Analysis Using R 2024

Faculty: Shraddha Pai, Chaitra Sarathy, Ian Cheong, Zoe Klein, Amin Nooranikhojasteh, Nia Hu

June 13, 2024 - June 14, 2024

Contents

Part I

Introduction

Chapter 1

Workshop Info

Welcome to the 2024 Analysis Using R Canadian Bioinformatics Workshop webpage!

1.1 Schedule

Time (EDT)	June 13	Time (EDT)	June 14
8:30	Arrivals & Check-in	8:30	Arrivals
9:00	Welcome (Nia Hughes)	9:00	Day 1 review Linear mixed models and evaluation (Shraddha Pai)
9:30	Exploratory data analysis and introducing clustering (Shraddha Pai)	10:00	Break (30 mins)
10:30	Break (30 mins)	10:30	Short linear model exercise (Shraddha Pai, Ian Cheong)
11:00	Exploratory data analysis and clustering project (Shraddha Pai, Ian Cheong)	12:30	Class photo + Lunch (1 hr)
13:00	Lunch (1 hr)	13:30	Bioconductor for differential expression (Shraddha Pai)
14:00	Dimensionality reduction (Chaitra Sarathy)	14:30	Break (30 mins)

Time (EDT)	June 13	Time (EDT)	June 14
15:00	Break (30 mins)	15:00	Short differential expression exercise (Shraddha Pai, Ian Cheong)
15:30	Short dimensionality reduction and clustering project (Chaitra Sarathy, Ian Cheong)	17:00	Survey & Closing Remarks
17:30	Finished	17:30	Finished

1.2 Pre-work

You can find your pre-work here.

Chapter 2

Meet Your Faculty

2.0.0.1 Shraddha Pai

Investigator I, OICR Assistant Professor, University of Toronto

Dr. Pai integrates genomics and computational methods to advance precision medicine. Her previous work involves DNA methylome-based biomarker discovery in psychosis, and building machine learning algorithms for patient classification from multi-modal data. The Pai Lab at the Ontario Institute for Cancer Research focuses on biomarker discovery for detection, diagnosis and prognosis in brain cancers and other brain-related disorders.

2.0.0.2 Chaitra Sarathy

Bioinformatics Specialist Krembil Research Institute

Dr. Sarathy is a computational biologist with industry experience in software development. Her previous research centered around developing multi-scale mathematical models of human systems to characterise biochemical changes in obesity. In addition, she has developed methods based on machine learning and multi-omics integration to identify drug targets in cancer and stratify patients for clinical trials. She currently focuses on characterizing genetic malfunctions in neurological diseases.

2.0.0.3 Ian Cheong

MSc. Candidate University of Toronto

Ian is a Master's level candidate in the Department of Medical Biophysics at the University of Toronto. His thesis work in the Pai lab involves analysis of single-cell transcriptomes to find the link between brain development and the development of childhood brain cancer.

2.0.0.4 Zoe Klein

PhD. Candidate University of Toronto

Zoe is a PhD level candidate in the Department of Molecular Genetics at the University of Toronto. Her thesis work in the Reimand lab involves using computational tools to investigate the role of non-coding RNA in cancer.

2.0.0.5 Nia Hughes

Platform Training Manager, Canadian Bioinformatics Hub Ontario Institute for Cancer Research Toronto, ON, Canada

— training@bioinformatics.ca

Nia is the Platform Training Manager for the Canadian Bioinformatics Hub, where she coordinates the Canadian Bioinformatics Workshop Series. Prior to starting at OICR, she completed her M.Sc. in Bioinformatics from the University of Guelph in 2020 before working there as a bioinformatician studying epigenetic and transcriptomic patterns across maize varieties.

Chapter 3

Data

3.0.0.1 Course Data Downloads

mouse_pheno.txt titanic.csv tomerge1.csv tomerge2.txt

Part II

Modules

Chapter 4

Module 1: Exploratory Data Analysis and Clustering

4.1 Lecture

Canadian Bioinformatics Workshops

www.bioinformatics.ca
bioinformaticsdotca.github.io

4.2 Lab

The goal of this lab is to examine correlation structure in a sample transcriptomic dataset using clustering.

We will start using a small mouse expression dataset from two tissues: * We will briefly explore the dataset using `dim()`, `head()`, and `summary()` * We will visualize correlation using `pairs()` and `corrplot()` * We will cluster samples using k-means and hierarchical clustering methods using `dist()` and `hclust()` * We will assign samples to clusters using `cutree()` and visualize these using `heatmap()` * We will use `clValid` to find out what cluster number best separates groups

Finally we will explore clustering in a more complex bladder cancer dataset.

4.2.1 Load `mouse` Data

For this exercise we are going to load a dataset built into the `clValid` package. This dataset measures 147 genes and expressed sequence tags in two developing mouse lineages: the neural crest cells and mesoderm-derived cells. There are three samples per group.

Let's load the data.

```
suppressMessages(library(clValid))
data(mouse)
```

Use `str()` to see what types of data the columns have:

```
str(mouse)
```

```
## 'data.frame':    147 obs. of  8 variables:
##  $ ID : Factor w/ 147 levels "1415787_at","1415904_at",..: 111 88 93 74 138 103 46 1
##  $ M1 : num  4.71 3.87 2.88 5.33 5.37 ...
##  $ M2 : num  4.53 4.05 3.38 5.5 4.55 ...
##  $ M3 : num  4.33 3.47 3.24 5.63 5.7 ...
##  $ NC1: num  5.57 5 3.88 6.8 6.41 ...
##  $ NC2: num  6.92 5.06 4.46 6.54 6.31 ...
##  $ NC3: num  7.35 5.18 4.85 6.62 6.2 ...
##  $ FC : Factor w/ 9 levels "ECM/Receptors",..: 3 8 6 6 1 3 1 6 5 6 ...
```

Another command is `head()`:

```
head(mouse)
```

```
##             ID       M1       M2       M3      NC1      NC2      NC3
## 1   1448995_at 4.706812 4.528291 4.325836 5.568435 6.915079 7.353144
## 2 1436392_s_at 3.867962 4.052354 3.474651 4.995836 5.056199 5.183585
## 3 1437434_a_at 2.875112 3.379619 3.239800 3.877053 4.459629 4.850978
## 4   1428922_at 5.326943 5.498930 5.629814 6.795194 6.535522 6.622577
## 5 1452671_s_at 5.370125 4.546810 5.704810 6.407555 6.310487 6.195847
## 6   1448147_at 3.471347 4.129992 3.964431 4.474737 5.185631 5.177967
##                       FC
## 1 Growth/Differentiation
## 2   Transcription factor
## 3          Miscellaneous
## 4          Miscellaneous
## 5          ECM/Receptors
## 6 Growth/Differentiation
```

Summary provides useful information about the distribution of variables. Note that `FC` has categorical variables:

```
summary(mouse)
```

```
##           ID            M1              M2              M3
##  1415787_at:  1   Min.   :2.352   Min.   :2.139   Min.   :2.500
##  1415904_at:  1   1st Qu.:4.188   1st Qu.:4.151   1st Qu.:4.207
##  1415993_at:  1   Median :4.994   Median :5.043   Median :5.054
##  1416164_at:  1   Mean   :5.166   Mean   :5.140   Mean   :5.231
##  1416181_at:  1   3rd Qu.:6.147   3rd Qu.:6.015   3rd Qu.:6.129
##  1416221_at:  1   Max.   :9.282   Max.   :9.273   Max.   :9.228
##  (Other)   :141
##       NC1             NC2             NC3                              FC
##  Min.   :2.100   Min.   :1.996   Min.   :2.125   EST                    :31
##  1st Qu.:4.174   1st Qu.:4.136   1st Qu.:4.293   Transcription factor   :28
##  Median :4.996   Median :5.056   Median :4.974   Miscellaneous          :25
##  Mean   :5.120   Mean   :5.134   Mean   :5.118   ECM/Receptors          :16
##  3rd Qu.:5.860   3rd Qu.:5.920   3rd Qu.:5.826   Growth/Differentiation:16
##  Max.   :8.905   Max.   :8.954   Max.   :9.251   Unknown                :10
##                                                  (Other)                :21
```

What are the values in `FC`?

```r
table(mouse$FC)
```

```
##
##          ECM/Receptors                    EST Growth/Differentiation
##                     16                     31                     16
##   Kinases/Phosphatases             Metabolism          Miscellaneous
##                      7                      8                     25
##         Stress-induced   Transcription factor                Unknown
##                      6                     28                     10
```

Usually the information about samples ("metadata") is in a different table.

Let's load the sample information about the mouse dataset:

```r
pheno <- read.delim("datasets/mouse_pheno.txt",sep="\t",h=T,as.is=T)
```

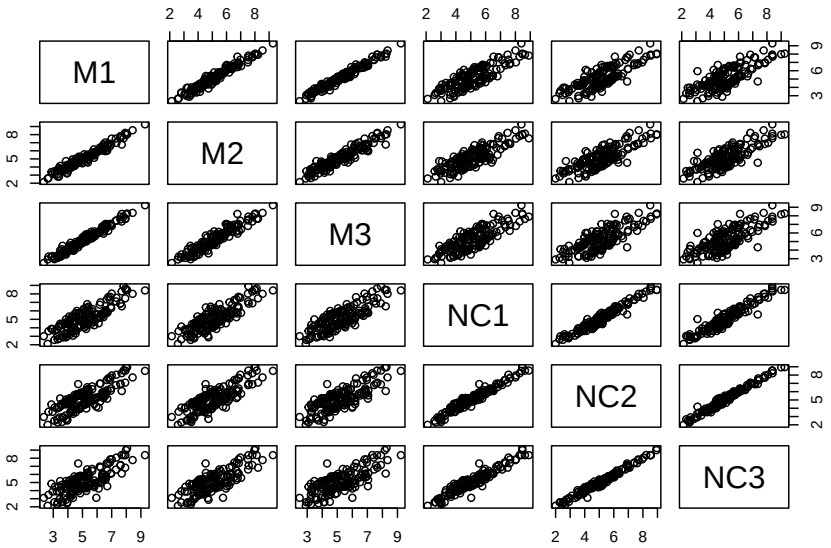
Let's look at it:

```r
head(pheno)
```

```
##   SampleID         Type
## 1       M1  Mesenchymal
## 2       M2  Mesenchymal
## 3       M3  Mesenchymal
## 4      NC1 Neural-crest
## 5      NC2 Neural-crest
## 6      NC3 Neural-crest
```

Let's use `pairs()` to look at pairwise scatterplots of the expression data in a single plot. We need to subset the columns with the expression data first:

```r
mouse_exp = mouse[,c("M1","M2","M3","NC1","NC2","NC3")]
pairs(mouse_exp)
```



4.2.2 Correlations, Distances, and Clustering

Let's look at sample correlation matrix. This should be a square matrix, equal to the number of samples. We have six samples, so the correlation matrix should be 6x6.

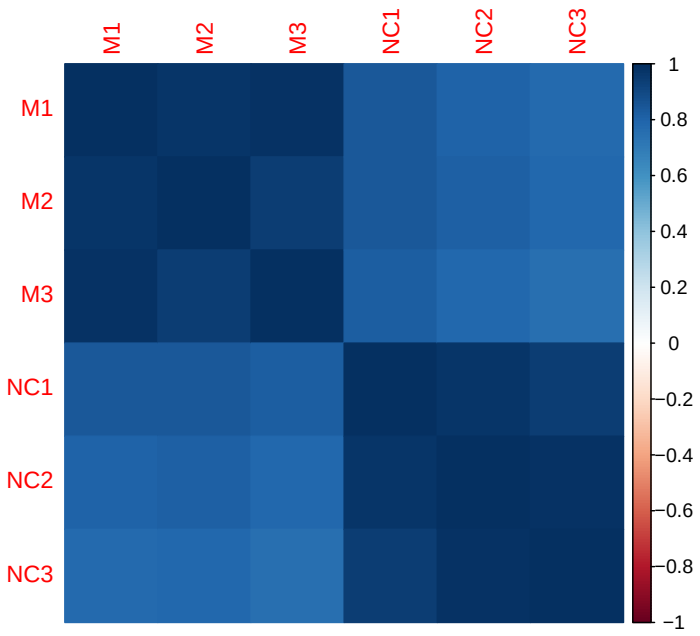
```
library(corrplot)
mouse_cor <- cor(mouse_exp)
dim(mouse_cor)
```

```
## [1] 6 6
```

```
round(mouse_cor,2)
```

```
##       M1   M2   M3  NC1  NC2  NC3
## M1  1.00 0.98 0.98 0.84 0.81 0.78
## M2  0.98 1.00 0.95 0.84 0.81 0.79
## M3  0.98 0.95 1.00 0.82 0.78 0.75
## NC1 0.84 0.84 0.82 1.00 0.97 0.95
## NC2 0.81 0.81 0.78 0.97 1.00 0.99
## NC3 0.78 0.79 0.75 0.95 0.99 1.00
```

```
corrplot(mouse_cor, method="color")
```



- Which samples appear to be best correlated with each other?
- Which samples don't appear to be as well correlated with each other?

4.2.3 Hierarchical Clustering

Hierarchical clustering requires distances between samples. Let's use `dist()` to compute these distances, and `hclust()` to generate the hierarchical clustering object.

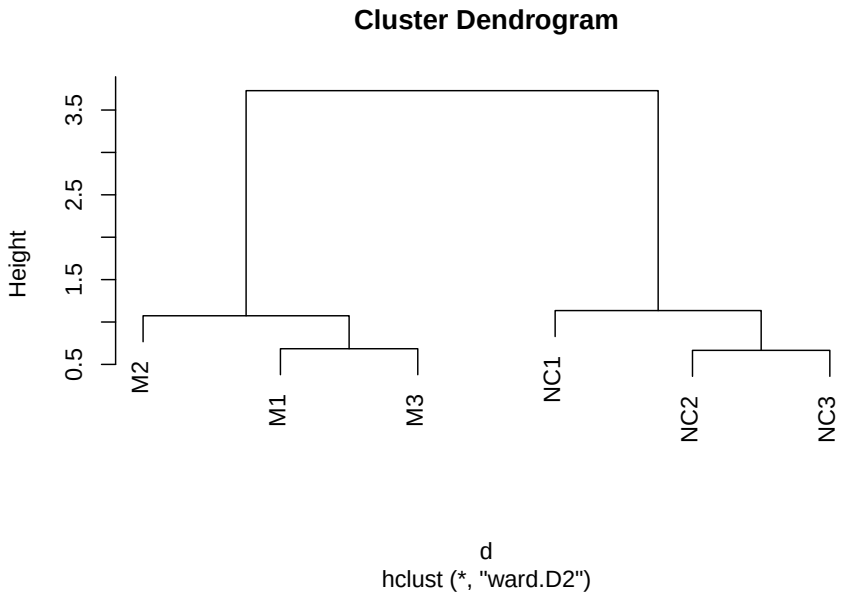
```
d <- dist(t(log(mouse_exp)))
```

Using these distances, we cluster the samples using hierarchical clustering:

```
h <- hclust(d,method="ward.D2")
```

The output of this can be plotted:

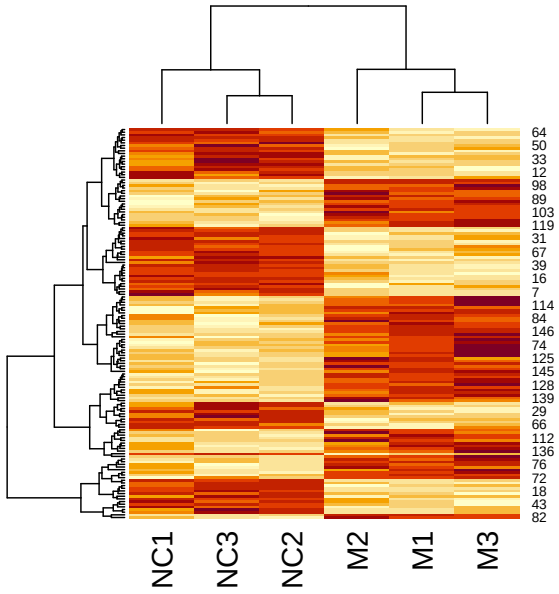
```
plot(h)
```



Can you guess how many clusters could best fit the data?

Now let's add a heatmap to this dendrogram, so we can see the values of genes in each cluster. For this we will use the `heatmap()` function. First let's just try the heatmap function:

```
mouse_exp <- as.matrix(mouse_exp)
heatmap(mouse_exp)
```



We can clearly see the data separated into two. But now let's colour-code samples based on cluster assignments. We get cluster assignments by "cutting" the dendrogram for two clusters (something we expect from our experimental design). We use `cutree()` for this.

```
h2 <- cutree(h, k = 2)
```

Let's look at our assigned labels:

```
h2
```

```
##  M1  M2  M3 NC1 NC2 NC3
##   1   1   1   2   2   2
```

Let's assign colours to the clusters so that cluster 1 is in pink, and cluster 2 is in green:

```
clust_colours <- c("pink","green")[h2]
```

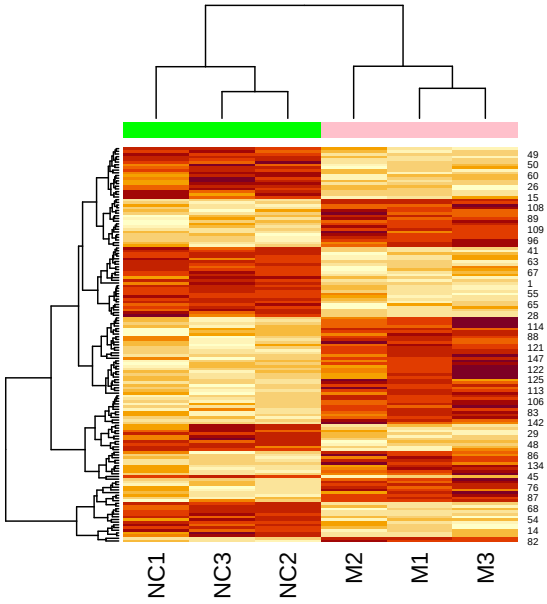
Look at clust_colours:

```
clust_colours
```

```
## [1] "pink"  "pink"  "pink"  "green" "green" "green"
```

Now let's plot the heatmap using these assigned cluster labels:

```
heatmap(mouse_exp,
    ColSideColors=clust_colours)
```



Note that the two colours are completely divided (i.e., there is no interspersed pink and green).

However, not all datasets are this simple. Let's cluster a bladder cancer gene expression dataset. This is in the R package, `bladderbatch` which we have already installed.

```
(library(bladderbatch))
```

```
##  [1] "bladderbatch" "Biobase"      "BiocGenerics" "corrplot"     "clValid"
##  [6] "cluster"      "stats"        "graphics"     "grDevices"    "utils"
## [11] "datasets"     "methods"      "base"
```

Load the dataset:

```
data(bladderdata)
```

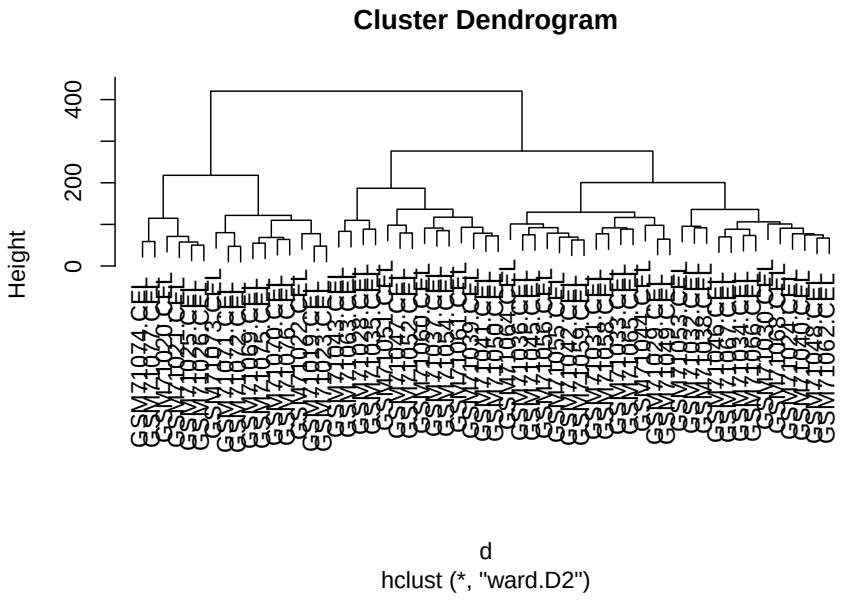
We will use specialized functions to get the expression data and sample information.

```
bexprs <- exprs(bladderEset)
bpheno <- pData(bladderEset)
```

How many genes and samples do we have in this dataset?

Let us use the same code as above to cluster these samples:

```
d <- dist(t(bexprs))
h <- hclust(d, method="ward.D2")
plot(h)
```



How many clusters do we see?

Let's assume three clusters, and assign colours to these. As before we use `cutree()`:

```
h3 <- cutree(h, k=3)
clust_colours <- c("red","green","blue")[h3]
```

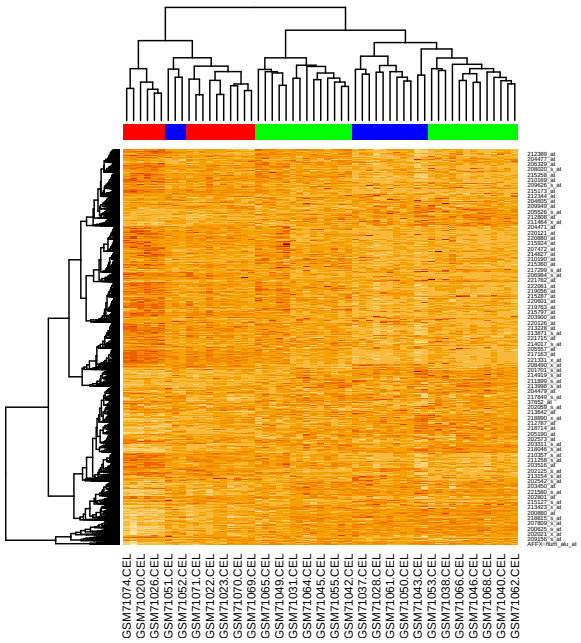
Look at the colour assignments?

```
table(h3)
```

```
## h3
##  1  2  3
## 16 27 14
```

Let's just plot the heatmap.

```
heatmap(bexprs,
    ColSideColors=clust_colours)
```



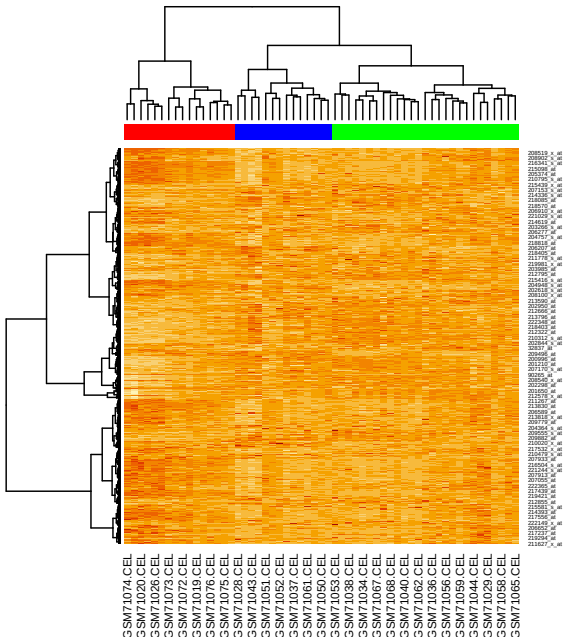
Why aren't the samples clustering?

Now try providing the `hclustfun` to `heatmap()` so it uses the same method to cluster as we did. For this we will create a custom function:

```
myhclust <- function(x) {
    hclust(x,method="ward.D2")
}
```

And now we run heatmap again, using our clustering function:

```
heatmap(bexprs,
    ColSideColors=clust_colours,
    hclustfun=myhclust
)
```



4.2.4 K-means Clustering

Let's try using k-means clustering, asking for three clusters:

```
kclust <- kmeans(
    mouse_exp,
    centers = 3
)
kclust
```

```
## K-means clustering with 3 clusters of sizes 61, 64, 22
##
## Cluster means:
##         M1       M2       M3      NC1      NC2      NC3
## 1 3.947440 3.946048 4.012209 3.922949 3.950984 4.004362
## 2 5.553148 5.499583 5.642404 5.426931 5.435363 5.353342
## 3 7.416536 7.406216 7.414799 7.548674 7.534414 7.520608
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   2   1   1   2   2   1   2   2   2   1   3   1   2   3   1   2   3   3   1   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   2   2   1   1   1   1   3   2   1   2   2   3   1   2   2   1   1   1   2   1
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   2   3   3   1   1   3   1   1   1   1   2   1   3   3   2   2   1   1   2   1
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   1   3   2   1   2   1   2   3   2   1   2   2   2   2   2   3   1   1   2   3
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
##   2   3   1   2   2   1   3   2   1   2   1   1   1   1   1   1   1   2   1   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   1   2   1   1   1   1   3   2   1   2   2   1   2   2   2   2   3   3   1   2
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   2   2   3   1   2   2   2   1   2   2   2   2   2   1   1   1   1   1   1   2
## 141 142 143 144 145 146 147
##   2   1   2   3   2   2   2
##
## Within cluster sum of squares by cluster:
## [1] 193.59229 166.24343  81.44264
##  (between_SS / total_SS =  74.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

4.2.5 Using `clValid` to Determine Number of Clusters

Use the `clValid()` function to validate clusters using the:

- Dunn index,
- silhouette scores, and
- connectivity

```
validation_data <- clValid(
    mouse_exp,
    2:6, # num. clusters to evaluate
    clMethods = c("hier","kmeans"), # methods to eval.
    validation = "internal"
)
```

Let's look at the results:

```
summary(validation_data)
```

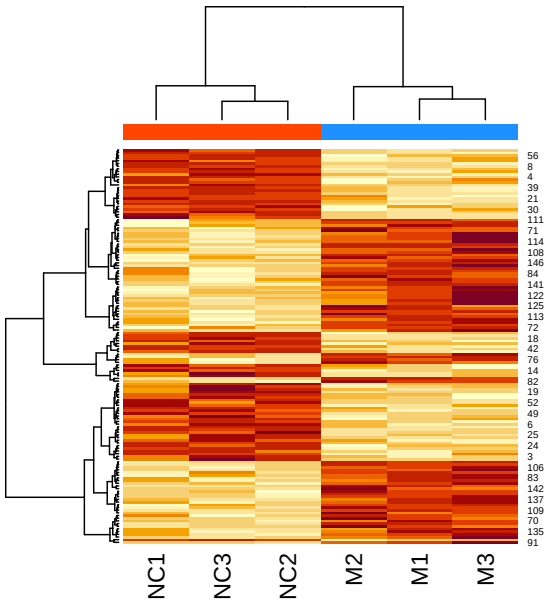
```
##
## Clustering Methods:
##  hierarchical kmeans
##
## Cluster sizes:
##  2 3 4 5 6
##
## Validation Measures:
##                                  2       3       4       5       6
##
## hierarchical Connectivity   5.3270 14.2528 20.7520 27.0726 30.6194
##              Dunn           0.1291  0.0788  0.0857  0.0899  0.0899
##              Silhouette     0.5133  0.4195  0.3700  0.3343  0.3233
## kmeans       Connectivity  13.2548 17.6651 37.3980 43.2655 50.6095
##              Dunn           0.0464  0.0873  0.0777  0.0815  0.0703
##              Silhouette     0.4571  0.4182  0.3615  0.3367  0.3207
##
## Optimal Scores:
##
##              Score  Method       Clusters
## Connectivity 5.3270 hierarchical 2
## Dunn         0.1291 hierarchical 2
## Silhouette   0.5133 hierarchical 2
```

All measures of clustering consistently indicate that **two** clusters best fit the data.

Now let's cluster:

```
d <- dist(t(log(mouse_exp)))
h <- hclust(d,method="ward.D2")
```

```
cluster_ids <- cutree(h, k = 2)
clust_colors <- c("dodgerblue","orangered")[cluster_ids]

heatmap(
    as.matrix(mouse_exp),
    hclustfun = myhclust,
    ColSideColors = clust_colors
)
```



4.2.6 Bonus Exercise

For your exercise, try the following:

- Load the MASS package using: `library(MASS)`
- Import `crabs` dataset using: `data(crabs)`
- Learn about this dataset using: `?crabs`
- Extract the numeric columns describing the crab measurements ("FL", "RW", "CL", "CW", "BD")
- Cluster the numeric columns using your method of choice
- Plot and color your data by clusters, by species (`sp`), and `sex`
- Do your clusters seem to separate these groups in the same way?

4.2.7 Bonus Exercise Results

Load packages and data, subset needed columns:

```
library(MASS)
data(crabs)
```

Learn more about the data:

```
?crabs
head(crabs)
```

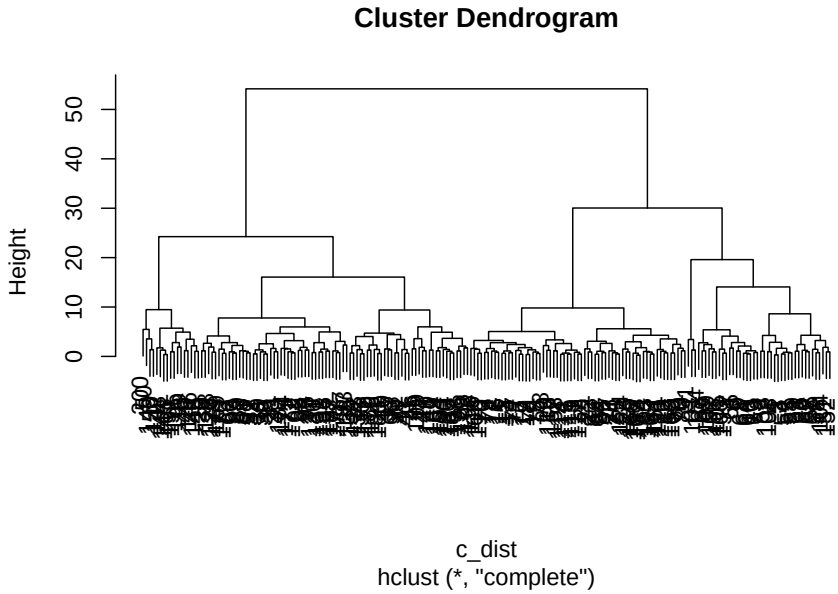
```
##   sp sex index   FL  RW   CL   CW  BD
## 1  B   M     1  8.1 6.7 16.1 19.0 7.0
## 2  B   M     2  8.8 7.7 18.1 20.8 7.4
## 3  B   M     3  9.2 7.8 19.0 22.4 7.7
## 4  B   M     4  9.6 7.9 20.1 23.1 8.2
## 5  B   M     5  9.8 8.0 20.3 23.0 8.2
## 6  B   M     6 10.8 9.0 23.0 26.5 9.8
```

Subset needed columns:

```
crabs_meas <-  crabs[,c("FL","RW","CL","CW","BD")]
```

Perform hierarchical clustering:

```
c_dist <- dist(crabs_meas)
c_hclust <- hclust(c_dist)
plot(c_hclust)
```



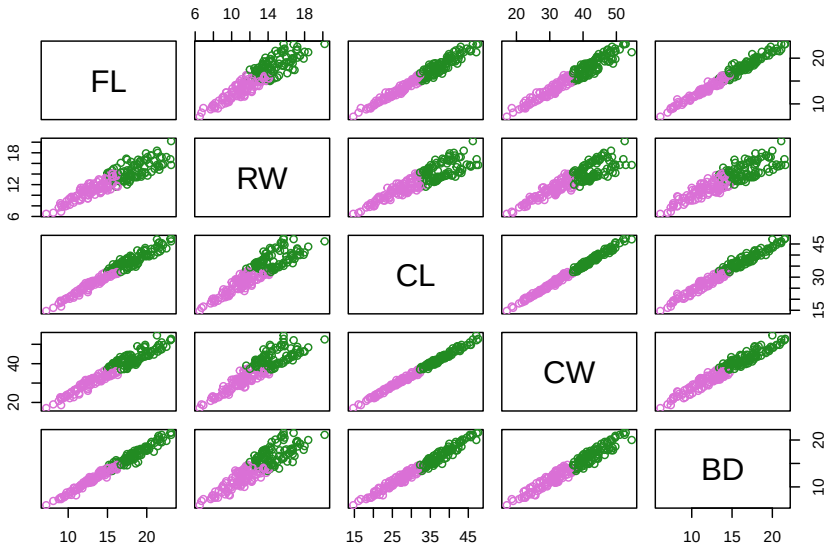
Colour-code samples based on cluster assignment. Assume there are two clusters.

```
c_clusters = cutree(c_hclust,k = 2)
```

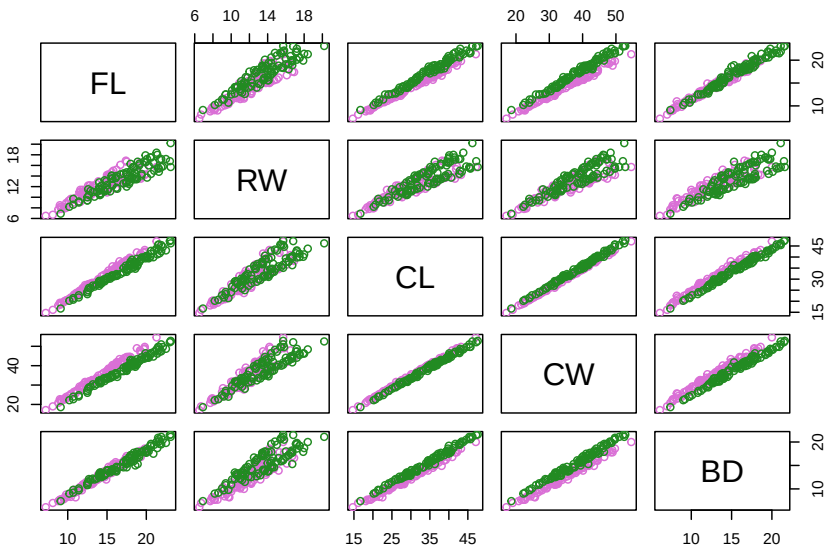
Now create a pairs plot, but colour-code by:

1. by gene-expression based clusters
2. by species
3. by sex

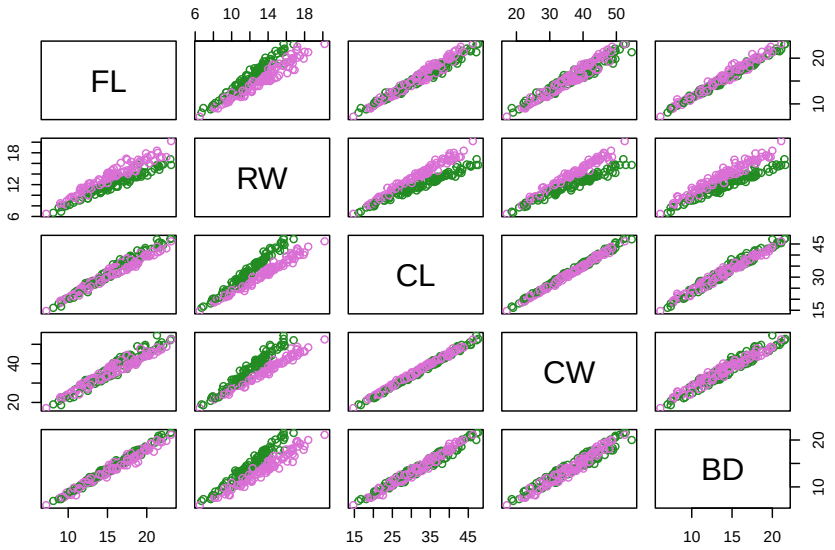
```
pairs(
    crabs_meas,
    col = c("orchid","forestgreen")[c_clusters]
)
```




```
pairs(
    crabs_meas,
    col = c("orchid","forestgreen")[factor(crabs$sp)]
)
```

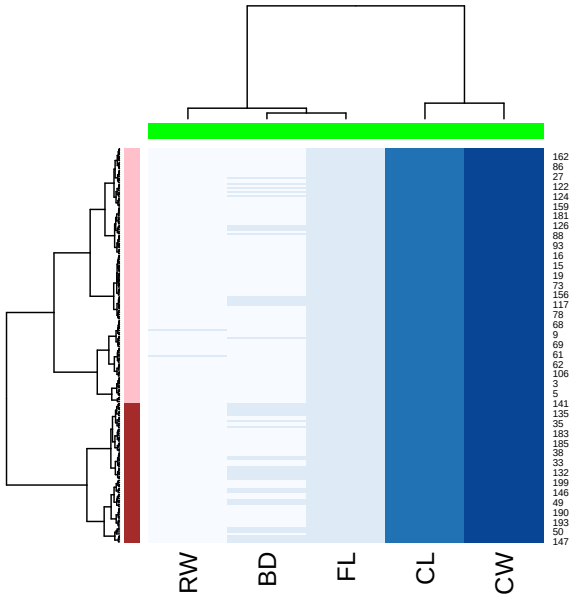



```
pairs(
    crabs_meas,
    col = c("orchid","forestgreen")[factor(crabs$sex)]
)
```



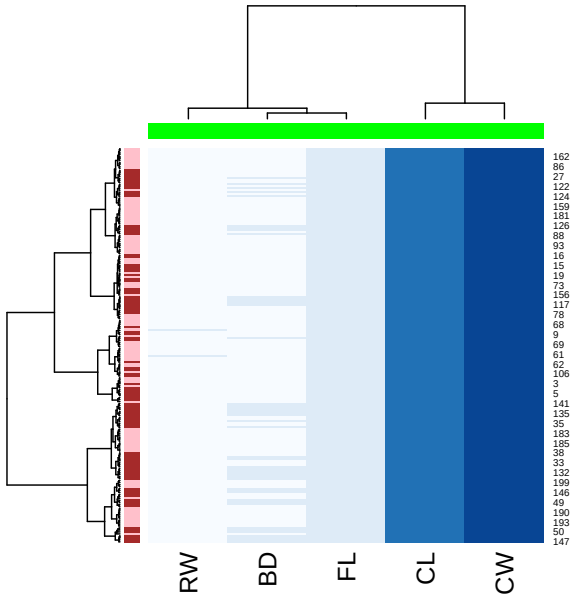
Hierarchical clustering:

```
h <- hclust(dist(crabs_meas),method="ward.D2")
c2 <- cutree(h,k=2)

hclust_fun <- function(x){
    f <- hclust(x, method = "ward.D2");
    return(f)
}

library(RColorBrewer)
heatmap(
    as.matrix(crabs_meas),
    hclustfun = hclust_fun,
    col = brewer.pal("Blues",n=8),
    RowSideColors = c("pink","brown")[c2],
    ColSideColors = rep("green",5)
)
```



Plot by sex:

```
heatmap(
    as.matrix(crabs_meas),
    hclustfun = hclust_fun,
    col = brewer.pal("Blues",n=8),
    RowSideColors = c("pink","brown")[factor(crabs$sex)],
    ColSideColors = rep("green",5)
)
```


162
86
27
122
124
159
181
126
88
93
16
15
19
73
156
117
78
68
9
69
61
62
106
3
5
141
135
35
183
185
38
33
132
199
146
49
190
193
50
147
RW
BD
FL
CL
CW

Chapter 5

Module 2: Dimensionality Reduction

5.1 Lecture

Canadian Bioinformatics Workshops

www.bioinformatics.ca
bioinformaticsdotca.github.io

5.2 Lab

The goal of this lab is to learn how to reduce the dimension of your dataset.

We will learn three different methods commonly used for dimension reduction:

1. Principal Component Analysis
2. t-stochastic Neighbor Embedding (tSNE)
3. Uniform Manifold Approximation (UMAP)

5.2.1 Principal Component Analysis

Let's start with PCA. PCA is commonly used as one step in a series of analyses. The goal of PCA is to explain most of the variability in the data with a smaller number of variables than the original data set. You can use PCA to explain the variability in your data using fewer variables. Typically, it is useful to identify outliers and determine if there's batch effect in your data.

Data: We will use the dataset that we used for exploratory analysis in Module 1. Load the mouse data.

```
library(clValid)
data("mouse")
mouse_exp <- mouse[,c("M1","M2","M3","NC1","NC2","NC3")]
head(mouse_exp)
```

```
##         M1       M2       M3      NC1      NC2      NC3
## 1 4.706812 4.528291 4.325836 5.568435 6.915079 7.353144
## 2 3.867962 4.052354 3.474651 4.995836 5.056199 5.183585
## 3 2.875112 3.379619 3.239800 3.877053 4.459629 4.850978
## 4 5.326943 5.498930 5.629814 6.795194 6.535522 6.622577
## 5 5.370125 4.546810 5.704810 6.407555 6.310487 6.195847
## 6 3.471347 4.129992 3.964431 4.474737 5.185631 5.177967
```

5.2.1.1 Step 1. Preparing our Data

It is important to make sure that all the variables in your dataset are on the same scale to ensure they are comparable. So, let us check if that is the case with our dataset. To do that, we will first compute the means and variances of each variable using `apply()`.

```
apply(mouse_exp, 2, mean)
```

```
##       M1       M2       M3      NC1      NC2      NC3
## 5.165708 5.140265 5.231185 5.120369 5.133540 5.117914
```

```
apply(mouse_exp, 2, var)
```

```
##       M1       M2       M3      NC1      NC2      NC3
## 1.858482 1.848090 1.869578 2.005517 2.080473 2.083073
```

As you can see, the means and variances for all the six variables are almost the same and on the same scale, which is great!

However, keep in mind that, the variables need not always be on the same scale in other non-omics datasets. PCA is influenced by the magnitude of each variable. So, it is important to include a scaling step during data preparation. Ideally, it is great to have variables centered at zero for PCA because it makes comparing each principal component to the mean straightforward. Scaling can be done either using `scale()`.

5.2.1.2 Step 2. Apply PCA

Since our variables are on the same scale, we can directly apply PCA using `prcomp()`.

```
pc_out <- prcomp(t(mouse_exp))
```

The output of `prcomp()` is a list. Examine the internal structure of `pc_out`.

```
str(pc_out)
```

```
## List of 5
##  $ sdev    : num [1:6] 5.577 1.764 1.583 0.758 0.667 ...
##  $ rotation: num [1:147, 1:6] 0.218 0.128 0.127 0.111 0.101 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:147] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:6] "PC1" "PC2" "PC3" "PC4" ...
##  $ center  : Named num [1:147] 5.57 4.44 3.78 6.07 5.76 ...
##   ..- attr(*, "names")= chr [1:147] "1" "2" "3" "4" ...
##  $ scale   : logi FALSE
##  $ x       : num [1:6, 1:6] -5.09 -4.46 -5.56 3.78 5.34 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:6] "M1" "M2" "M3" "NC1" ...
##   .. ..$ : chr [1:6] "PC1" "PC2" "PC3" "PC4" ...
##  - attr(*, "class")= chr "prcomp"
```

The output of `prcomp()` contains five elements `sdev`, `rotation`, `center`, `scale` and `x`. Let us examine what each looks like.

```
pc_out$sdev
```

```
## [1] 5.576793e+00 1.764207e+00 1.582710e+00 7.576131e-01 6.668424e-01
## [6] 6.513478e-15
```

`sddev` gives standard deviation (used for computing variance explained). We will see how in sections below.

```
head(pc_out$rotation)
```

```
##          PC1          PC2          PC3          PC4           PC5            PC6
## 1 0.2175336 -0.09843120  0.24646637 -0.18360970 -0.004759212 -0.1282942645
## 2 0.1277130 -0.05971284 -0.05865888 -0.04302434  0.069398716  0.0190233560
## 3 0.1274181 -0.06356995  0.12354321  0.18188678 -0.083764114  0.0646766982
## 4 0.1113816  0.07285611 -0.05940135  0.14861908  0.016695673 -0.0008885424
## 5 0.1010094  0.24317614  0.03484815 -0.06467411  0.047649021 -0.0883151388
## 6 0.1125121 -0.05572681  0.08624863  0.17407046 -0.264955100  0.0373735665
```

After PCA, the observations are expressed in new axes and the loadings are provided in `pc_out$rotation`. Each column of `pc_out$rotation` contains the corresponding principal component loading vector.

We see that there are six distinct principal components, as indicated by column names of `pc_out$rotation`.

```
pc_out$center
```

```
##        1        2        3        4        5        6        7        8
## 5.566266 4.438431 3.780365 6.068163 5.755939 4.400684 5.160500 5.683933
##        9       10       11       12       13       14       15       16
## 5.021818 3.586855 7.078354 4.669260 5.831477 8.181517 4.614013 5.382728
##       17       18       19       20       21       22       23       24
## 7.529027 7.313685 4.547701 5.290136 5.187301 6.013839 3.985299 3.642647
##       25       26       27       28       29       30       31       32
## 4.154045 4.674278 8.505242 4.899224 3.882938 4.964233 5.587100 8.420071
##       33       34       35       36       37       38       39       40
## 4.668532 5.571026 5.341486 4.246074 4.603076 3.906143 5.290324 3.205139
##       41       42       43       44       45       46       47       48
```

```
## 5.501723 6.599761 7.957662 4.147284 2.674647 7.029644 4.297315 3.550112
##       49       50       51       52       53       54       55       56
## 4.440525 4.440604 5.233737 4.589023 6.559673 7.563419 5.203153 5.563289
##       57       58       59       60       61       62       63       64
## 4.172330 4.363869 5.816609 4.678070 4.538479 8.210925 5.701295 4.330346
##       65       66       67       68       69       70       71       72
## 5.057213 3.462456 6.015641 7.096558 5.494757 2.779979 5.011157 6.460525
##       73       74       75       76       77       78       79       80
## 5.065625 6.057910 5.358162 7.126652 3.547180 3.157770 5.342622 7.277169
##       81       82       83       84       85       86       87       88
## 6.229511 8.134572 3.942388 5.476810 6.147777 2.856113 6.580211 5.616440
##       89       90       91       92       93       94       95       96
## 4.708294 5.244003 2.387781 3.769301 4.025543 3.968695 3.848995 4.564350
##       97       98       99      100      101      102      103      104
## 4.546153 4.840263 4.257428 6.249407 3.007570 5.067437 4.271637 4.436552
##      105      106      107      108      109      110      111      112
## 4.231309 3.973251 6.599789 4.837524 4.322573 4.867455 5.161541 3.307676
##      113      114      115      116      117      118      119      120
## 6.143019 5.071244 6.177220 4.827735 7.376264 7.533733 4.374265 5.183771
##      121      122      123      124      125      126      127      128
## 5.625725 5.976763 8.844863 4.056325 5.954586 4.811972 4.748867 4.025010
##      129      130      131      132      133      134      135      136
## 5.284776 5.388265 5.895763 6.181208 6.034146 3.303646 3.136626 3.336961
##      137      138      139      140      141      142      143      144
## 4.592918 3.190322 3.742253 5.845806 5.574498 3.444501 5.058708 6.899120
##      145      146      147
## 5.868390 5.154152 5.004526
```

```
pc_out$scale
```

```
## [1] FALSE
```

The `center` and `scale` elements correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA.

```
# See the principal components
dim(pc_out$x)
```

```
## [1] 6 6
```

```
head(pc_out$x)
```

```
##           PC1         PC2        PC3        PC4        PC5           PC6
## M1  -5.091712  0.07170858  0.1415984 -1.2149422  0.5781693 6.039314e-15
## M2  -4.463273 -2.78279960 -1.0114746  0.5067161 -0.2847184 5.893565e-15
## M3  -5.560192  2.16495377  1.2469598  0.6895931 -0.3109790 5.855662e-15
## NC1  3.781742  1.47081980 -2.5212675  0.2858078  0.3841018 6.158759e-15
## NC2  5.338015  0.05493898  0.2748255 -0.6551843 -1.0483358 6.282292e-15
## NC3  5.995420 -0.97962153  1.8693583  0.3880095  0.6817622 5.522265e-15
```

Let's now see the summary of the analysis using the `summary()` function!

```
summary(pc_out)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5       PC6
## Standard deviation     5.5768 1.76421 1.58271 0.75761 0.66684 6.513e-15
## Proportion of Variance 0.8242 0.08248 0.06638 0.01521 0.01178 0.000e+00
## Cumulative Proportion  0.8242 0.90663 0.97301 0.98822 1.00000 1.000e+00
```

The first row gives the `Standard deviation` of each component, which is the same as the result of `pc_out$sdev`.

The second row, `Proportion of Variance`, shows the percentage of explained variance, also obtained as variance/sum(variance) where variance is the square of sdev.

Compute variance

```
pc_out$sdev^2 / sum(pc_out$sdev^2)
```

```
## [1] 8.241484e-01 8.247748e-02 6.638030e-02 1.521008e-02 1.178373e-02
## [6] 1.124249e-30
```

From the second row you can see that the first principal component explains over 82.4% of the total variance (Note: multiply each number by 100 to get the percentages).

The second principal component explains 8.2% of the variance, and the amount of variance explained reduces further down with each component.

Finally, the last row, `Cumulative Proportion`, calculates the cumulative sum of the second row.

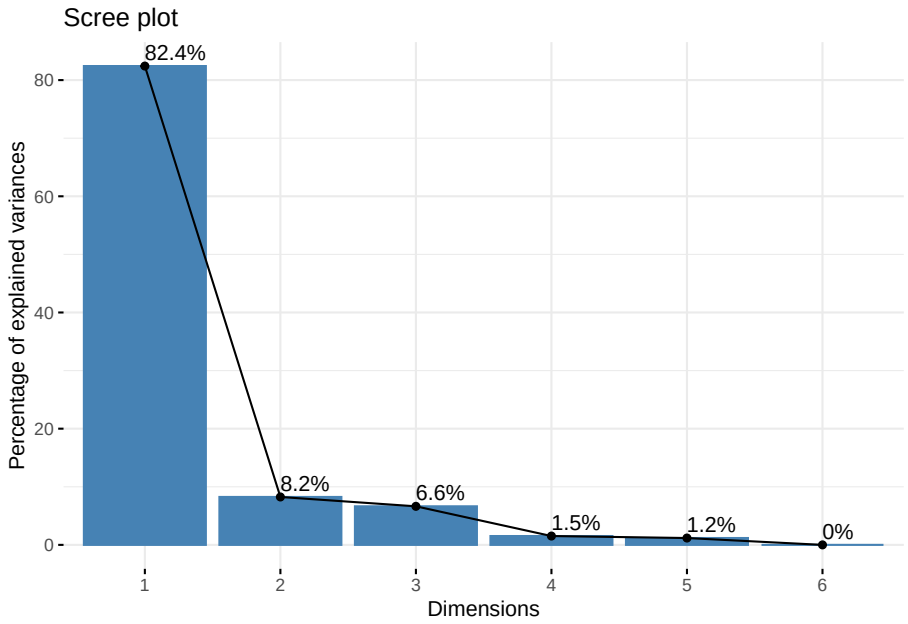
Now, let's have some fun with visualising the results of PCA.

5.2.1.3 Step 3. Visualisation of PCA Results

A. Scree Plot

We can visualize the percentage of explained variance per principal component by using what is called a scree plot. We will call the `fviz_eig()` function of the factoextra package for the application. You may need to install the package using `install.packages("factoextra")`.

```
library(factoextra)
fviz_eig(pc_out,
         addlabels = TRUE)
```



The x-axis shows the PCs and the y-axis shows the percentage of variance explained that we saw above. Percentages are listed on top of the bars. It's common to see that the first few principal components explain the major amount of variance.

Scree plot can also be used to decide the number of components to keep for rest of your analysis. One of the ways is using the elbow rule. This method is about looking for the "elbow" shape on the curve and retaining all components before the point where the curve flattens out. Here, the elbow appears to occur at the second principal component.

Note that we will NOT remove any components for the current analysis since our goal is to understand how PCA can be used to identify batch effect in the

data.

B. Scatter Plot

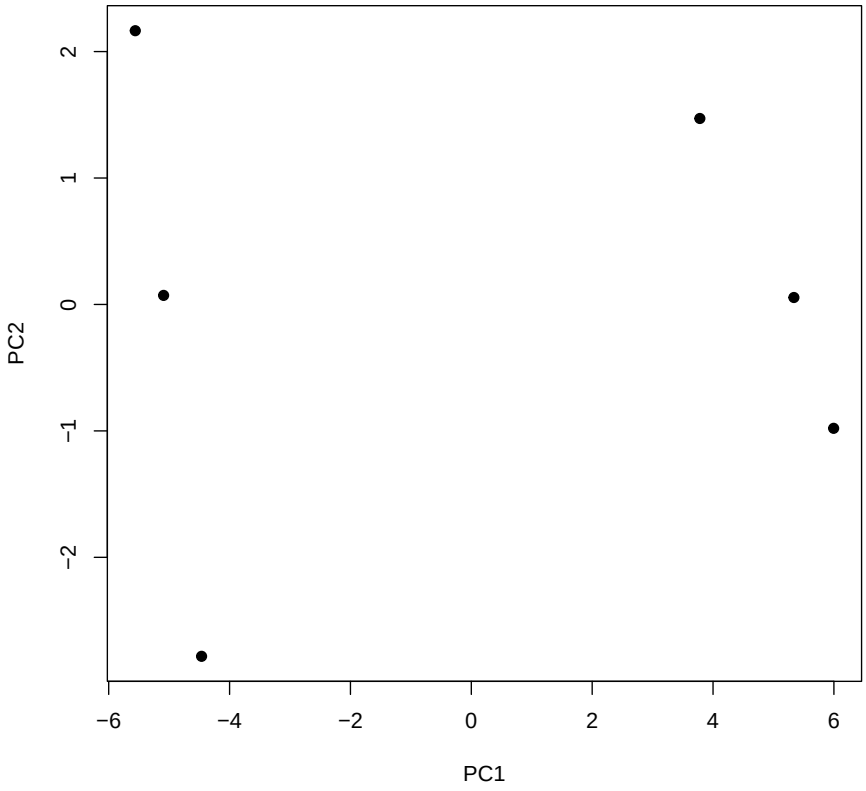
After a PCA, the observations are expressed in principal component scores (as we saw above in `pc_out$rotation`). So, it is important to visualize the observations along the new axes (principal components) how observations have been transformed and to understand the relations in the dataset.

This can be achieved by drawing a scatterplot. To do so, first, we need to extract and the principal component scores in `pc_out$rotation`, and then we will store them in a data frame called PC.

```
PC <- as.data.frame(pc_out$x)
```

Plot the first two principal components as follows:

```
plot(x = PC$PC1,
     y = PC$PC2,
     pch = 19,
     xlab="PC1",
     ylab="PC2")
```



We see six points in two different groups. The points correspond to six samples. But, we don't know what group/condition they belong to.

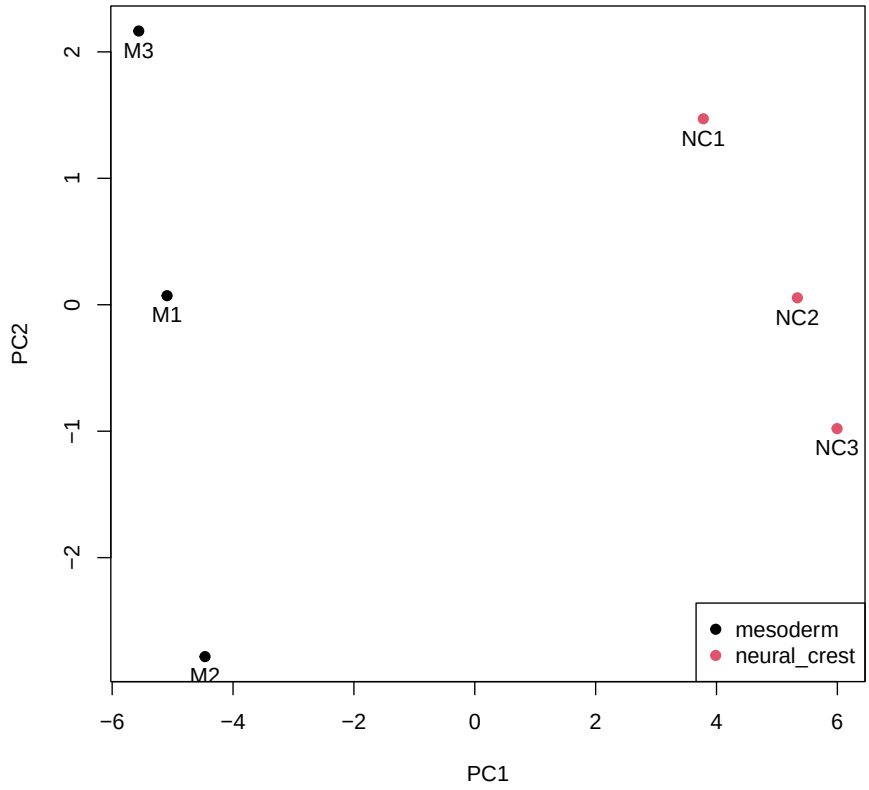
That can be done by adding sample-related information to the data.frame `PC` (such as cell type, treatment type, batch they were processed etc) as new variables. Here we will add the sample names and the cell types.

```
PC$sample <- factor(rownames(PC))
PC$cells <- factor(c(rep("mesoderm", 3), rep("neural_crest", 3)))
```

Plot the scatterplot again and now, colour the points by cell type. Then, add sample names and legend.

```
plot(x = PC$PC1,
     y = PC$PC2,
```

```
     col = PC$cells,
     pch = 19,
     xlab="PC1",
     ylab="PC2")

text(x= PC$PC1,
     y = PC$PC2-0.15,
     labels = PC$sample)

legend("bottomright",
       legend = levels(PC$cells),
       col = seq_along(levels(PC$cells)),
       pch = 19)
```

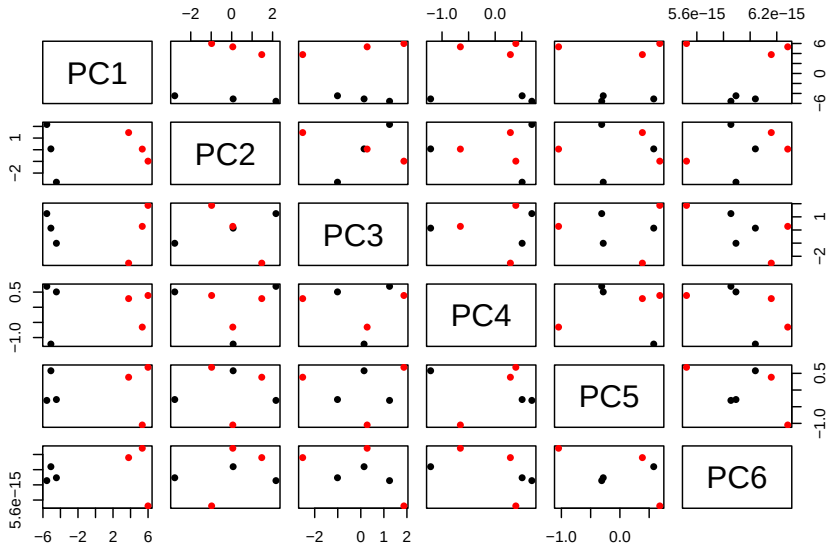


Samples from each cell type are closer together on the scatter plot. If the batch

information is available, it can also be used to colour the scatterplot. Ideally, samples from different conditions should cluster together, irrespective of the batch they were processed in.

You can also plot other PCs such as PC2 vs PC3 by changing the x and y variables above. Another way to plot all PCs is using `pairs()`

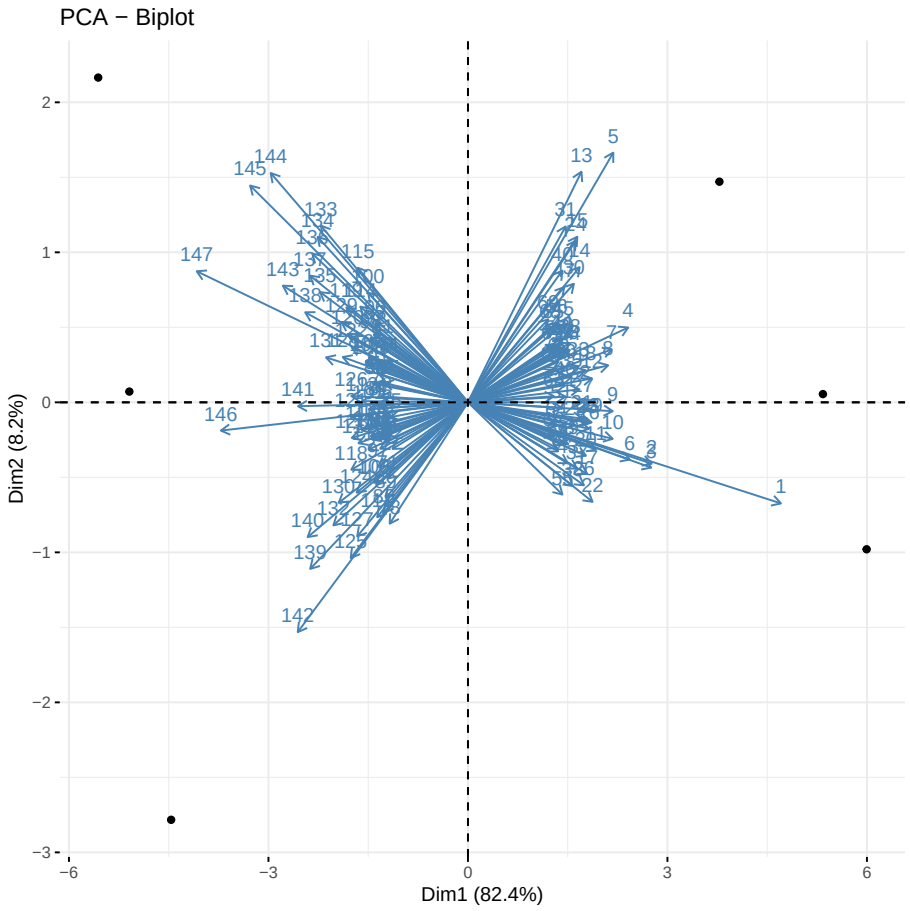
```
pairs(PC[,1:6],
      col=c("black","red")[PC$cells],
      pch=16)
```



C. Biplot

Another useful plots to understand the results are biplots. We will use the `fviz_pca_biplot()` function of the factoextra package. We will set label="var" argument to label the variables.

```
fviz_pca_biplot(pc_out, label = "var")
```



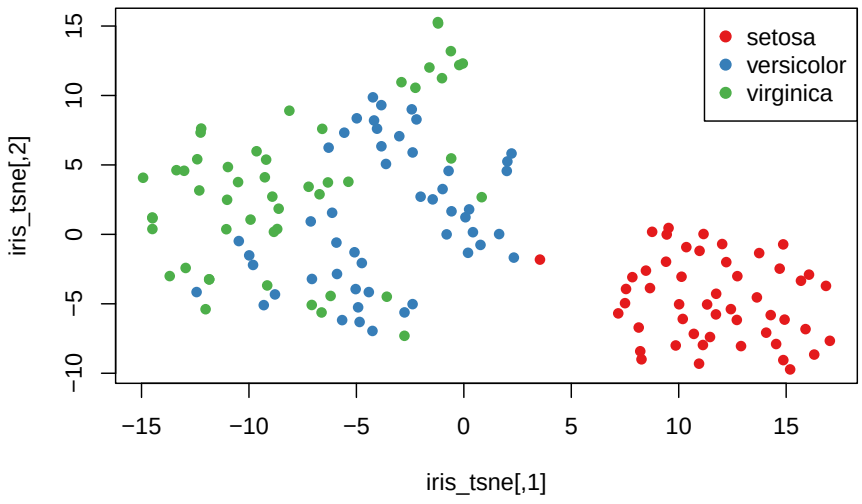
The axes show the principal component scores, and the vectors are the loading vectors, whose components are in the magnitudes of the loadings. Vectors indicate that samples from each cell type are closer together.

5.2.2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets.

There are several packages that have implemented t-SNE. Here we are going to use the package `tsne` and the function `tsne`. Let's run the t-SNE algorithm on the iris dataset and generate a t-SNE plot.

```r
library(tsne)
library(RColorBrewer)

### load the input data
data(iris)
iris_data <- iris[,-5]

# set colours of the plot
my_cols_vec <- brewer.pal("Set1",n = length(levels(iris$Species)))
species_cols <- my_cols_vec[factor(iris$Species)]

# run t-SNE
iris_tsne <- tsne(iris_data)

plot(iris_tsne,
     pch=16,
     col=species_cols)
legend("topright",
       legend = levels(iris$Species),
       col = my_cols_vec,
       pch = 19)
```

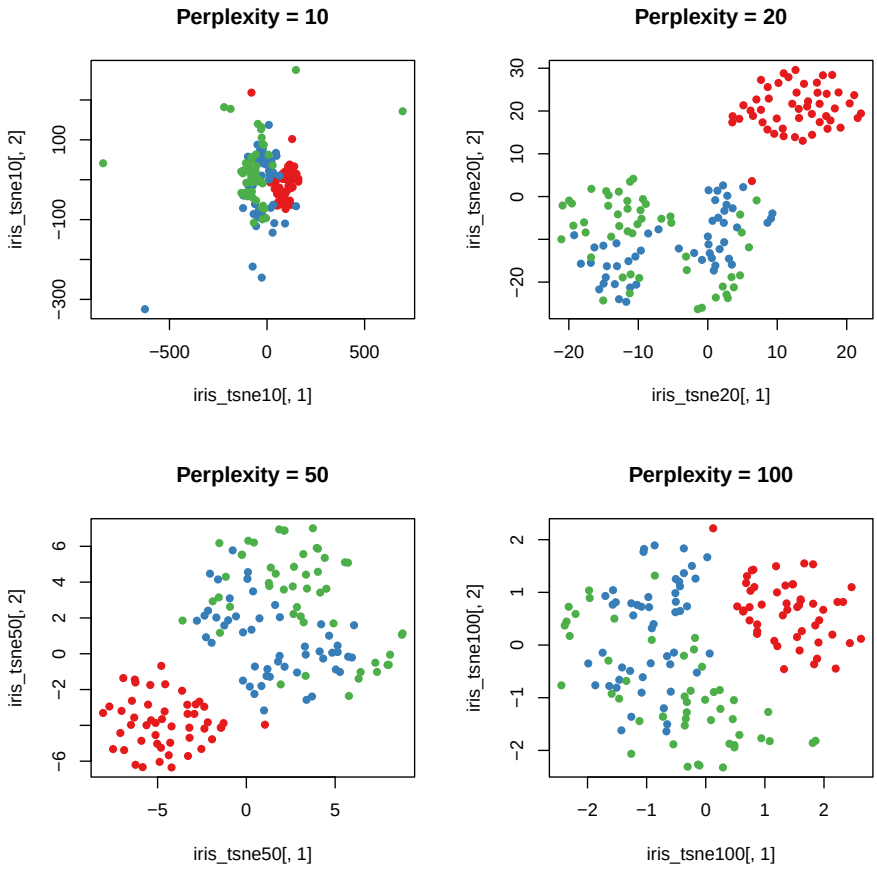


The `tsne` function has a parameter called `perplexity` which determines how to balance attention to neighborhood vs global structure. Default value is 30

which was used above. Set perplexity to 10, 20, 50, 100 and rerun tsne. Then visualise each result.

```
iris_tsne10 <- tsne(iris_data,perplexity = 10)
iris_tsne20 <- tsne(iris_data,perplexity = 20)
iris_tsne50 <- tsne(iris_data,perplexity = 50)
iris_tsne100 <- tsne(iris_data,perplexity = 100)
```

```
par(mfrow=c(2,2))

plot(iris_tsne10[,1],
     iris_tsne10[,2],
     main = "Perplexity = 10",
     col = species_cols,
     pch=16)

plot(iris_tsne20[,1],
     iris_tsne20[,2],
     main = "Perplexity = 20",
     col = species_cols,
     pch=16)

plot(iris_tsne50[,1],
     iris_tsne50[,2],
     main = "Perplexity = 50",
     col = species_cols,
     pch=16)

plot(iris_tsne100[,1],
     iris_tsne100[,2],
     main = "Perplexity = 100",
     col = species_cols,
     pch=16)
```



Higher perplexity leads to higher spread in your data.

5.2.3 Uniform Manifold Approximation and Projection (UMAP)

UMAP is another dimension reduction method and it uses similar neighborhood approach as t-SNE except uses Riemannian geometry.

Here we are going to use the package `umap` and the function `umap`. Let's apply UMAP on the iris dataset and generate a UMAP plot.

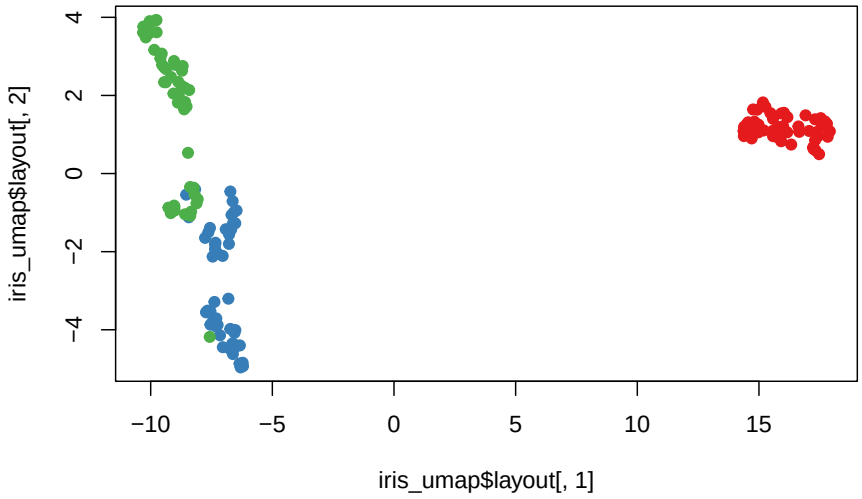
```
## umap
library(umap)
```

```
iris_umap <- umap(iris_data)
str(iris_umap)
```

```
## List of 4
##  $ layout: num [1:150, 1:2] 15.8 17.3 17.5 17.6 15.6 ...
##  $ data  : num [1:150, 1:4] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##  $ knn   :List of 2
##   ..$ indexes  : int [1:150, 1:15] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ distances: num [1:150, 1:15] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "class")= chr "umap.knn"
##  $ config:List of 24
##   ..$ n_neighbors         : int 15
##   ..$ n_components        : int 2
##   ..$ metric              : chr "euclidean"
##   ..$ n_epochs            : int 200
##   ..$ input               : chr "data"
##   ..$ init                : chr "spectral"
##   ..$ min_dist            : num 0.1
##   ..$ set_op_mix_ratio    : num 1
##   ..$ local_connectivity  : num 1
##   ..$ bandwidth           : num 1
##   ..$ alpha               : num 1
##   ..$ gamma               : num 1
##   ..$ negative_sample_rate: int 5
##   ..$ a                   : num 1.58
##   ..$ b                   : num 0.895
##   ..$ spread              : num 1
##   ..$ random_state        : int 981281951
##   ..$ transform_state     : int NA
##   ..$ knn                 : logi NA
##   ..$ knn_repeats         : num 1
##   ..$ verbose             : logi FALSE
##   ..$ umap_learn_args     : logi NA
##   ..$ method              : chr "naive"
##   ..$ metric.function     :function (m, origin, targets)
##   ..- attr(*, "class")= chr "umap.config"
##  - attr(*, "class")= chr "umap"
```

```
par(mfrow=c(1,1))
plot(iris_umap$layout[,1],
```

```
      iris_umap$layout[,2],
      col = species_cols, pch = 19)
```




```
# legend("topright",
#          legend = levels(mouse$FC),
#          col = species_cols,
#          pch = 19)
```

5.2.4 Bonus Exercise

For your exercise, try the following:

- Return to your crabs data
- Compute the principle components (PCs) for the numeric columns
- Plot these PCs and color them by species ("sp") and sex
- Now compute 2 t-SNE components for these data and color by species and sex
- Finally compute 2 UMAP components for these data and color by species and sex
- Do any of these dimensionality reduction methods seem to segregate sex/species groups?

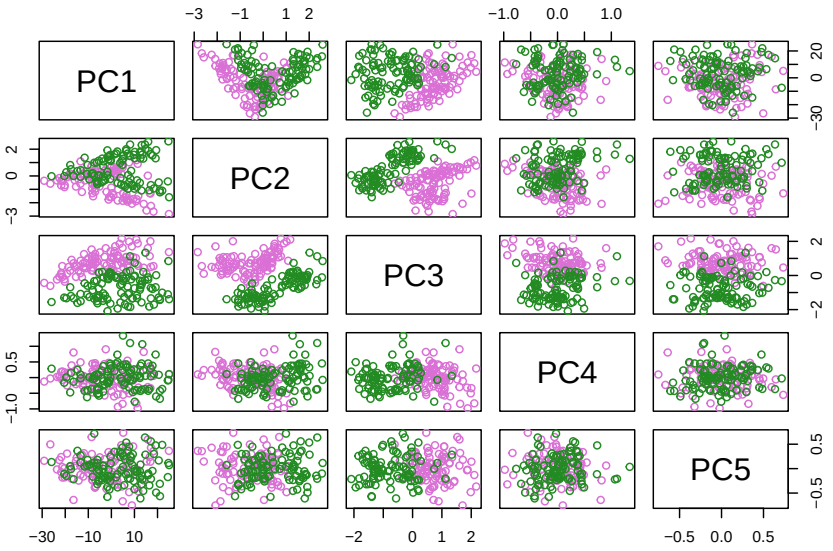
5.2.5 Bonus Exercise Results

PCA

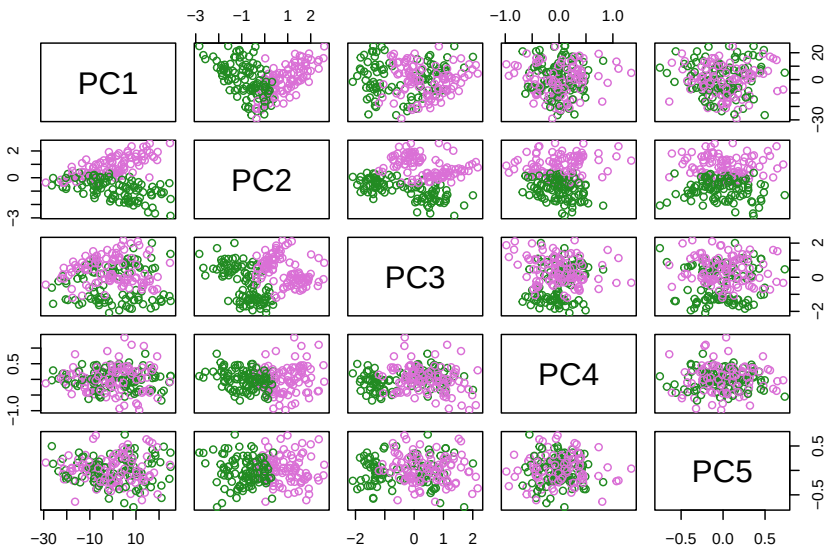
```
c_pcs = prcomp(crabs_meas)
```

Plot PC projections (embeddings).

```
pairs(c_pcs$x, col = c("orchid","forestgreen")[factor(crabs$sp)])
```




```
pairs(c_pcs$x, col = c("orchid","forestgreen")[factor(crabs$sex)])
```



tSNE:

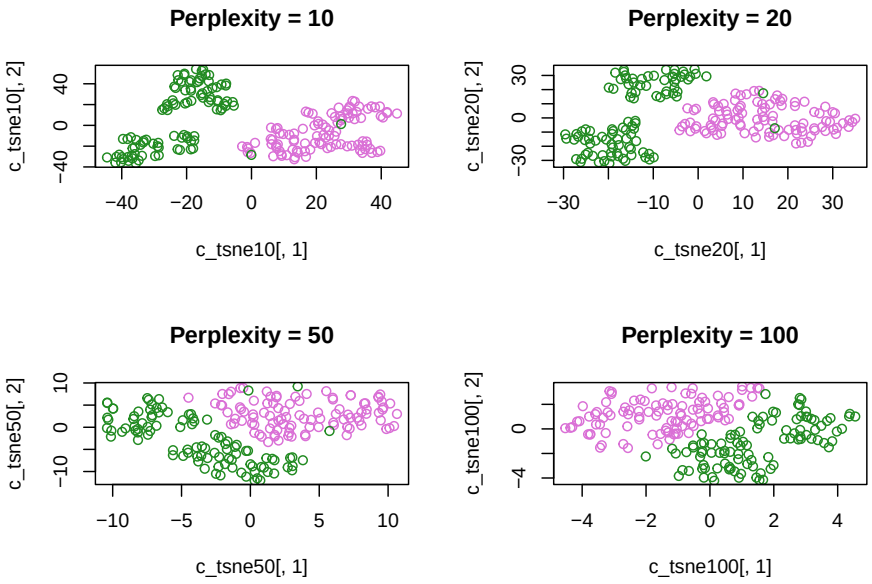
```
library(tsne)
c_tsne10 = tsne(crabs_meas,perplexity = 10)
c_tsne20 = tsne(crabs_meas,perplexity = 20)
c_tsne50 = tsne(crabs_meas,perplexity = 50)
c_tsne100 = tsne(crabs_meas,perplexity = 100)
```

sex_cols = c("orchid","forestgreen")[factor(crabs$sex)]

Color-code tSNE plot by species, try various perplexity levels:

```
species_cols = c("orchid","forestgreen")[factor(crabs$sp)]
par(mfrow=c(2,2))
plot(c_tsne10[,1],
     c_tsne10[,2],
     main = "Perplexity = 10",
     col = species_cols)

plot(c_tsne20[,1],
     c_tsne20[,2],
     main = "Perplexity = 20",
     col = species_cols)
```

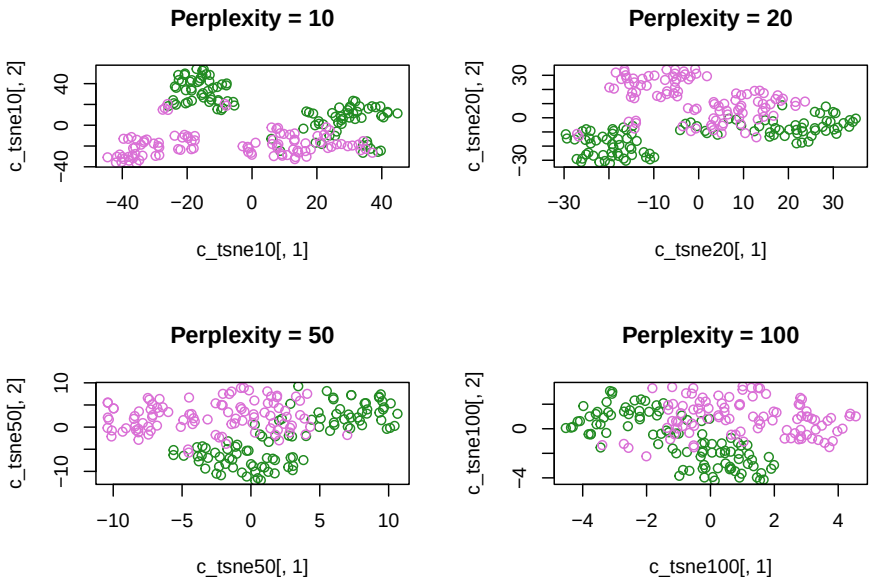
```
plot(c_tsne50[,1],
     c_tsne50[,2],
     main = "Perplexity = 50",
     col = species_cols)
plot(c_tsne100[,1],
     c_tsne100[,2],
     main = "Perplexity = 100",
     col = species_cols)
```



Now do the same, but colour-code for sex:

```
sex_cols = c("orchid","forestgreen")[factor(crabs$sex)]
par(mfrow=c(2,2))
plot(c_tsne10[,1],
     c_tsne10[,2],
     main = "Perplexity = 10",
     col = sex_cols)
plot(c_tsne20[,1],
     c_tsne20[,2],
     main = "Perplexity = 20",
     col = sex_cols)
plot(c_tsne50[,1],
     c_tsne50[,2],
```

```
     main = "Perplexity = 50",
     col = sex_cols)
plot(c_tsne100[,1],
     c_tsne100[,2],
     main = "Perplexity = 100",
     col = sex_cols)
```



Run UMAP

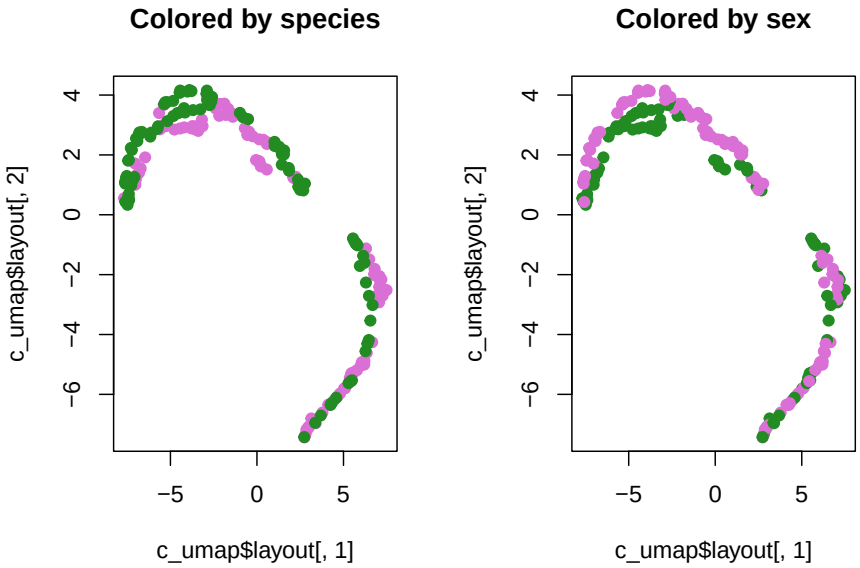
```
library(umap)
c_umap <- umap(crabs_meas)
str(c_umap)
```

```
## List of 4
##  $ layout: num [1:200, 1:2] 2.8 2.88 2.98 3.14 3.4 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:200] "1" "2" "3" "4" ...
##   .. ..$ : NULL
##  $ data  : num [1:200, 1:5] 8.1 8.8 9.2 9.6 9.8 10.8 11.1 11.6 11.8 11.8 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:200] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:5] "FL" "RW" "CL" "CW" ...
```

```
##  $ knn   :List of 2
##   ..$ indexes  : int [1:200, 1:15] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:200] "1" "2" "3" "4" ...
##   .. .. ..$ : NULL
##   ..$ distances: num [1:200, 1:15] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:200] "1" "2" "3" "4" ...
##   .. .. ..$ : NULL
##   ..- attr(*, "class")= chr "umap.knn"
##  $ config:List of 24
##   ..$ n_neighbors         : int 15
##   ..$ n_components        : int 2
##   ..$ metric              : chr "euclidean"
##   ..$ n_epochs            : int 200
##   ..$ input               : chr "data"
##   ..$ init                : chr "spectral"
##   ..$ min_dist            : num 0.1
##   ..$ set_op_mix_ratio    : num 1
##   ..$ local_connectivity  : num 1
##   ..$ bandwidth           : num 1
##   ..$ alpha               : num 1
##   ..$ gamma               : num 1
##   ..$ negative_sample_rate: int 5
##   ..$ a                   : num 1.58
##   ..$ b                   : num 0.895
##   ..$ spread              : num 1
##   ..$ random_state        : int 686219013
##   ..$ transform_state     : int NA
##   ..$ knn                 : logi NA
##   ..$ knn_repeats         : num 1
##   ..$ verbose             : logi FALSE
##   ..$ umap_learn_args     : logi NA
##   ..$ method              : chr "naive"
##   ..$ metric.function     :function (m, origin, targets)
##   ..- attr(*, "class")= chr "umap.config"
##  - attr(*, "class")= chr "umap"
```

```
par(mfrow=c(1,2))
plot(c_umap$layout[,1],
     c_umap$layout[,2],
     col = species_cols, pch = 19,
     main = "Colored by species")

plot(c_umap$layout[,1],
```

```
    c_umap$layout[,2],
    col = sex_cols, pch = 19,
    main = "Colored by sex")
```



Chapter 6

Module 3: Generalized Linear Models

6.1 Lecture

Canadian Bioinformatics Workshops

www.bioinformatics.ca

bioinformaticsdotca.github.io

6.2 Lab

In this module we're going to cover:

- Reading data from files and evaluating missingness

- Creating publication-quality plots with `ggplot2`
- Fitting models with binary outcomes using generalized linear models

6.2.1 Essential R: Reading Tables From Files, Merging, Basic Data Exploration

For this we're going to use two data files available in the course data directory.

Download the following datasets and put them in your R working directory.

- `tomerge1.csv` : comma-separated values
- `tomerge2.txt` : space-delimited values

Use `read.delim()` to read in tables. Note the use of the `sep=` parameter to indicate what the column separator is:

```
x1 <- read.delim("datasets/tomerge1.csv",sep=",")
head(x1)
```

```
##   Sample_ID Exposure Biomarker_value
## 1         A        0              35
## 2         B        0              22
## 3         C        0              91
## 4         D        0               3
## 5         E        1              56
## 6         F        1              37
```

```
x2 <- read.delim("datasets/tomerge2.txt",sep=" ")
head(x2)
```

```
##   sampleID Exposure_level Biomarker2_detected
## 1        A     0.65405517                   1
## 2        B     0.67202852                   0
## 3        C     0.88646372                   1
## 4        D     0.28433256                   0
## 5        E     0.04166839                   0
## 6        F     0.45263534                   1
```

Use `merge()` to combine the two tables by sample ID. Note the use of `by.x` and `by.y` to tell `merge()` which columns are equivalent:

```
x_merge <- merge(x1, x2, by.x = "Sample_ID", by.y = "sampleID")
head(x_merge)
```

```
##   Sample_ID Exposure Biomarker_value Exposure_level Biomarker2_detected
## 1         A        0              35     0.65405517                   1
## 2         B        0              22     0.67202852                   0
## 3         C        0              91     0.88646372                   1
## 4         D        0               3     0.28433256                   0
## 5         E        1              56     0.04166839                   0
## 6         F        1              37     0.45263534                   1
```

We're going to use a popular dataset for data analysis, pertaining to survival of passengers aboard the Titanic.

Download the following dataset and put them in your R working directory.

- `titanic.csv`

Let's read in the data from a file:

```
dat <- read.delim(
    "datasets/titanic.csv",
    sep="," # indicate the column separator
    )
```

Examine the columns:

```
head(dat)
```

```
##   PassengerId Survived Pclass
## 1           1        0      3
## 2           2        1      1
## 3           3        1      3
## 4           4        1      1
## 5           5        0      3
## 6           6        0      3
##                                                  Name    Sex Age SibSp Parch
## 1                             Braund, Mr. Owen Harris   male  22     1     0
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) female  38     1     0
## 3                              Heikkinen, Miss. Laina female  26     0     0
## 4        Futrelle, Mrs. Jacques Heath (Lily May Peel) female  35     1     0
```

```
## 5                            Allen, Mr. William Henry   male  35     0     0
## 6                                    Moran, Mr. James   male  NA     0     0
##             Ticket    Fare Cabin Embarked
## 1        A/5 21171  7.2500              S
## 2         PC 17599 71.2833   C85        C
## 3 STON/O2. 3101282  7.9250              S
## 4           113803 53.1000  C123        S
## 5           373450  8.0500              S
## 6           330877  8.4583              Q
```

Some of the columns are categorical, use `table()` to look at the tallies. Examine the columns:

```
table(dat$Survived)
```

```
##
##   0   1
## 549 342
```

```
table(dat$Pclass)
```

```
##
##   1   2   3
## 216 184 491
```

Use `summary()` to look at continuous-valued data:

```
summary(dat$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.42   20.12   28.00   29.70   38.00   80.00     177
```

Notice that there are 177 `NA` (missing) values for age. Let's visualize the missing data more systematically.

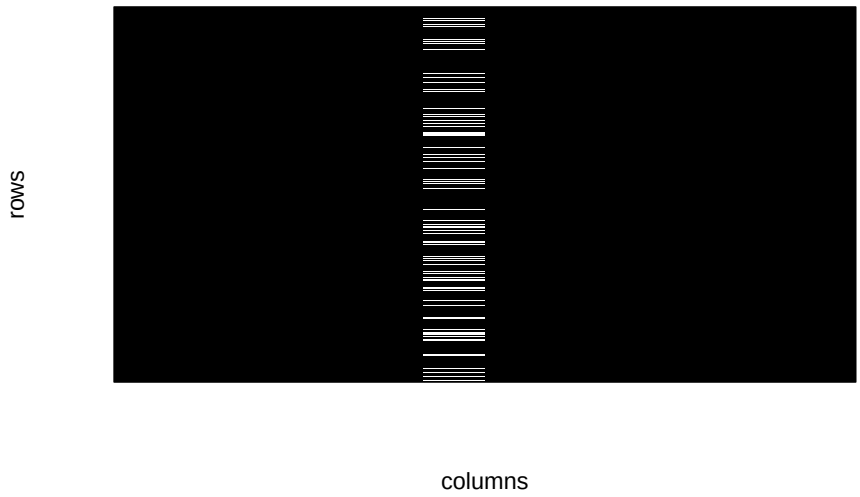
6.2.2 Explore Missing Data

For this let's use a little script that converts a table into black and white squares to visualize missing data. For this, install the `plotrix` package.

```
if (!requireNamespace("plotrix", quietly = TRUE)) install.packages("plotrix")

suppressMessages(library(plotrix))

#' show data missingness as a chequered matrix
#'
#' @param x (matrix) data matrix.
#' @param outFile (char) path to file for printing graph
#' @param wd (numeric) width in inches
#' @param ht (numeric) height in inches
#' @return plots missingness matrix to file
#' @import plotrix
#' @export
plotMissMat <- function(x,xlab="columns",
        ylab="rows",border=NA) {

    x <- !is.na(x)
    class(x) <- "numeric"
    color2D.matplot(x,show.values=FALSE,axes=FALSE,
        cs1=c(1,0),cs2=c(1,0),cs3=c(1,0),border=border,
        cex=0.8,
        xlab=xlab,ylab=ylab)
}
```

Let's look at the missingness in the Titanic dataset. Missing data is shown as a white cell, and non-missing data is shown in black.

```
plotMissMat(dat)
```



We can see a column with many missing values. This is probably the "age" data. Let's count the number of missing values on a per-column level.

For this we combine `is.na()`, which returns a TRUE/FALSE value for NA values, and `colSums()` which adds up the TRUE values down the columns.

```
colSums(is.na(dat))
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0           0           0           0           0         177
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##           0           0           0           0           0           0
```

This confirms that `Age` is the only column with missing data.

Now let's explore the data using plots.

6.2.3 Essential R: Plots with `ggplot2`

`ggplot2` is a popular plotting package that uses an additive paradigm to build plots.

Useful websites: * ggplot2 cheatsheet * The ggplot2 website is a wealth of reference information, so here we just touch on the basics to get you started.

Anytime you need to generate a specific kind of plot, the website will most likely have documentation for how to achieve it.

Let's start by creating a scatterplot with two **continuous variables**. For this let's load a dataset measuring statistics around quality of life in US states in the late 70's:

```
state.x77 <- as.data.frame(state.x77)
head(state.x77)
```

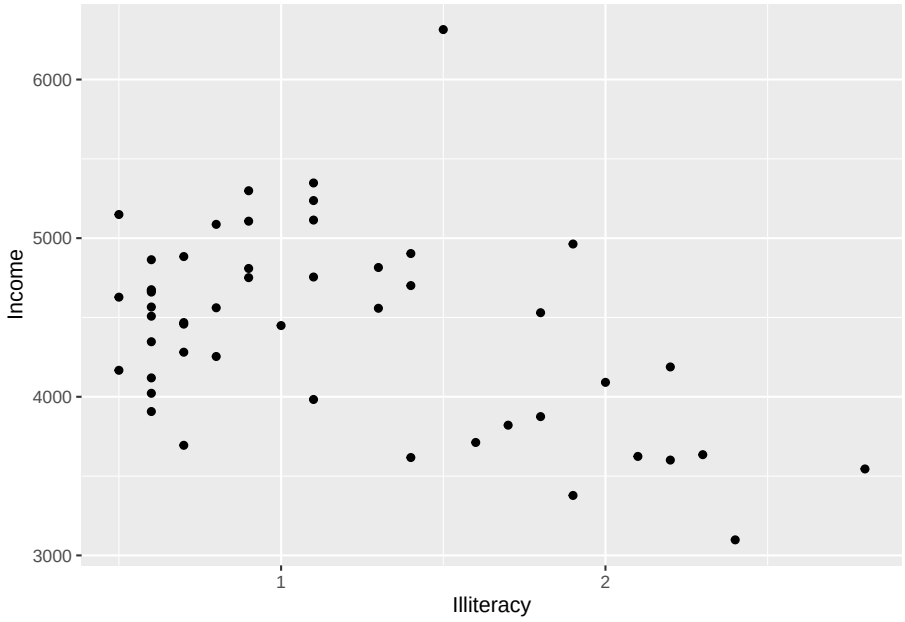
```
##            Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166 103766
```

Create a base plot using the `ggplot` function. Then "add" a scatterplot to it. Notice that the plot has been assigned to a variable named `p`.

This setup is standard for `ggplot2` and allows multiple visualizations to be applied to the same base plot.

We use `aes` to tell ggplot what the `x` and `y` axes are, and later, if we want to colour-code by a particular column.

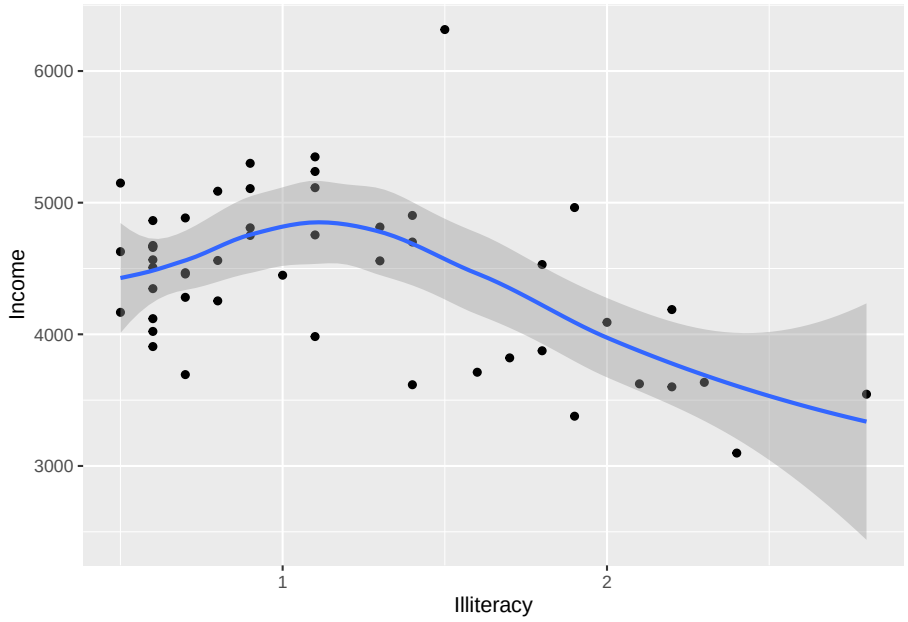
```
library(ggplot2)
p <- ggplot(state.x77,
    aes(x = Illiteracy,y = Income)
    )

p <- p + geom_point() # scatter plot
p
```



Now let's add confidence intervals:

```
p + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



It looks like there is a negative relationship between illiteracy and income. We can confirm this by looking at correlation:

```
x <- state.x77$Illiteracy
y <- state.x77$Income
cor.test(x,y)
```

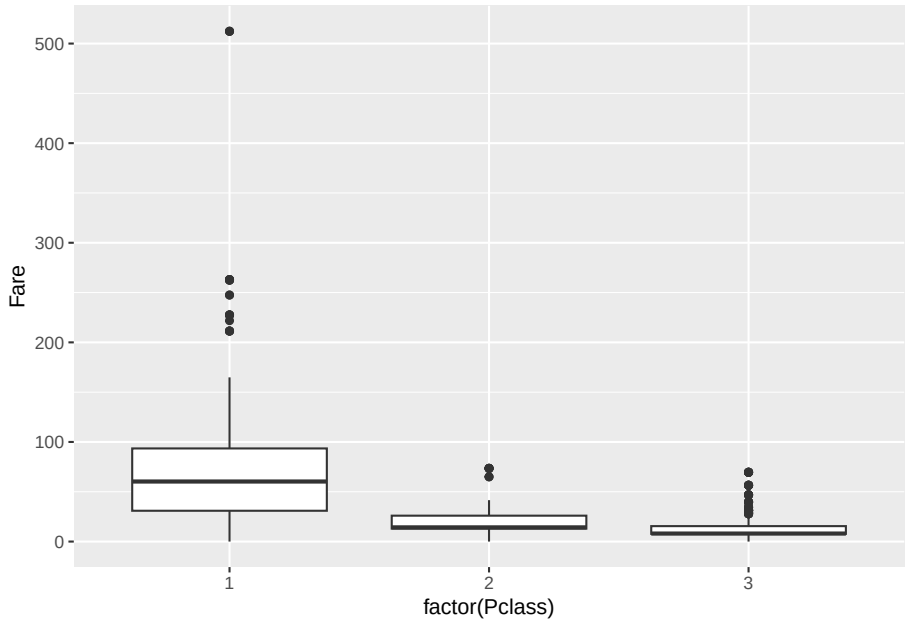
```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = -3.3668, df = 48, p-value = 0.001505
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.6378257 -0.1807128
## sample estimates:
##        cor
## -0.4370752
```

```
cor.test(x,y)$p.value
```

```
## [1] 0.001505073
```

Let's now examine **categorical variables**. In the titanic set, let's look at the fare paid based on passenger class:

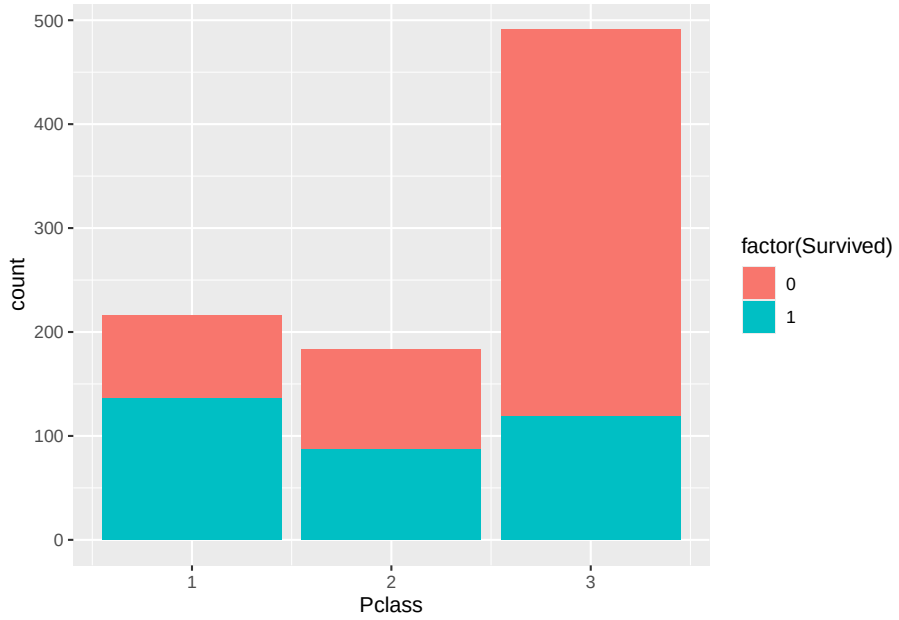
```
p <- ggplot(dat)
p + geom_boxplot(
    aes(x = factor(Pclass), # "factor()" makes a data column a categorical variable
        y = Fare))
```



We can use barplots to examine counts and proportions. Let's look at number of survivors, split by passenger class.

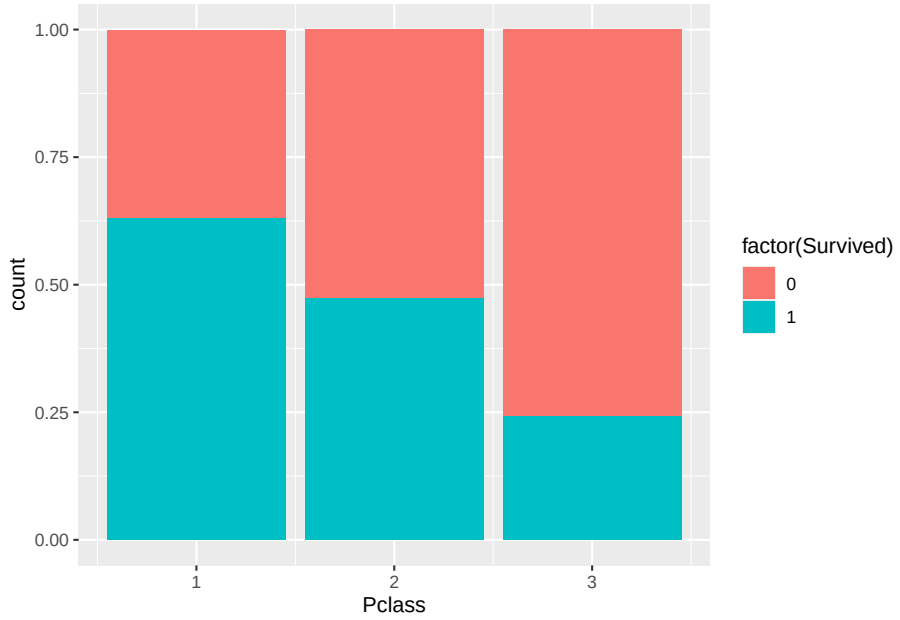
Here the `fill` command is used to to split each barplot by the category, "Survived". So you can see number of passengers by "Pclass" split by survival.

```
p + geom_bar(
    aes(fill=factor(Survived),
        x = Pclass)
)
```



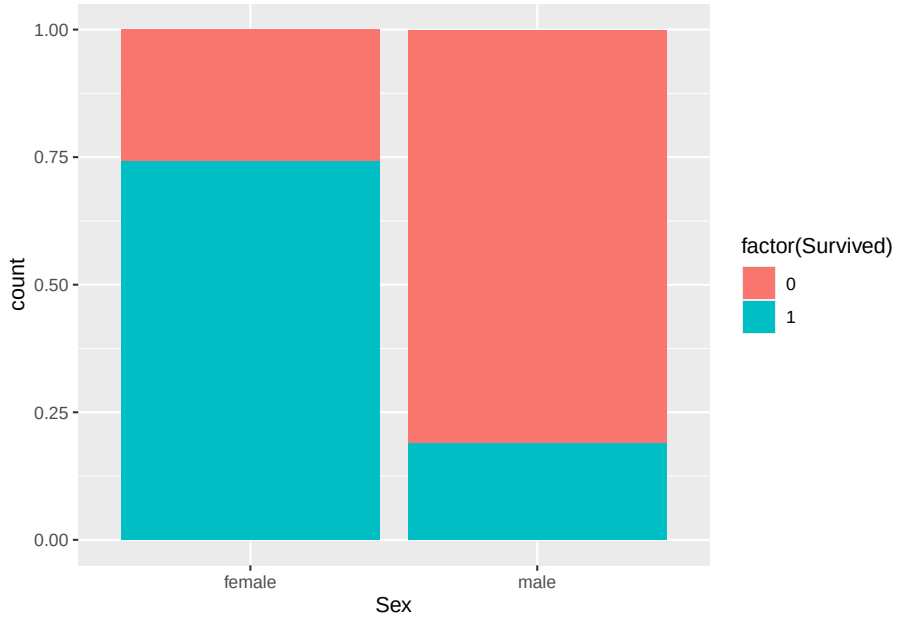
The plot above shows count data. Let's convert this to proportions. We can see that the fraction of non-survivors in "Class 3" is high.

```
p + geom_bar(
    aes(fill=factor(Survived), x = Pclass),
    position = "fill"
)
```



How about males versus females?

```
p + geom_bar(
    aes(fill=factor(Survived), x = Sex),
    position = "fill"
)
```



Exercise for later: Try other `ggplot` functions on these data or on a small data table of your project. (avoid using large genomics datasets, because those are going to be harder to interpret)

6.2.4 Fit Binary Response Variable Using `glm()` and Logistic Regression

Let's fit a model to a binary outcome. For this we load a dataset that measures physiological variables in a cohort of Pima Indians.

```
library(mlbench)
data(PimaIndiansDiabetes2)
# type ?PimaIndiansDiabetes2 to learn more about the dataset.

dat <- PimaIndiansDiabetes2
head(dat)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35      NA 33.6    0.627  50      pos
## 2        1      85       66      29      NA 26.6    0.351  31      neg
## 3        8     183       64      NA      NA 23.3    0.672  32      pos
## 4        1      89       66      23      94 28.1    0.167  21      neg
## 5        0     137       40      35     168 43.1    2.288  33      pos
```

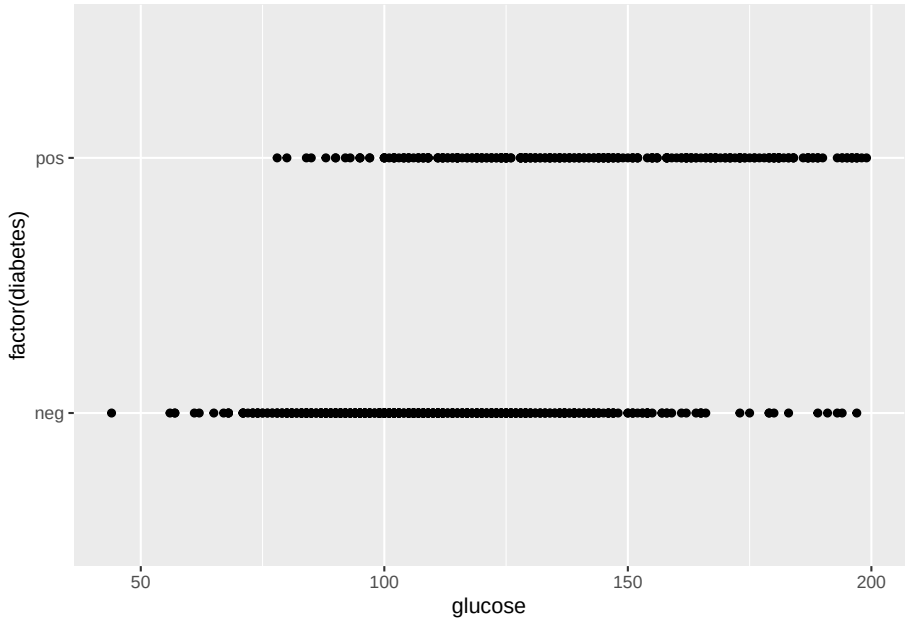
```
## 6           5     116        74      NA       NA 25.6    0.201  30       neg
```

Let's look at the impact of blood glucose levels on diabetes diagnosis.

First let's make a scatterplot. Could there be a relationship?

```
p <- ggplot(dat, aes(x = glucose, y = factor(diabetes)))
p + geom_point()
```

```
## Warning: Removed 5 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

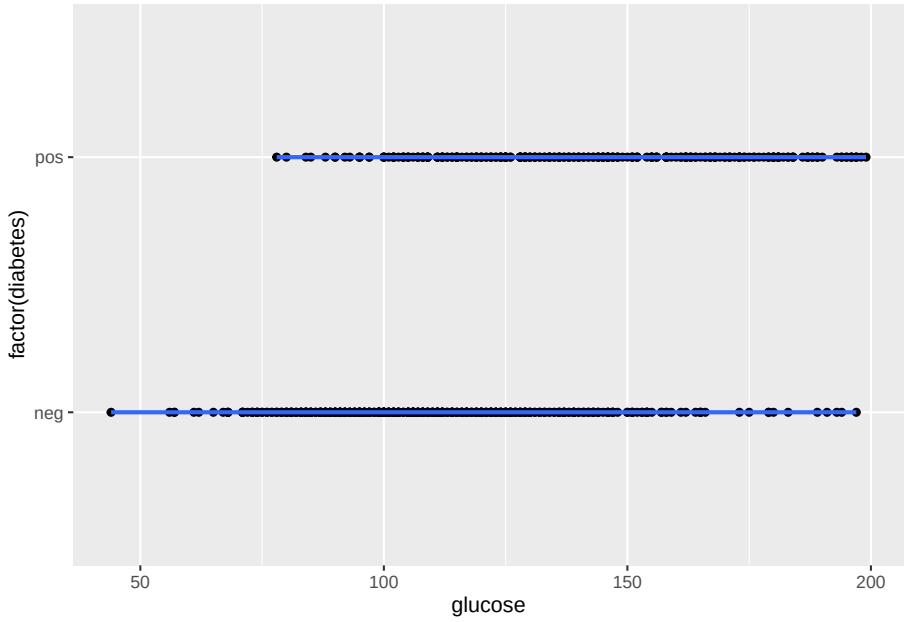


Could this be fit with a linear model?

Is there a continuous line that could reasonably fit the data?

```
p <- ggplot(dat, aes(x = glucose, y = factor(diabetes)))
p + geom_point() + geom_smooth()
```

```
## Warning: Removed 5 rows containing non-finite outside the scale range
## (`stat_smooth()`).

## Warning: Removed 5 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



This is a situation where a logistic regression model would be an appropriate choice because of the binary outcome.

We're going to use `glm()` to fit a model to these data:

```
mod <- glm(factor(diabetes)~glucose,
    dat,
    family = "binomial" # set to model binary outcome
)
summary(mod)
```

```
##
## Call:
## glm(formula = factor(diabetes) ~ glucose, family = "binomial",
##     data = dat)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.715088   0.438100  -13.04   <2e-16 ***
## glucose      0.040634   0.003382   12.01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```