Background
1. Exploratory data analysis
2. Principal Component Analysis
3. Hierarchical clustering
4. K-means clustering
5. Combining methods
6. Sensitivity/Specificity
7. Prediction
8. OPTIONAL
About this document

BIMM-143, Lecture 9

Code

Unsupervised Learning Analysis of Human Breast Cancer Cells

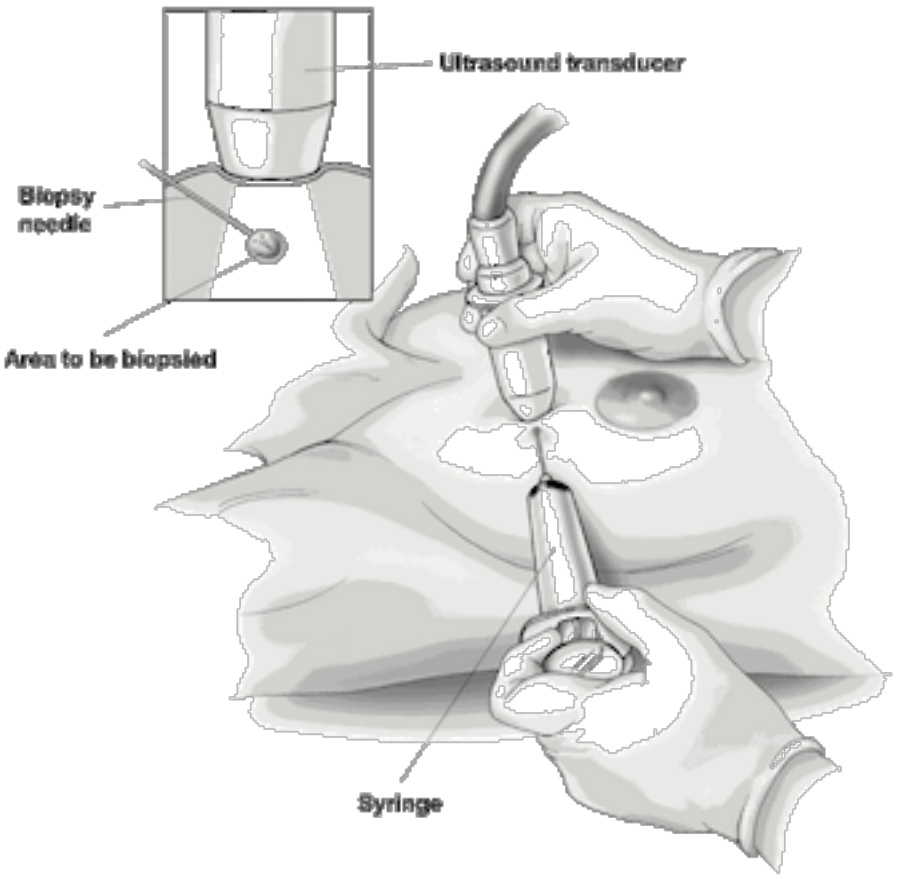
Barry Grant < http://thegrantlab.org/bimm143/ (http://thegrantlab.org/bimm143/) >
2018-12-07 (12:33:41 PST on Fri, Dec 07)

Background

The goal of this hands-on session is for you to explore a complete analysis using the unsupervised learning techniques covered in the last class. You'll extend what you've learned by combining PCA as a preprocessing step to clustering using data that consist of measurements of cell nuclei of human breast masses. This expands on our RNA-Seq analysis from last day.

The data itself comes from the *Wisconsin Breast Cancer Diagnostic Data Set* first reported by *K. P. Benne and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets"* (http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%2528Diagnostic%2529).

Values in this data set describe characteristics of the cell nuclei present in digitized images of a fine needle aspiration (FNA) of a breast mass.



FNA is a type of biopsy procedure where a very thin needle is inserted into an area of abnormal tissue or cells with a guide of CT scan or ultrasound monitors (**Figure 1**). The collected sample is then transferred to a pathologist to study it under a microscope and examine whether cells in the biopsy are normal or not.

For example `radius` (i.e. mean of distances from center to points on the perimeter), `texture` (i.e. standard deviation of gray-scale values), and `smoothness` (local variation in radius lengths). Summary information is also provided for each group of cells including `diagnosis` (i.e. benign (not cancerous) and and malignant (cancerous)).

1. Exploratory data analysis

Preparing the data

Before we can begin our analysis we first have to download and import our data correctly into our R session.

For this we can use the `read.csv()` function to read the CSV (comma-separated values) file containing the data (available from our class website: WisconsinCancer.csv (https://bioboot.github.io/bimm143_S18/class-material/WisconsinCancer.csv))

Assign the result to an object called `wisc.df` .

Hide

Hide

```
# Save your input data file to a new 'data' directory
fna.data <- "data/WisconsinCancer.csv"

# Complete the following code to input the data and store as wisc.df
wisc.df <- ___
```

Examine your input data to ensure column names are set correctly. The `id` and `diagnosis` columns will not be used for most of the following steps (you can use the **View()** or **head()** functions here).

Hide

Hide

```
wisc.df
```

id <int>	diagnosis <fctr>	radius_m... <dbl>	texture_me... <dbl>	perimeter_m... <dbl>	area_m... <dbl>	smoo
842302	M	17.990	10.38	122.80	1001.0	
842517	M	20.570	17.77	132.90	1326.0	
84300903	M	19.690	21.25	130.00	1203.0	
84348301	M	11.420	20.38	77.58	386.1	
84358402	M	20.290	14.34	135.10	1297.0	
843786	M	12.450	15.70	82.57	477.1	
844359	M	18.250	19.98	119.60	1040.0	
84458202	M	13.710	20.83	90.20	577.9	
844981	M	13.000	21.82	87.50	519.8	
84501001	M	12.460	24.04	83.97	475.9	

1-10 of 569 rows | 1-7 of 33 columns Previous **1** 2 3 4 5 6 ... 57 Next

Next use `as.matrix()` to convert the other features (i.e. columns) of the data (in columns 3 through 32) to a matrix. Store this in a variable called `wisc.data`.

Hide

Hide

```
# Convert the features of the data: wisc.data
wisc.data <- as.matrix( ___ )
```

Assign the row names of `wisc.data` the values currently contained in the `id` column of `wisc.df`. While not strictly required, this will help you keep track of the different observations throughout the modeling process.

Hide

Hide

```
# Set the row names of wisc.data
row.names(wisc.data) <- wisc.df$id
#head(wisc.data)
```

Finally, setup a separate new vector called `diagnosis` to be `1` if a diagnosis is malignant (`"M"`) and `0` otherwise. Note that R coerces `TRUE` to `1` and `FALSE` to `0`.

Hide

Hide

```
# Create diagnosis vector by completing the missing code
diagnosis <- as.numeric( ___ )
```

Exploratory data analysis

The first step of any data analysis, unsupervised or supervised, is to familiarize yourself with the data.

Explore the data you created before (`wisc.data` and `diagnosis`) to answer the following questions:

- **Q1**. How many observations are in this dataset?
- **Q2**. How many variables/features in the data are suffixed with `_mean` ?
- **Q3**. How many of the observations have a malignant diagnosis?

> The functions `dim()`, `length()`, `grep()` and `sum()` may be useful for answering the first 3 questions above.

2. Principal Component Analysis

Performing PCA

The next step in your analysis is to perform principal component analysis (PCA) on `wisc.data`.

It is important to check if the data need to be scaled before performing PCA. Recall two common reasons for scaling data include:

- The input variables use different units of measurement.
- The input variables have significantly different variances.

Check the mean and standard deviation of the features (i.e. columns) of the `wisc.data` to determine if the data should be scaled. Use the `colMeans()` and `apply()` functions like you've done before.

Hide

Hide

```
# Check column means and standard deviations
colMeans(wisc.data)

apply(wisc.data,2,sd)
```

Execute PCA with the `prcomp()` function on the `wisc.data`, scaling if appropriate, and assign the output model to `wisc.pr`.

Hide

Hide

```
# Perform PCA on wisc.data by completing the following code
wisc.pr <- prcomp( ___ )
```

Inspect a summary of the results with the `summary()` function.

Hide

Hide

```
# Look at summary of results
summary(wisc.pr)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     3.6444 2.3857 1.67867 1.40735 1.28403 1.09880
## Proportion of Variance 0.4427 0.1897 0.09393 0.06602 0.05496 0.04025
## Cumulative Proportion  0.4427 0.6324 0.72636 0.79239 0.84734 0.88759
##                            PC7     PC8    PC9    PC10   PC11    PC12
## Standard deviation     0.82172 0.69037 0.6457 0.59219 0.5421 0.51104
## Proportion of Variance 0.02251 0.01589 0.0139 0.01169 0.0098 0.00871
## Cumulative Proportion  0.91010 0.92598 0.9399 0.95157 0.9614 0.97007
##                           PC13    PC14    PC15    PC16    PC17    PC
18
## Standard deviation     0.49128 0.39624 0.30681 0.28260 0.24372 0.229
39
## Proportion of Variance 0.00805 0.00523 0.00314 0.00266 0.00198 0.001
75
## Cumulative Proportion  0.97812 0.98335 0.98649 0.98915 0.99113 0.992
88
##                           PC19    PC20   PC21    PC22    PC23   PC24
## Standard deviation     0.22244 0.17652 0.1731 0.16565 0.15602 0.1344
## Proportion of Variance 0.00165 0.00104 0.0010 0.00091 0.00081 0.0006
## Cumulative Proportion  0.99453 0.99557 0.9966 0.99749 0.99830 0.9989
##                           PC25    PC26    PC27    PC28    PC29    PC
30
## Standard deviation     0.12442 0.09043 0.08307 0.03987 0.02736 0.011
53
## Proportion of Variance 0.00052 0.00027 0.00023 0.00005 0.00002 0.000
00
## Cumulative Proportion  0.99942 0.99969 0.99992 0.99997 1.00000 1.000
00
```

- **Q4**. From your results, what proportion of the original variance is captured by the first principal components (PC1)?
- **Q5**. How many principal components (PCs) are required to describe at least 70% of the original variance in the data?
- **Q6**. How many principal components (PCs) are required to describe at least 90% of the original variance in the data?

Interpreting PCA results

Now you will use some visualizations to better understand your PCA model. A common visualization for PCA results is the so-called biplot.

However, you will often run into some common challenges with using biplots on real-world data containing a non-trivial number of observations and variables. Here we will need to look at some alternative visualizations. You are encouraged to experiment with additional visualizations before moving on to the next section

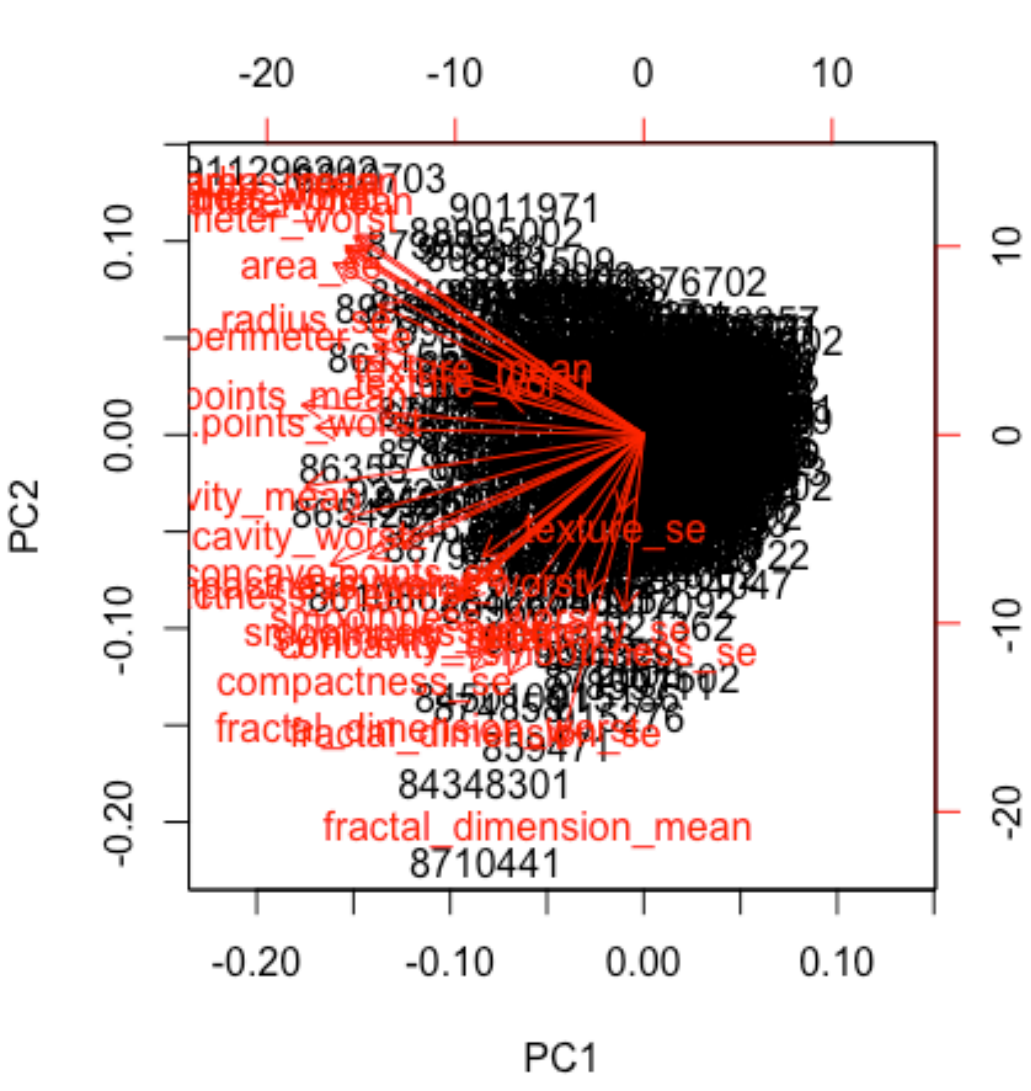
Create a biplot of the `wisc.pr` using the `biplot()` function.

- **Q7.** What stands out to you about this plot? Is it easy or difficult to understand? Why?

Hide

Hide

```
biplot(___)
```

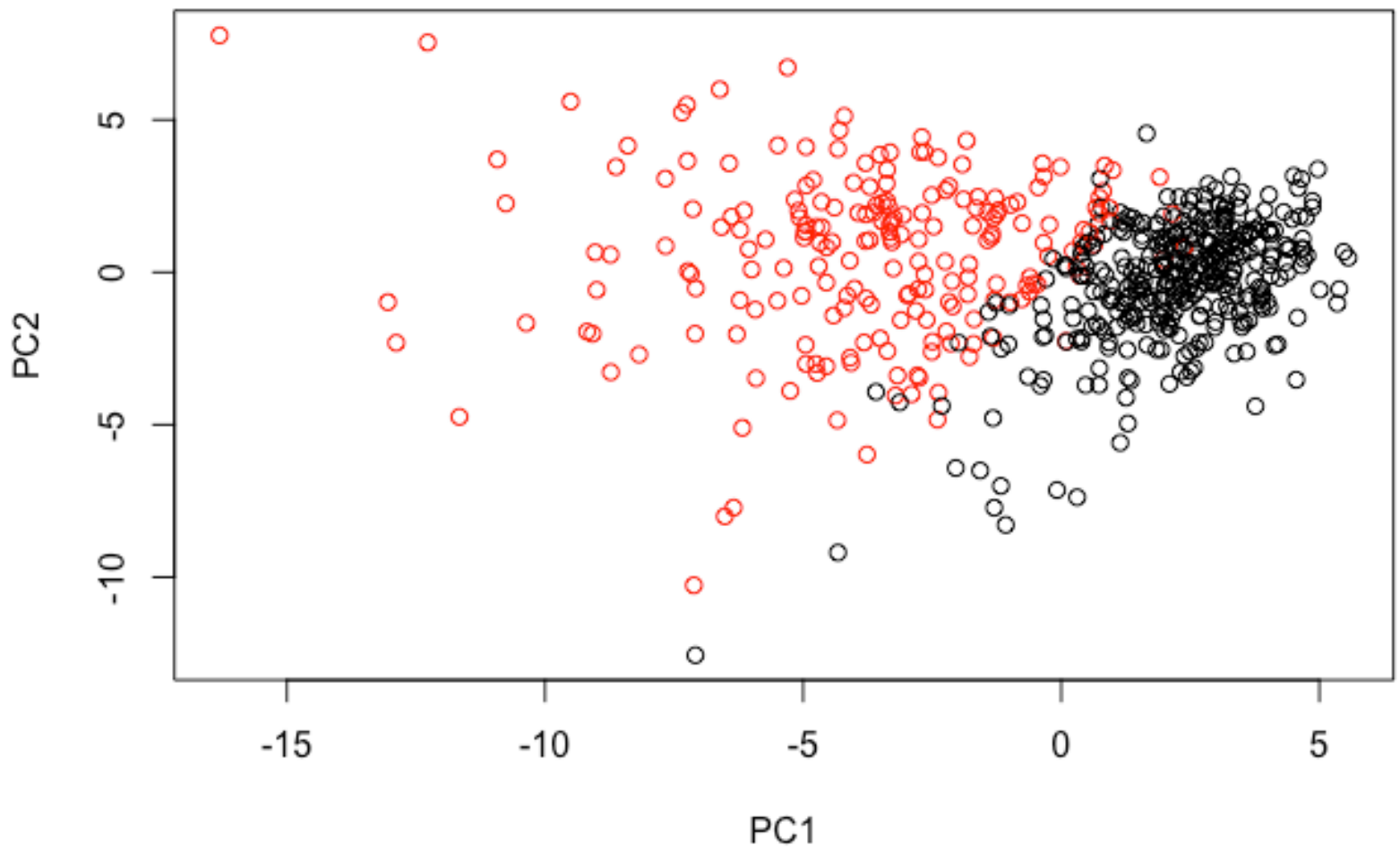


Rownames are used as the plotting character for biplots like this one which can make trends rather hard to see. In fact, this plot is very poor. So lets generate a more standard scatter plot of each observation along principal components 1 and 2 (i.e. a plot of PC1 vs PC2 available as the first two columns of `wisc.pr$x`) and color the points by the diagnosis (available in the `diagnosis` vector you created earlier).

Hide

Hide

```
# Scatter plot observations by components 1 and 2
plot( ___ , col = ___ ,
     xlab = "PC1", ylab = "PC2")
```

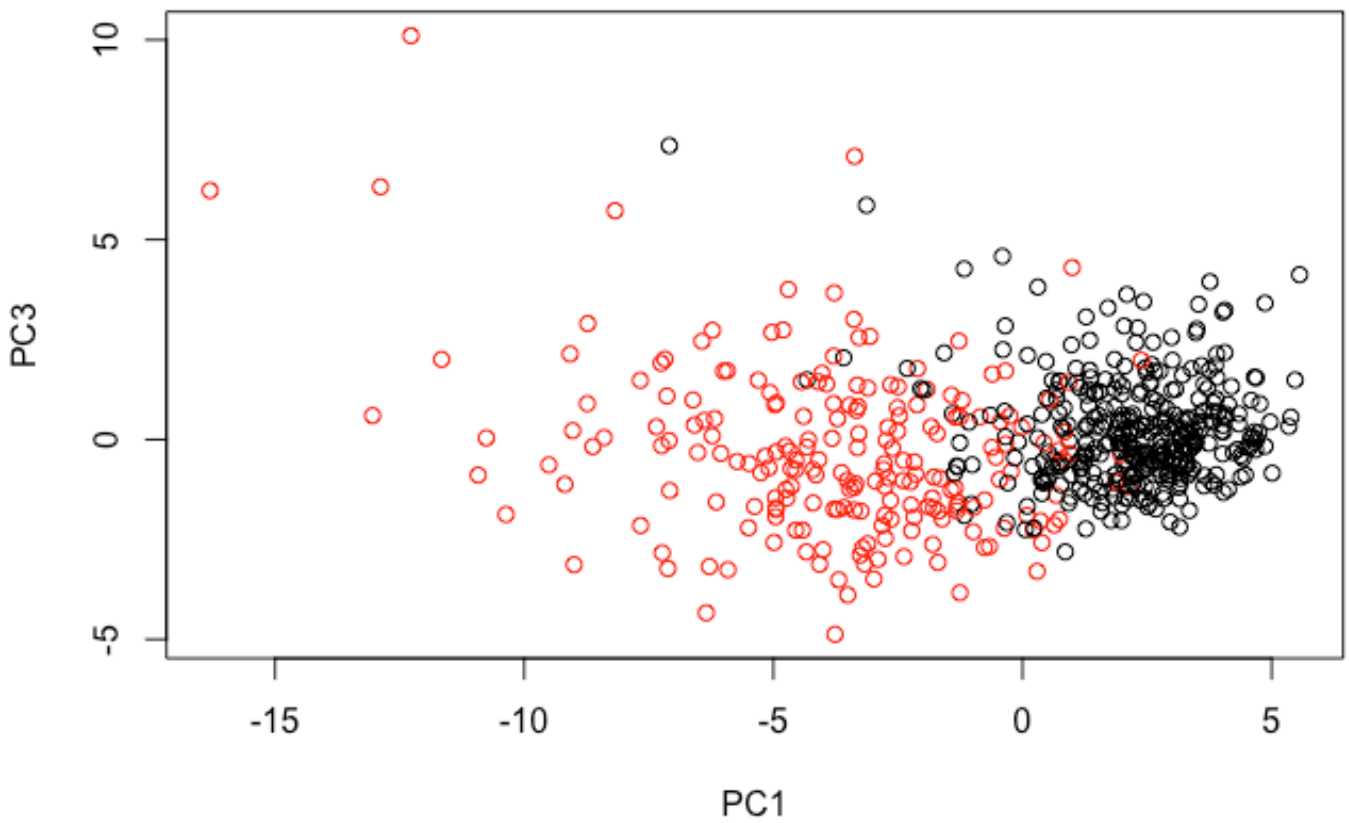


- **Q8.** Generate a similar plot for principal components 1 and 3. What do you notice about these plots?

Hide

Hide

```
# Repeat for components 1 and 3
plot(wisc.pr$x[, ___ ], col = (diagnosis + 1),
     xlab = "PC1", ylab = "PC3")
```



Because principal component 2 explains more variance in the original data than principal component 3, you can see that the first plot has a cleaner cut separating the two subgroups.

> Overall, the plots indicate that principal component 1 is capturing a separation of malignant from benign samples. This is an important and interesting result worthy of further exploration - as we will do in the next sections!

Variance explained

In this exercise, you will produce scree plots showing the proportion of variance explained as the number of principal components increases. The data from PCA must be prepared for these plots, as there is not a built-in function in base R to create them directly from the PCA model.

As you look at these plots, ask yourself if there's an 'elbow' in the amount of variance explained that might lead you to pick a natural number of principal components. If an obvious elbow does not exist, as is typical in some real-world datasets, consider how else you might determine the number of principal components to retain based on the scree plot.

Calculate the variance of each principal component by squaring the sdev component of `wisc.pr` (i.e. `wisc.pr$sdev^2`). Save the result as an object called `pr.var`.

Hide

Hide

```
# Calculate variance of each component
pr.var <- ___
head(pr.var)
```

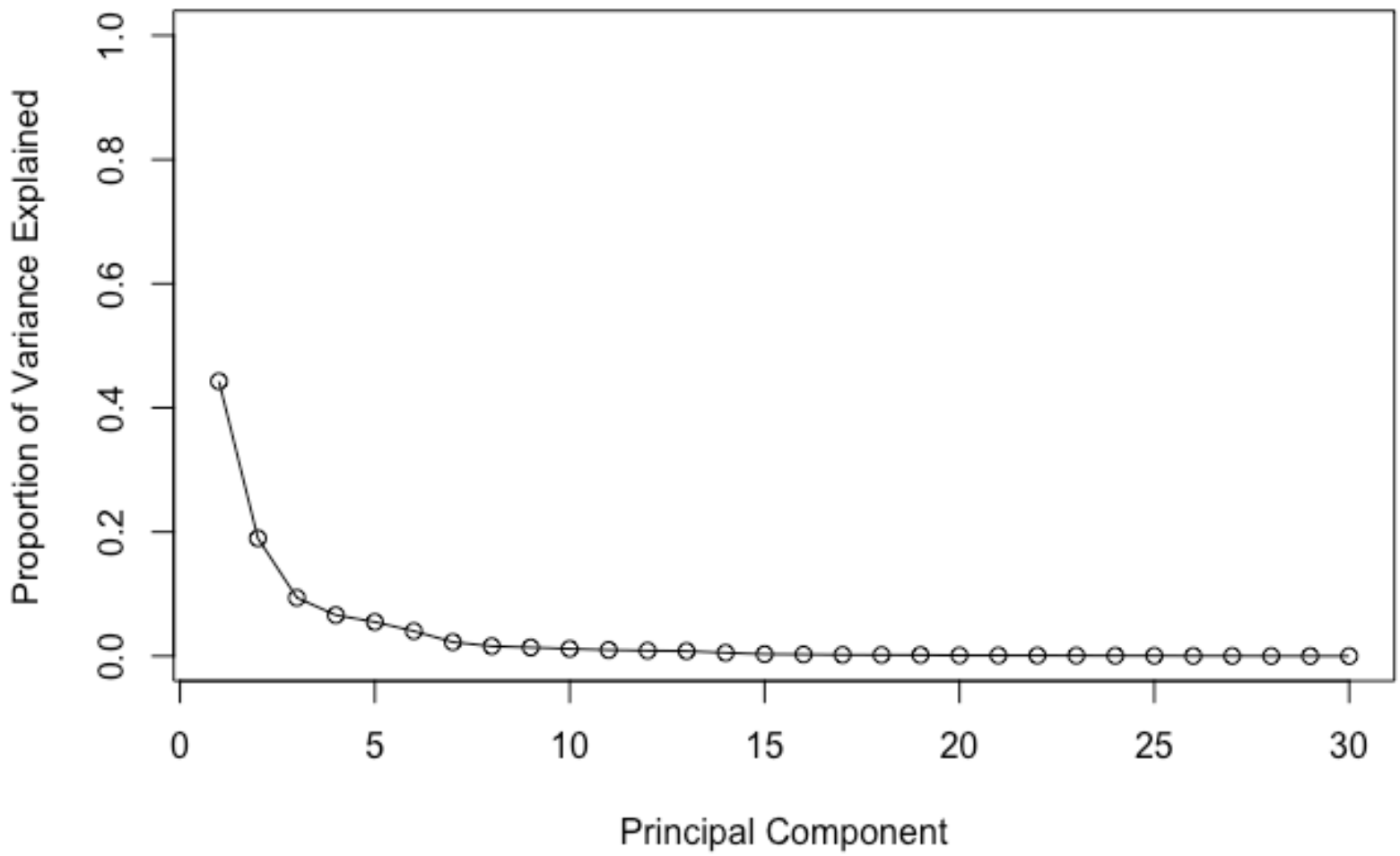
```
## [1] 13.281608  5.691355  2.817949  1.980640  1.648731  1.207357
```

Calculate the variance explained by each principal component by dividing by the total variance explained of all principal components. Assign this to a variable called `pve` and create a plot of variance explained for each principal component.

Hide

Hide

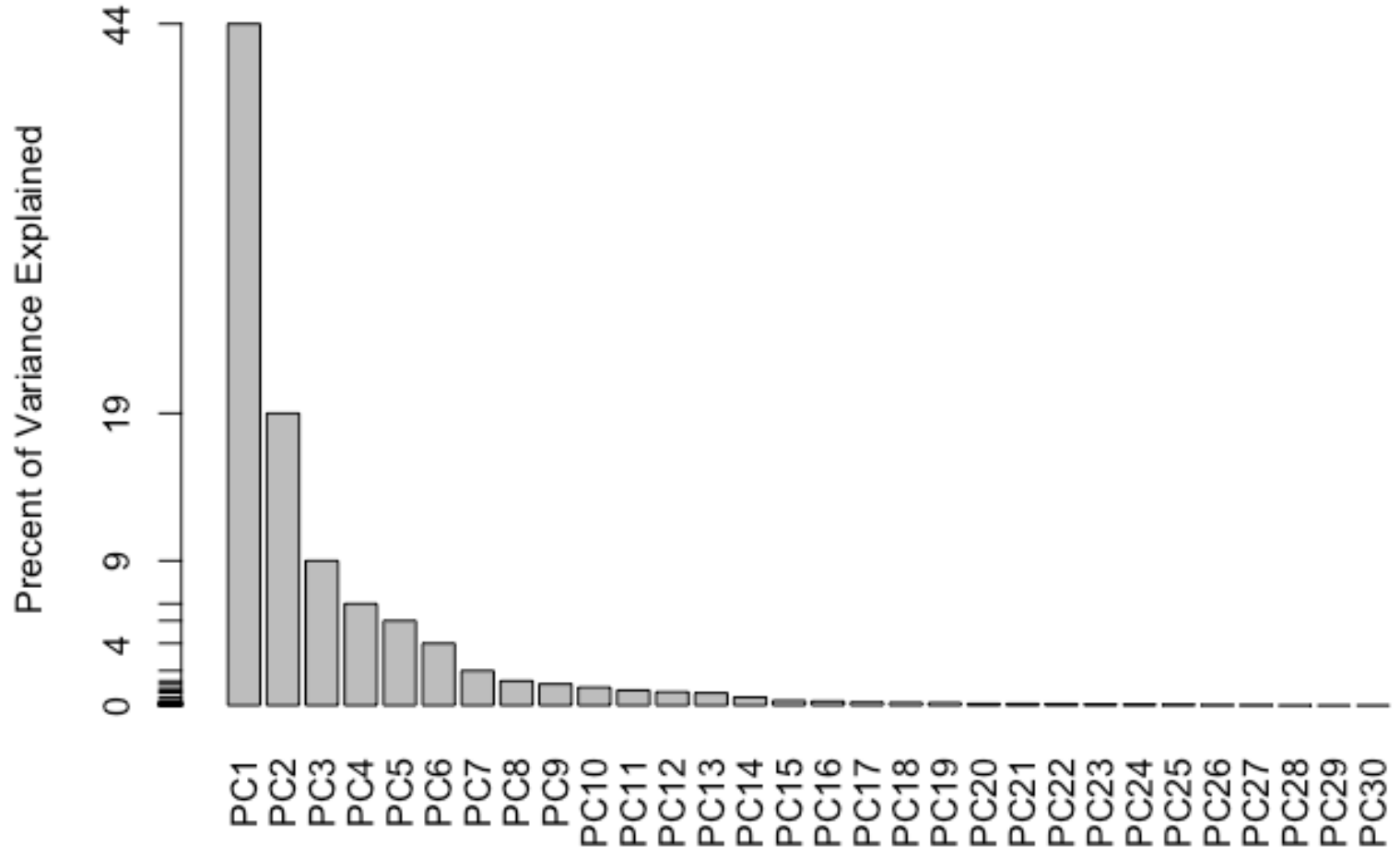
```
# Variance explained by each principal component: pve
pve <- ___ / ___

# Plot variance explained for each principal component
plot(pve, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     ylim = c(0, 1), type = "o")
```



Hide

Hide

```
# Alternative scree plot of the same data, note data driven y-axis
barplot(pve, ylab = "Precent of Variance Explained",
     names.arg=paste0("PC",1:length(pve)), las=2, axes = FALSE)
axis(2, at=pve, labels=round(pve,2)*100 )
```

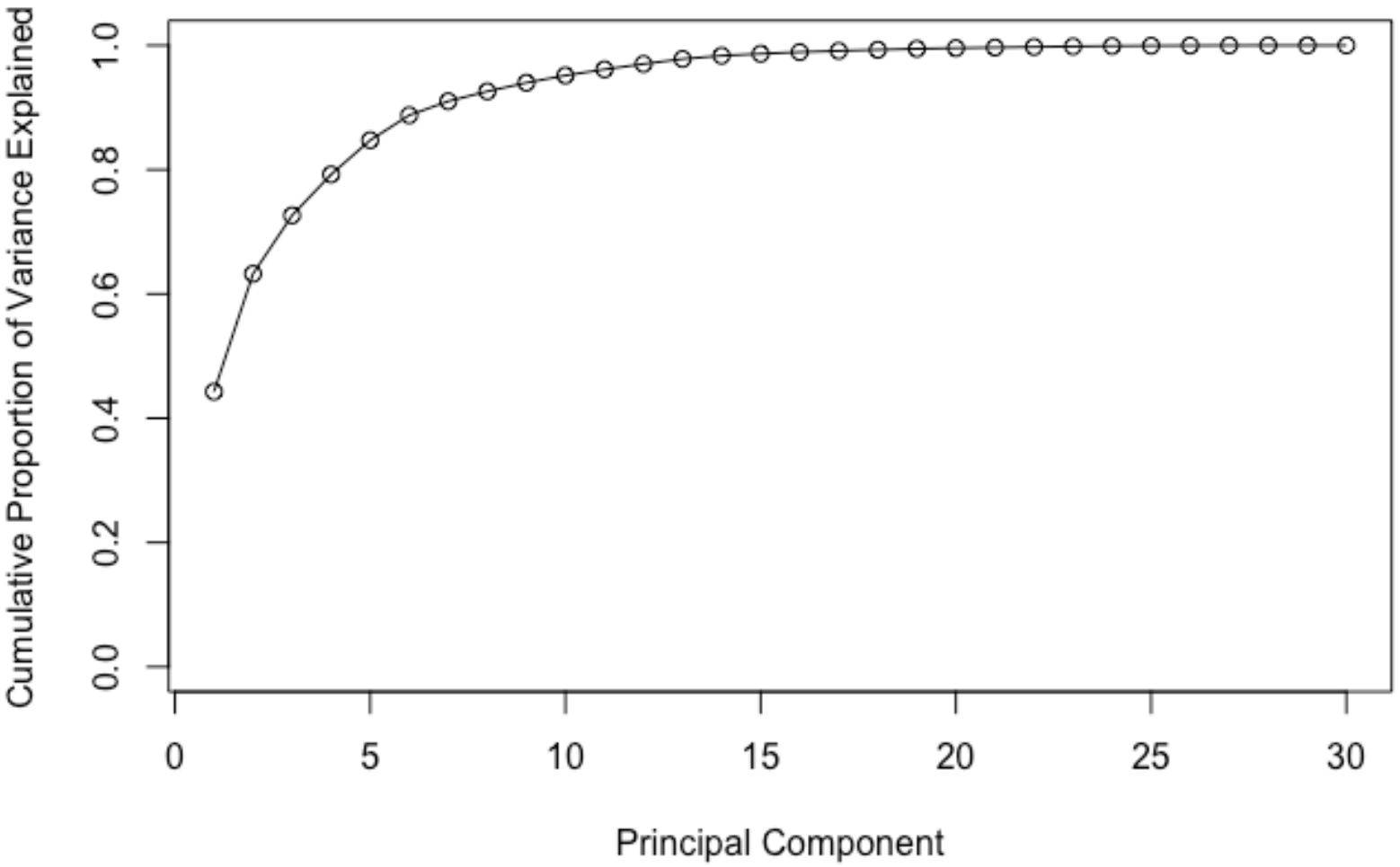


Using the `cumsum()` function, create a plot of cumulative proportion of variance explained.

Hide

Hide

```
# Plot cumulative proportion of variance explained
plot(cumsum(pve), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     ylim = c(0, 1), type = "o")
```

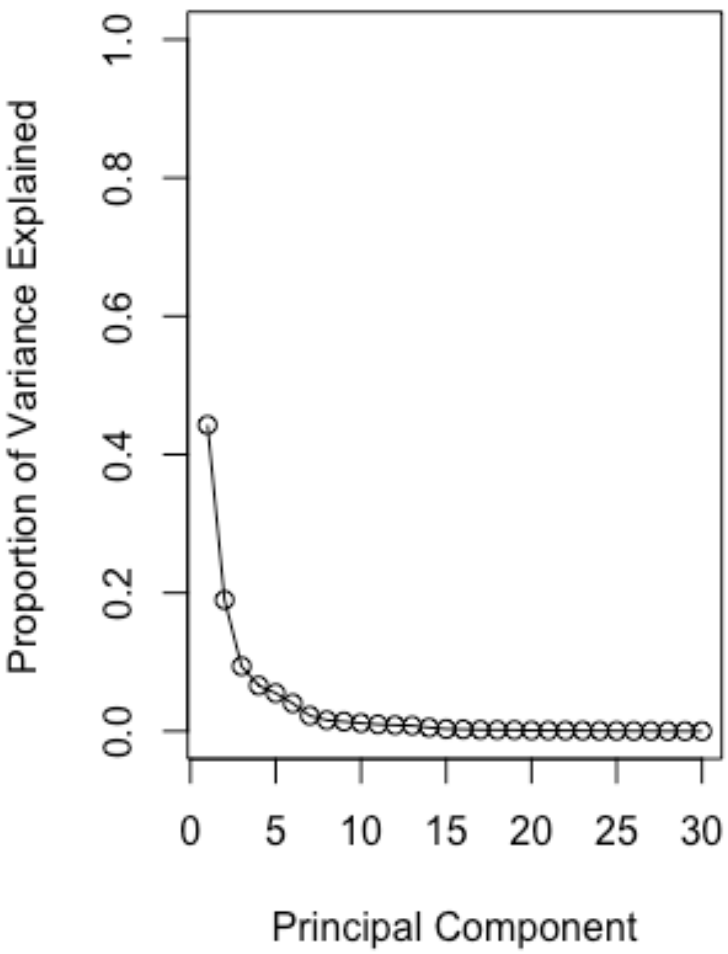


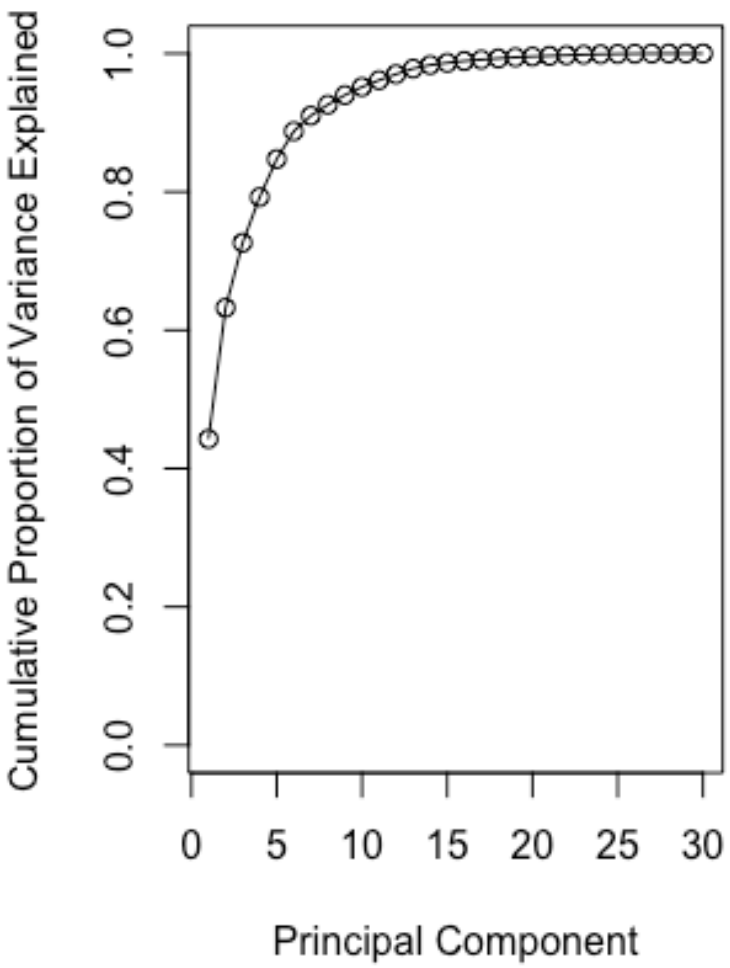
Hide

Hide

```
# Plot cumulative proportion of variance explained
plot( ___ , xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     ylim = c(0, 1), type = "o")
```

Use the `par()` function to create a side by side plot (i.e. 1 row 2 column arrangement) of these two graphs.





Hide

Hide

```
## ggplot based graph
#install.packages("factoextra")
library(factoextra)
```

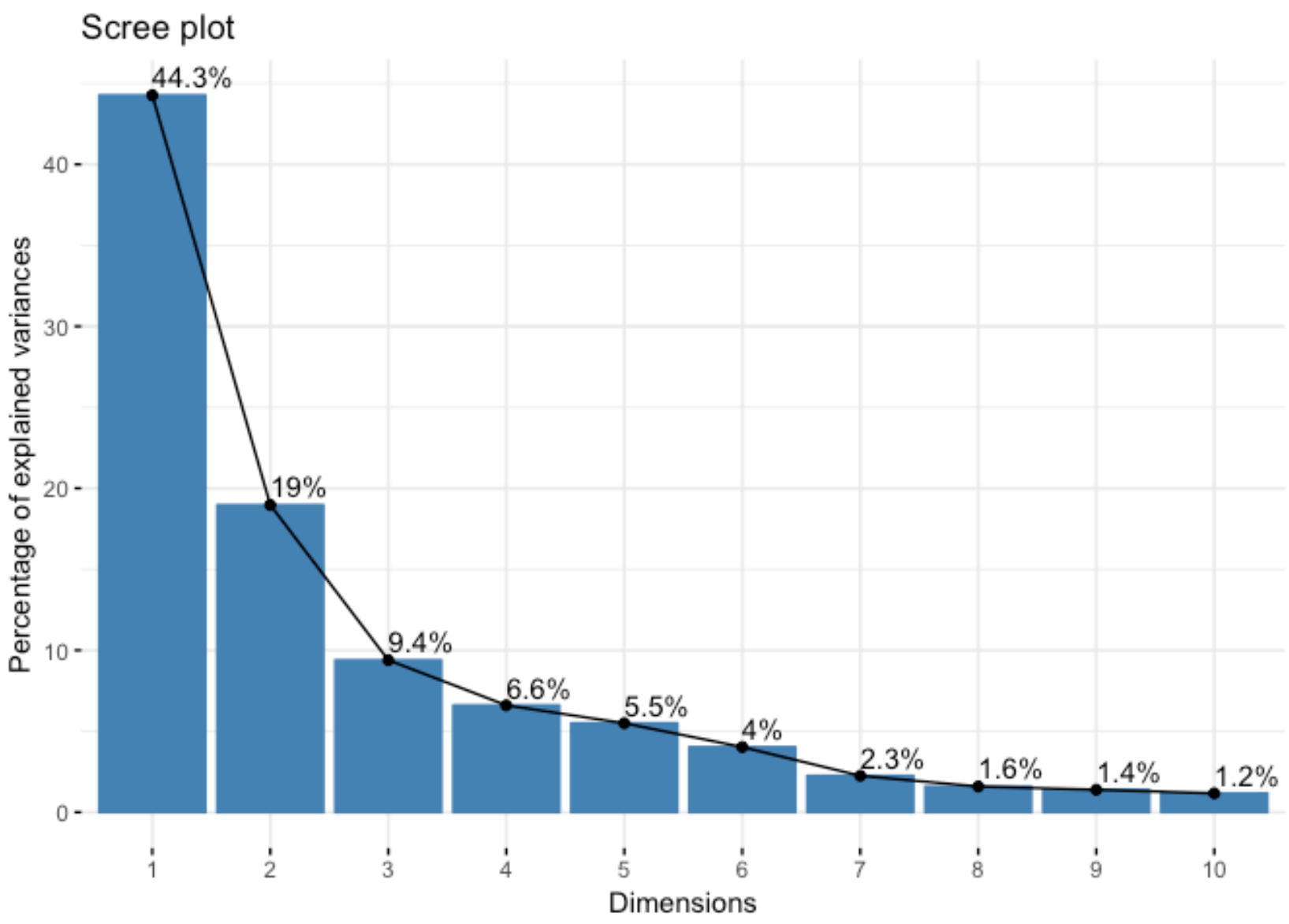
```
## Loading required package: ggplot2
```

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

Hide

Hide

```
fviz_eig(wisc.pr, addlabels = TRUE)
```



Communicating PCA results

In this section we will check your understanding of the PCA results, in particular the loadings and variance explained. The loadings, represented as vectors, explain the mapping from the original features to the principal components. The principal components are naturally ordered from the most variance explained to the least variance explained.

- **Q9.** For the first principal component, what is the component of the loading vector (i.e. `wisc.pr$rotation[,1]`) for the feature `concave.points_mean` ?
- **Q10.** What is the minimum number of principal components required to explain 80% of the variance of the data?

3. Hierarchical clustering

Hierarchical clustering of case data

The goal of this section is to do hierarchical clustering of the observations. Recall from our last class that this type of clustering does not assume in advance the number of natural groups that exist in the data.

As part of the preparation for hierarchical clustering, the distance between all pairs of observations are computed. Furthermore, there are different ways to link clusters together, with single, complete, and average being the most common linkage methods.

Scale the `wisc.data` data and assign the result to `data.scaled`.

Hide

Hide

```
# Scale the wisc.data data: data.scaled
data.scaled <- ___(wisc.data)
```

Calculate the (Euclidean) distances between all pairs of observations in the new scaled dataset and assign the result to `data.dist`.

Hide

Hide

```
data.dist <- ___(data.scaled)
```

Create a hierarchical clustering model using complete linkage. Manually specify the method argument to hclust() and assign the results to `wisc.hclust`.

Hide

Hide

```
wisc.hclust <- ___(data.dist, ___)
```

Results of hierarchical clustering

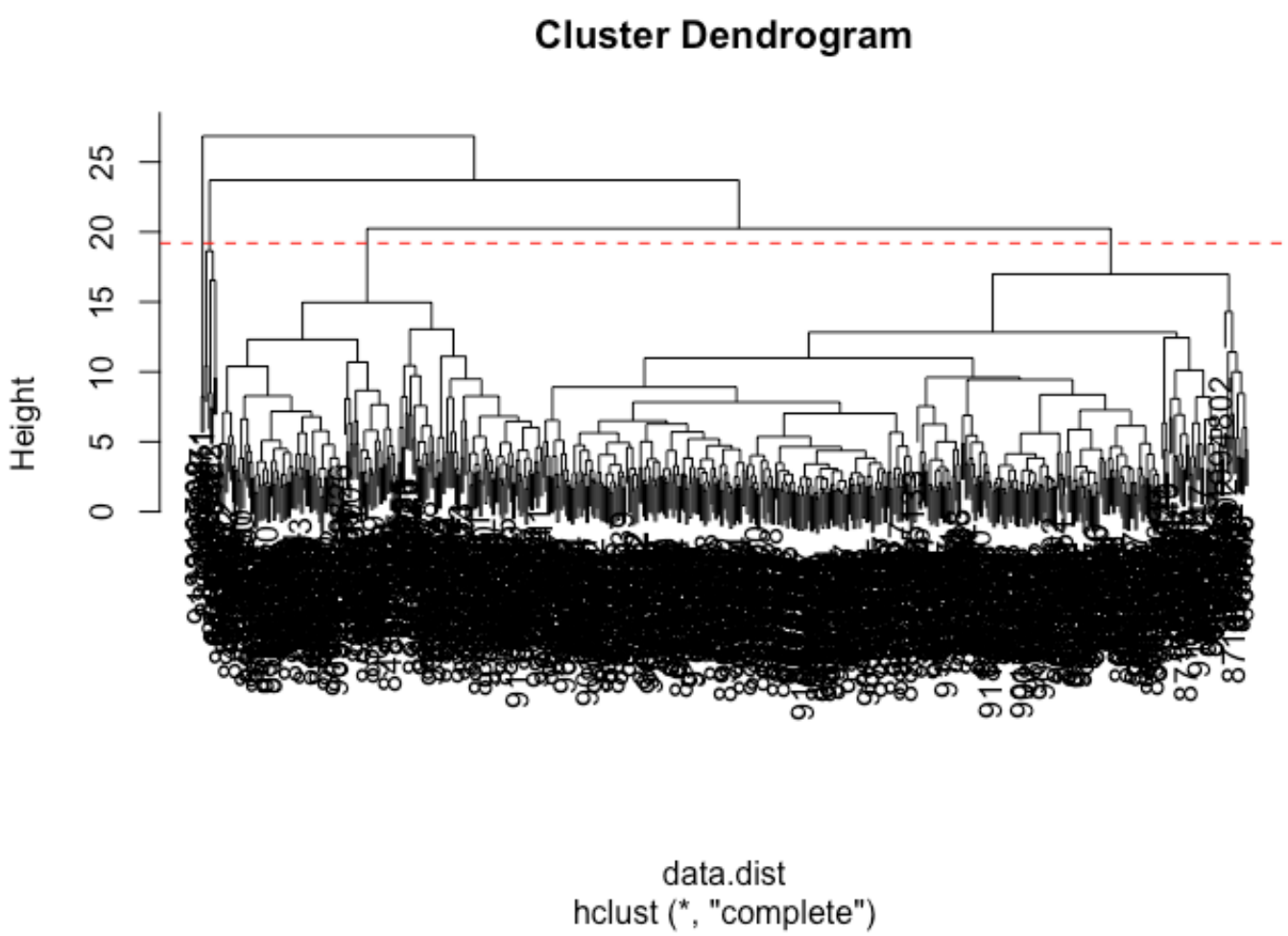
Let's use the hierarchical clustering model you just created to determine a height (or distance between clusters) where a certain number of clusters exists.

- **Q11.** Using the `plot()` and `abline()` functions, what is the height at which the clustering model has 4 clusters?

Hide

Hide

```
plot(___)
abline(___, col="red", lty=2)
```



Selecting number of clusters

In this section, you will compare the outputs from your hierarchical clustering model to the actual diagnoses. Normally when performing unsupervised learning like this, a target variable (i.e. known answer or labels) isn't available. We do have it with this dataset, however, so it can be used to check the performance of the clustering model.

When performing supervised learning - that is, when you're trying to predict some target variable of interest and that target variable is available in the original data - using clustering to create new features may or may not improve the performance of the final model.

This exercise will help you determine if, in this case, hierarchical clustering provides a promising new feature.

Use `cutree()` to cut the tree so that it has 4 clusters. Assign the output to the variable `wisc.hclust.clusters`.

Hide

Hide

```
wisc.hclust.clusters <- ___
```

We can use the `table()` function to compare the cluster membership to the actual diagnoses.

Hide

Hide

```
table(wisc.hclust.clusters, diagnosis)
```

```
##                     diagnosis
## wisc.hclust.clusters   0   1
##                    1  12 165
##                    2   2   5
##                    3 343  40
##                    4   0   2
```

Here we picked four clusters and see that cluster 1 largely corresponds to malignant cells (with `diagnosis` values of 1) whilst cluster 3 largely corresponds to benign cells (with `diagnosis` values of 0).

Before moving on, explore how different numbers of clusters affect the ability of the hierarchical clustering to separate the different diagnoses.

- **Q12.** Can you find a better cluster vs diagnoses match with by cutting into a different number of clusters between 2 and 10?

4. K-means clustering

K-means clustering and comparing results

As you now know, there are two main types of clustering: hierarchical and k-means.

In this section, you will create a k-means clustering model on the Wisconsin breast cancer data and compare the results to the actual diagnoses and the results of your hierarchical clustering model. Take some time to see how each clustering model performs in terms of separating the two diagnoses and how the clustering models compare to each other.

Create a k-means model on `wisc.data`, assigning the result to `wisc.km`. Be sure to create 2 clusters, corresponding to the actual number of diagnosis. Also, remember to scale the data (with the `scale()` function and repeat the algorithm 20 times (by setting setting the value of the `nstart` argument appropriately). Running multiple times such as this will help to find a well performing model.

Hide

Hide

```
wisc.km <- kmeans(___, centers= ___, nstart= ___)
```

Use the `table()` function to compare the cluster membership of the k-means model (`wisc.km$cluster`) to the actual diagnoses contained in the `diagnosis` vector.

Hide

Hide

```
table(___, ___)
```

- **Q13**. How well does k-means separate the two diagnoses? How does it compare to your hclust results?

Use the table() function to compare the cluster membership of the k-means model (`wisc.km$cluster`) to your hierarchical clustering model from above (`wisc.hclust.clusters`). Recall the cluster membership of the hierarchical clustering model is contained in `wisc.hclust.clusters` object.

Hide

Hide

```
table(___, ___)
```

```
##
## wisc.hclust.clusters   1   2
##                    1  17 160
##                    2   0   7
##                    3 363  20
##                    4   0   2
```

Looking at the second table you generated, it looks like clusters 1, 2, and 4 from the hierarchical clustering model can be interpreted as the cluster 1 equivalent from the k-means algorithm, and cluster 3 can be interpreted as the cluster 2 equivalent.

5. Combining methods

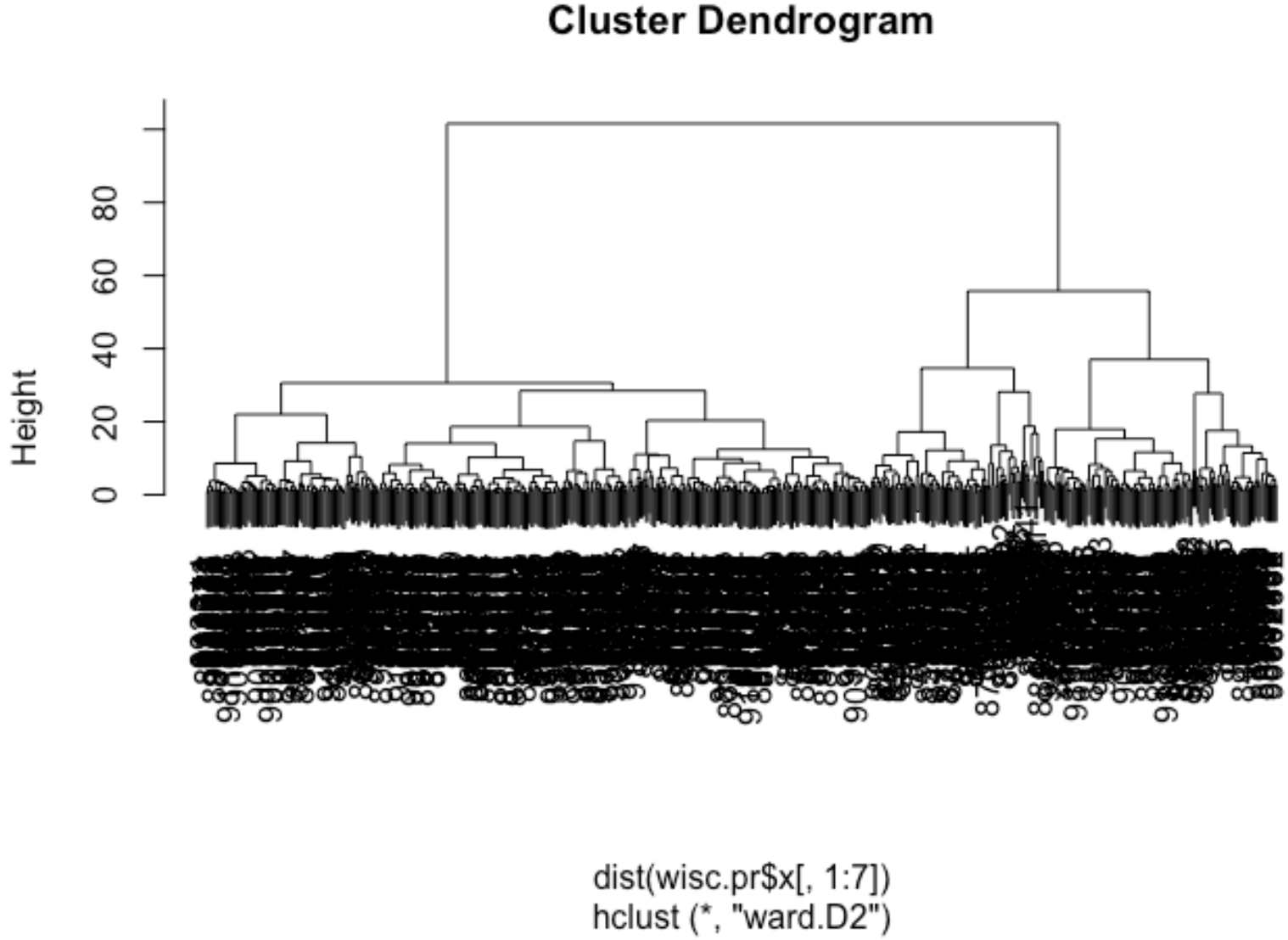
Clustering on PCA results

In this final section, you will put together several steps you used earlier and, in doing so, you will experience some of the creativity and open endedness that is typical in unsupervised learning.

Recall from earlier sections that the PCA model required significantly fewer features to describe 70%, 80% and 95% of the variability of the data. In addition to normalizing data and potentially avoiding over-fitting, PCA also uncorrelates the variables, sometimes improving the performance of other modeling techniques.

Let's see if PCA improves or degrades the performance of hierarchical clustering.

Using the minimum number of principal components required to describe at least 90% of the variability in the data, create a hierarchical clustering model with the linkage `method="ward.D2"` . We use Ward's criterion here because it is based on multidimensional variance like principal components analysis. Assign the results to `wisc.pr.hclust` .



Hide

Hide

```
grps <- cutree(wisc.pr.hclust, k=2)
table(grps)
```

```
## grps
##   1   2
## 216 353
```

Hide

Hide

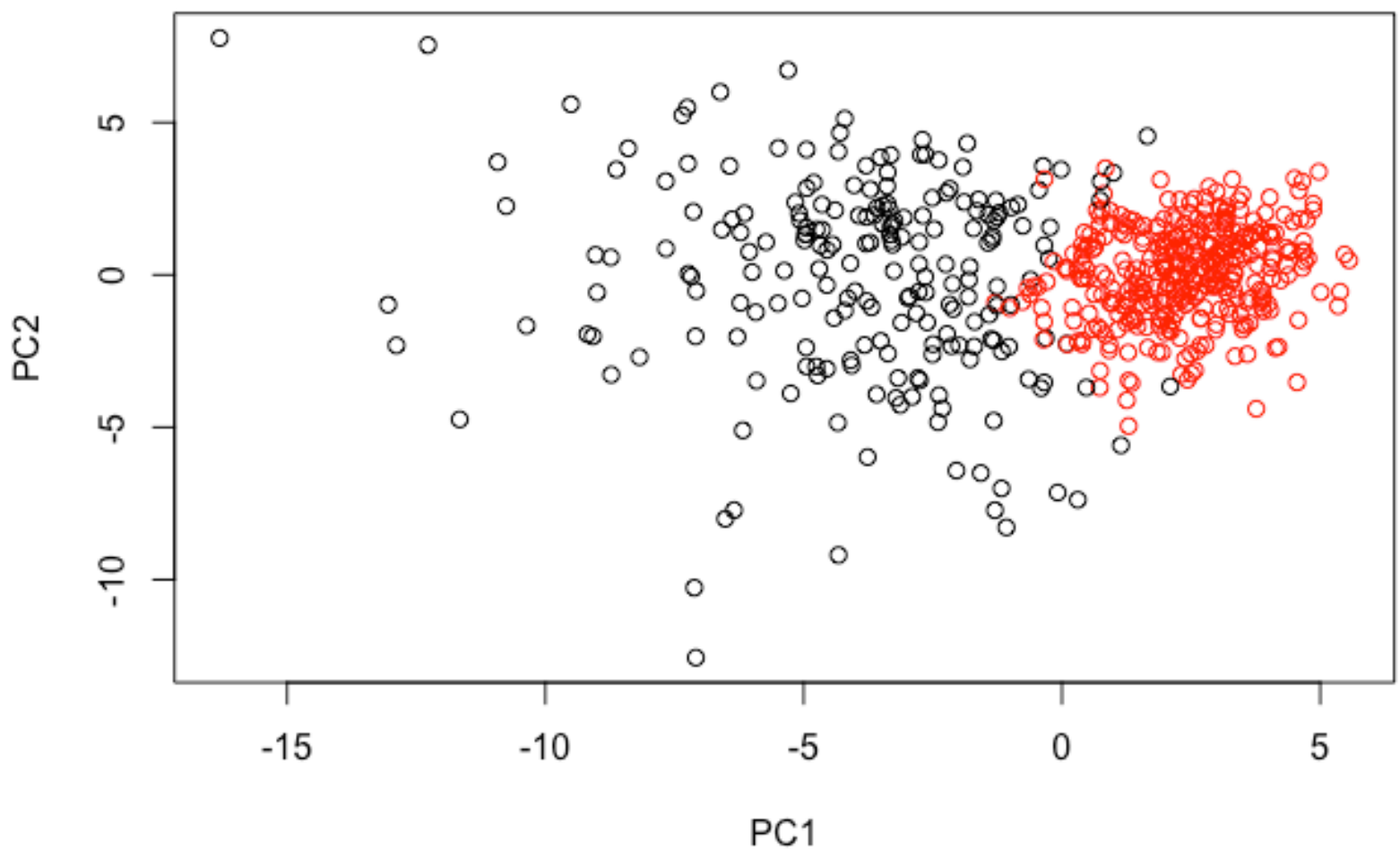
```
table(grps, diagnosis)
```

```
##     diagnosis
## grps   0   1
##    1  28 188
##    2 329  24
```

Hide

Hide

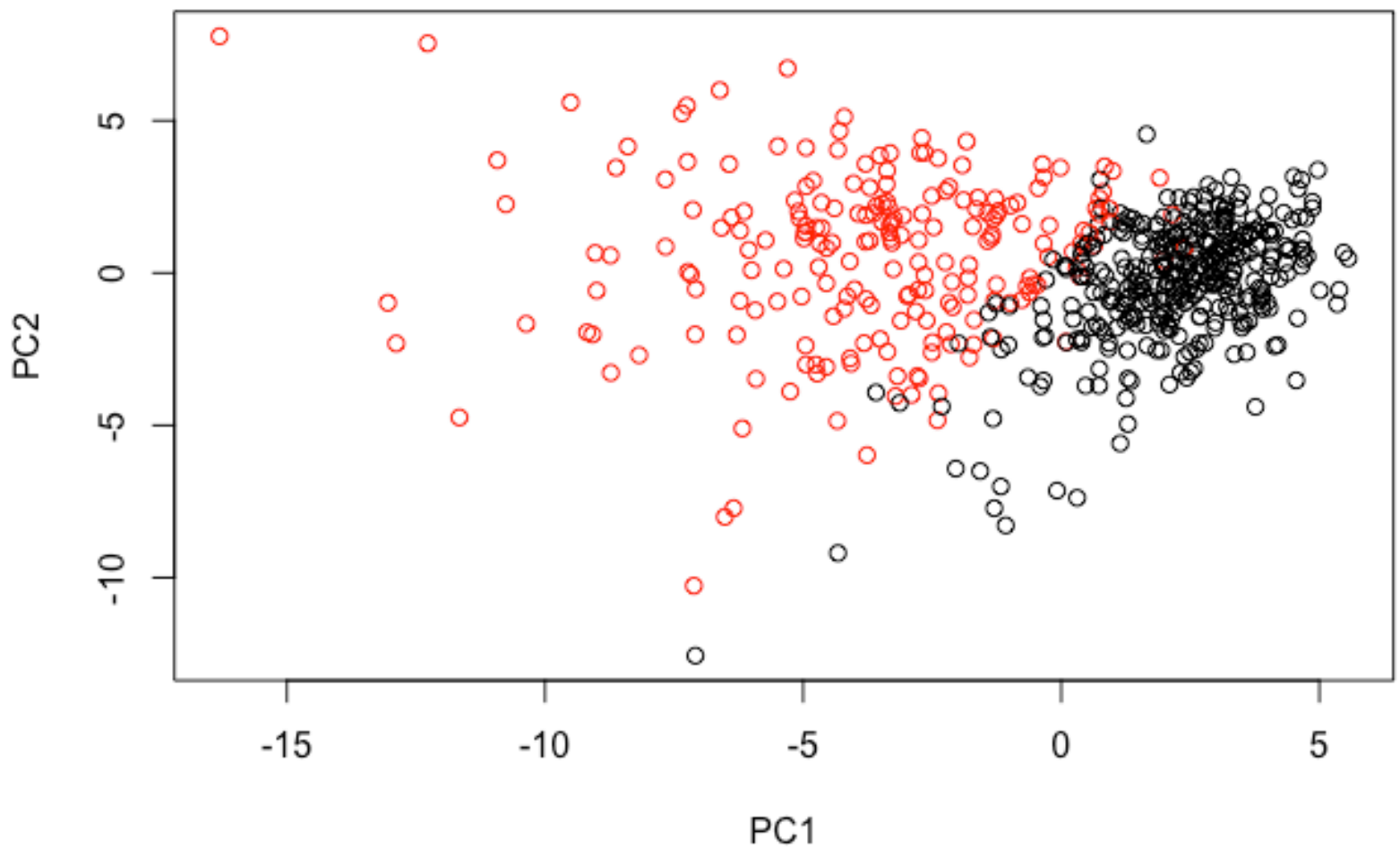
```
plot(wisc.pr$x[,1:2], col=grps)
```



Hide

Hide

```
plot(wisc.pr$x[,1:2], col=diagnosis+1)
```



Lets be fancy and look in 3D with the **rgl** package we learned about in a previous class. Feel free to skip this optional step if you have difficulty installing the **rgl** package on windows machines.

Hide

Hide

```
library(rgl)
plot3d(wisc.pr$x[,1:3], xlab="PC 1", ylab="PC 2", zlab="PC 3", cex=1.5,
size=1, type="s", col=diagnosis+1)
```

To include the interactive **rgl** plot in your HTML renderd lab report you can add the R code `rglwidget(width = 400, height = 400)` after you call the `plot3d()` function. It will look just like the plot above. **Try rotating and zooming on this 3D plot**.

Hide

Hide

```
## Use the distance along the first 7 PCs for clustering i.e. wisc.pr$x
[, 1:7]
wisc.pr.hclust <- hclust(___, method="ward.D2")
```

Cut this hierarchical clustering model into 2 clusters and assign the results to `wisc.pr.hclust.clusters`.

Hide

Hide

```
wisc.pr.hclust.clusters <- cutree(wisc.pr.hclust, k=2)
```

Using `table()`, compare the results from your new hierarchical clustering model with the actual diagnoses.

- **Q14**. How well does the newly created model with four clusters separate out the two diagnoses?

Hide

Hide

```
# Compare to actual diagnoses
table(___, diagnosis)
```

```
##                         diagnosis
## wisc.pr.hclust.clusters   0   1
##                       1  28 188
##                       2 329  24
```

- **Q15**. How well do the k-means and hierarchical clustering models you created in previous sections (i.e. before PCA) do in terms of separating the diagnoses? Again, use the `table()` function to compare the output of each model (`wisc.km$cluster` and `wisc.hclust.clusters`) with the vector containing the actual diagnoses.

Hide

Hide

```
table(___, diagnosis)
table(___, diagnosis)
```

```
##    diagnosis
##        0   1
##   1 343  37
##   2  14 175
```

```
##                      diagnosis
## wisc.hclust.clusters   0   1
##                    1  12 165
##                    2   2   5
##                    3 343  40
##                    4   0   2
```

6. Sensitivity/Specificity

Sensitivity refers to a test's ability to correctly detect ill patients who do have the condition. In our example here the sensitivity is the total number of samples in the cluster identified as predominantly malignant (cancerous) divided by the total number of known malignant samples.

Specificity relates to a test's ability to correctly reject healthy patients without a condition. In our example specificity is the proportion of benign (not cancerous) samples in the cluster identified as predominantly benign that are known to be benign.

- **Q16.** Which of your analysis procedures resulted in a clustering model with the best specificity? How about sensitivity?

7. Prediction

We will use the `predict()` function that will take our PCA model from before and new cancer cell data (https://tinyurl.com/new-samples-CSV) and project that data onto our PCA space.

Hide

Hide

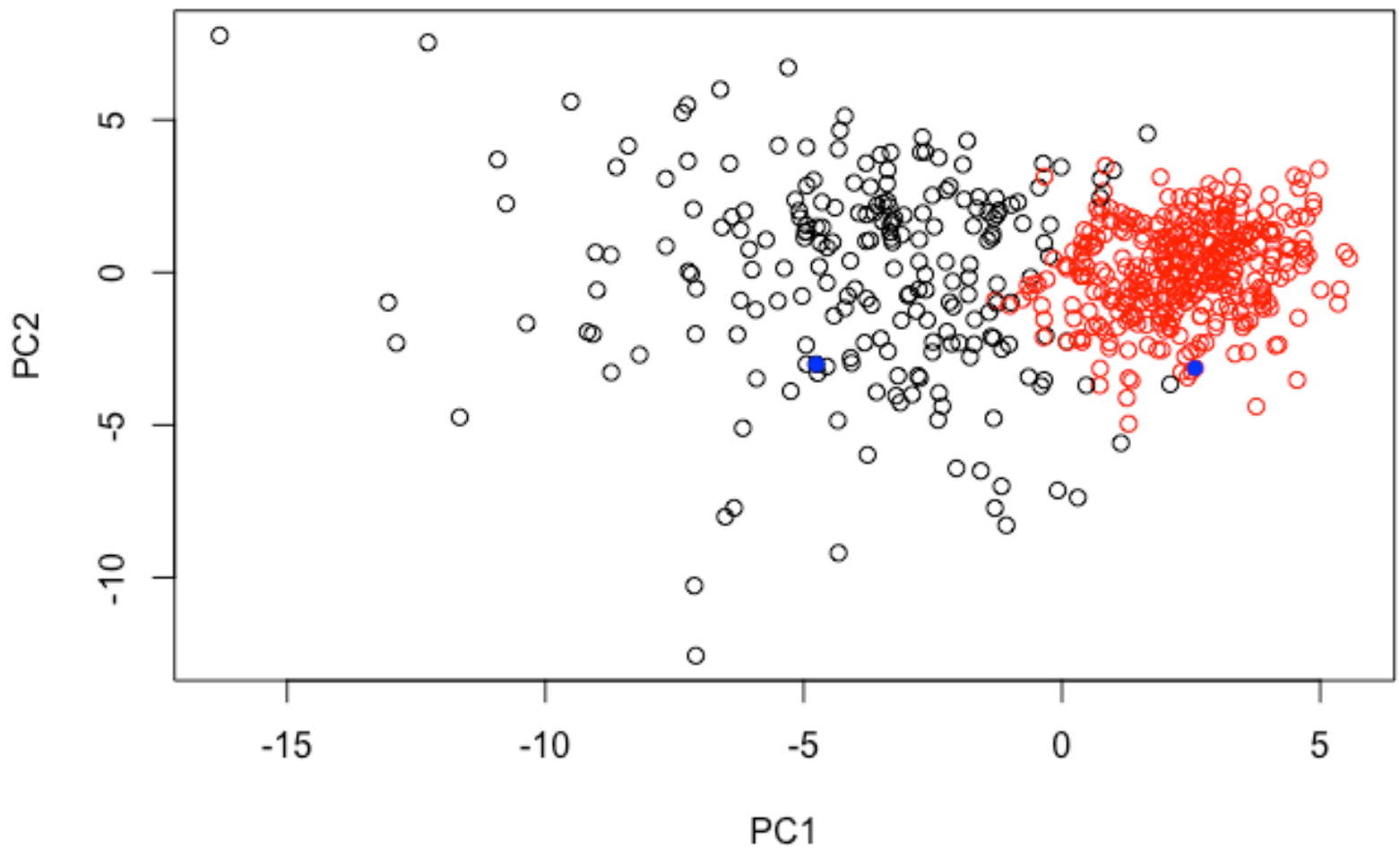
```
#url <- "new_samples.csv"
url <- "https://tinyurl.com/new-samples-CSV"
new <- read.csv(url)
npc <- predict(wisc.pr, newdata=new)
npc
```

```
##             PC1       PC2        PC3        PC4       PC5        PC6
## [1,]  2.576616 -3.135913  1.3990492 -0.7631950  2.781648 -0.8150185
## [2,] -4.754928 -3.009033 -0.1660946 -0.6052952 -1.140698 -1.2189945
##             PC7        PC8       PC9       PC10      PC11      PC12
## [1,] -0.3959098 -0.2307350 0.1029569 -0.9272861 0.3411457  0.375921
## [2,]  0.8193031 -0.3307423 0.5281896 -0.4855301 0.7173233 -1.185917
##            PC13     PC14      PC15       PC16        PC17        PC18
## [1,] 0.1610764 1.187882 0.3216974 -0.1743616 -0.07875393 -0.11207028
## [2,] 0.5893856 0.303029 0.1299153  0.1448061 -0.40509706  0.06565549
##             PC19       PC20       PC21       PC22       PC23       P
C24
## [1,] -0.08802955 -0.2495216  0.1228233 0.09358453 0.08347651  0.1223
396
## [2,]  0.25591230 -0.4289500 -0.1224776 0.01732146 0.06316631 -0.2338
618
##             PC25         PC26         PC27        PC28         PC29
## [1,]  0.02124121  0.078884581  0.220199544 -0.02946023 -0.015620933
## [2,] -0.20755948 -0.009833238 -0.001134152  0.09638361  0.002795349
##              PC30
## [1,]  0.005269029
## [2,] -0.019015820
```

Hide

Hide

```
plot(wisc.pr$x[,1:2], col=grps)
points(npc[,1], npc[,2], col="blue", pch=16)
```



- **Q17.** Which of these new patients should we prioritize for follow up based on your results?

8. OPTIONAL

PCA of protein structure data

Visit the Bio3D-web PCA app http://bio3d.ucsd.edu (http://bio3d.ucsd.edu) and explore how PCA of large protein structure sets can provide considerable insight into major features and trends with clear biological mechanistic insight. Note that the final report generated from this app contains all the R code required to run the analysis yourself - including PCA and clustering. We will delve more into this type of analysis in the next class.

About this document

Here we use the `sessionInfo()` function to report on our R systems setup at the time of document execution.

Hide

Hide

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/lib
Rblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/l
ibRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] rgl_0.99.16      factoextra_1.0.5 ggplot2_3.0.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.19             later_0.7.5
##  [3] pillar_1.3.0             compiler_3.5.1
##  [5] plyr_1.8.4               bindr_0.1.1
##  [7] ggpubr_0.1.8             tools_3.5.1
##  [9] digest_0.6.18            jsonlite_1.5
## [11] evaluate_0.12            tibble_1.4.2
## [13] gtable_0.2.0             pkgconfig_2.0.2
## [15] rlang_0.3.0.1            shiny_1.1.0
## [17] crosstalk_1.0.0          ggrepel_0.8.0
## [19] yaml_2.2.0               bindrcpp_0.2.2
## [21] withr_2.1.2              dplyr_0.7.7
## [23] stringr_1.3.1            knitr_1.20
## [25] htmlwidgets_1.3          webshot_0.5.1
## [27] manipulateWidget_0.10.0 rprojroot_1.3-2
## [29] grid_3.5.1               tidyselect_0.2.5
## [31] glue_1.3.0               R6_2.3.0
## [33] rmarkdown_1.10           purrr_0.2.5
```

```
## [35] magrittr_1.5            promises_1.0.1
## [37] backports_1.1.2         scales_1.0.0
## [39] htmltools_0.3.6         assertthat_0.2.0
## [41] xtable_1.8-3            mime_0.6
## [43] colorspace_1.3-2        httpuv_1.4.5
## [45] labeling_0.3            miniUI_0.1.1.1
## [47] stringi_1.2.4           lazyeval_0.2.1
## [49] munsell_0.5.0           crayon_1.3.4
```