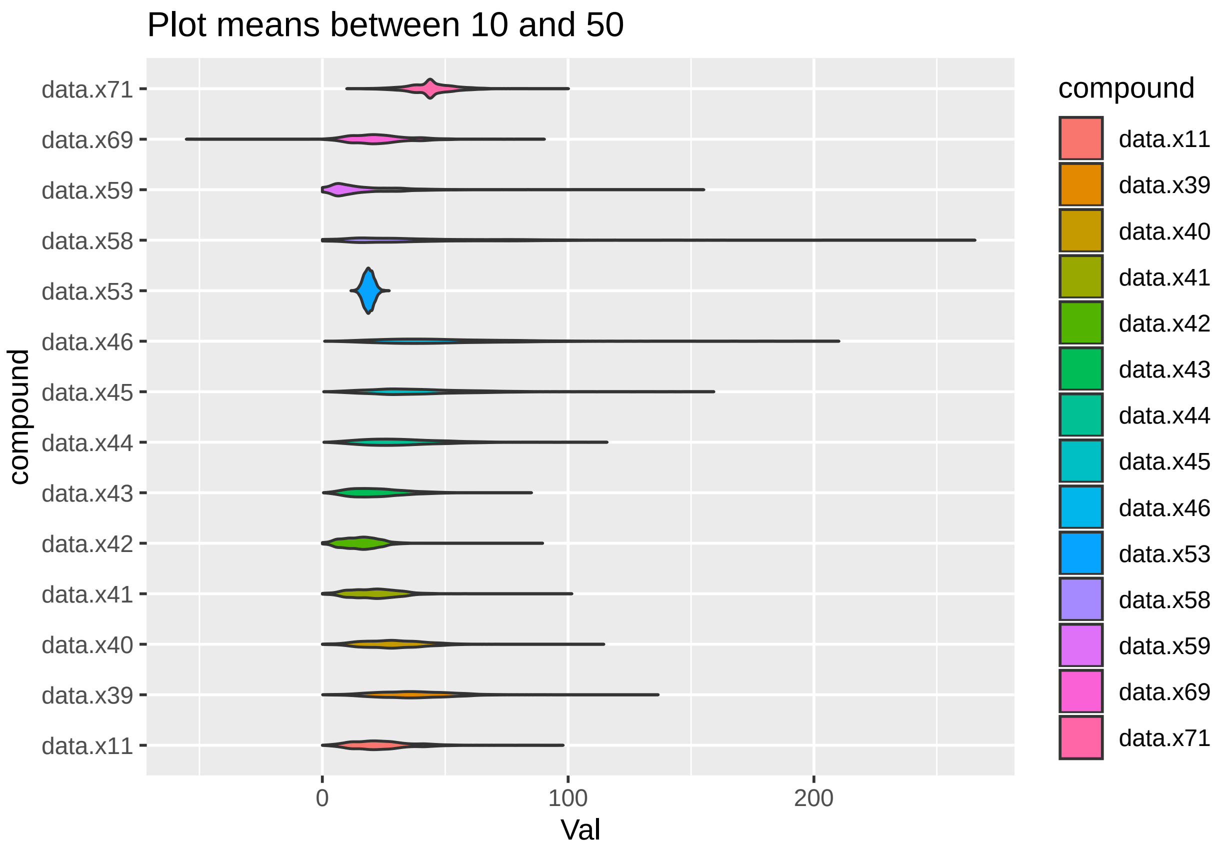

```r
subset1.1 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means lower than 2")
```



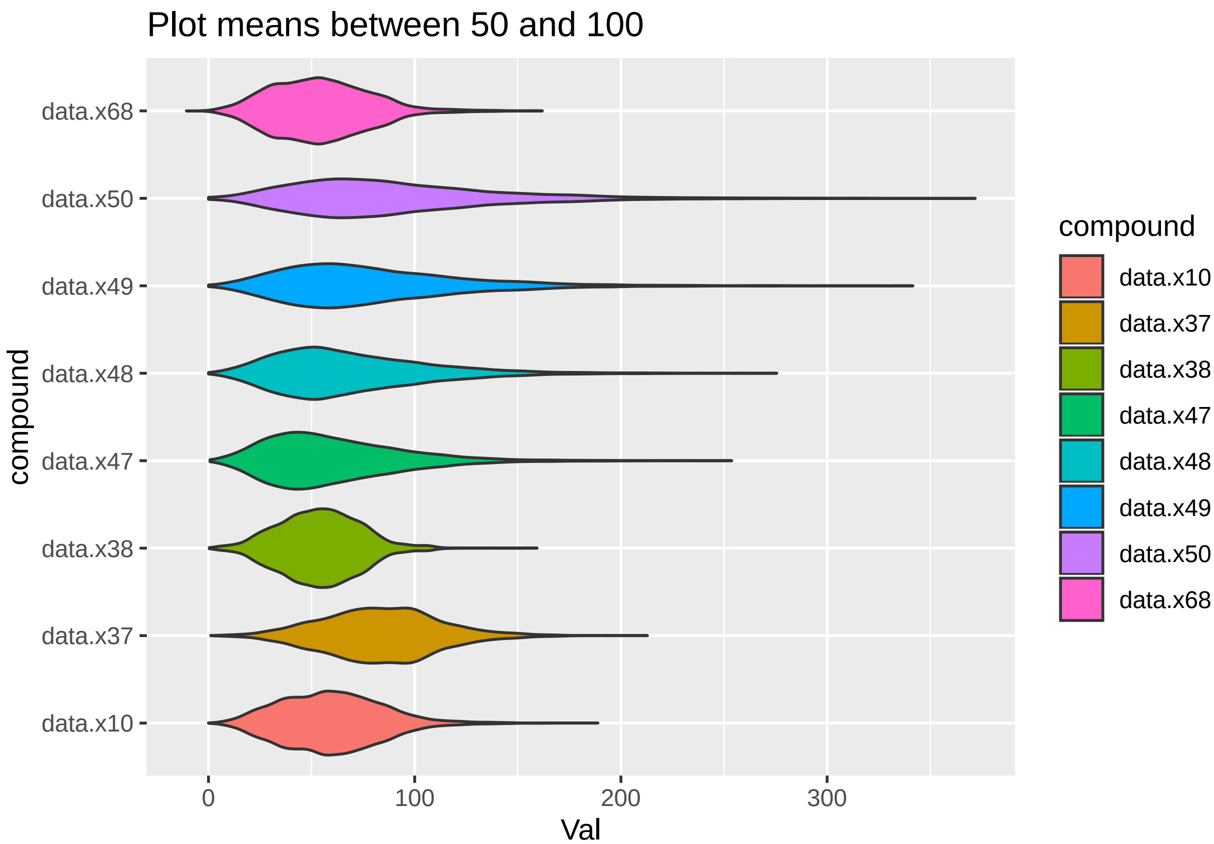

```
subset2 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means between 2 and 10")
```



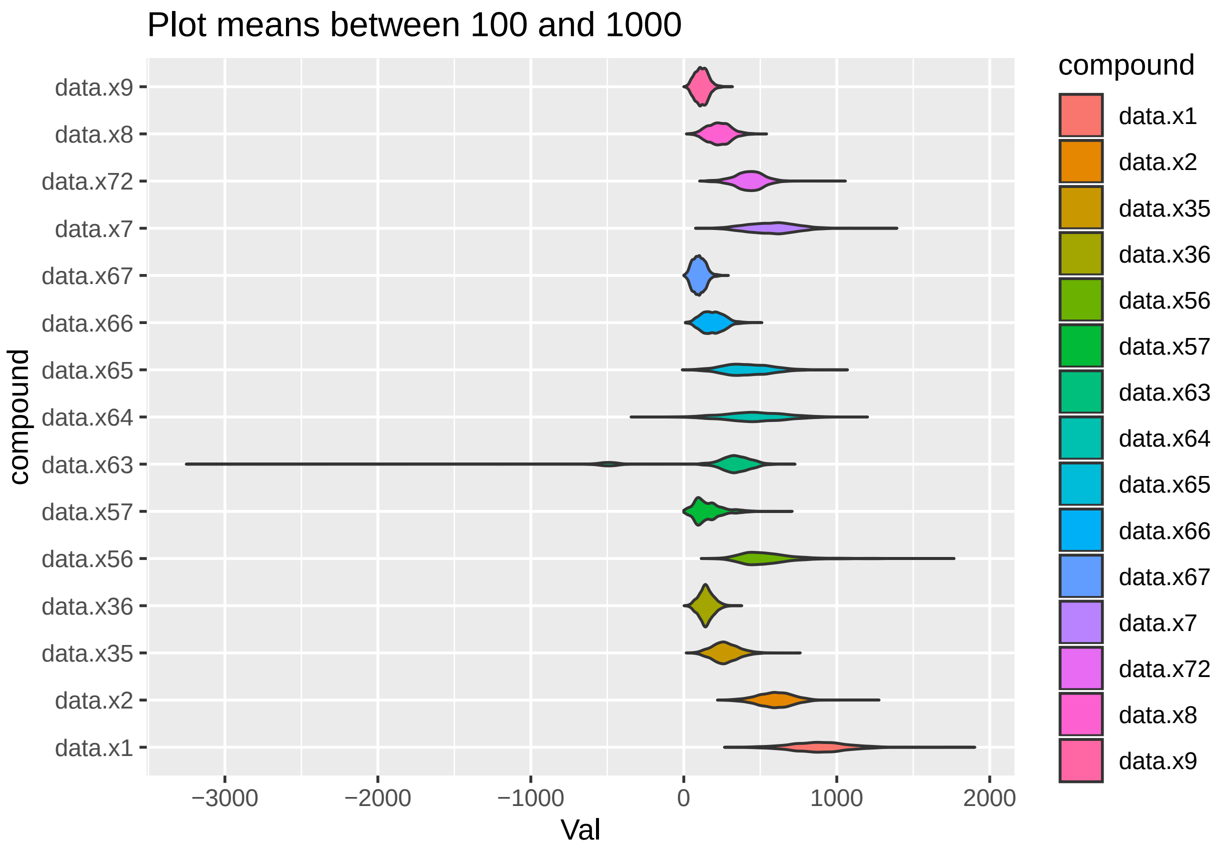

```
subset3 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means between 10 and 50")
```



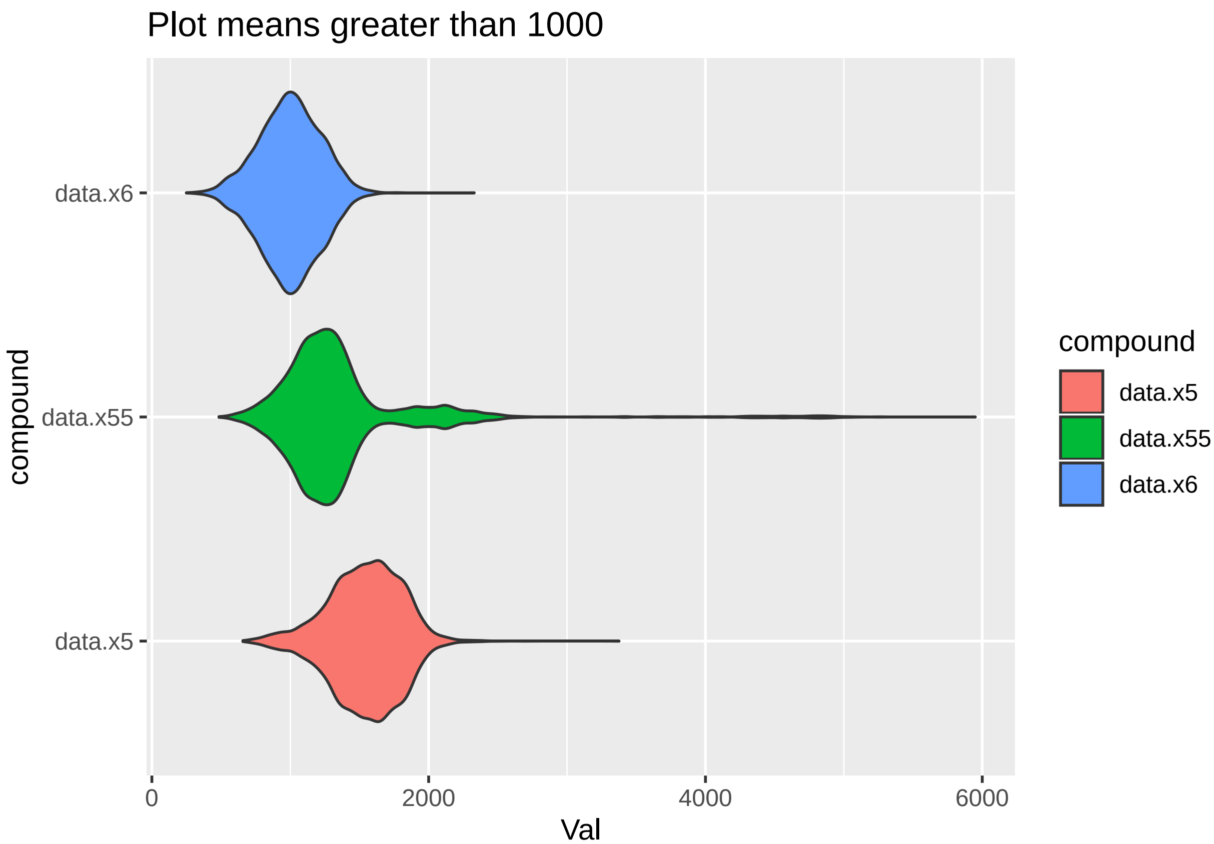

```
subset4 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means between 50 and 100")
```




```
subset5 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means between 100 and 1000")
```




```r
subset6 %>%
  gather(key="compound", value="Val") %>%
  ggplot( aes(x=compound, y=Val, fill=compound)) +
    geom_violin()+
  coord_flip()+
  ggtitle("Plot means greater than 1000")
```



Looking at this violin plots it can be seen that a lot of the components have quite small variance. However we can see some high outliers which could be problematic. This is noted when moving on to firther analysis.

A correlation table is done in order to check if there is a linear relation between the variables.

```
X <- data[,-73]
X <- scale(X)

C<-cor(X)
head(C)
```

```
##           x1        x2        x3        x4        x5        x6         x7
## x1 1.0000000 0.9922019 0.6617702 0.9132550 0.8548888 0.5157994 0.39866996
## x2 0.9922019 1.0000000 0.5686335 0.9562784 0.8789283 0.5519158 0.43840048
## x3 0.6617702 0.5686335 1.0000000 0.3229651 0.4089550 0.1306643 0.03110228
## x4 0.9132550 0.9562784 0.3229651 1.0000000 0.8826500 0.6045916 0.50541191
## x5 0.8548888 0.8789283 0.4089550 0.8826500 1.0000000 0.8586513 0.74276859
## x6 0.5157994 0.5519158 0.1306643 0.6045916 0.8586513 1.0000000 0.96017475
##            x8        x9       x10       x11       x12        x13        x14
## x1 0.37053247 0.3508783 0.3566473 0.3637045 0.3214353 -0.4874010 -0.4412800
## x2 0.39631857 0.3671058 0.3684962 0.3668381 0.3211978 -0.4959966 -0.4548166
## x3 0.07737273 0.1140231 0.1409133 0.1934261 0.1811551 -0.3076249 -0.2418498
## x4 0.43192191 0.3826158 0.3760261 0.3567108 0.3072873 -0.4949219 -0.4678345
## x5 0.61762760 0.5403826 0.5212293 0.4700830 0.3621697 -0.4756300 -0.5057987
## x6 0.82088533 0.7145723 0.6662820 0.5418295 0.3727688 -0.3217159 -0.4390745
##           x15        x16        x17        x18        x19        x20
## x1 -0.4197133 -0.4694502 -0.4998835 -0.5332395 -0.5931547 -0.5397476
## x2 -0.4387986 -0.4850215 -0.5105356 -0.5436700 -0.6023948 -0.5401801
```

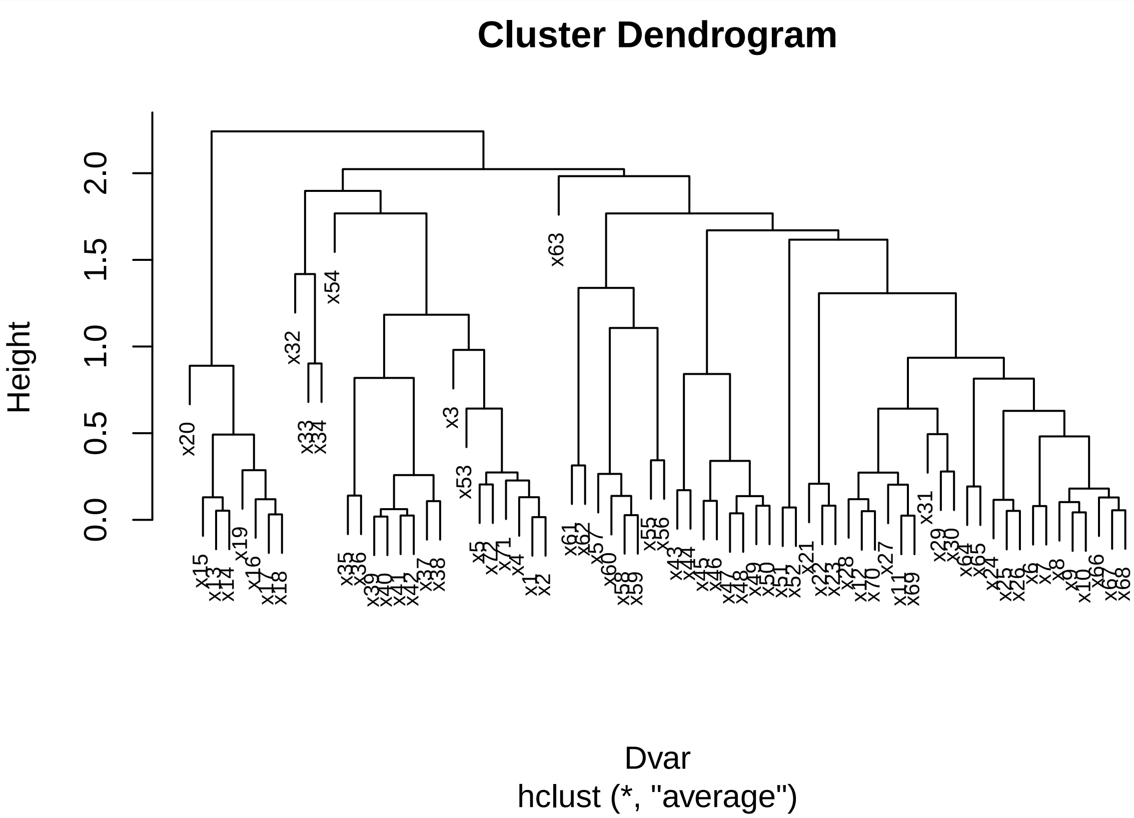
```
## x3 -0.1829922 -0.2194932 -0.2595881 -0.2826530 -0.3369743 -0.3639192
## x4 -0.4658221 -0.4985059 -0.5089568 -0.5401488 -0.5942427 -0.5137649
## x5 -0.5110266 -0.4933989 -0.4697766 -0.4881575 -0.5128573 -0.4132727
## x6 -0.4954873 -0.4454225 -0.3917520 -0.3887633 -0.3432310 -0.2144998
##            x21        x22        x23         x24        x25         x26
## x1 -0.49678070 -0.4039201 -0.3237346 -0.24248698 -0.1924284 -0.13328099
## x2 -0.47403638 -0.3764538 -0.2923797 -0.22321575 -0.1811279 -0.12459983
## x3 -0.43498135 -0.4160630 -0.3947351 -0.28985380 -0.2160468 -0.16022319
## x4 -0.39236690 -0.2892218 -0.2015362 -0.16667586 -0.1455783 -0.09564338
## x5 -0.01120148  0.0705358  0.1109908  0.07521345  0.0586746  0.09325173
## x6  0.40825157  0.5423688  0.5802171  0.48708716  0.4154461  0.40732757
##            x27        x28        x29        x30       x31        x32        x33
## x1 -0.01551995 0.09940273 0.19675200 0.25112251 0.3008533 0.04373356 0.09138912
## x2 -0.01315373 0.10066136 0.20717237 0.26451246 0.3111918 0.04239189 0.08635702
## x3 -0.04147939 0.04079677 0.05430859 0.06614591 0.1159653 0.03081442 0.08641229
## x4 -0.00166872 0.10543667 0.22382082 0.28590732 0.3224404 0.03570327 0.07001654
## x5  0.14658482 0.17917834 0.25848656 0.31333691 0.3548803 0.03961313 0.08446086
## x6  0.35259032 0.26818768 0.28139613 0.32224847 0.3591088 0.04386218 0.08465656
##           x34        x35         x36       x37       x38       x39       x40
## x1 0.08896521  0.6820756 0.588985593 0.5412972 0.4387331 0.3510472 0.3527750
## x2 0.08388428  0.6545158 0.567458738 0.5330149 0.4452651 0.3653052 0.3660019
## x3 0.08159868  0.5876487 0.511231062 0.4178851 0.2814430 0.1921597 0.1988093
## x4 0.06602553  0.5503497 0.484876674 0.4822526 0.4348051 0.3774941 0.3753558
## x5 0.07562523  0.3920232 0.433586764 0.5038924 0.5154822 0.4957246 0.4869050
## x6 0.06449795 -0.1050620 0.006272814 0.1746825 0.2945501 0.3649717 0.3589584
##          x41       x42          x43          x44          x45          x46
## x1 0.3262755 0.3329922 -0.113127876 -0.075759064 -0.052079217 -0.017285553
## x2 0.3383110 0.3451567 -0.091967702 -0.055255219 -0.033390059 -0.004909783
## x3 0.1928582 0.1939332 -0.199395100 -0.174369793 -0.145792121 -0.090074135
## x4 0.3457763 0.3520189 -0.036802398 -0.003972388  0.013455127  0.025741495
## x5 0.4500299 0.4490919  0.009131573 -0.003541028  0.005375077  0.002855975
## x6 0.3366080 0.3374267  0.112826503  0.088358902  0.106307012  0.080491815
##             x47         x48          x49         x50         x51         x52
## x1  0.004612659  0.01290031  0.040463095  0.05750260 -0.12846114 -0.10529052
## x2  0.012436612  0.01863229  0.047561663  0.06365525 -0.12346116 -0.10419555
## x3 -0.058349687 -0.04449105 -0.034091551 -0.01518383 -0.15534114 -0.11855845
## x4  0.030363690  0.03080259  0.059784152  0.07348788 -0.10701945 -0.09779046
## x5 -0.012612957 -0.02070397  0.002859267  0.01161077 -0.06988990 -0.08493151
## x6  0.040425827  0.01920912  0.026639917  0.02167635  0.08409449  0.01540715
##          x53          x54       x55       x56         x57          x58
## x1 0.7088191  0.022163378 0.4054590 0.3331118  0.02361152 -0.091706155
## x2 0.7303931  0.015854938 0.4010131 0.3255883  0.01361591 -0.106121159
## x3 0.3254089  0.061837413 0.2935718 0.2590556  0.05826200 -0.004781807
## x4 0.7368525 -0.001592405 0.3633431 0.2863657 -0.01229390 -0.134420485
## x5 0.6599917 -0.030763847 0.4483741 0.4891974  0.19720934  0.031590740
## x6 0.4096258 -0.143687954 0.3615647 0.5053489  0.34541697  0.196832038
##             x59         x60          x61          x62          x63         x64
## x1 -0.098068220 -0.07000824  0.022004579 -0.005932764  0.008117355  0.26195724
## x2 -0.112731072 -0.08542620  0.008192976 -0.007850912  0.026062013  0.30795156
## x3 -0.008472751  0.01504489  0.073769032  0.006262774 -0.107475937 -0.09021364
## x4 -0.141075427 -0.11728239 -0.026746464 -0.014402946  0.070183183  0.40035722
## x5  0.012165637  0.02778423  0.069042310  0.035211567  0.038459279  0.49464514
## x6  0.181469488  0.19220134  0.141332366  0.061730205  0.059218953  0.63378100
##             x65        x66       x67       x68       x69       x70       x71
```

```
## x1  0.398177970 0.42471230 0.4003946 0.3869621 0.3663280 0.3217195 0.9284626
## x2  0.445344790 0.45797004 0.4225859 0.4030531 0.3716906 0.3219106 0.9122037
## x3 -0.003135893 0.08206638 0.1216271 0.1423389 0.1850402 0.1791729 0.6795407
## x4  0.530384895 0.50775816 0.4489942 0.4187291 0.3668999 0.3095062 0.8187867
## x5  0.648976376 0.62295240 0.5574405 0.5333876 0.4670753 0.3531464 0.7866585
## x6  0.784080981 0.75661837 0.6771354 0.6431088 0.5286311 0.3577971 0.4749514
##          x72
## x1 0.9068178
## x2 0.9010083
## x3 0.6009858
## x4 0.8355009
## x5 0.8979010
## x6 0.6817726
```

We have some variables that are linearly correlated. For example `x1` and `x2` have a correlation of NA.

Next a Cluster Dendrogram is produced in order to check graphically the distances between the variables. It clusters the variables with respect to their column mean.

```
Dvar<-as.dist(2*(1-C)) # as.dist  is a generic function. Its default method handles objects inheriting
clusterVar<-hclust(Dvar,method= "average")
plot(clusterVar,labels=colnames(X),cex=0.65)
```



In this dendrogram it can be seen that there are a lot of clusters meaning a lot of the variables are close to each other with respect to the correlation mean distance. So it's saying, as we commented before that there is a lot of variables that are closly correlated.

As we have a lot of vairables PCA analysis is preformed in order to check the which variables explain much of the variablility of the data.

```r
pca <- princomp(X)
summary(pca)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3     Comp.4     Comp.5
## Standard deviation     4.2765825 3.8201475 2.61795519 2.36376354 2.17066070
## Proportion of Variance 0.2540612 0.2027239 0.09520704 0.07761626 0.06545284
## Cumulative Proportion  0.2540612 0.4567851 0.55199212 0.62960837 0.69506122
##                            Comp.6    Comp.7    Comp.8     Comp.9    Comp.10
## Standard deviation     2.04486950 1.4696980 1.3852587 1.29838683 1.28371474
## Proportion of Variance 0.05808658 0.0300055 0.0266567 0.02341816 0.02289189
## Cumulative Proportion  0.75314780 0.7831533 0.8098100 0.83322816 0.85612006
##                           Comp.11    Comp.12   Comp.13    Comp.14    Comp.15
## Standard deviation     1.23466275 1.09170557 1.0285509 0.92836109 0.91026746
## Proportion of Variance 0.02117587 0.01655601 0.0146959 0.01197232 0.01151019
## Cumulative Proportion  0.87729593 0.89385194 0.9085478 0.92052017 0.93203036
##                           Comp.16     Comp.17     Comp.18     Comp.19
## Standard deviation     0.88929080 0.725888210 0.699707547 0.642445464
## Proportion of Variance 0.01098581 0.007319546 0.006801078 0.005733465
## Cumulative Proportion  0.94301617 0.950335719 0.957136797 0.962870262
##                            Comp.20     Comp.21     Comp.22     Comp.23
## Standard deviation     0.610738144 0.603206698 0.516239961 0.483958538
## Proportion of Variance 0.005181491 0.005054485 0.003702098 0.003253576
## Cumulative Proportion  0.968051753 0.973106238 0.976808336 0.980061912
##                            Comp.24     Comp.25   Comp.26     Comp.27
## Standard deviation     0.435023558 0.406718003 0.3561279 0.345128596
## Proportion of Variance 0.002628877 0.002297902 0.0017618 0.001654651
## Cumulative Proportion  0.982690789 0.984988690 0.9867505 0.988405142
##                            Comp.28     Comp.29    Comp.30      Comp.31
## Standard deviation     0.318381534 0.296303303 0.26995339 0.2587110170
## Proportion of Variance 0.001408122 0.001219601 0.00101233 0.0009297678
## Cumulative Proportion  0.989813264 0.991032865 0.99204519 0.9929749626
##                             Comp.32      Comp.33      Comp.34    Comp.35
## Standard deviation     0.2415662321 0.2287125425 0.2213461521 0.19680264
## Proportion of Variance 0.0008106196 0.0007266489 0.0006805947 0.00053803
## Cumulative Proportion  0.9937855822 0.9945122311 0.9951928258 0.99573086
##                             Comp.36      Comp.37     Comp.38      Comp.39
## Standard deviation     0.1924752313 0.1728686108 0.163466497 0.1564707226
## Proportion of Variance 0.0005146291 0.0004151231 0.000371195 0.0003401033
## Cumulative Proportion  0.9962454849 0.9966606080 0.997031803 0.9973719063
##                             Comp.40      Comp.41      Comp.42      Comp.43
## Standard deviation     0.1551350603 0.1470969135 0.1430641473 0.1249421784
## Proportion of Variance 0.0003343217 0.0003005742 0.0002843192 0.0002168517
## Cumulative Proportion  0.9977062280 0.9980068022 0.9982911215 0.9985079732
##                             Comp.44      Comp.45      Comp.46      Comp.47
## Standard deviation     0.1192421416 0.1136811903 0.1019380236 0.0988889400
## Proportion of Variance 0.0001975169 0.0001795237 0.0001443501 0.0001358439
## Cumulative Proportion  0.9987054900 0.9988850138 0.9990293638 0.9991652077
##                             Comp.48      Comp.49      Comp.50      Comp.51
## Standard deviation     0.0913520744 0.0884807831 0.0849282758 8.036697e-02
## Proportion of Variance 0.0001159262 0.0001087533 0.0001001957 8.972218e-05
## Cumulative Proportion  0.9992811339 0.9993898872 0.9994900830 9.995798e-01
##                             Comp.52      Comp.53      Comp.54      Comp.55
## Standard deviation     7.670105e-02 7.404935e-02 6.740711e-02 6.365644e-02
```

```
## Proportion of Variance 8.172356e-05 7.617055e-05 6.311842e-05 5.628975e-05
## Cumulative Proportion  9.996615e-01 9.997377e-01 9.998008e-01 9.998571e-01
##                             Comp.56      Comp.57      Comp.58      Comp.59
## Standard deviation     5.988363e-02 4.475628e-02 4.326690e-02 3.956368e-02
## Proportion of Variance 4.981509e-05 2.782612e-05 2.600496e-05 2.174393e-05
## Cumulative Proportion  9.999069e-01 9.999347e-01 9.999608e-01 9.999825e-01
##                             Comp.60      Comp.61      Comp.62      Comp.63
## Standard deviation     2.580847e-02 1.602058e-02 1.168558e-02 1.086949e-02
## Proportion of Variance 9.252713e-06 3.565341e-06 1.896903e-06 1.641205e-06
## Cumulative Proportion  9.999918e-01 9.999953e-01 9.999972e-01 9.999989e-01
##                             Comp.64      Comp.65      Comp.66 Comp.67 Comp.68
## Standard deviation     9.084114e-03 4.675022e-08 3.102016e-08       0       0
## Proportion of Variance 1.146330e-06 3.036072e-17 1.336696e-17       0       0
## Cumulative Proportion  1.000000e+00 1.000000e+00 1.000000e+00       1       1
##                        Comp.69 Comp.70 Comp.71 Comp.72
## Standard deviation           0       0       0       0
## Proportion of Variance       0       0       0       0
## Cumulative Proportion        1       1       1       1
```

Looking to the principal components one can see that 17 components are needed in order to explain the 95% of the data variation.

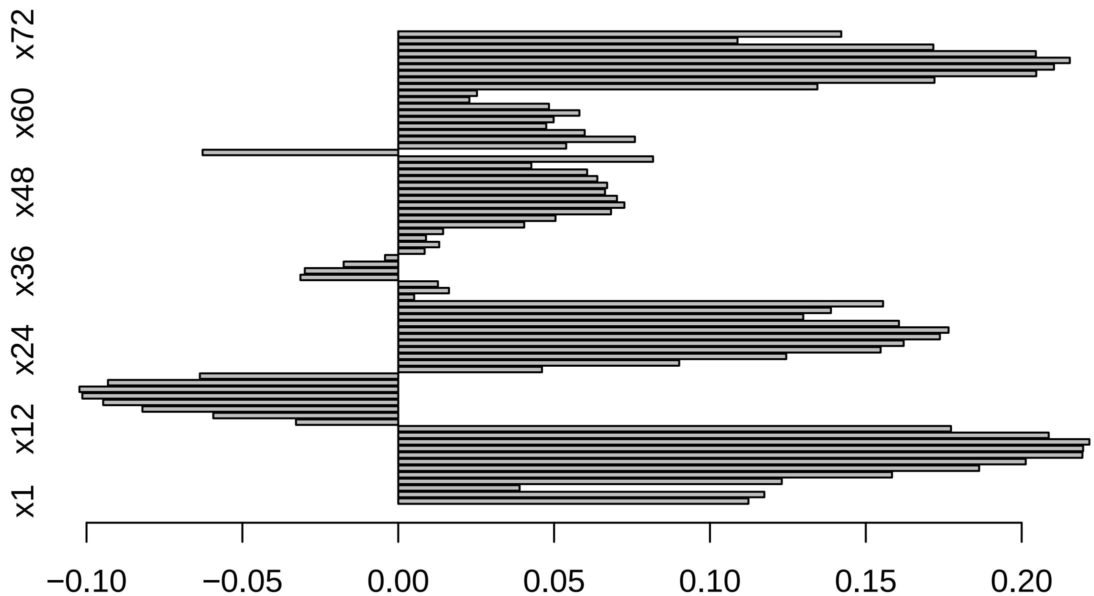
The next step is to check the importance that each variable has in each of the 17th principal components which we consider significant.

```r
l <- data.frame(pca$loadings[,1:17])
head(l)
```

```
##        Comp.1     Comp.2       Comp.3      Comp.4       Comp.5      Comp.6
## x1 0.11234649 0.19199901  0.119100563 0.090254064  0.002409457  0.03854674
## x2 0.11744560 0.19155408  0.114877746 0.077021320 -0.003949036  0.02017171
## x3 0.03893855 0.13330638  0.092953342 0.127367942  0.038217303  0.11488100
## x4 0.12303975 0.17952297  0.098858603 0.040173983 -0.019053747 -0.02201165
## x5 0.15840681 0.16232073 -0.008475662 0.062796554  0.002610569 -0.07775550
## x6 0.18635406 0.06927434 -0.121906994 0.009531728 -0.040445182 -0.12250458
##           Comp.7      Comp.8     Comp.9      Comp.10    Comp.11      Comp.12
## x1  0.0799879285 0.136441085 0.03387566  0.023998931 0.08782882  0.006751505
## x2  0.0631368031 0.147832556 0.03114743  0.035611194 0.09003051 -0.024520631
## x3  0.1317884804 0.008044929 0.01837754 -0.072087509 0.02471100  0.187158157
## x4  0.0139682347 0.163066990 0.02441388  0.057984524 0.08283012 -0.097545867
## x5 -0.0005471538 0.194991159 0.05373537  0.020375652 0.04721878 -0.038316382
## x6 -0.0263493959 0.206543316 0.02617906  0.002537998 0.03023852 -0.043768161
##         Comp.13     Comp.14       Comp.15      Comp.16     Comp.17
## x1  0.007703451  0.02072805  1.012833e-02  0.026647670  0.07246095
## x2  0.002068803  0.02363510  4.046959e-05 -0.057899578  0.04172298
## x3  0.035755218 -0.01189155  6.349231e-02  0.546938191  0.22467173
## x4 -0.012076600  0.02588064 -2.206488e-02 -0.242715135 -0.03588205
## x5  0.093104799 -0.03530430 -1.567965e-02 -0.009450296 -0.07807828
## x6  0.121889939 -0.04036149 -1.993386e-02  0.011038690 -0.07530793
```

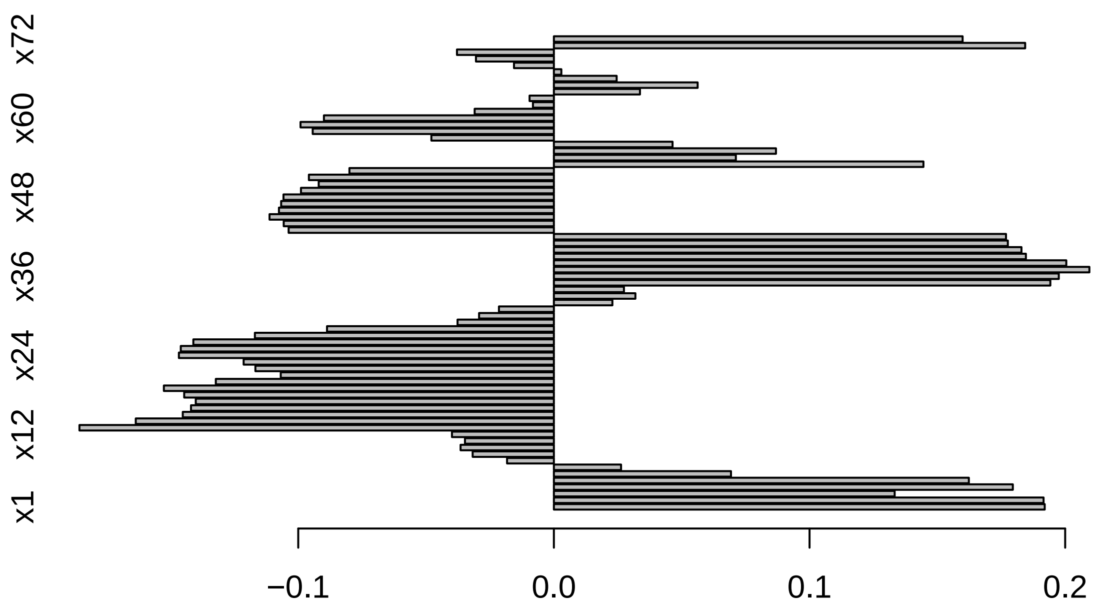
```r
names <- row.names(l)
barplot(l[,1], names.arg = rownames(l), horiz = T)
title("1st Principal Component")
```

1st Principal Component

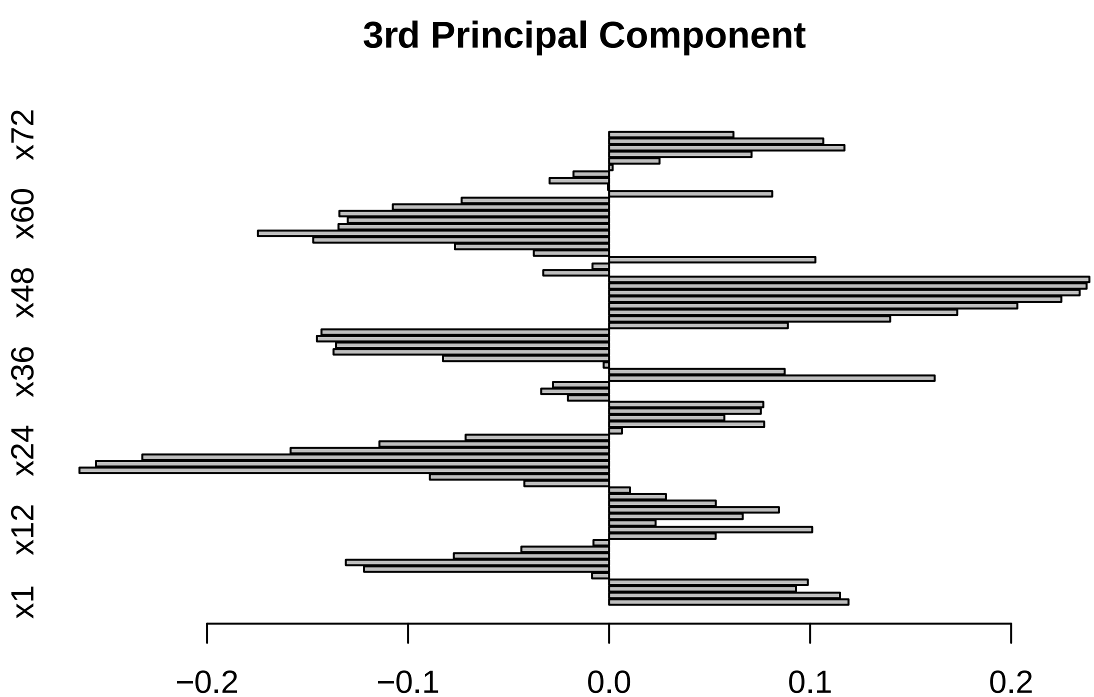



```r
barplot(l[,2], names.arg = rownames(l), horiz = T)
title("2nd Principal Component")
```

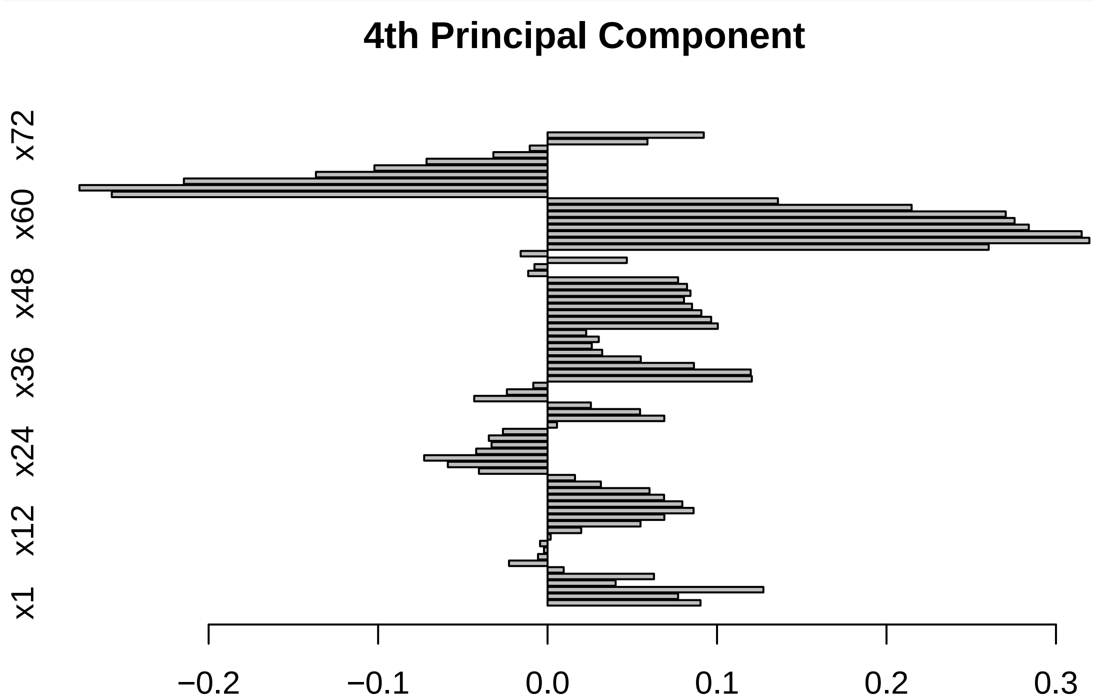
2nd Principal Component



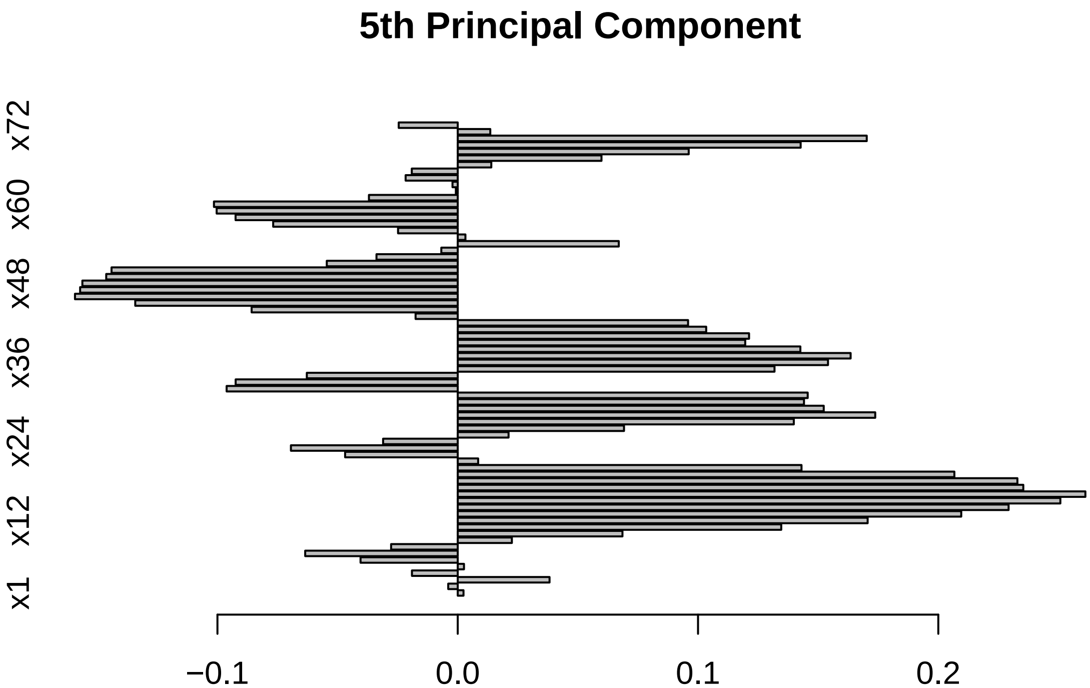

```r
barplot(l[,3], names.arg = rownames(l), horiz = T)
title("3rd Principal Component")
```




```r
barplot(l[,4], names.arg = rownames(l), horiz = T)
title("4th Principal Component")
```




```r
barplot(l[,5], names.arg = rownames(l), horiz = T)
title("5th Principal Component")
```

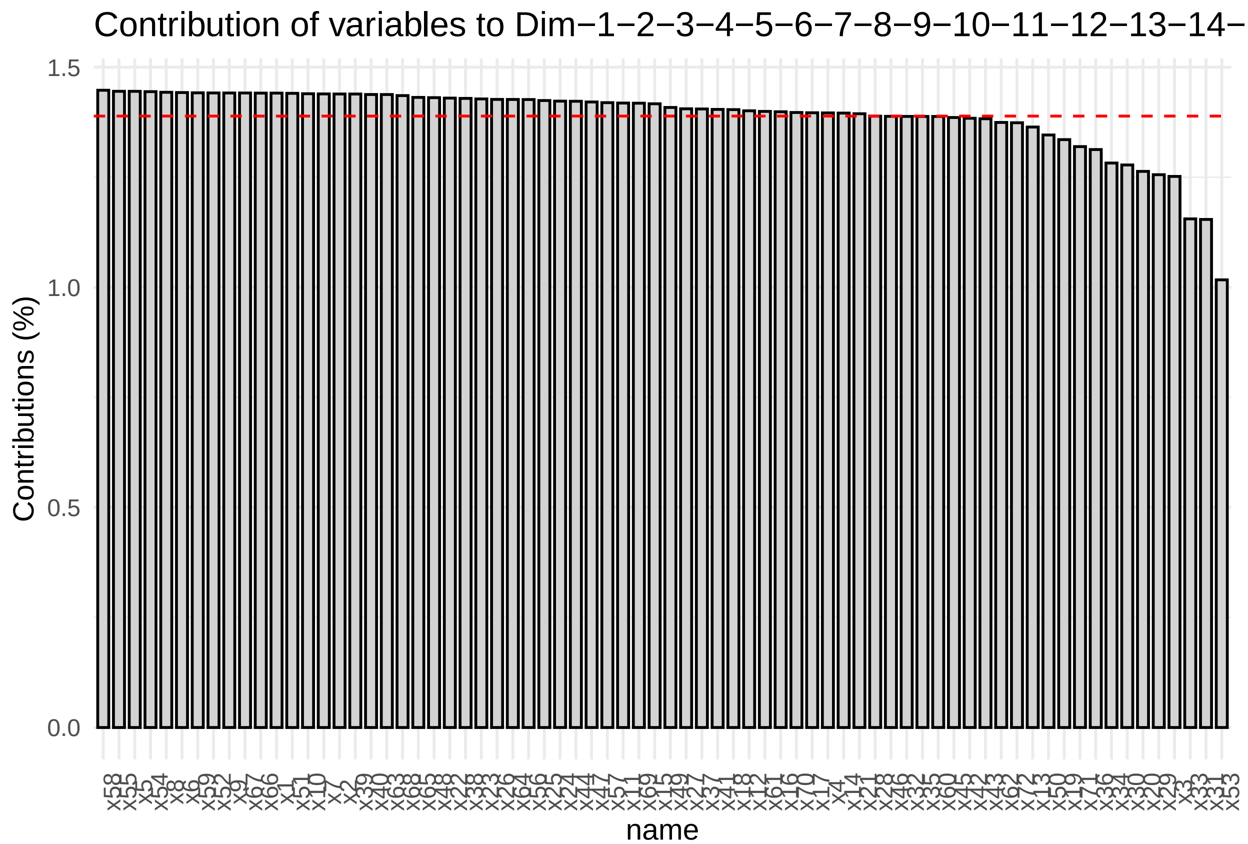


In this plots it can be seen that there is a lot of variables that contribute in each of the principal components. To know which are the most important variables and if there is any variable that can be removed from the dataset I am going to do a contribution plot in order to check if there is any variable that is not significantly contributing to any of the 17PC which explain the 95% of the variance.

```
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
contrib <- fviz_contrib(pca, choice="var", axes = 1:17,
         fill = "lightgray", color = "black") +
         theme_minimal() +
         theme(axis.text.x = element_text(angle=90))
contrib
```




```
contrib.data <- contrib$data[with(contrib$data, order(contrib)), ]
nsign <- contrib.data[1:13,]
nsign <- c(nsign$name)
```

We can see that there are 13 variables (53, 31, 33, 3, 29, 20, 30, 34, 36, 71, 19, 50, 13) with a non significant contribution to the principal components that explain the 95% of the variance of our dataset. This are the variables that following the PCA analysis can be removed. As it's seen there are a lot of variables whom one could consider removing from the original data however we deciside to keep them and use this analysis to compare with the variables that are demed unimportend when we do the prunned tree and the random forest analysisi.

2 Splitting data

```
data$y=as.factor(data$y)

set.seed(params$seed)
pt <- params$partition

inTrain <- createDataPartition(y=data$y, p=pt, list=FALSE)
str(inTrain)
```

```
##  int [1:2816, 1] 2 6 11 13 15 16 19 21 22 23 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "Resample1"
```

```r
training <- data[-inTrain,]
testing <- data[inTrain,]
nrow(training)
```

```
## [1] 2815
```

3 Pruned single tree:

```r
tree.data <- tree(y~., data, subset= inTrain, split="deviance")
summary(tree.data)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = data, subset = inTrain, split = "deviance")
## Variables actually used in tree construction:
## [1] "x38" "x37" "x65" "x12" "x40"
## Number of terminal nodes:  6
## Residual mean deviance:  1.099 = 3089 / 2810
## Misclassification error rate: 0.2923 = 823 / 2816
```

```r
set.seed(params$seed)
cv.data=cv.tree(tree.data)
names(cv.data)
```

```
## [1] "size"   "dev"    "k"      "method"
```

```r
cv.data
```

```
## $size
## [1] 6 5 4 3 2 1
##
## $dev
## [1] 3150.043 3176.539 3246.135 3253.552 3353.560 3740.975
##
## $k
## [1]      -Inf  39.60449  49.90155  54.89454 117.21261 388.48898
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

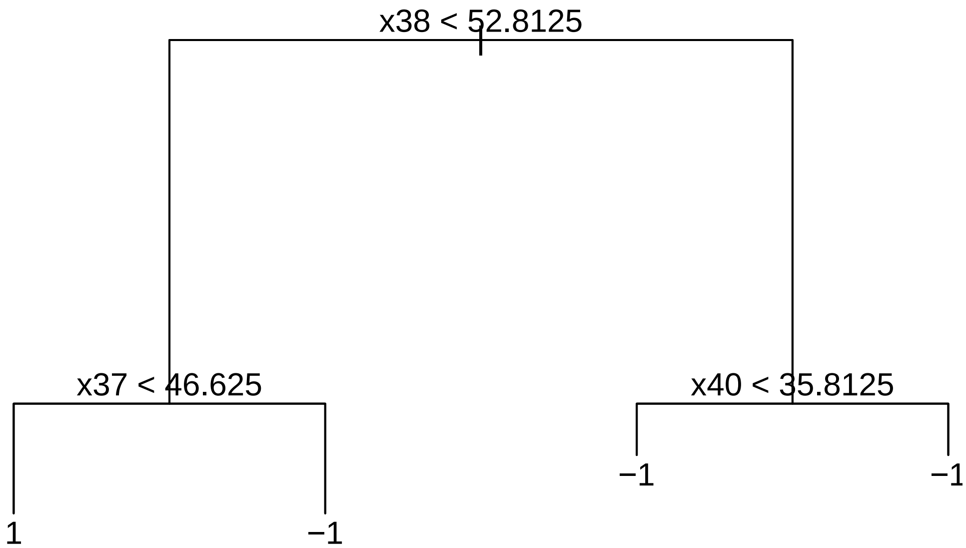
One can see there is an increase in deveiance as the number of nodes get smaller. Since it does not have that much sence to choose the actual mimimum sinse this would not prune the tree at all we chose $n = 4$.

```r
prune.data=prune.tree(tree.data,best=4)
summary(prune.data)
```

```
##
## Classification tree:
## snip.tree(tree = tree.data, nodes = 5:4)
## Variables actually used in tree construction:
## [1] "x38" "x37" "x40"
## Number of terminal nodes:  4
```

```
## Residual mean deviance:  1.13 = 3178 / 2812
## Misclassification error rate: 0.3132 = 882 / 2816
```

```r
plot(prune.data)
text(prune.data,pretty=0)
```



Make predictions on the test set to evaluate the classifier.

```r
yhat_1=predict(prune.data,newdata=testing, type = "class")
res <- table(yhat_1,testing$y)
res
```

```
##
## yhat_1   -1    1
##     -1 1703  838
##     1    44  231
```

```r
accrcy <- sum(diag(res)/sum(res))
```

<<<<<<< HEAD The accuracy that we obtain in this model is 0.6867898.

Random forrest:

In the random forrest model one needs to decide how many variables *mtry* are assesed in each node and how many trees *ntree* are used in the forrest. Caret only has tuning for the number of parameters assessed in each node and this tuningprocess is quite slow. Therefor the decition is made to take a gready approch towards tuning these parameters. First we use tuning by caret to tune the number of variables. Then the optimum is chosen and used when tuning the number of trees. The order is decided this way since the number of parameters assesed in each node is said to hava a bigger inpack on the result than the number of trees.

A tunegrid is created with 15 values from 1:15 for *mtry* to tun the model. Our train function will change number of entry variable at each split according to the tunegrid. The default numebr of trees is 500.

```r
#Very slow
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(mlbench)
library(caret)
library(e1071)

control <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3,
                        search='grid')

tunegrid <- expand.grid(.mtry = (1:15))

rf_gridsearch <- train(y ~ .,
                       data = training,
                       method = 'rf',
                       metric = 'Accuracy',
                       tuneGrid = tunegrid,
                       trControl=control)
print(rf_gridsearch)
```

```
## Random Forest
##
## 2815 samples
##   72 predictor
##    2 classes: '-1', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 2533, 2534, 2533, 2534, 2533, 2533, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.7806923  0.5099224
##    2    0.7848357  0.5232310
##    3    0.7848303  0.5246080
##    4    0.7849514  0.5253865
##    5    0.7867316  0.5301324
##    6    0.7838863  0.5242217
##    7    0.7840045  0.5244932
##    8    0.7830610  0.5230448
##    9    0.7860190  0.5298642
##   10    0.7844773  0.5263509
##   11    0.7823467  0.5223274
##   12    0.7856636  0.5294921
##   13    0.7869731  0.5324345
##   14    0.7824687  0.5231927
```

```
##   15    0.7832966  0.5249542
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 13.
```

Eventually the optimum value for *mtry* si chosen automatically by the algoritm.

A loop is made to tune for *ntree* with the optimum value for *mtry*. This is also a rather slow process hence the decition is made to only chech for number of trees that are multiples of 50.

```
mtry.opt <- which.max(rf_gridsearch$results$Accuracy)
accuratcy.list <- rep(0, 20)

control <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3)

for (i in 1:20){
  model <- train(y ~ .,
                 data = training,
                 method = 'rf',
                 metric = 'Accuracy',
                 tuneGrid = data.frame(mtry = mtry.opt),
                 ntree = i*50,
                 trControl=control)
  yhat <- predict(model, newdata = testing, type = "raw")
  res <- table(yhat,testing$y)
  accuratcy.list[i] <- sum(diag(res)/sum(res))
  #print(sum(diag(res)/sum(res)))
}
```

As one can see with the exeption of $ntree= 50 $ there is very little difference in the performance of each itteration. This supports the assumption that *mtry* has a greater inpack on the result.

```
x = seq(1,20)

ntree.opt = which.max(accuratcy.list)
ntree.opt
```

```
## [1] 16
```

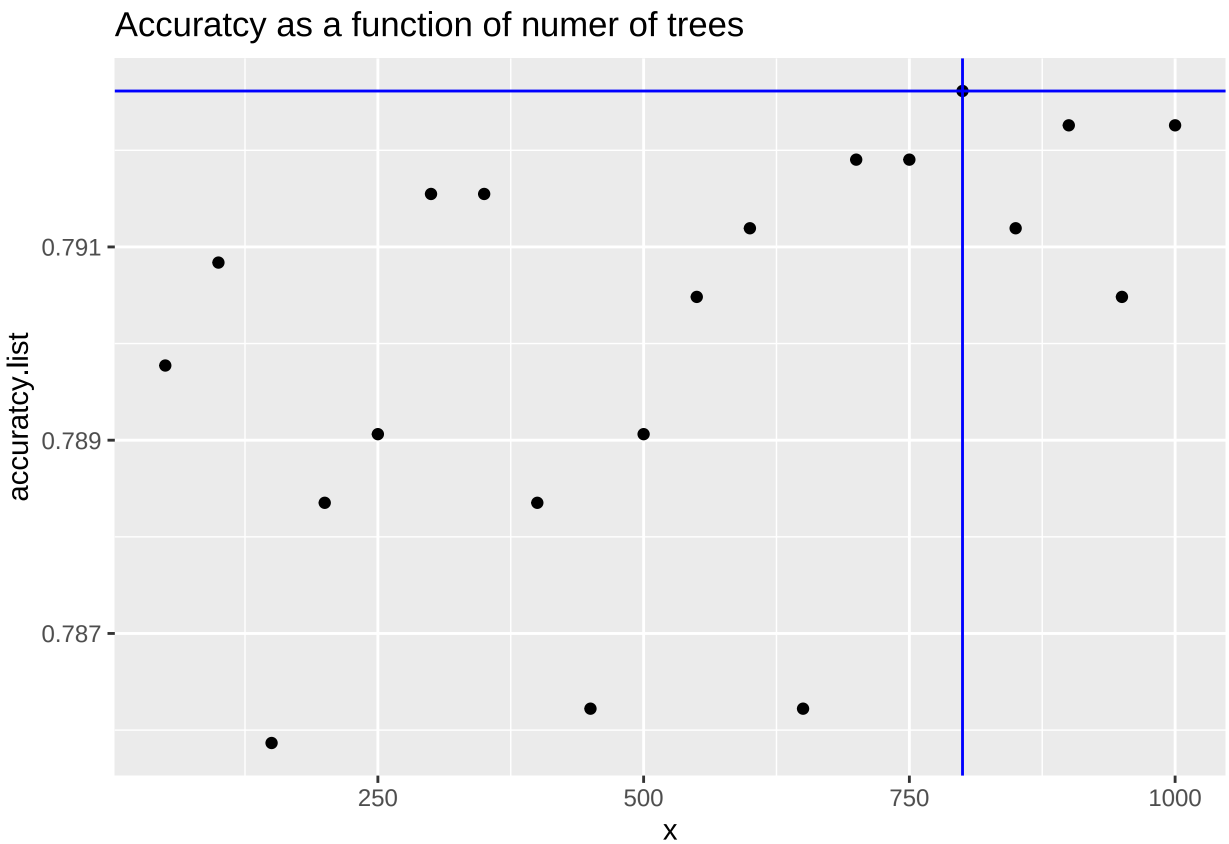
```
length(x)
```

```
## [1] 20
```

```
length(accuratcy.list)
```

```
## [1] 20
```

```
ntree <- which.max(accuratcy.list)*50

ggplot(data = tibble(x = seq(1, 20)*50, accuratcy.list), aes(x, accuratcy.list)) + geom_point() + geom_
```



There is not a very big difference in accuratcy for any number above 50.

```
control <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3)


rf_final <- train(y ~ .,
                       data = training,
                       method = 'rf',
                       metric = 'Accuracy',
                       ntree = ntree.opt,
                       tuneGrid = data.frame(mtry = mtry.opt),
                       trControl=control)
```

Make predictions with the final model and get accuratcy statistics

```
yhat_final <- predict(rf_final, newdata = testing, type = "raw")
res <- table(yhat_final,testing$y)
res
```

```
##
## yhat_final   -1    1
##         -1 1531  409
##         1   216  660
```

```
accrcy_rf <- sum(diag(res)/sum(res))
```

Now the accuratcy obtained is 0.778054. This is much higher thtn with the single tree from earlier.

Use varImp to get the variable importance, the 10 most important are shown.

```r
varImp(rf_final)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 72)
##
##     Overall
## x38  100.00
## x40   97.54
## x37   80.62
## x41   64.60
## x72   48.90
## x42   42.14
## x59   42.09
## x64   40.57
## x71   37.88
## x36   36.72
## x39   34.70
## x57   33.17
## x58   32.75
## x21   30.41
## x50   30.04
## x63   30.03
## x47   29.36
## x65   28.64
## x51   28.51
## x19   28.34
```

Variables originally deamed less important by PCA analytics:

```r
nsign
```

```
##  [1] 53 31 33  3 29 20 30 34 36 71 19 50 13
```

As one can see there are three variables $x36$, $x20$ and $x53$ which are not important in the PCA analysis but when we do the random forest they are important.

5 Comparation of models:

The pruend tree had an accuratcy of 0.6867898 while the random forrest has an accuratcy of 0.778054. The random forrest preforms significally better.

6 Gradiant Boosting

We are using gbm function: Generalized Boosted Regression Modeling To run gradient boosting algorithm we use a GLM funcion.

Concerning the use of the function :

- n.trees is equivalent to the number of iterations and the number of basis functions in the additive expansion.
- we use stumps to study iteration influence. This is obtained by choosing interaction.depth equal to 1.

Moreover, it is important to replace binary variables (due to the fact that we are working on a classification) {-1,1} by {0,1} from our training data set at the begining to use this function and to fit the model.

```
library("ISLR")
library("gbm")
```

```
## Loaded gbm 2.1.5
```

```
library("pROC")
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(tidyr)
library(ggplot2)
library(dplyr)

set.seed(2)
#Replace binary variables by {0,1}
training$y <- ifelse(training$y==-1,0,1)

#Fitting the model & prediction
boost.compounds=gbm(y~.,data=training,distribution="adaboost",n.trees=2000,interaction.depth=1)

#Evaluating the error for testing data set
yhat_gbm_2000 =predict(boost.compounds,newdata=testing,n.trees=2000, type = "response")
yhat_gbm_2000 <- ifelse(yhat_gbm_2000 < 0.5,0,1)
res <- table(yhat_gbm_2000,testing$y)
accrcy_gbm_test_2000 <- sum(diag(res)/sum(res))
accrcy_gbm_test_2000
```

```
## [1] 0.7574574
```

```
#Evaluating the error for training data set
yhat_gbm_train_2000 =predict(boost.compounds,newdata=training,n.trees=2000, type = "response")
yhat_gbm_train_2000 <- ifelse(yhat_gbm_train_2000 < 0.5,0,1)
res <- table(yhat_gbm_train_2000,training$y)
accrcy_gbm_train_2000 <- sum(diag(res)/sum(res))
accrcy_gbm_train_2000
```

```
## [1] 0.8650089
```

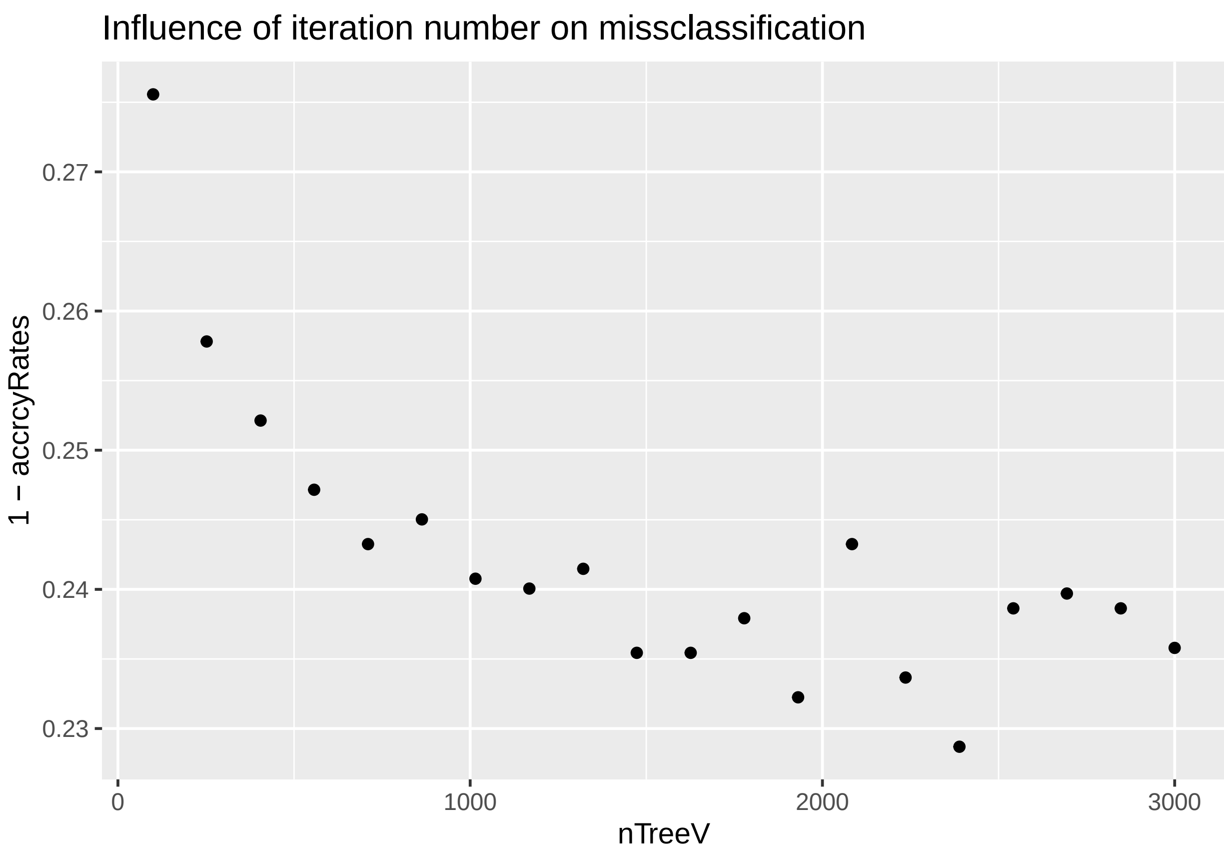
As we can see, the accuracy rate is really higher on training data (0.7574574) set than on testing data set (0.8650089), which is expected due to the fact that the model is fitted on the training data set.

We should retain the missclassification rate of testing data set. We can notice that the model for 2000 iterations with stumps as trees is more efficient than a pruned single tree model but less efficient that with a forest trees model.

Then we can put forward the influence of the number of iterations (ie. the number of trees) on the missclassification on the testing data set:

```
sampling = 20
accrcyRates <- rep(0,sampling)
```

```
firstValue = 100
lastValue = 3000
nTreeV = floor(seq(from = firstValue, to = lastValue, by = (lastValue - firstValue)/(sampling-1)))

for (i in 1:sampling) {
  boost.compounds=gbm(y~.,data=training,distribution="adaboost",n.trees=nTreeV[i],interaction.depth=1)
  yhat_gbm =predict(boost.compounds,newdata=testing,n.trees=nTreeV[i], type = "response")
  yhat_gbm <- ifelse(yhat_gbm < 0.5,0,1)
  res <- table(yhat_gbm,testing$y)
  err_gbm <- sum(diag(res)/sum(res))
  accrcyRates[i] <- err_gbm
}

x <- data.frame(nTreeV,1-accrcyRates)
ggplot(x) + aes(x=nTreeV, y=1-accrcyRates) + geom_point()+ggtitle("Influence of iteration number on mis
```

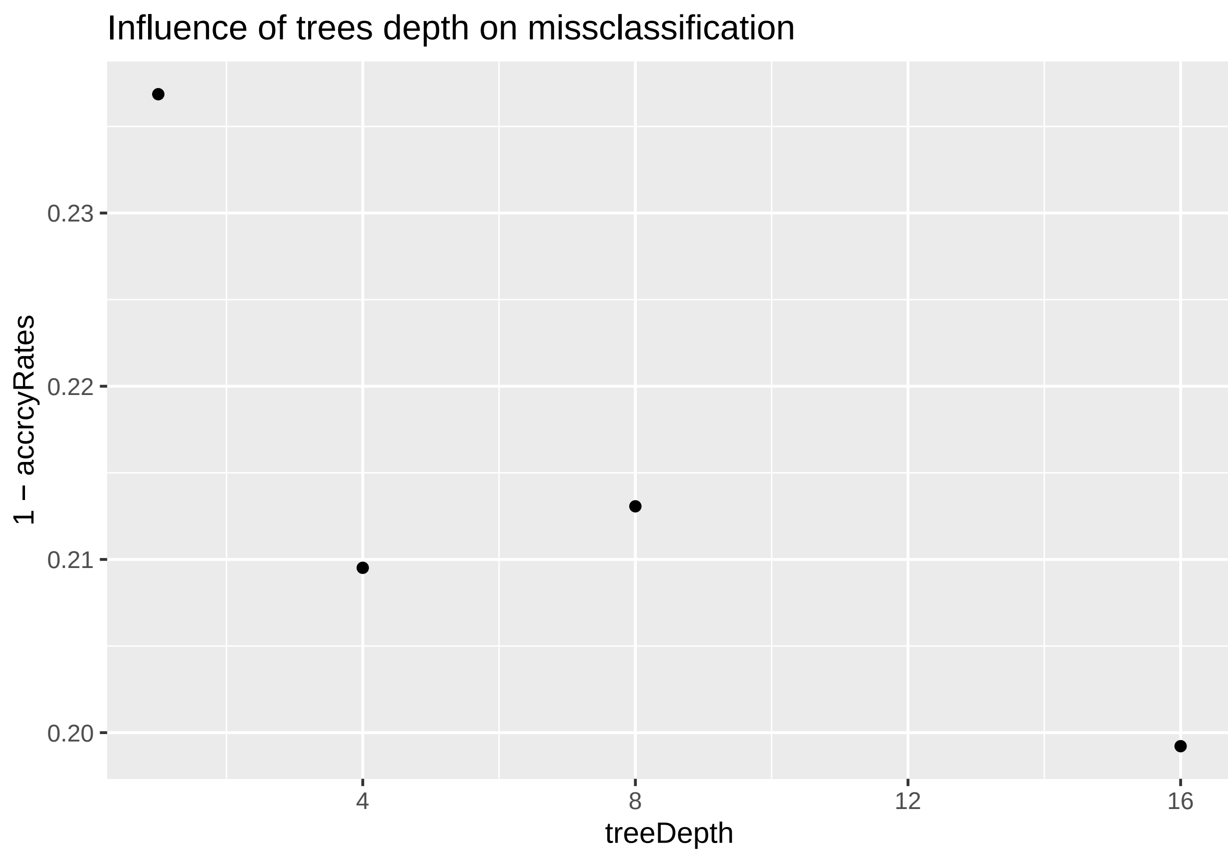


On this plot, as expected, we can notice that missclassification as a function of iteration number is more or less a decreasing function. We can also notice that above a certain number of iteration, the functions is almost constant.

We can then study the influence of trees maximum depth to optimize our solution :

```
#We take the n.tree value with the best accrcyRates
nTree <- nTreeV[which(max(accrcyRates)==accrcyRates)]
treeDepth <- c(1,4,8,16)
accrcyRates <- rep(0,length(treeDepth))

k <- 0
```

```r
for (i in treeDepth) {
  k = k +1
  boost.compounds=gbm(y~.,data=training,distribution="adaboost",n.trees=nTree,interaction.depth=i)
  yhat_gbm =predict(boost.compounds,newdata=testing,n.trees=nTree, type = "response")
  yhat_gbm <- ifelse(yhat_gbm < 0.5,0,1)
  res <- table(yhat_gbm,testing$y)
  err_gbm <- sum(diag(res)/sum(res))
  accrcyRates[k] <- err_gbm
}

x <- data.frame(treeDepth,1-accrcyRates)
ggplot(x, aes(x=treeDepth, y=1-accrcyRates)) + geom_point()+ggtitle("Influence of trees depth on misscla
```



On this plot the decrease is also more or less decreadig with a minimum at 16.

In the best condition (which means with trees depth equal to 16) we can reach 0.8007812 of accuracy rate. This means that this algorithm is the most efficient among those explored in this report.

To conclude we can say that trees depth & iteration number are two factor which can be used to optimize our classification. Nevertheless, it important to be careful because both two factors are very greedy in ressource.