Solutions Lecture 8

Exercise 1

Question 1

In exercise 2 of lecture 3, we created the following functions.

```
centering <- function(data){
  centered_data <- data - mean(data)
  return(centered_data)
}

scaling <- function(data){
  scaled_data <- data/sd(data)
  return(scaled_data)
}

normalization <- function(data){
  return(scaling(centering(data)))
}
```

Question 2

For the first requirement we need to check that v is vector and numeric. If both are true we will not display the error. For the second requirement we need to check if `any()` of the elements are `NA`.

```
centering <- function(data){
  if(!(is.vector(data) & is.numeric(data))){
    stop("Function input should be a numeric vector \n")
  }
  if(any(is.na(data))){
    warning("The supplied vector contains missing values \n")
  }
  centered_data <- data - mean(data)
  return(centered_data)
}

scaling <- function(data){
  if(!(is.vector(data) & is.numeric(data))){
    stop("Function input should be a numeric vector \n")}
    if(any(is.na(data))){
    warning("The supplied vector contains missing values \n")
  }
  scaled_data <- data/sd(data)
  return(scaled_data)
}

normalization <- function(data){
```

```
  if(!(is.vector(data) & is.numeric(data))){
    stop("Function input should be a numeric vector \n")}
    if(any(is.na(data))){
    warning("The supplied vector contains missing values \n")
  }
  return(scaling(centering(data)))
}
```

Question 3

```
obj1 = matrix(1:10, 5, 2)
obj2 = c(5, 7, 10, -25)
obj3 = c(42, NA, 3, 7)
obj4 = c(pi, 42, 'apple', sqrt(3))
```

```
centering(obj1)
```

```
## Error in centering(obj1): Function input should be a numeric vector
```

```
centering(obj2)
```

```
## [1]   5.75   7.75  10.75 -24.25
```

```
centering(obj3)
```

```
## Warning in centering(obj3): The supplied vector contains missing values
```

```
## [1] NA NA NA NA
```

```
centering(obj4)
```

```
## Error in centering(obj4): Function input should be a numeric vector
```

```
scaling(obj1)
```

```
## Error in scaling(obj1): Function input should be a numeric vector
```

```
scaling(obj2)
```

```
## [1]  0.3068101  0.4295341  0.6136201 -1.5340503
```

```
scaling(obj3)
```

```
## Warning in scaling(obj3): The supplied vector contains missing values
```

```
## [1] NA NA NA NA
```

```
scaling(obj4)
```

```
## Error in scaling(obj4): Function input should be a numeric vector
```

```
normalization(obj1)
```

```
## Error in normalization(obj1): Function input should be a numeric vector
```

```
normalization(obj2)
```

```
## [1]  0.3528316  0.4755556  0.6596416 -1.4880288
```

```
normalization(obj3)
```

```
## Warning in normalization(obj3): The supplied vector contains missing values
```

```
## Warning in centering(data): The supplied vector contains missing values

## Warning in scaling(centering(data)): The supplied vector contains missing values

## [1] NA NA NA NA
```

Note that this function gives the warning message three times, because it uses the previously made functions!

```
normalization(obj4)
```

```
## Error in normalization(obj4): Function input should be a numeric vector
```

The functions behave as expected, however because the functions call each other some warnings are displayed several times.

Exercise 2

Question 1

```r
my_skewness <- function(v){
  # number of observations
  n <- length(v)

  # calculating both parts of the ratio
  numerator <- (1 / n) * sum((v - mean(v))^3)
  denominator <- ((1 / n) * sum((v - mean(v))^2))^(3/2)

  # calculating final product
  skew <- numerator / denominator

  return(skew)
}
```

We could also just use:

```r
my_skewness <- function(data){

   (mean((data - mean(data))^3))/
  (mean(((data- mean(data))^2))^(3/2))
}
```

Question 2

In the code below, the first `if` condition checks whether v is not a numeric vector and not a matrix.

In the second `if` condition, the function calls itself inside `apply`. `apply` applies `my_skewness` column wise so it will return the value by each column vector.

Note that in this case, the order in which the if conditions are specified matters.

```r
my_skewness <- function(data){
  if(!(is.numeric(data) & is.vector(data)) & !is.matrix(data)){
    stop("Wrong type of input \n")
  }
  if(is.matrix(data)){
    warning("Supplied input is a matrix.\nSkewness by column returned. \n")
    return(apply(data, 2, my_skewness))
  }
  if(any(is.na(data))){
    message("The supplied vector contains missing values \n")
  }
    return((mean((data - mean(data))^3))/
    (mean(((data- mean(data))^2))^(3/2)))
}
```

```r
# checking the function with a matrix
my_skewness(matrix(rnorm(100), 10, 10))
```

```
## Warning in my_skewness(matrix(rnorm(100), 10, 10)): Supplied input is a matrix.
## Skewness by column returned.

##  [1] -0.45715443  0.55989376  0.10379070  0.25294160 -0.20866157  0.73400511
```

```
##  [7]  0.39624036  0.01064357 -0.05180157  0.11836546
```

Exercise 3

Question 1

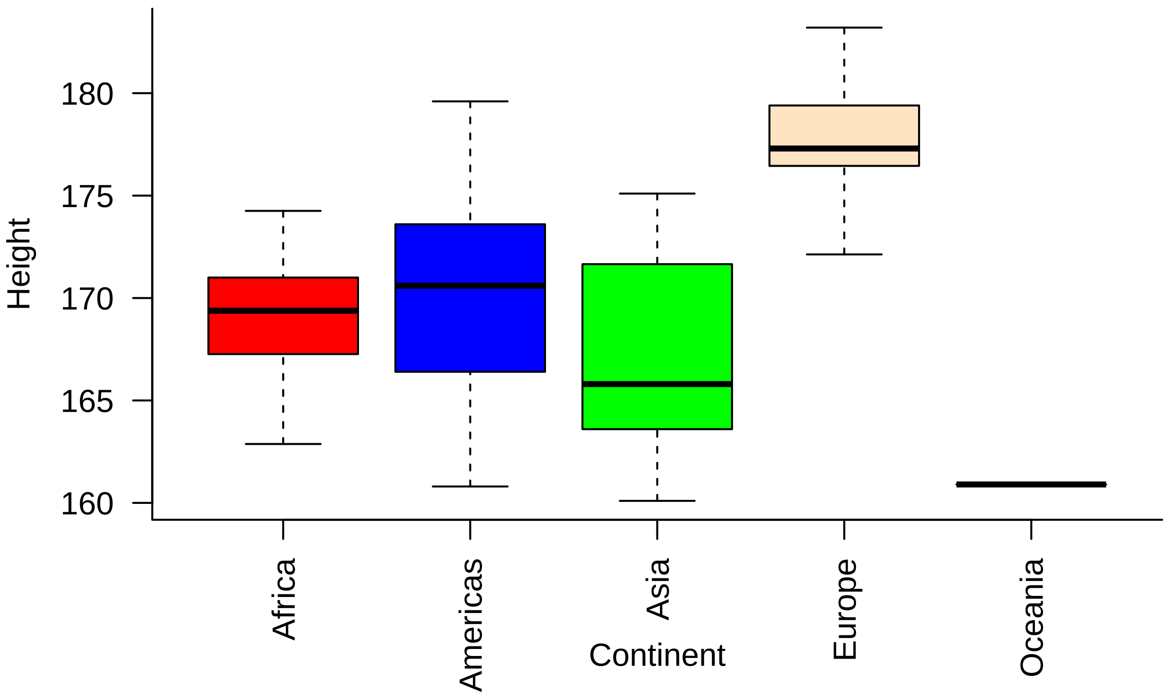
```
library(brolgar)
data(heights)
heights = as.data.frame(heights)
data(wages)
wages = as.data.frame(wages)
```

```
head(heights)
```

```
##       country continent year height_cm
## 1 Afghanistan      Asia 1870    168.40
## 2 Afghanistan      Asia 1880    165.69
## 3 Afghanistan      Asia 1930    166.80
## 4 Afghanistan      Asia 1990    167.10
## 5 Afghanistan      Asia 2000    161.40
## 6     Albania    Europe 1880    170.10
```

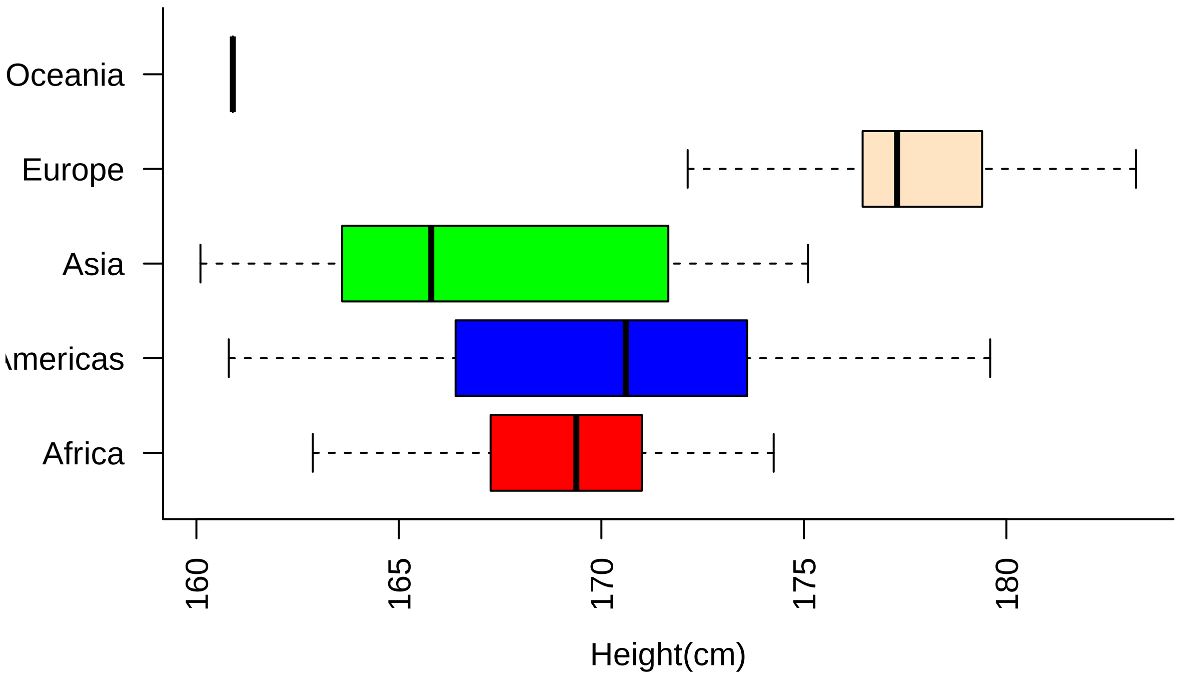
Boxplots:

```
# vertical
par(bty = 'l')
boxplot(height_cm ~ continent,
        data = subset(heights, year == 1980),
        ylab = "Height",
        xlab = "Continent",
        col = c("red", "blue", "green", "bisque1", "darkorange3"),
        las = 2)
```



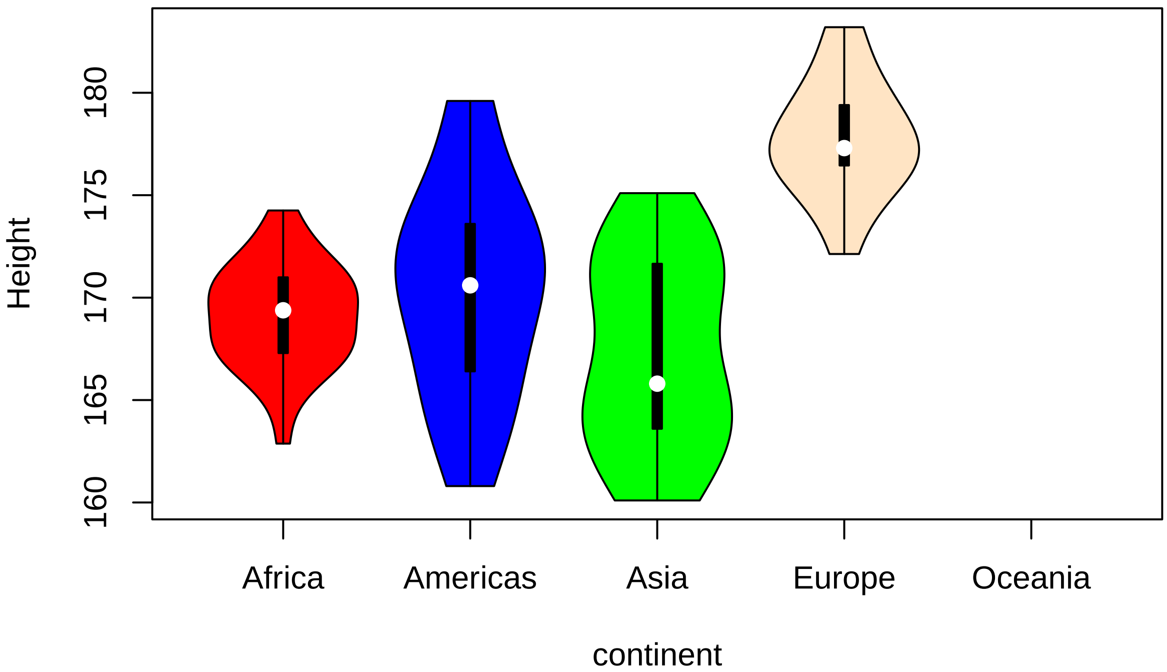

```
# horizontal
par(bty = 'l')
boxplot(height_cm ~ continent,
```

```
        data = subset(heights, year == 1980),
        ylab = "",
        xlab = "Height(cm)",
        col = c("red", "blue", "green", "bisque1", "darkorange3"),
        las = 2,
        horizontal = T)
```



Violin plot:

```
# install.packages("vioplot")
library(vioplot)
vioplot(height_cm ~ continent,
        data = subset(heights, year == 1980),
        ylab = "Height",
        col = c("red", "blue", "green", "bisque1", "darkorange3"))
```

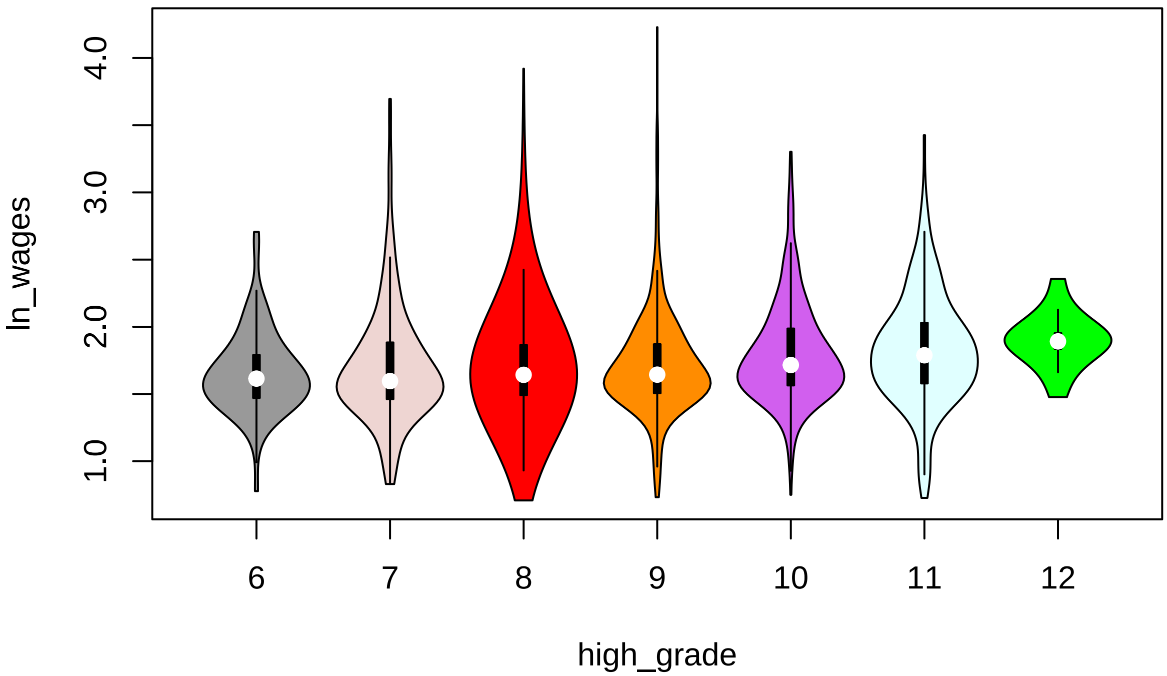


Question 2

```
head(wages)
```

```
##   id ln_wages    xp ged xp_since_ged black hispanic high_grade unemploy_rate
## 1 31    1.491 0.015   1        0.015     0        1          8          3.21
## 2 31    1.433 0.715   1        0.715     0        1          8          3.21
## 3 31    1.469 1.734   1        1.734     0        1          8          3.21
## 4 31    1.749 2.773   1        2.773     0        1          8          3.30
## 5 31    1.931 3.927   1        3.927     0        1          8          2.89
## 6 31    1.709 4.946   1        4.946     0        1          8          2.49
```

```
vioplot(ln_wages ~ high_grade,
        data = subset(wages, xp <= 2),
        col = c("gray60", "mistyrose2", "red", "darkorange", "mediumorchid2",
                "lightcyan1", "green"))
```



ln_wages
1.0
2.0
3.0
4.0
6
7
8
9
10
11
12
high_grade

Exercise 4

```
library(mvtnorm)
set.seed(8)
Sigma1 = matrix(c(1, 0.5, -0.3,
                  0.5, 1, -0.6,
                  -0.3, -0.6, 1), 3, 3)
m1 = rmvnorm(500, mean = rep(0, 3), sigma = Sigma1)
Sigma2 = matrix(c(1, 0.4, 0.4, 1), 2, 2)
m2 = rmvnorm(500, mean = rep(0, 2), sigma = Sigma2)
all.vars = cbind(m1, m2)
colnames(all.vars) = paste('X', 1:5, serp = '')
```

Question 1

The supplied code creates a covariance matrix:

	X1	X2	X3
X1	1.0	0.5	-0.3
X2	0.5	1.0	-0.6
X3	-0.3	-0.6	1.0

Then 500 samples are drawn from a normal distribution with 3 correlated variables using the above covariance matrix.

Then the same for the 2 correlated variables with covariance matrix:

	X1	X2
X1	1.0	0.4
X2	0.4	1.0

The output of the multivariate normal are 3 and 2 values respectively, therefore after using `cbind` we have a matrix (`all.vars`) with 5 columns.

Question 2

We can compute the correlation between the 5 variables using `cor`.

```
cor(all.vars)
```

```
##              X 1         X 2         X 3         X 4         X 5
## X 1   1.00000000  0.49602696 -0.33569496 -0.07637467 -0.08984472
## X 2   0.49602696  1.00000000 -0.66982079 -0.07231346 -0.06348094
## X 3  -0.33569496 -0.66982079  1.00000000  0.06248103  0.11240236
## X 4  -0.07637467 -0.07231346  0.06248103  1.00000000  0.38312350
## X 5  -0.08984472 -0.06348094  0.11240236  0.38312350  1.00000000
```

Question 3

The real covariance matrix for the first 3 columns is:

```
matrix(c(1, 0.5, -0.3,
         0.5, 1, -0.6,
         -0.3, -0.6, 1), 3, 3)
```

```
##      [,1] [,2] [,3]
## [1,]  1.0  0.5 -0.3
## [2,]  0.5  1.0 -0.6
## [3,] -0.3 -0.6  1.0
```

And we can see how, approximately, this is true in:

```
round(cor(all.vars)[1:3, 1:3], 2)
```

```
##       X 1   X 2   X 3
## X 1   1.00  0.50 -0.34
## X 2   0.50  1.00 -0.67
## X 3  -0.34 -0.67  1.00
```

The same for the two last columns:

```
matrix(c(1, 0.4, 0.4, 1), 2, 2)
```

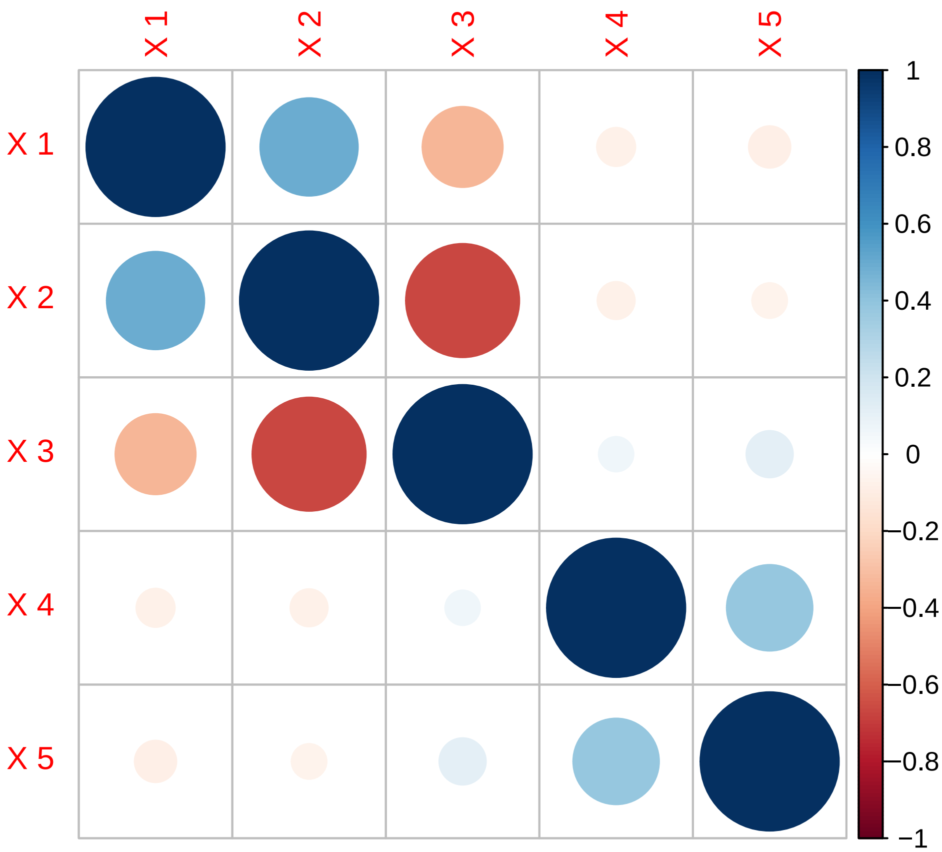
```
##      [,1] [,2]
## [1,]  1.0  0.4
## [2,]  0.4  1.0
```

```
round(cor(all.vars)[4:5, 4:5], 2)
```

```
##      X 4  X 5
## X 4  1.00 0.38
## X 5  0.38 1.00
```

Question 4

```
library(corrplot)
corrplot(cor(all.vars))
```

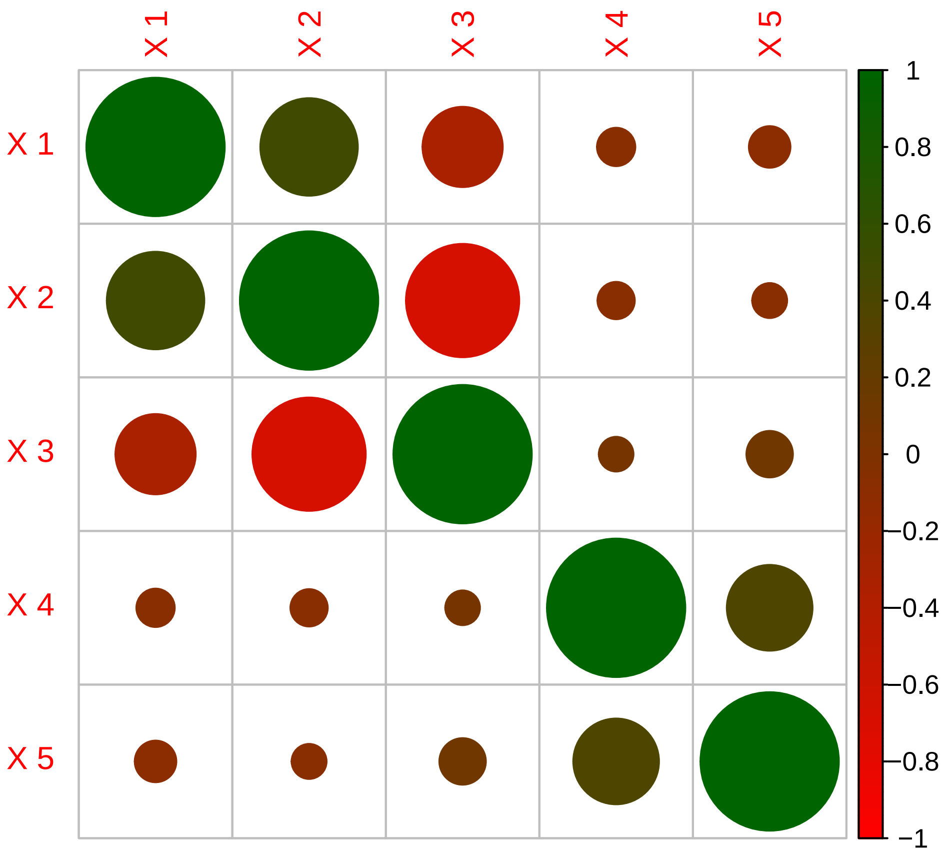


5

To see how the following works please see `?colorRampPalette` and `?corrplot`.

```
# generate 100 colours (in order) between 'red' and 'darkgreen'
my_col = colorRampPalette(c("red", "darkgreen"))( 100 )

# apply these colours to the plot
corrplot(cor(all.vars), col = my_col)
```


X 1
X 2
X 3
X 4
X 5
X 1
X 2
X 3
X 4
X 5
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1