Lecture 5: solutions

Statistical Computing with R

Exercise 2

1

```
x = c(3, 4, 6, 2, 4, 3, 8, 1, 3)
# t = 5, k = 3
sum(x[(5-3+1):5])/3
```

```
## [1] 4
```

In this second case $max(2-3+1, 1) = 1$

```
# t = 2, k = 3
sum(x[1:2])/3
```

```
## [1] 2.333333
```

2

```
MA_1 <- function(x, t, k){
  return(sum(x[max((t-k+1), 1):t])/k)
}
```

3

```
for (i in 1:length(x)){
  print(MA_1(x, i, 3))
}
```

```
## [1] 1
## [1] 2.333333
## [1] 4.333333
## [1] 4
## [1] 4
## [1] 3
## [1] 5
## [1] 4
## [1] 4
```

Exercise 3

1

For $t = 5$ and $k = 3$ the result is the same as before.

```r
# t = 5, k = 3
sum(x[(5-3+1):5])/3
```

```
## [1] 4
```

However, for $t = 2$ and $k = 3$ the result is different, now we would get `NA` because $t < k$.

2

We can return `NA` if the condition $t < k$ is satisfied, and the moving average otherwise.

```r
MA_2 <- function(x, t, k){
  if (t < k){
    return(NA)
  }else{
    return(sum(x[(t-k+1):t])/k)
  }
}
```

3

Here we can see how the first two values have changed but the remaining ones are the same.

```r
for (i in 1:length(x)){
  print(MA_2(x, i, 3))
}
```

```
## [1] NA
## [1] NA
## [1] 4.333333
## [1] 4
## [1] 4
## [1] 3
## [1] 5
## [1] 4
## [1] 4
```

Exercise 4

```r
set.seed(4)
r = 100
A = matrix(NA, nrow = r, ncol = 5)
A[ , 1:2] = rpois(2*r, 6)
A[ , 3] = rgamma(r, shape = 2, scale = 1)
A[ , 4:5] = rbinom(r, 2, 0.6)
```

1

```r
for (i in 1:ncol(A)){
  print(mean(A[, i]))
}
```

```
## [1] 6.38
## [1] 5.85
## [1] 1.998239
## [1] 1.25
## [1] 1.25
```

2

```r
x <- numeric(nrow(A))
for (i in 1:nrow(A)){
  x[i] <- mean(A[i, ])
}
x
```

```
##   [1] 3.291189 2.272466 2.640902 3.111095 3.626159 2.785598 3.287181 3.515471
##   [9] 5.013893 3.110099 3.349365 3.179451 2.672720 3.795876 3.119570 3.667153
##  [17] 3.750258 3.420071 5.334524 2.823049 2.168831 4.313738 4.170264 3.922343
##  [25] 3.057442 5.011309 4.680214 3.940540 3.171850 3.249605 3.128821 2.071929
##  [33] 3.800002 2.472575 2.986846 4.915670 2.733382 3.562307 3.904345 1.419388
##  [41] 4.458485 3.205480 3.487084 3.835591 3.389630 2.832274 3.915139 4.057130
##  [49] 3.440784 2.912621 2.545943 2.636555 5.399106 4.273108 3.336074 3.208792
##  [57] 2.317662 2.400785 3.434188 3.591113 4.200431 3.417701 3.966343 3.057654
##  [65] 3.173224 3.525466 1.651295 2.847296 3.137991 4.321664 2.176795 4.481994
##  [73] 2.871325 3.040652 2.476331 2.141397 3.193488 3.632953 2.846659 3.463482
##  [81] 3.393715 3.698968 3.409908 4.833123 3.480056 1.833386 2.970404 4.638822
##  [89] 3.500193 5.074874 2.475508 3.390183 4.076886 1.689733 2.497737 3.116681
##  [97] 3.756823 3.038489 2.319318 2.620802
```

3

```r
my_ColMeans <- function(A){
    x <- numeric(ncol(A))
    for (i in 1:ncol(A)){
      x[i] <- mean(A[, i])
    }
    return(x)
}
```

```
my_ColMeans(A)
```

```
## [1] 6.380000 5.850000 1.998239 1.250000 1.250000
```

```
colMeans(A)
```

```
## [1] 6.380000 5.850000 1.998239 1.250000 1.250000
```

4

```
my_statistics <- function(A){
    x <- matrix(NA, ncol = 3, nrow = ncol(A))
    for (i in 1:ncol(A)){
      x[i, 1] <- i
      x[i, 2] <- mean(A[, i])
      x[i, 3] <- sd(A[, i])
    }
    colnames(x) <- c("column", "mean", "standard deviation")

    return(data.frame(x))
}

my_statistics(A)
```

```
##   column     mean standard.deviation
## 1      1 6.380000          2.5654267
## 2      2 5.850000          2.3926445
## 3      3 1.998239          1.7092407
## 4      4 1.250000          0.7159792
## 5      5 1.250000          0.7159792
```

Exercise 5

1

We can use the read.csv function to import the irish_polls.csv file. Note that in this case the defaults of the function are appropriate and we don't need to change them (`?read.csv`).

```
# setwd()
df <- read.csv("irish_polls.csv")
```

2 & 3

As is often the case at the beginning of a data analysis project, our data are in a format that cannot be directly used for our analyses. Therefore, we need to "polish" our data frame before proceeding to analyze it. The main problem is that the vote percentages are strings including an % sign. This needs to be removed. The following loop checks for each column if it contains values equal to the string "Not Available". If it is, "Not Available" is replaced with `NA`. Then, it takes each row of that column, and with `nchar()` it removes the last character, the %. Lastly, the column is converted from character to numeric.

```
for (i in 10:21) {
  na.pos = which(df[ , i] == 'Not Available')
  if (length(na.pos) > 0) df[na.pos, i] = NA
  df[ , i] = substr(df[ , i], 1, nchar(df[ , i]) - 1)
  df[ , i] = as.numeric(df[ , i])
}
```

4 & 5

We can calculate the mean of the top 10 rows by using loops like we've learned, or by using the `apply` function *(?apply)*, with the `mean` as the input function. Then we combine all parties with less than 6% of the votes in an *other* category.

```
# Using what we've learned the obvious approach might be:

# Take a subset of the columns we want to work with
subset_df <- df[1:10, 10:21]

# Calculate the means
party.mean = numeric(ncol(subset_df))
for (col in 1:ncol(subset_df)) {
  party.mean[col] = mean(subset_df[, col], na.rm = T)
}
names(party.mean) <- colnames(subset_df)

# Alternatively, we can get the same means using the apply() family of functions
# This is both faster and uses less code!

party.mean <- apply(df[1:10, 10:21], 2, function(x) mean(x, na.rm = T))
```

Then we continue with the other steps:

```
# subset all parties with more than 6% votes
party.mean[which(party.mean > 6)]
```

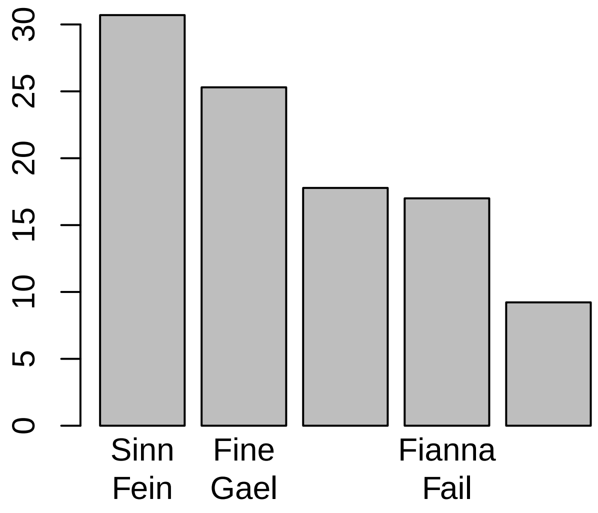
```
##    Fine.Gael  Fianna.Fáil    Sinn.Féin Independents
##    25.300000    17.000000    30.700000     9.222222
```

```
# Combining and adding other parties category
party <- party.mean[which(party.mean > 6)]
names(party) <- c("Fine Gael", "Fianna Fail", "Sinn Fein", "Independent")
party <- c(party, "Others" = 100 - sum(party.mean[which(party.mean > 6)]))
```

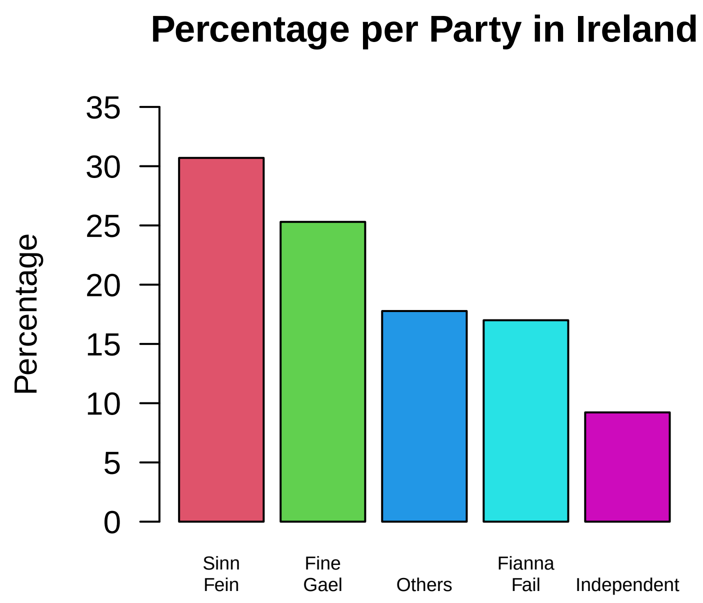
6

We first need to order the data based on vote percentage. Then we can create a barplot. You can customize your plot by adding color `col`, labels `names.arg` and font size `cex.names`, and the title `main`. Here we have also used `las = 1` to make the axis labels horizontal and `horiz = T` to draw the bars horizontally with the first on the bottom.

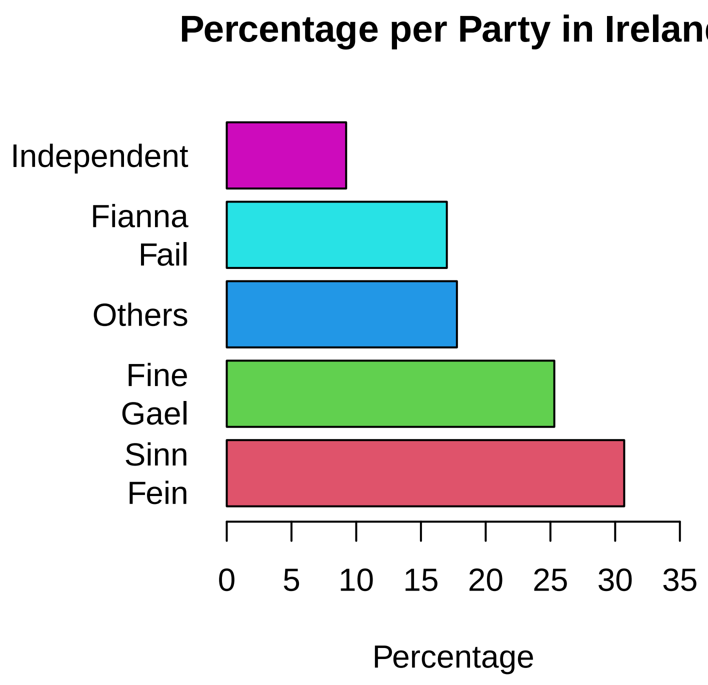
```r
# formatting data
party <- data.frame(percentage = party,
                    name = c("Fine\nGael", "Fianna\nFail", "Sinn\nFein",
                             "Independent", "Others"))
party <- party[order(party$percentage, decreasing = T),]

# simple bar plot
barplot(party$percentage, names.arg = party$name)
```



```r
# pimped bar plot
barplot(party$percentage,
        names.arg = party$name,
        col = 2:6,
        ylim = c(0, 35),
        las = 1,
        cex.names=0.55,
        ylab = "Percentage",
        main = "Percentage per Party in Ireland")
```



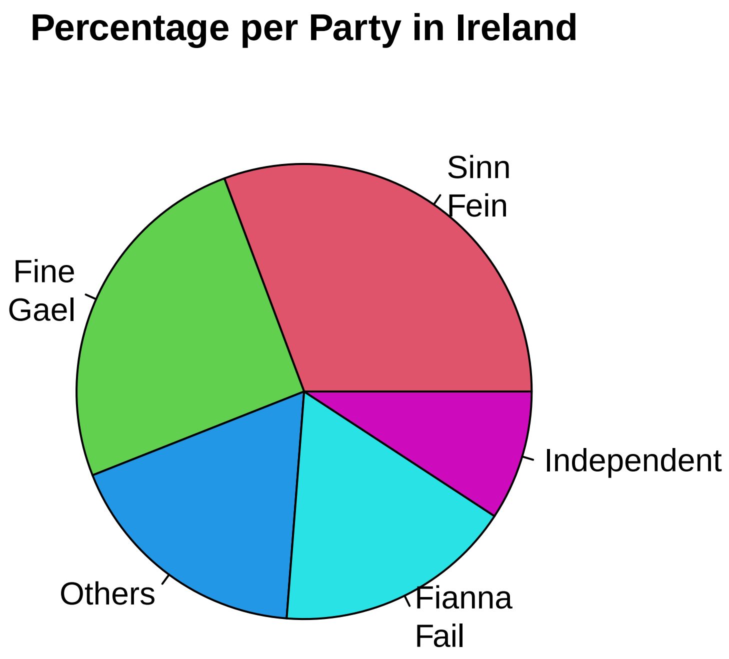
```
par(mar=c(5,6,4,2)+.1)
barplot(party$percentage,
        names.arg = party$name,
        col = 2:6,
        xlim = c(0, 35),
        las = 1,
        cex.names=1,
        xlab = "Percentage",
        main = "Percentage per Party in Ireland",
        horiz = T)
```



7

Using the `pie` we can create a pie chart. With many of the same arguments you can create nice looking graphs.

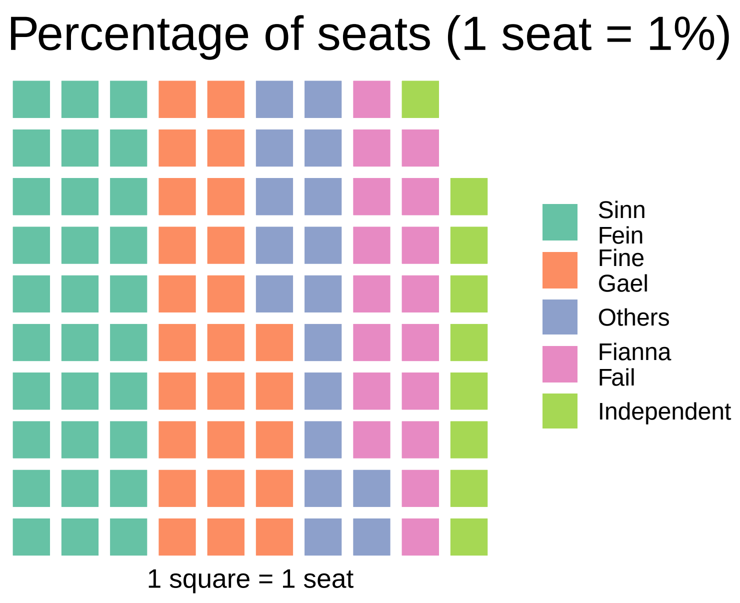
```
par(mar=c(1,1,1,3))
pie(party$percentage,
    labels = party$name,
    col = 2:6,
    main = "Percentage per Party in Ireland")
```



8

Install and load the `waffle` package. Remember that the function `waffle()` takes as input a named vector. Then you can create and customize your waffle plot.

```
# install.packages("waffle")
library(waffle)
party.waffle <- party$percentage
names(party.waffle) <- party$name

waffle(party.waffle,
       rows = 10,
       xlab = "1 square = 1 seat",
       title = "Percentage of seats (1 seat = 1%)")
```



9

To create a pdf with the plots, simply start by inputting a string with (the location and) the file name, and end it with `.pdf`, to indicate the file type, into the `pdf()` function. In the next lines, run the code for the plots. Lastly, use `dev.off()` to end the document.

```
pdf('some_charts.pdf')
barplot(party$percentage,
        names.arg = party$name,
        col = 2:6,
        ylim = c(0, 35),
        las = 1,
        cex.names=.525,
        ylab = "Percentage",
        main = "Percentage per Party in Ireland")
pie(party$percentage,
    labels = party$name,
    col = 2:6,
    main = "Percentage per Party in Ireland")
waffle(party.waffle,
       rows = 10,
       xlab = "1 square = 1 seat",
       title = "Percentage of seats (1 seat = 1%)")
dev.off()
```

10

Similarly as to the pdf, you start by opening the graphical device by using the function `jpeg()`. Here you indicate the name and end it with the file type `.jpg`. Then you run the code with your plot and close the graphical device with `dev.off()`.

```
jpeg("barplot.jpg")
barplot(party$percentage,
        names.arg = party$name,
        col = 2:6,
        xlim = c(0, 35),
        las = 1,
        cex.names=.525,
        xlab = "Percentage",
        main = "Percentage per Party in Ireland",
        horiz = T)
dev.off()

jpeg("pie-chart.jpg")
pie(party$percentage,
    labels = party$name,
    col = 2:6,
    main = "Percentage per Party in Ireland")
dev.off()

jpeg("waffle.jpg")
waffle(party.waffle,
       rows = 10,
       xlab = "1 square = 1 seat",
       title = "Percentage of seats (1 seat = 1%)")
dev.off()
```

Exercise 6

1

```
# install.packages("psych")
library(psych)
data(sat.act)
```

2

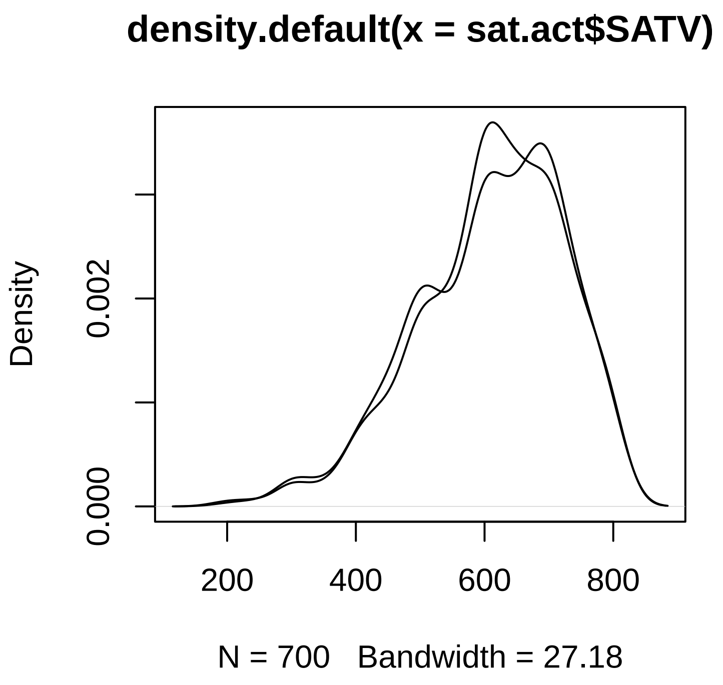
```
?sat.act
```

Looking at the help page, we can see that the ACT scores range from 1-36 and both the SATV and the SATQ scores range from 200-800. Since the SATV and SATQ scores have the same range it would make sense to compare their marginal distributions in one graph.

3

Note that the variable of the SATQ scores contains some NAs. When we check the help page of the function density() we can see that it contains the default argument na.rm = FALSE. So we set it to na.rm = TRUE to be able to plot the scores.

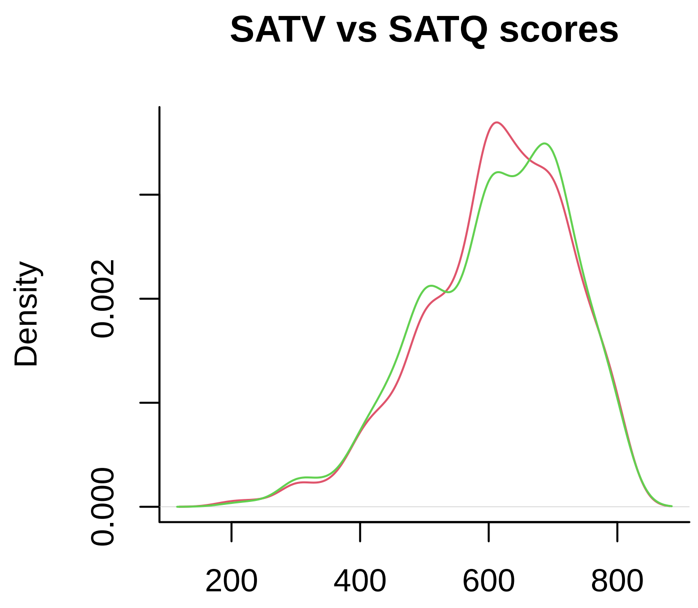
```
plot(density(sat.act$SATV))
lines(density(sat.act$SATQ, na.rm = T))
```



4

Use the function par() to remove the top and the right border of the graph that you create afterwards. You will learn more about par() in the next lecture. For the colours using simply numbers is the easiest way. You can also check colours() and then set the col = argument to the name of a colour you like, such as "darkred" or "forestgreen".

```
par(bty = 'l')
plot(density(sat.act$SATV), main = "SATV vs SATQ scores", xlab = "", col = 2)
lines(density(sat.act$SATQ, na.rm = T), col = 3)
```

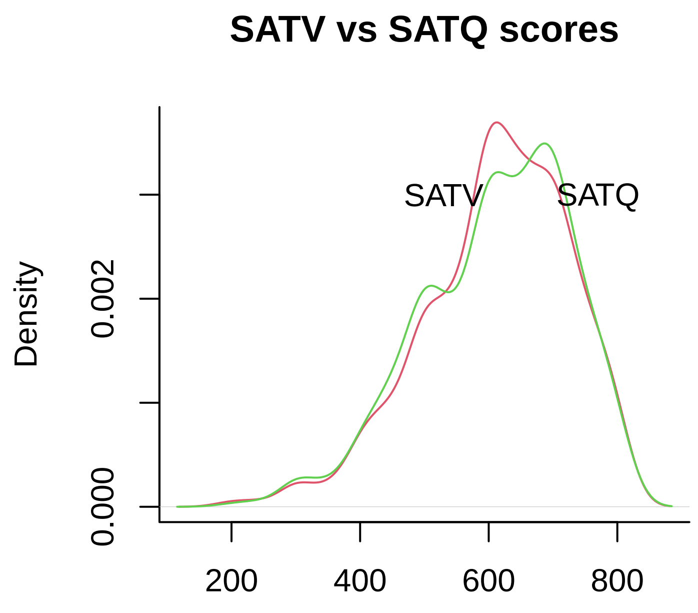


5

```
# Execute in the console:
par(bty = 'l')
plot(density(sat.act$SATV), main = "SATV vs SATQ scores", xlab = "", col = 2)
lines(density(sat.act$SATQ, na.rm = T), col = 3)
locator(2)
```

Using `locator` we find out that we want to place the text “SATV” at the coordinates (530, 0.003) and the text “SATQ” at the coordinates (770, 0.003).

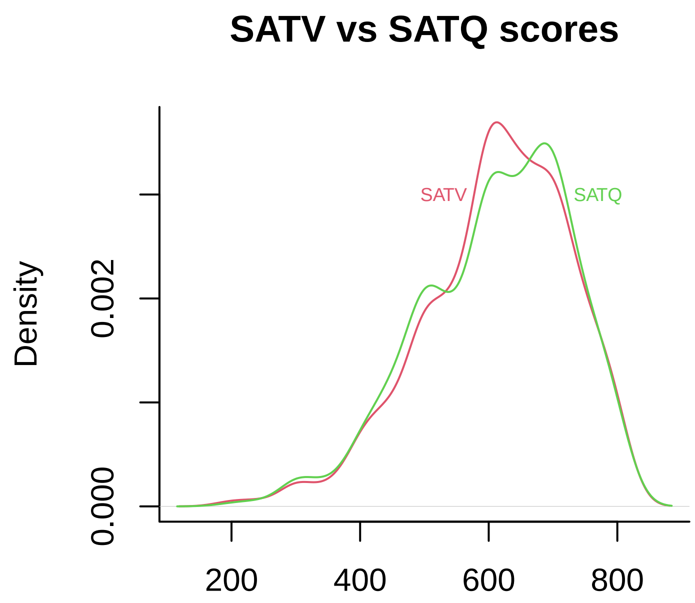
```
par(bty = 'l')
plot(density(sat.act$SATV), main = "SATV vs SATQ scores", xlab = "", col = 2)
lines(density(sat.act$SATQ, na.rm = T), col = 3)
text(530,0.003, "SATV")
text(770,0.003, "SATQ")
```



6

Note that to change the text in our graph, we need to create a new graph, otherwise, the text will just be written on top of the previous text.

```
par(bty = 'l')
plot(density(sat.act$SATV), main = "SATV vs SATQ scores", xlab = "", col = 2)
lines(density(sat.act$SATQ, na.rm = T), col = 3)
text(530,0.003, "SATV", cex = 0.6, col = 2,)
text(770,0.003, "SATQ", cex =  0.6, col = 3)
```


SATV vs SATQ scores
Density
0.000
0.002
200
400
600
800
SATV
SATQ