# Exercise:
# Multidimensional Data and Dimensionality Reduction

Introduction to Machine Learning with R

SOLUTION

## Exercise 1: Wisconsin breast cancer data

The Wisconsin data set describes features computed from digitized images of fine needle asprirates (FNA) of breast mass. In addition to the extracted features, the data set includes an ID number of the image and the diagnosis of the sample (1 = benign, 2 = malignant). For more information see https://www.kaggle.com/uciml/breast-cancer-wisconsin-data.

a) Load the "breastCancer_Wisconsin.rda" file using the `load('breastCancer_Wisconsin.rda')` command. The file contains the `wisco` data frame which stores the Wisconsin breast cancer data. First, take a look at the data. To do this, apply the `head` function to the data frame, which returns the first six rows of each column.

```
load('breastCancer_Wisconsin.rda')
head(wisco)
```

```
##         id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1   842302         2       17.99        10.38         122.80    1001.0
## 2   842517         2       20.57        17.77         132.90    1326.0
## 3 84300903         2       19.69        21.25         130.00    1203.0
## 4 84348301         2       11.42        20.38          77.58     386.1
## 5 84358402         2       20.29        14.34         135.10    1297.0
## 6   843786         2       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
```

```
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst perimeter_worst
## 1     0.03003             0.006193        25.38         17.33          184.60
## 2     0.01389             0.003532        24.99         23.41          158.80
## 3     0.02250             0.004571        23.57         25.53          152.50
## 4     0.05963             0.009208        14.91         26.50           98.87
## 5     0.01756             0.005115        22.54         16.67          152.20
## 6     0.02165             0.005082        15.47         23.75          103.40
##   area_worst smoothness_worst compactness_worst concavity_worst
## 1     2019.0           0.1622            0.6656          0.7119
## 2     1956.0           0.1238            0.1866          0.2416
## 3     1709.0           0.1444            0.4245          0.4504
## 4      567.7           0.2098            0.8663          0.6869
## 5     1575.0           0.1374            0.2050          0.4000
## 6      741.6           0.1791            0.5249          0.5355
##   concave.points_worst symmetry_worst fractal_dimension_worst
## 1               0.2654         0.4601                 0.11890
## 2               0.1860         0.2750                 0.08902
## 3               0.2430         0.3613                 0.08758
## 4               0.2575         0.6638                 0.17300
## 5               0.1625         0.2364                 0.07678
## 6               0.1741         0.3985                 0.12440
```

b) What are the dimensions of the data set, i.e. how many rows and columns are there? (**Hint**: `dim()`)

```
dim(wisco)
```

```
## [1] 569  32
```

c) Next, apply the `str` function to the data frame. Try to understand what the output of the function means. Why could it be useful?

```
str(wisco)
```

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id                      : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844
##  $ diagnosis               : num  2 2 2 2 2 2 2 2 2 2 ...
##  $ radius_mean             : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean            : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean          : num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean               : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean         : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean        : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean          : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points_mean     : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean           : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se               : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se              : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se            : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                 : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se           : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
```

```
##  $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst             : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave.points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

The str function returns information about the structure of an R object. In the case of a data frame, it tells the dimensions of the data, and lists the names of all columns. Additionally, it shows the format of each column, i.e. whether a column is for example numeric or a factor. The function is a useful tool to see whether certain variables are coded in a wrong format (e.g. a categorical variable stored as numbers).

d) The main variable of interest, the `diagnosis` variable, is a categorical variable which tells us whether an image shows a benign or malignant tumor. However, as we can see in the output from the previous exercise, it is coded as a `numerical` variable (with the value 1 indicating a benign and the value 2 indicating a malignant tumor). This is usually not advisable, because it makes working with the categorical variable more difficult. For example, we always need a legend to understand the meaning of the numbers (1 = benign, 2 = malignant). In the worst case, using numbers for categorical variables can lead to a wrong analysis because a function might mistakenly treat a categorical variable as if it were continuous. The intended format in R for categorical variables is `factor`. Turn `diagnosis` into a `factor`, with the value 1 taking on the label `benign` and the value 2 taking on the label `malignant` (**Hint**: `wisco$diagnosis <- factor(...)`). Run again the `str` command to check if the conversion worked.

```
wisco$diagnosis <- factor(wisco$diagnosis, levels = 1:2, labels = c('benign', 'malignant'))
str(wisco[, 1:6])   # Only show first six variables to save space
```

```
## 'data.frame':    569 obs. of  6 variables:
##  $ id             : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844981 8450
##  $ diagnosis      : Factor w/ 2 levels "benign","malignant": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius_mean    : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean   : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean: num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean      : num  1001 1326 1203 386 1297 ...
```

e) The picuture-ID (`id`) is coded as an `integer` (whole numbers). Although this will not affect the work in this exercise, it is again not advisable to code such a categorical ID-variable as numbers. Turn the `id` variable into a `factor`. Since we don't need to give new labels, you can leave the `labels` and `levels` arguments away when applying the `factor` function. (**Hint**: `wisco$id <- factor(...)`)

```
wisco$id <- factor(wisco$id)
str(wisco[, 1:6])   # Only show first six variables to save space
```

```
## 'data.frame':    569 obs. of  6 variables:
##  $ id             : Factor w/ 569 levels "8670","8913",..: 42 43 489 490 491 44 45 492 46 493 ...
```

```
##  $ diagnosis     : Factor w/ 2 levels "benign","malignant": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius_mean   : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean  : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean: num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean     : num  1001 1326 1203 386 1297 ...
```

f) Perform a Principal Component Analysis (with scaling) on the data. Do not include the `id` factor in the PCA. Also exclude the `diagnosis` factor from the PCA. Why can't we include these factors in the PCA? Look at the created object. (**Hint**: `prcomp(..., scale.=TRUE)`)

```
pca_obj <- prcomp(wisco[,-c(1,2)], scale. = TRUE)
#pca_obj   # Not shown due to size
```

PCA only works with numerical values, therefore factors cannot be included. Also, the ID of the pictures is in itself not a meaningful variable (regarding the picture features) and, therefore, it doesn't make sense to include it.

g) Check the results of the PCA. What proportion of the variance is explained by the first PC? What proportion of the variance is explained by the first four PCs together? (**Hint**: `summary()`)

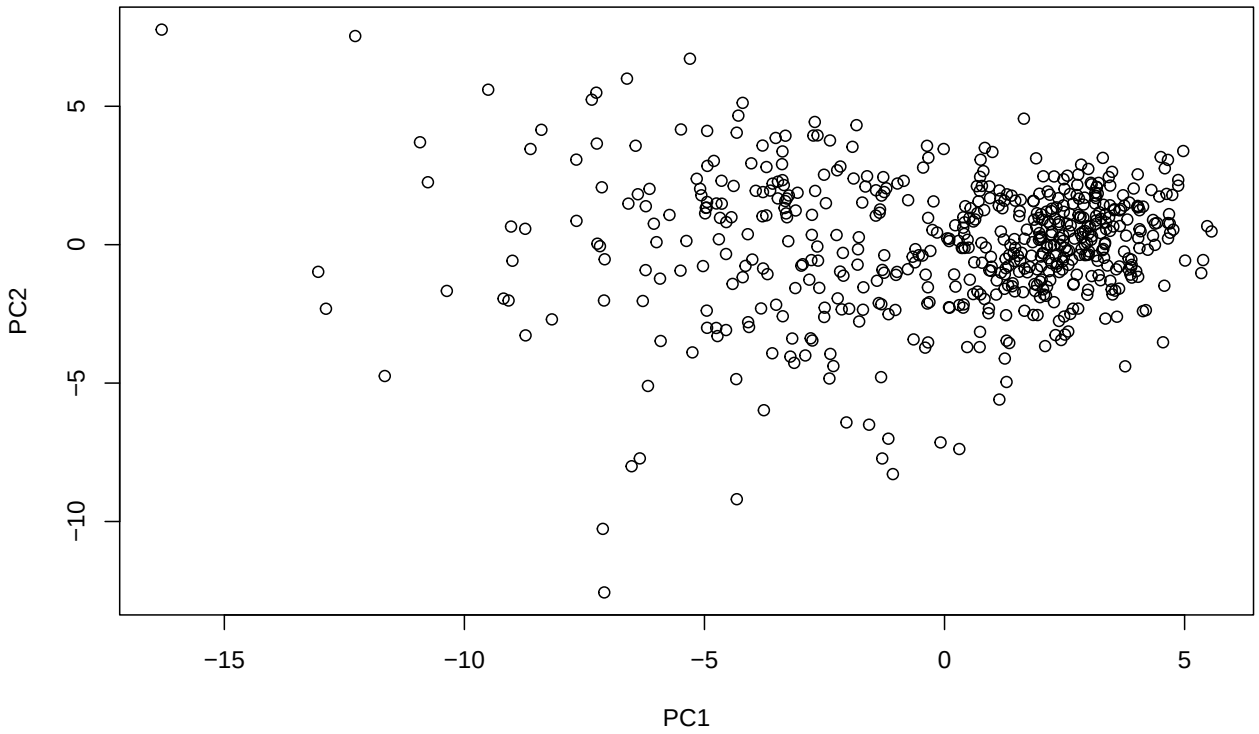
```
summary(pca_obj)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     3.6444 2.3857 1.67867 1.40735 1.28403 1.09880 0.82172
## Proportion of Variance 0.4427 0.1897 0.09393 0.06602 0.05496 0.04025 0.02251
## Cumulative Proportion  0.4427 0.6324 0.72636 0.79239 0.84734 0.88759 0.91010
##                           PC8    PC9    PC10   PC11    PC12    PC13    PC14
## Standard deviation     0.69037 0.6457 0.59219 0.5421 0.51104 0.49128 0.39624
## Proportion of Variance 0.01589 0.0139 0.01169 0.0098 0.00871 0.00805 0.00523
## Cumulative Proportion  0.92598 0.9399 0.95157 0.9614 0.97007 0.97812 0.98335
##                           PC15    PC16    PC17    PC18    PC19    PC20   PC21
## Standard deviation     0.30681 0.28260 0.24372 0.22939 0.22244 0.17652 0.1731
## Proportion of Variance 0.00314 0.00266 0.00198 0.00175 0.00165 0.00104 0.0010
## Cumulative Proportion  0.98649 0.98915 0.99113 0.99288 0.99453 0.99557 0.9966
##                           PC22    PC23   PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.16565 0.15602 0.1344 0.12442 0.09043 0.08307 0.03987
## Proportion of Variance 0.00091 0.00081 0.0006 0.00052 0.00027 0.00023 0.00005
## Cumulative Proportion  0.99749 0.99830 0.9989 0.99942 0.99969 0.99992 0.99997
##                           PC29    PC30
## Standard deviation     0.02736 0.01153
## Proportion of Variance 0.00002 0.00000
## Cumulative Proportion  1.00000 1.00000
```

The first PC explaines a proportion of 0.443 of the variance. The first four PCs cummulatively explain a proportion of 0.792 of the total variance.

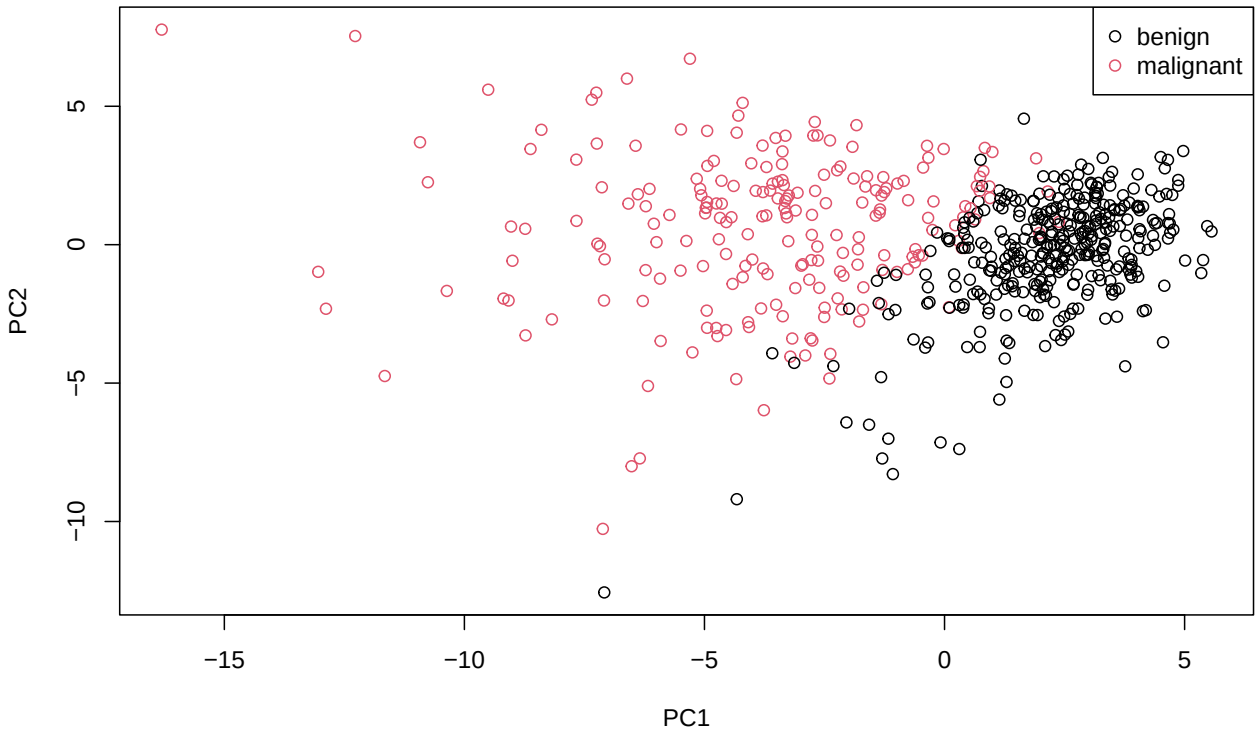
h) Perform a dimensionality reduction by looking only at the first two PCs. Plot the values of the two first PCs in a scatterplot.

```
plot(PC2~PC1, data = pca_obj$x)
```




i) Extend the plot by colouring the points according to the `diagnosis` variable. To do this, give the `diagnosis` vector as input to the `col` argument of the `plot` function (**Hint**: `plot(..., col=...)`). What can you see? We can also add a legend to show us the meaning of the colours. Check the solution and try to understand how the `legend` function works.

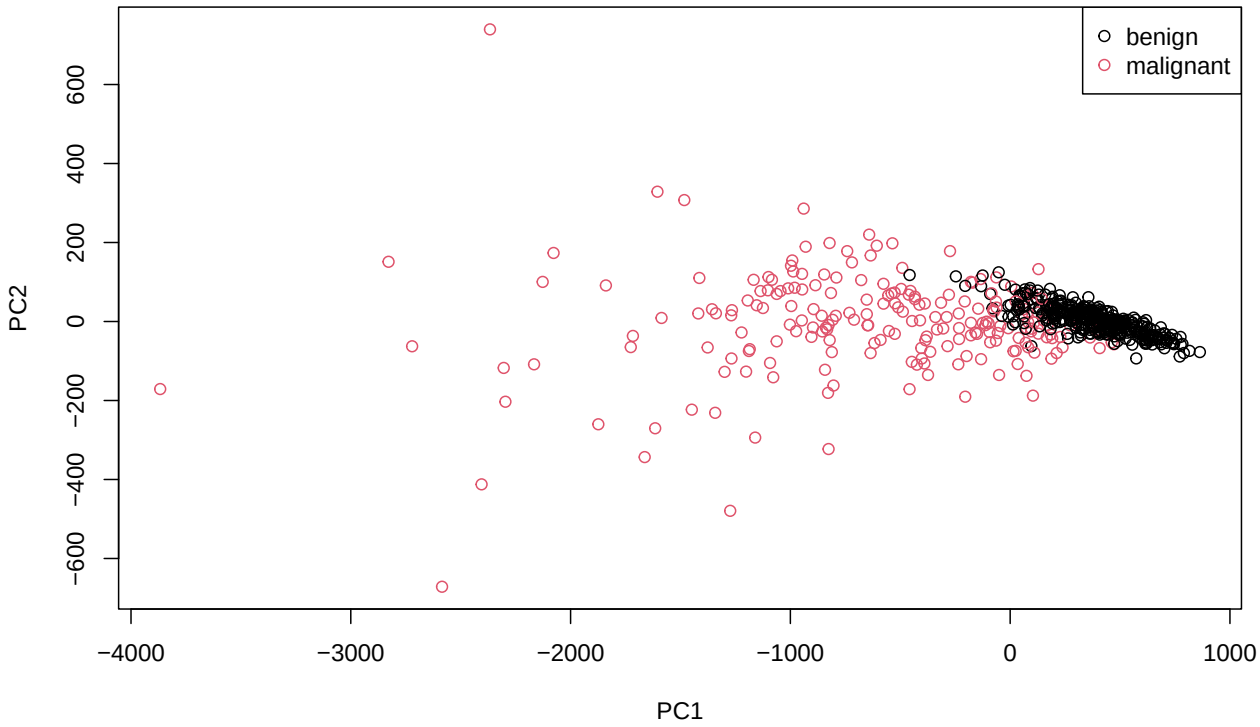
```
plot(PC2~PC1, data = pca_obj$x, col = wisco$diagnosis)
legend('topright',     # Position of legend
       legend = levels(wisco$diagnosis),    # Text in each row
       pch = c(1,1),    # Symbol in each row
       col = 1:nlevels(wisco$diagnosis))     # Colour in each row
```




It seems that in order to differentiate malignant from benign pictures, the original 30 dimensions are not necessarily needed. The two classes are quite well separated even in a two-dimensional representation based on the PCA.

j) We now want to see how the 2D representation changes when no scaling is used. Perform the PCA again but without scaling and create the 2D plot of the first two PCs. What can you observe?

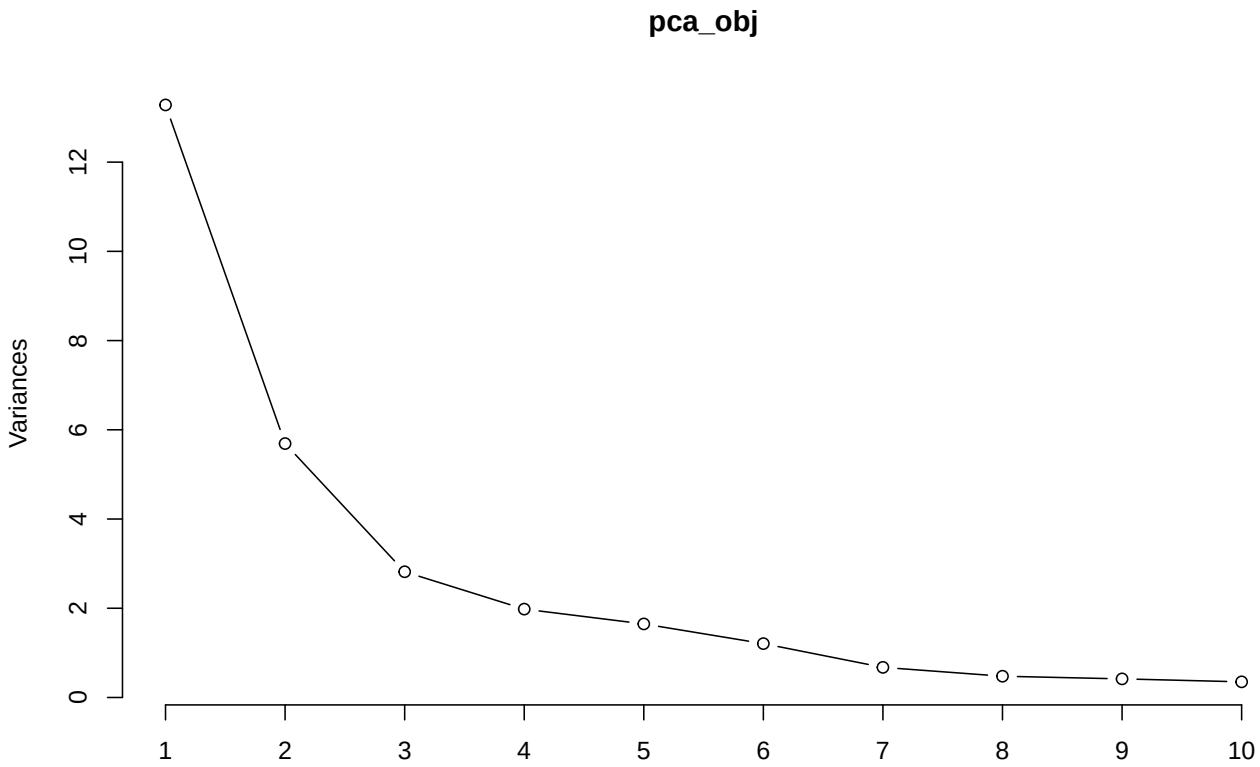
```
pca_noscal <- prcomp(wisco[, -c(1,2)], scale. = FALSE)
plot(PC2~PC1, data = pca_noscal$x, col = wisco$diagnosis)
legend('topright',    # Position of legend
       legend = levels(wisco$diagnosis),   # Text in each row
       pch = c(1,1),   # Symbol in each row
       col = 1:nlevels(wisco$diagnosis))    # Colour in each row
```




The plot does look a bit different from when performing the PCA with scaling. The malignant data points are much more spread than before. Also, it seems that the benign and malignant pictures are overlapping more with each other than before.

k) Using the PCA generated with scaling, we wish to find out how many PCs are needed to explain the data well. Create a scree-plot and interpret it.

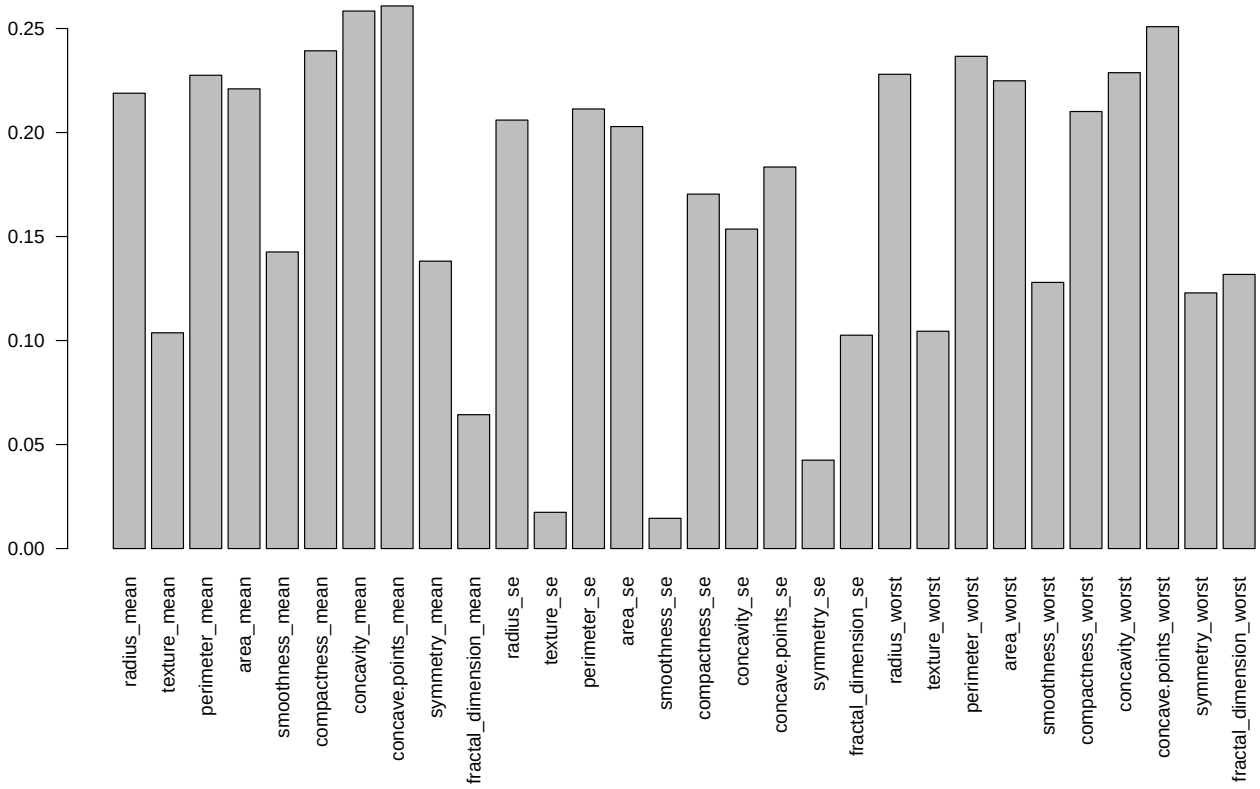
```
screeplot(pca_obj, type = 'l')
```




A possible bend in the curve is occurring at around the 3rd or 4th PC. Thus, one could choose to include the first two or three PCs.

l) We might be interested which variable contributes most strongly to the first PC. Plot the variables' loadings to the first PC (take absolute values of loadings) in a barplot. (**Hint**: `barplot()`, `abs()`, `pca_object$rotation`)

```
par(mar = c(12, 4, 4, 2))   # Increase plot margins to see labels
pc1_load <- pca_obj$rotation[, 'PC1']
barplot(abs(pc1_load), las=2)   # las=2 rotates xlabels
```




m) Which of the variables has the largest (absolute) loading on PC1? How large is this loading? (**Hint**: `which.max()`)

```
ind_m <- which.max(abs(pc1_load))    # Find the index of the maximum loading
ind_m
```

```
## concave.points_mean
##                   8
```

```
pca_obj$rotation[ind_m, 'PC1']    # Show the loading
```

```
## [1] -0.2608538
```

The variable with the highest loading is concave.points_mean, it has a loading of -0.261.

n) To make the situation a bit easier to oversee, we decrease the size of the data set by only working with the first eight columns. Create a new data set consisting only of those columns.

```
wisco_red <- wisco[, 1:8]
head(wisco_red)
```

```
##         id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1   842302 malignant       17.99        10.38         122.80    1001.0
## 2   842517 malignant       20.57        17.77         132.90    1326.0
## 3 84300903 malignant       19.69        21.25         130.00    1203.0
## 4 84348301 malignant       11.42        20.38          77.58     386.1
## 5 84358402 malignant       20.29        14.34         135.10    1297.0
## 6   843786 malignant       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean
## 1         0.11840          0.27760
```

```
## 2          0.08474          0.07864
## 3          0.10960          0.15990
## 4          0.14250          0.28390
## 5          0.10030          0.13280
## 6          0.12780          0.17000
```

o) Try to get a visual impression of the data by plotting it in a pairs plot. You can use the `ggpairs` function (you need to load the `ggplot2` and `GGally` package for the function to work). The `id` variable has to be removed before applying the function. Interpret the created figure. (**Hint**: `library(ggplot2)`, `library(GGally)`, `ggpairs(...)`)

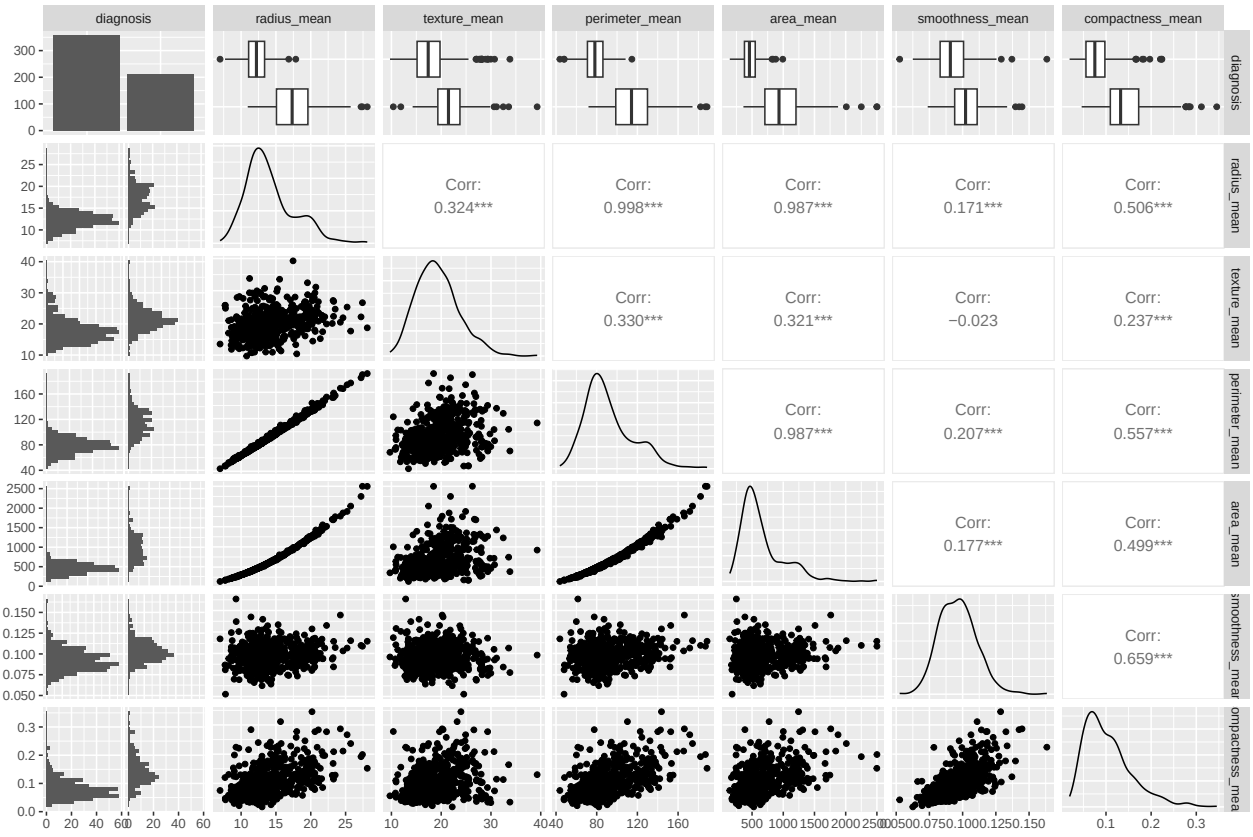
```
library(ggplot2)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
ggpairs(wisco_red[,-1])
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

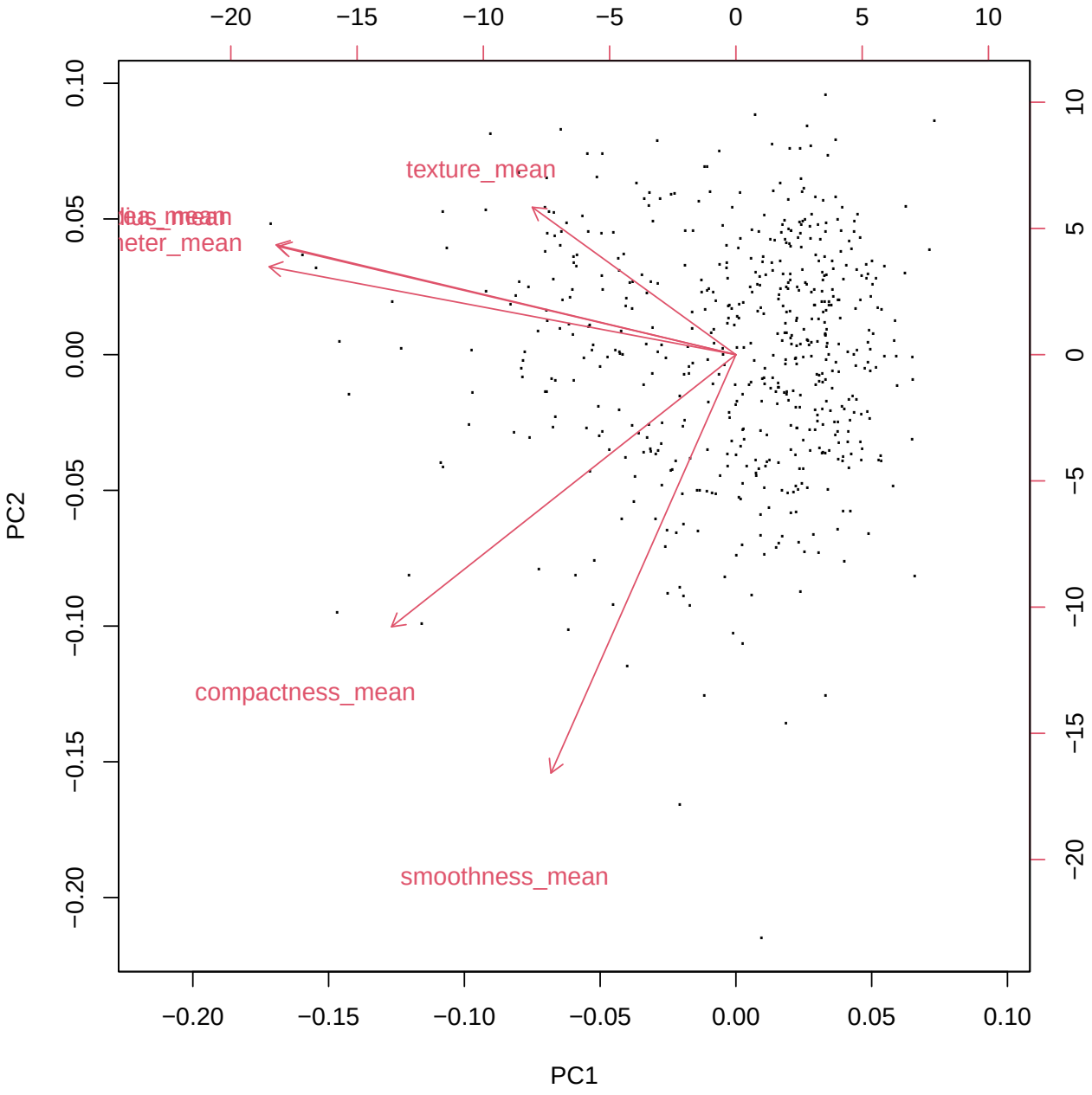



The pairs plot shows the pairwise relations of all variables. We can see that some of the variables are very

strongly correlated.

p) Perform a PCA with the reduced data (again excluding `id` and `diagnosis`, use scaling). Create a biplot of this PCA, showing the projections of the original variables. Which of the variables has the strongest contribution to PC2?

```
pca_red <- prcomp(wisco_red[, -c(1,2)], scale. = TRUE)
biplot(pca_red,
       xlabs = rep('.', nrow(wisco)))   # xlabs defines the symbols used inside the plot
```




Although some labels are a bit difficult to read, we can see that smoothness_mean is most strongly contributing to the second PC.