# Solution: Statistics, dimensionality reduction, and clustering

Author name goes here

2023-08-31

## Demo

### The standard normal distribution

**pnorm()** The quantiles represent a set of possible values in the distribution.

```
data_quantiles <- seq(-5, 5, 0.1)
head(data_quantiles)
```

```
## [1] -5.0 -4.9 -4.8 -4.7 -4.6 -4.5
```

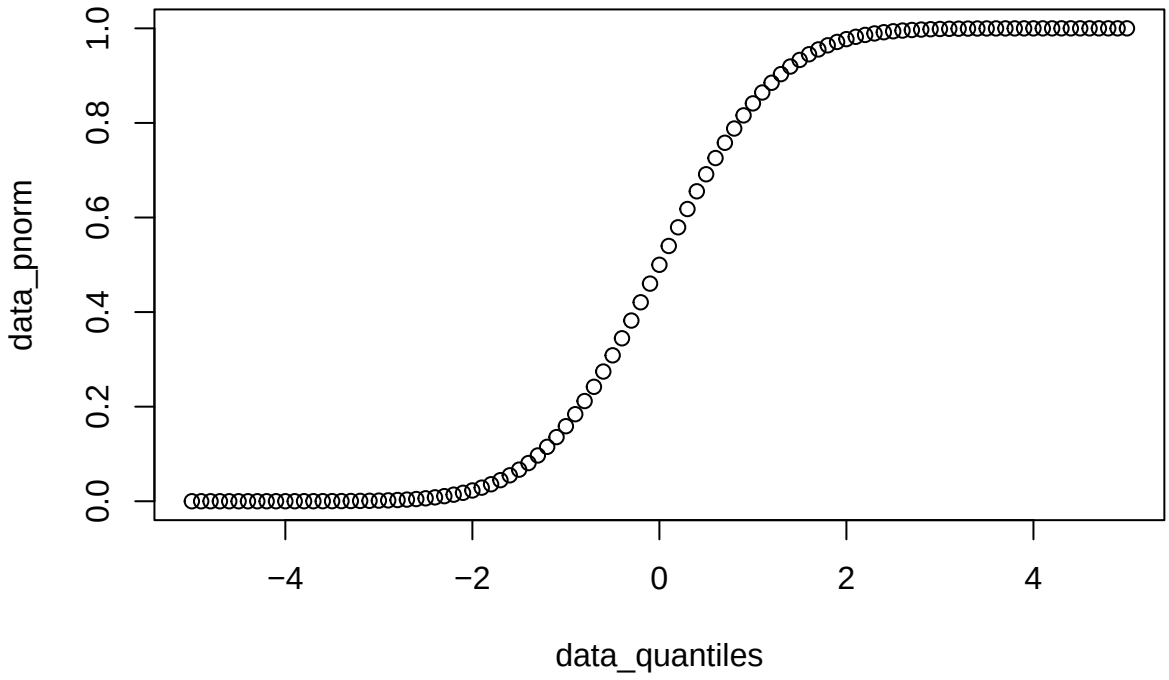
The function `pnorm()` returns the probability of a observing a value less or equal to each quantile given.

```
data_pnorm <- pnorm(q = data_quantiles)
head(data_pnorm)
```

```
## [1] 2.866516e-07 4.791833e-07 7.933282e-07 1.300807e-06 2.112455e-06
## [6] 3.397673e-06
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_quantiles, data_pnorm)
```




**qnorm()** The function `qnorm()` takes as input a vector of probabilities between 0 and 1.

```
data_probabilities <- seq(from = 0, to = 1, by = 0.02)
head(data_probabilities)
```

```
## [1] 0.00 0.02 0.04 0.06 0.08 0.10
```

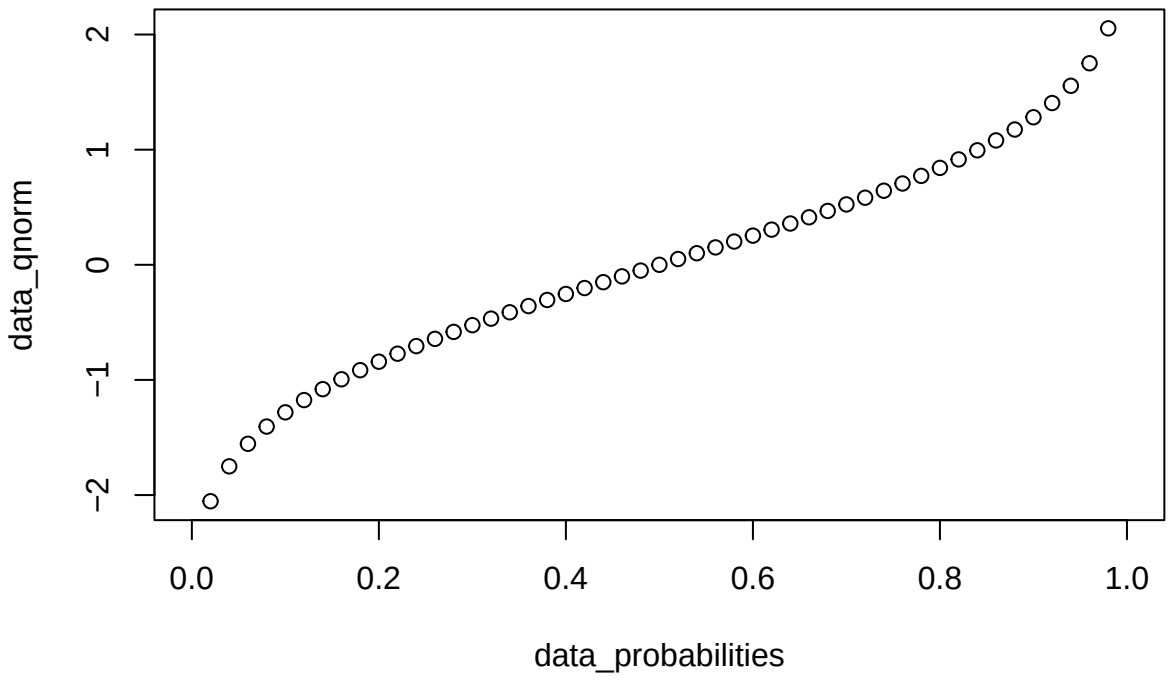
The function `qnorm()` the quantiles (i.e., values) that correspond to those probabilities.

```
data_qnorm <- qnorm(p = data_probabilities)
head(data_qnorm)
```

```
## [1]      -Inf -2.053749 -1.750686 -1.554774 -1.405072 -1.281552
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_probabilities, data_qnorm)
```




**dnorm** The quantiles represent a set of possible values in the distribution.

```
data_quantiles <- seq(-5, 5, 0.2)
head(data_quantiles)
```

```
## [1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0
```

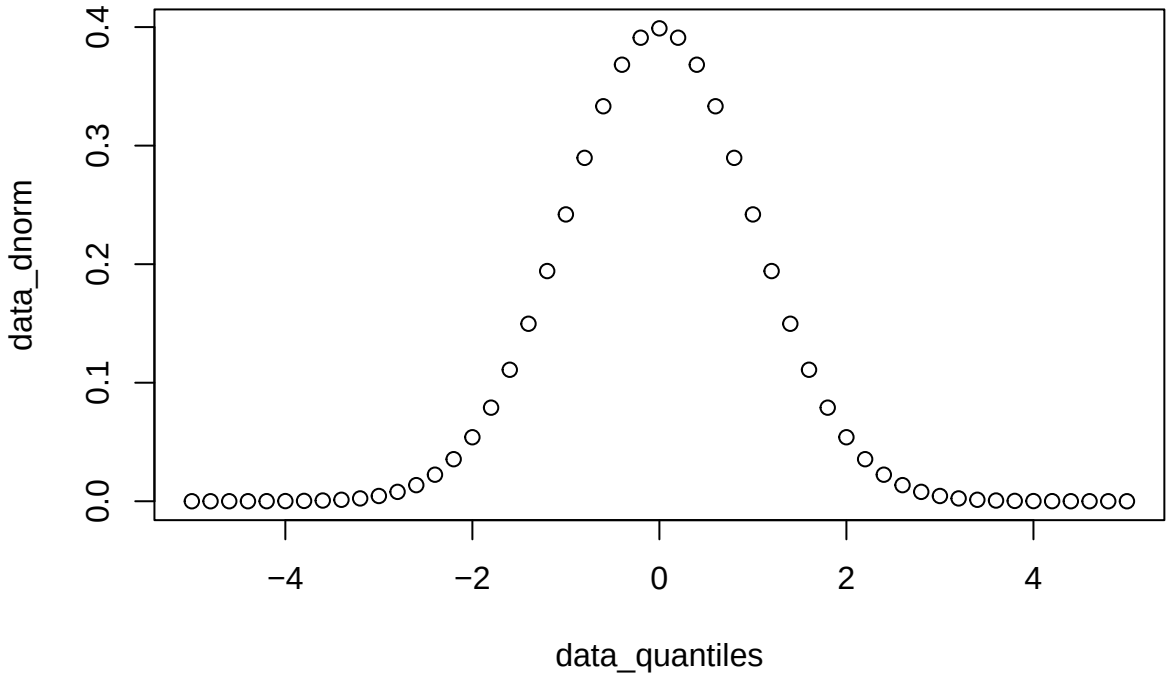
The function `pnorm()` returns the probability of a observing a value less or equal to each quantile given.

```
data_dnorm <- dnorm(x = data_quantiles)
head(data_dnorm)
```

```
## [1] 1.486720e-06 3.961299e-06 1.014085e-05 2.494247e-05 5.894307e-05
## [6] 1.338302e-04
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_quantiles, data_dnorm)
```

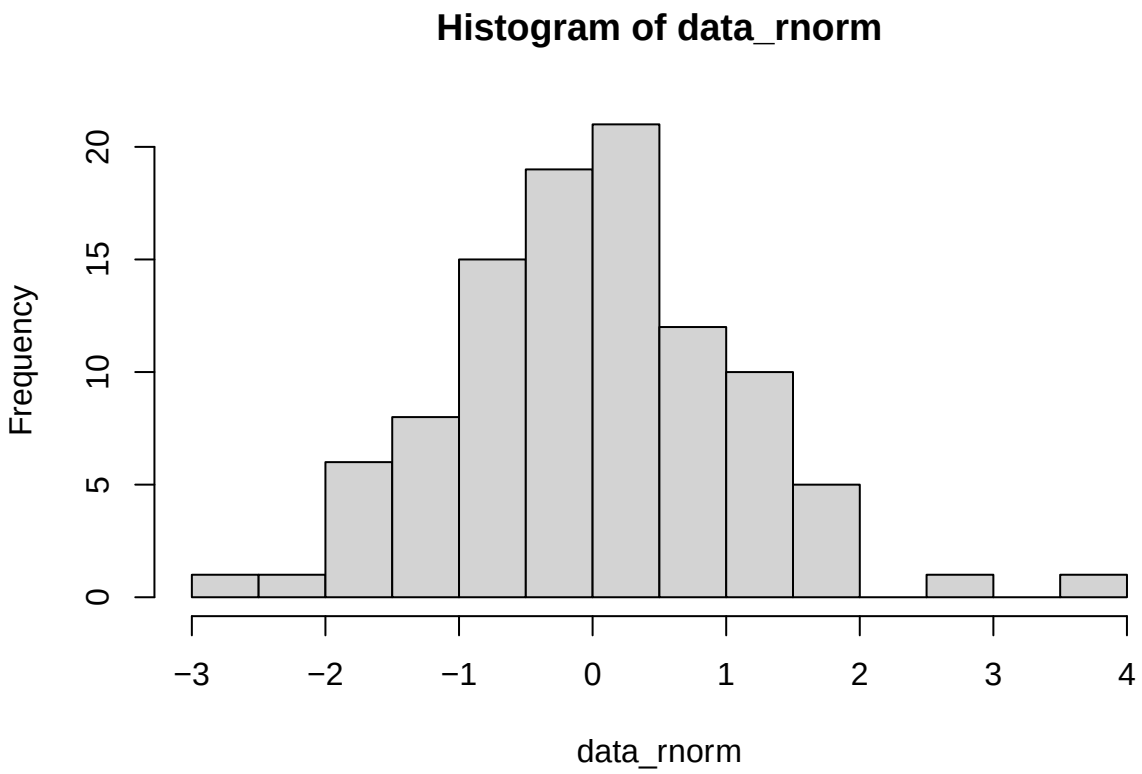



```
data_rnorm <- rnorm(n = 100)
head(data_rnorm)
```

**rnorm**

```
## [1] -0.397155166 -0.510064077  0.005875185  1.595404315  0.696670917
## [6] -1.214200932
```

```
hist(data_rnorm, breaks = 20)
```




## Demo

### A parameterised normal distribution

**pnorm()** The quantiles represent a set of possible values in the distribution.

```
data_quantiles <- seq(-500, 500, 50)
head(data_quantiles)
```

```
## [1] -500 -450 -400 -350 -300 -250
```

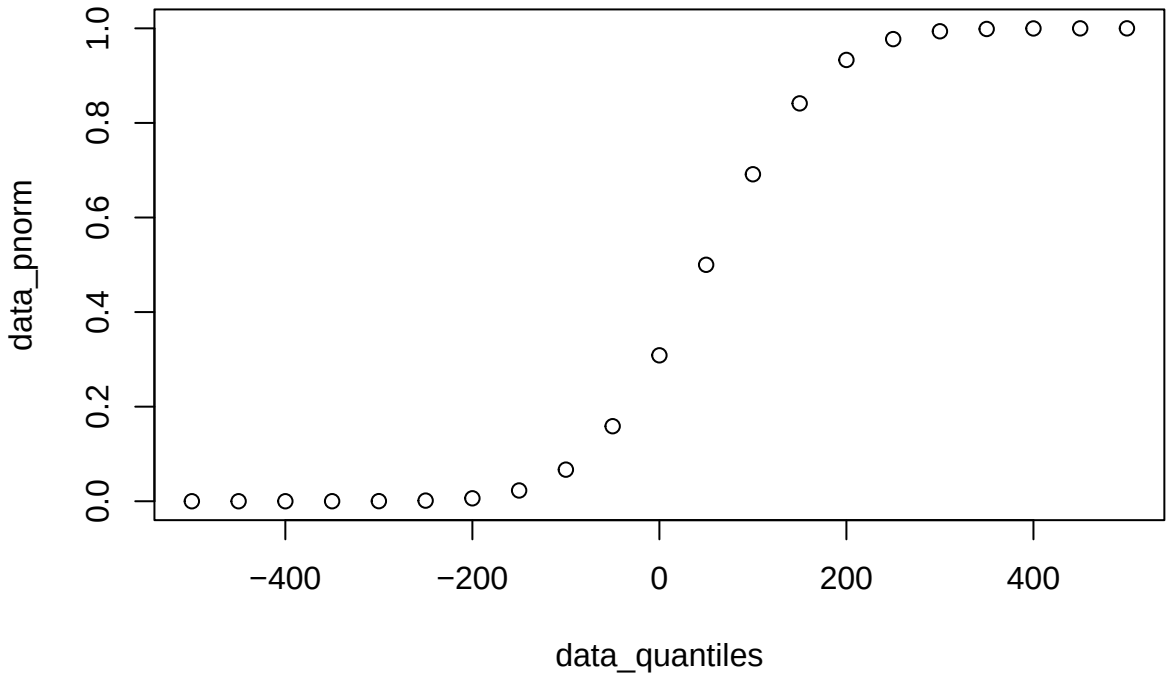
The function `pnorm()` returns the probability of a observing a value less or equal to each quantile given.

```
data_pnorm <- pnorm(q = data_quantiles, mean = 50, sd = 100)
head(data_pnorm)
```

```
## [1] 1.898956e-08 2.866516e-07 3.397673e-06 3.167124e-05 2.326291e-04
## [6] 1.349898e-03
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_quantiles, data_pnorm)
```




**qnorm()** The function `qnorm()` takes as input a vector of probabilities between 0 and 1.

```
data_probabilities <- seq(from = 0, to = 1, by = 0.02)
head(data_probabilities)
```

```
## [1] 0.00 0.02 0.04 0.06 0.08 0.10
```

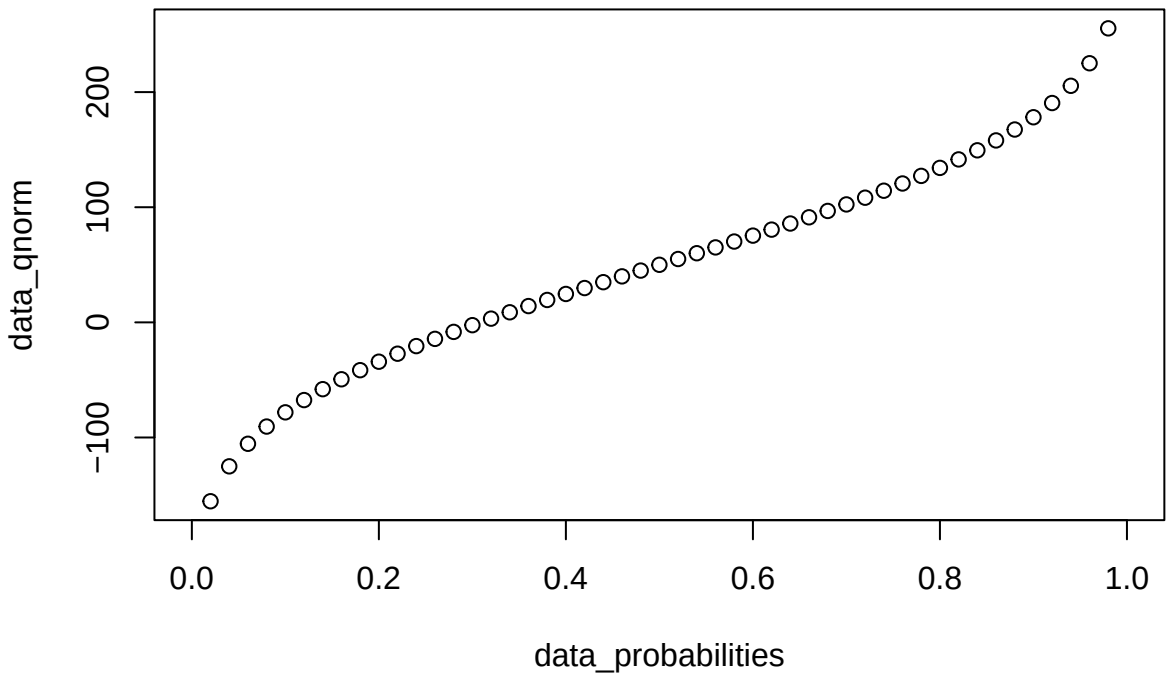
The function `qnorm()` the quantiles (i.e., values) that correspond to those probabilities.

```
data_qnorm <- qnorm(p = data_probabilities, mean = 50, sd = 100)
head(data_qnorm)
```

```
## [1]       -Inf -155.37489 -125.06861 -105.47736  -90.50716  -78.15516
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_probabilities, data_qnorm)
```




**dnorm** The quantiles represent a set of possible values in the distribution.

```
data_quantiles <- seq(-500, 500, 50)
head(data_quantiles)
```

```
## [1] -500 -450 -400 -350 -300 -250
```

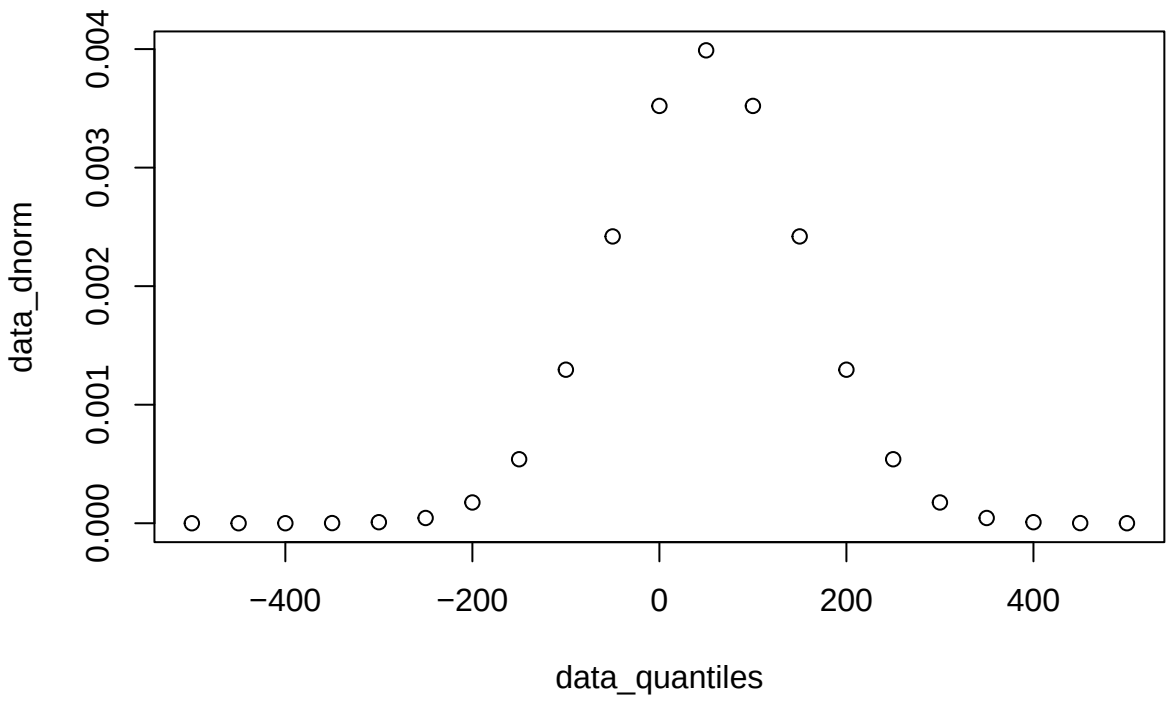
The function `pnorm()` returns the probability of a observing a value less or equal to each quantile given.

```
data_dnorm <- dnorm(x = data_quantiles, mean = 50, sd = 100)
head(data_dnorm)
```

```
## [1] 1.076976e-09 1.486720e-08 1.598374e-07 1.338302e-06 8.726827e-06
## [6] 4.431848e-05
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_quantiles, data_dnorm)
```

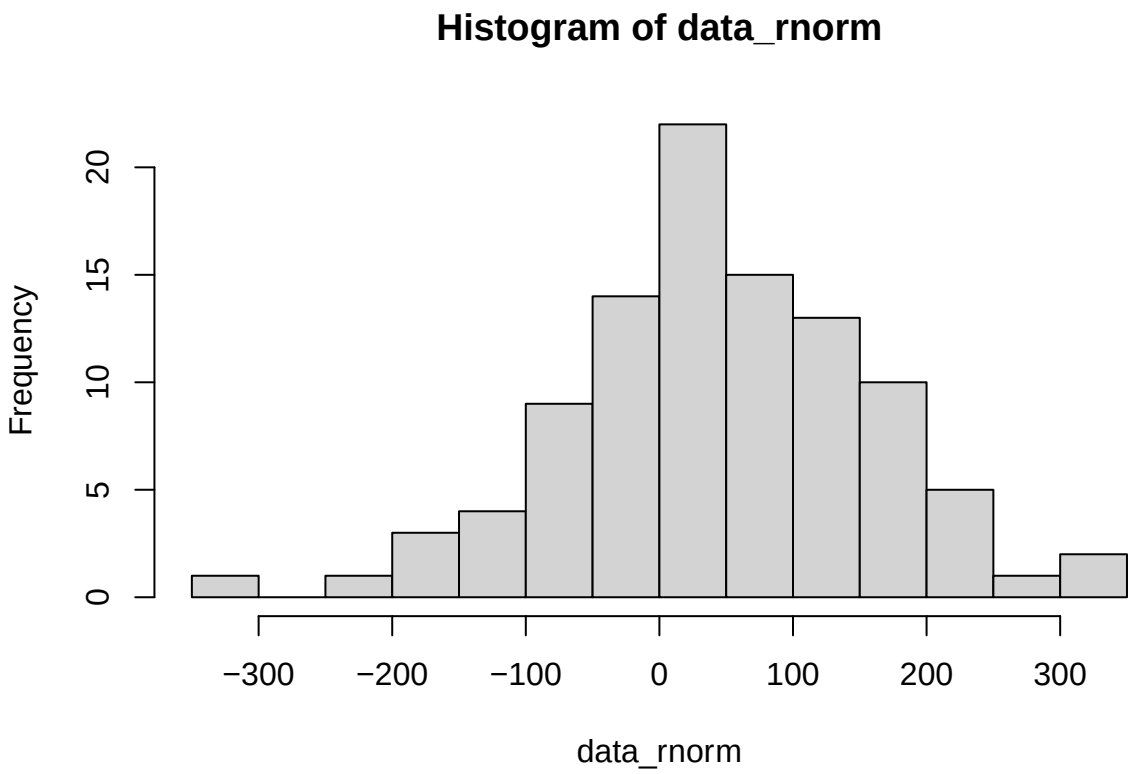



```
data_rnorm <- rnorm(n = 100, mean = 50, sd = 100)
head(data_rnorm)
```

**rnorm**

```
## [1] -30.26826 184.88101  29.86492 103.39879  81.53355  22.92193
```

```
hist(data_rnorm, breaks = 20)
```




## Demo

### A parameterised binomial distribution

**pnorm()** The quantiles represent a set of possible values in the distribution.

```
data_quantiles <- seq(0, 50, 1)
head(data_quantiles)
```

```
## [1] 0 1 2 3 4 5
```

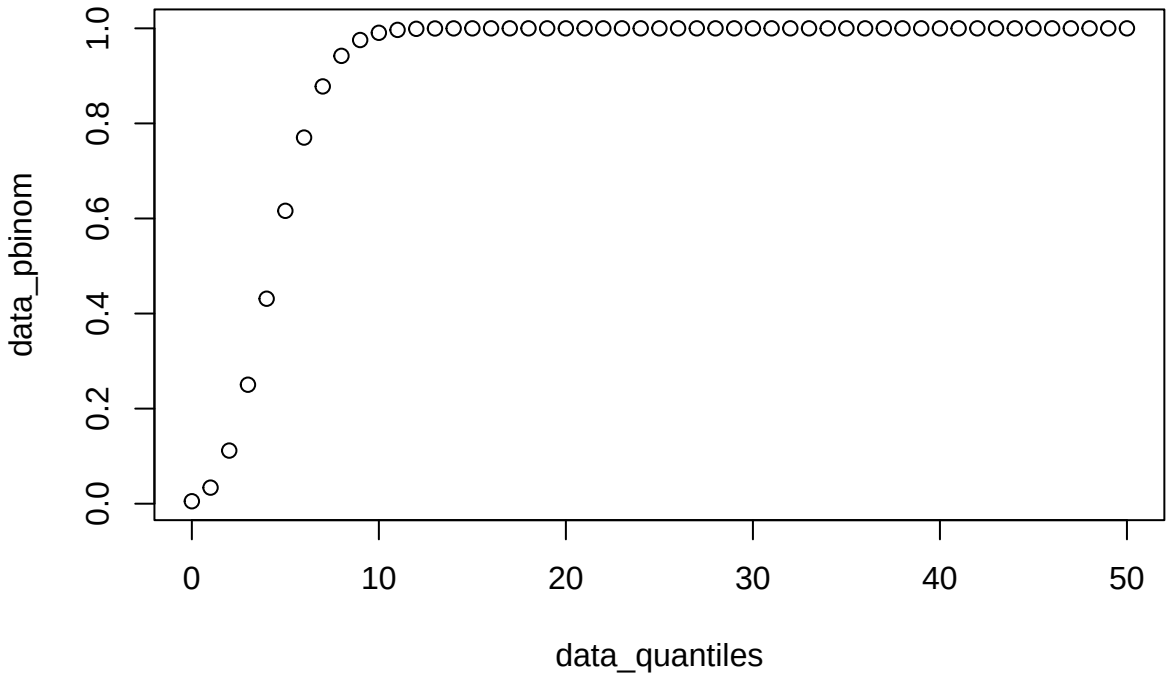
The function `pnorm()` returns the probability of a observing a value less or equal to each quantile given.

```
data_pbinom <- pbinom(q = data_quantiles, size = 50, prob = 0.1)
head(data_pbinom)
```

```
## [1] 0.005153775 0.033785860 0.111728756 0.250293906 0.431198407 0.616123008
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_quantiles, data_pbinom)
```




**qbinom()** The function `qbinom()` takes as input a vector of probabilities between 0 and 1.

```
data_probabilities <- seq(from = 0, to = 1, by = 0.02)
head(data_probabilities)
```

```
## [1] 0.00 0.02 0.04 0.06 0.08 0.10
```

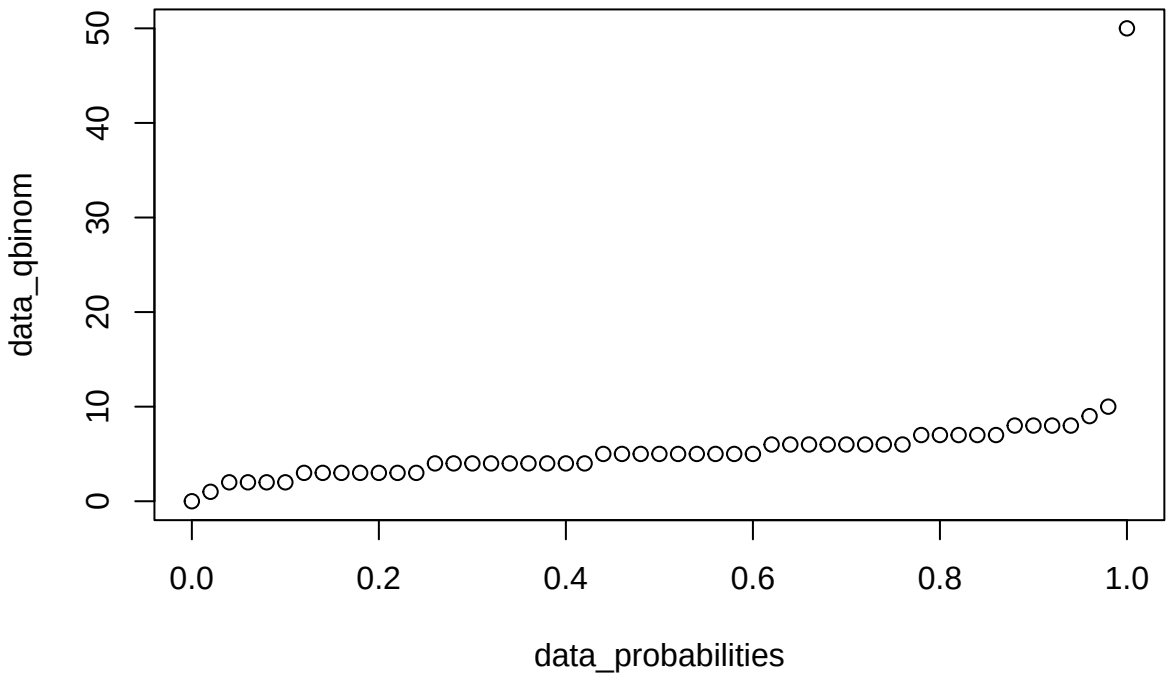
The function `qbinom()` the quantiles (i.e., values) that correspond to those probabilities.

```
data_qbinom <- qbinom(p = data_probabilities, size = 50, prob = 0.1)
head(data_qbinom)
```

```
## [1] 0 1 2 2 2 2
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_probabilities, data_qbinom)
```




**dbinom** The quantiles represent a set of possible values in the distribution.

```
data_quantiles <- seq(0, 50, 1)
head(data_quantiles)
```

```
## [1] 0 1 2 3 4 5
```

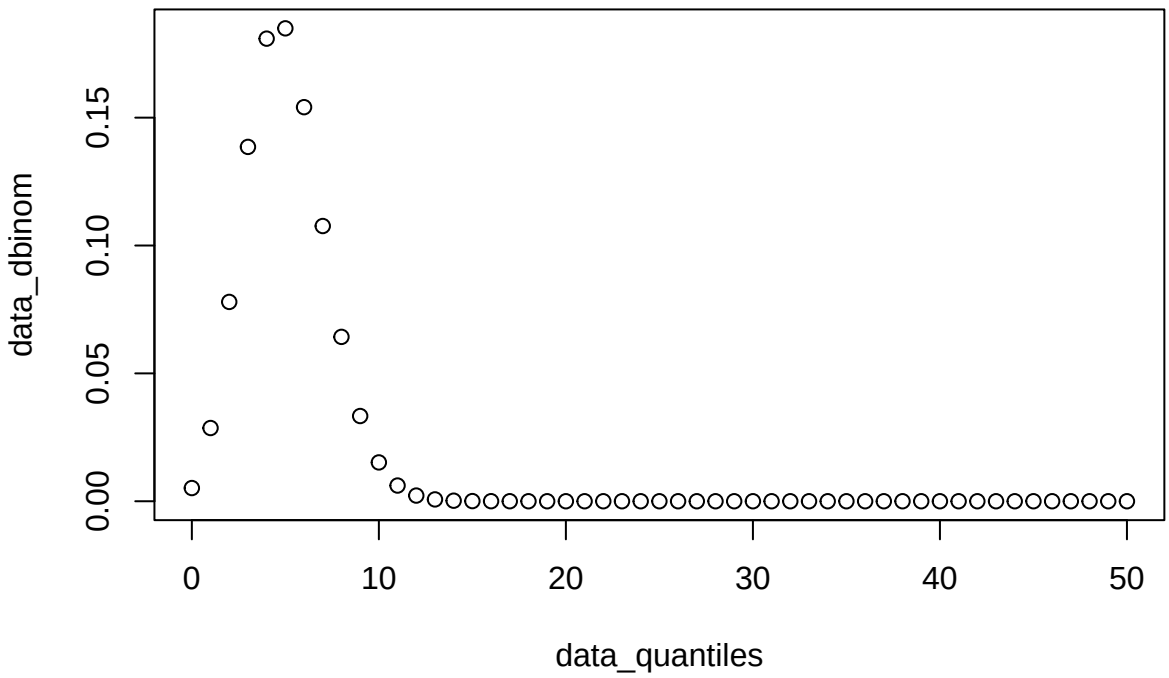
The function `pbinom()` returns the probability of a observing a value less or equal to each quantile given.

```
data_dbinom <- dbinom(x = data_quantiles, size = 50, prob = 0.1)
head(data_dbinom)
```

```
## [1] 0.005153775 0.028632084 0.077942897 0.138565150 0.180904501 0.184924601
```

The base R `plot()` function can be used to quickly plot each quantile and its probability.

```
plot(data_quantiles, data_dbinom)
```

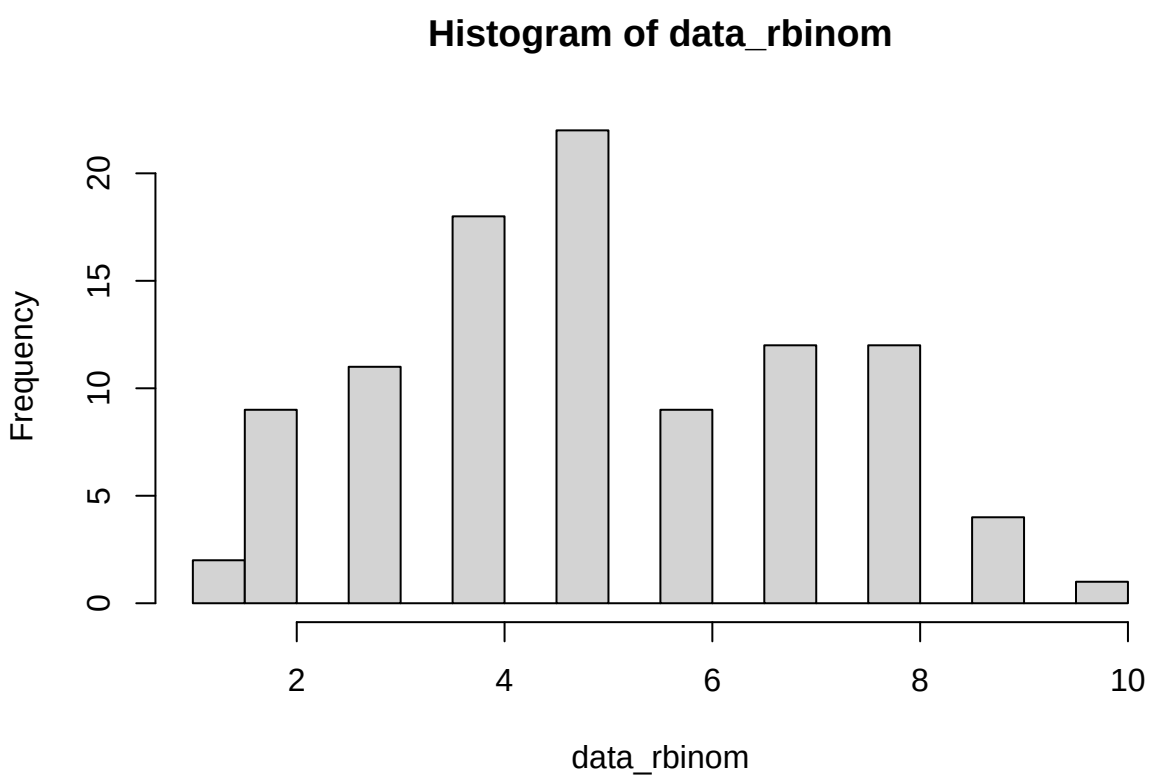



```
data_rbinom <- rbinom(n = 100, size = 50, prob = 0.1)
head(data_rbinom)
```

**rbinom**

```
## [1] 2 3 4 5 4 5
```

```
hist(data_rbinom, breaks = 20)
```



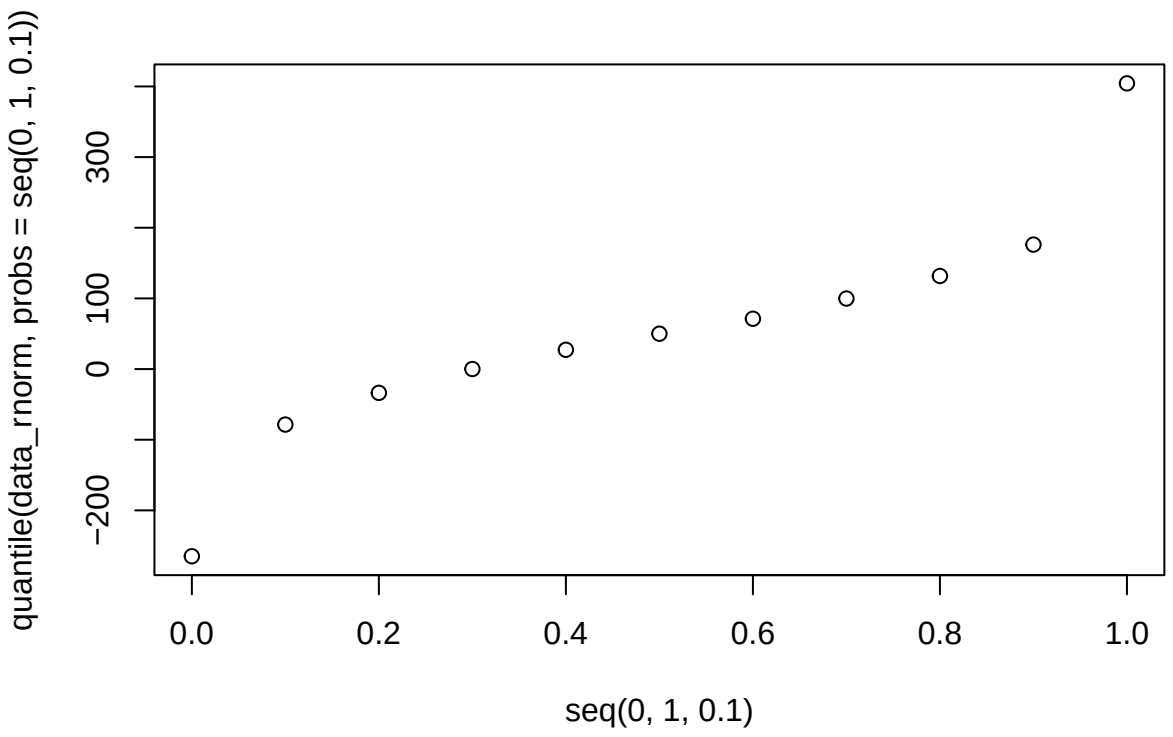

## Demo

### Quantiles

```
data_rnorm <- rnorm(n = 1000, mean = 50, sd = 100)
quantile(data_rnorm, probs = seq(0, 1, 0.1))
```

```
##            0%           10%           20%           30%           40%
## -264.91672087  -78.54178917  -33.68949604    0.07465986   27.28429276
##           50%           60%           70%           80%           90%
##   50.01638178   71.25849967   99.82666259  131.75714934  176.15793747
##          100%
##  404.37795917
```

```
plot(
  seq(0, 1, 0.1),
  quantile(data_rnorm, probs = seq(0, 1, 0.1))
)
```



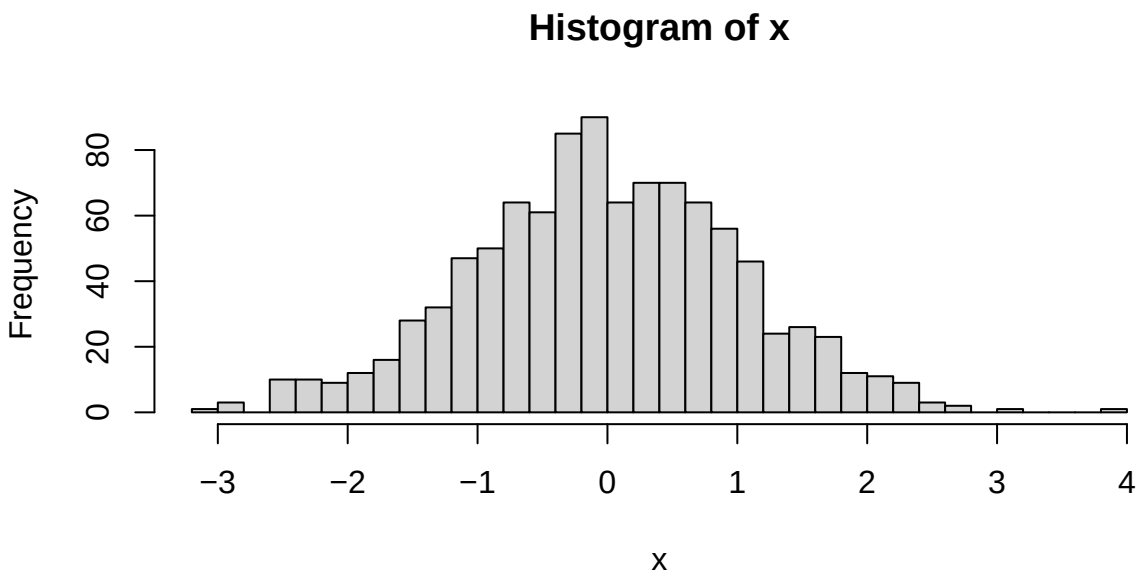

## Demo

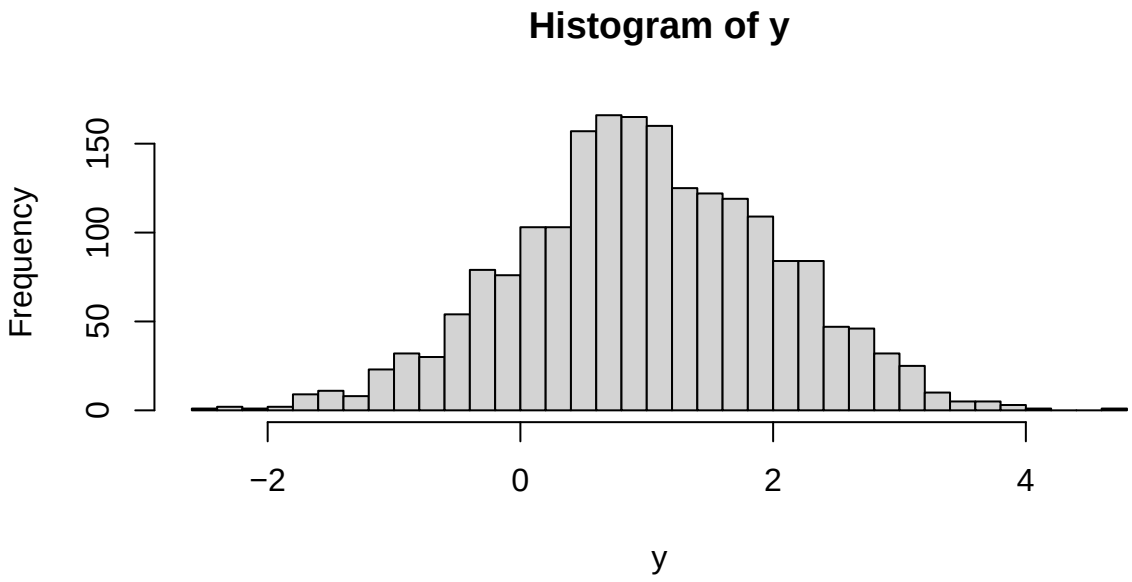
### Parametric t-test

```
set.seed(1)
x <- rnorm(n = 1000, mean = 0, sd = 1)
y <- rnorm(n = 2000, mean = 1, sd = 1)
```

In base R, `par(mfrow=c(i, j))` can be used to display plots in a grid of `i` rows and `j` columns.

```
par(mfrow = c(2, 1))
hist(x, breaks = 30)
hist(y, breaks = 30)
```




```
par(mfrow = c(1, 1 ))
```

```
t.test(x, y)
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -25.223, df = 1999, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.089791 -0.932552
```

```
## sample estimates:
##   mean of x   mean of y
## -0.01164814  0.99952356
```

```
t.test(y, x)
```

```
##
##  Welch Two Sample t-test
##
## data:  y and x
## t = 25.223, df = 1999, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.932552 1.089791
## sample estimates:
##   mean of x   mean of y
##  0.99952356 -0.01164814
```

## Demo

### Paired test

Simulate a vector of values `x`. Then simulate values `y` that are systematically 2 units greater than their `x` counterpart, with a bit of noise (normally distributed).

```
set.seed(1)
n_sample <- 10
x <- runif(n = n_sample, min = 10, max = 20)
y <- x + 2 + rnorm(n = n_sample, mean = 0, sd = 1)
```

The average difference between `y` and `x` values is approximately 2, as intended.

```
mean(y - x)
```

```
## [1] 2.086629
```

```
t.test(x, y, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  x and y
## t = -6.0238, df = 9, p-value = 0.0001967
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -2.870241 -1.303017
## sample estimates:
## mean difference
##       -2.086629
```

## Demo

### Non-parametric tests

```
set.seed(1)
x <- runif(n = 10, min = 1, max = 11)
y <- runif(n = 5, min = 3, max = 13)
```

```r
wilcox.test(x, y)
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  x and y
## W = 20, p-value = 0.5941
## alternative hypothesis: true location shift is not equal to 0
```

```r
wilcox.test(x, y, alternative = "less")
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  x and y
## W = 20, p-value = 0.297
## alternative hypothesis: true location shift is less than 0
```

## Demo

### Analysis of Variance (ANOVA)

```r
set.seed(1)
n_sample <- 1000
x1 <- rnorm(n = n_sample, mean = 10, sd = 2)
x2 <- x1 + 5 + rnorm(n = n_sample, mean = 0, sd = 1)
x3 <- x2 + 0 + rnorm(n = n_sample, mean = 0, sd = 0.5)
data_aov <- data.frame(
  value = c(x1, x2, x3),
  group = c(
    rep("x1", length(x1)),
    rep("x2", length(x2)),
    rep("x3", length(x3))
  )
)
head(data_aov)
```

```
##       value group
## 1  8.747092    x1
## 2 10.367287    x1
## 3  8.328743    x1
## 4 13.190562    x1
## 5 10.659016    x1
## 6  8.359063    x1
```

```r
aov_result <- aov(value ~ group, data_aov)
aov_result
```

```
## Call:
##    aov(formula = value ~ group, data = data_aov)
##
## Terms:
##                     group Residuals
## Sum of Squares  16583.9   15450.7
## Deg. of Freedom       2      2997
##
```

```
## Residual standard error: 2.270548
## Estimated effects may be unbalanced
```

```
summary(aov_result)
```

```
##               Df Sum Sq Mean Sq F value Pr(>F)
## group          2  16584    8292    1608 <2e-16 ***
## Residuals   2997  15451       5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## Demo

### Fisher's Exact Test

```
data_fisher <- matrix(
  data = c(12, 4, 3, 23),
  nrow = 2, ncol = 2,
  dimnames = list(
    c("DE", "Not DE"),
    c("In pathway", "Not in pathway")
  )
)
data_fisher
```

```
##        In pathway Not in pathway
## DE             12              3
## Not DE          4             23
```

```
fisher.test(data_fisher)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  data_fisher
## p-value = 4.983e-05
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##    3.592731 170.706615
## sample estimates:
## odds ratio
##   20.56889
```

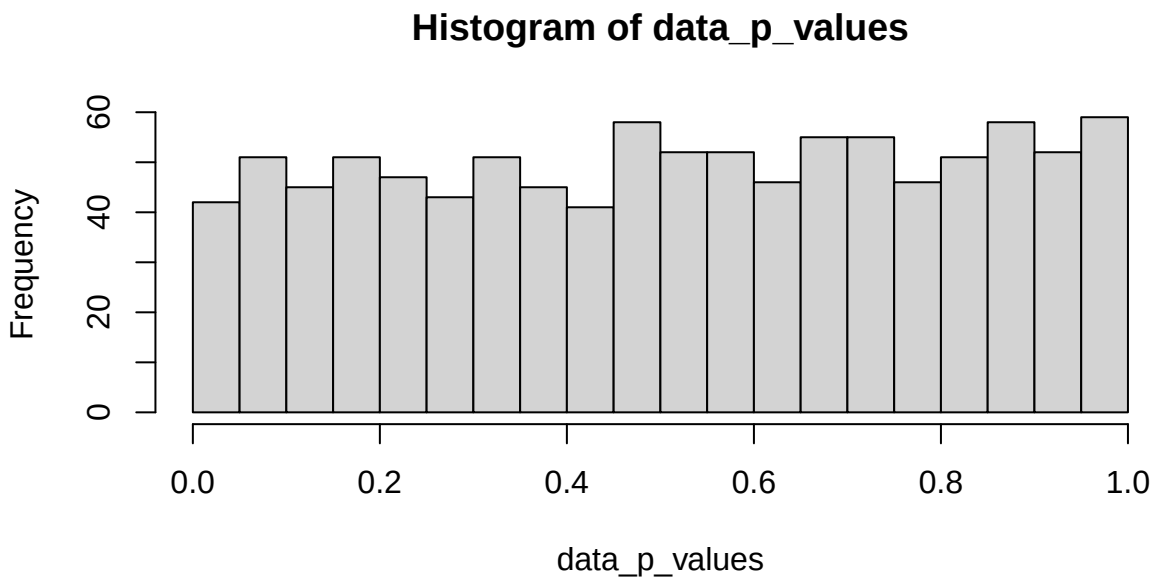
## Demo

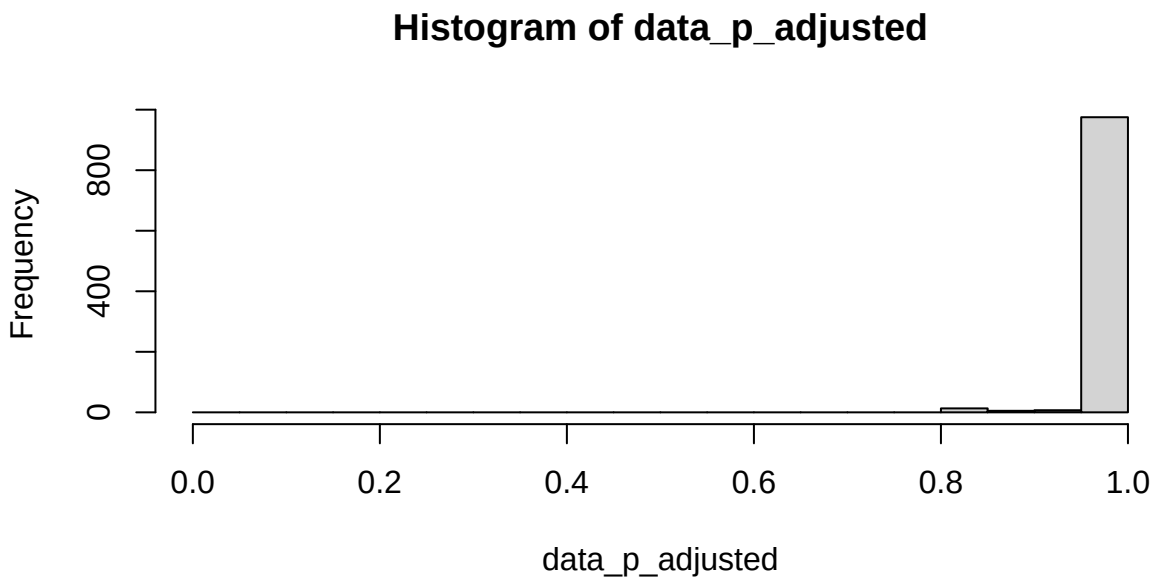
### Multiple-testing correction

```
data_p_values <- runif(1E3, min = 0, max = 1)
data_p_adjusted <- p.adjust(data_p_values, method = "BH")
head(sort(data_p_adjusted))
```

```
## [1] 0.8215773 0.8215773 0.8215773 0.8215773 0.8215773 0.8215773
```

```
par(mfrow = c(2, 1))
hist(data_p_values, xlim = c(0, 1), breaks = seq(0, 1, 0.05))
hist(data_p_adjusted, xlim = c(0, 1), breaks = seq(0, 1, 0.05))
```




```
par(mfrow = c(1, 1))
```

## Exercise

### Setup

- Import the `iris` data set.

```
data("iris")
```

- Separate the matrix of measurements in a new object named `iris_features`.

```r
iris_features <- iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")]
head(iris_features)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
```

## Exercise

### Apply Principal Components Analysis (PCA)

The `prcomp()` function allows you to standardise the data as part of the principal components analysis itself.

- Apply PCA, centering and scaling the matrix of features. Assign the result to an object called `pca_iris`.

```r
pca_iris <- prcomp(iris_features, center = TRUE, scale. = TRUE)
pca_iris
```

```
## Standard deviations (1, .., p=4):
## [1] 1.7083611 0.9560494 0.3830886 0.1439265
##
## Rotation (n x k) = (4 x 4):
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

- Examine the object `pca_iris`. Display the loading of each feature on each principal component.

```r
str(pca_iris)
```

```
## List of 5
##  $ sdev    : num [1:4] 1.708 0.956 0.383 0.144
##  $ rotation: num [1:4, 1:4] 0.521 -0.269 0.58 0.565 -0.377 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##   .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"
##  $ center  : Named num [1:4] 5.84 3.06 3.76 1.2
##   ..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##  $ scale   : Named num [1:4] 0.828 0.436 1.765 0.762
##   ..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##  $ x       : num [1:150, 1:4] -2.26 -2.07 -2.36 -2.29 -2.38 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"
##  - attr(*, "class")= chr "prcomp"
```

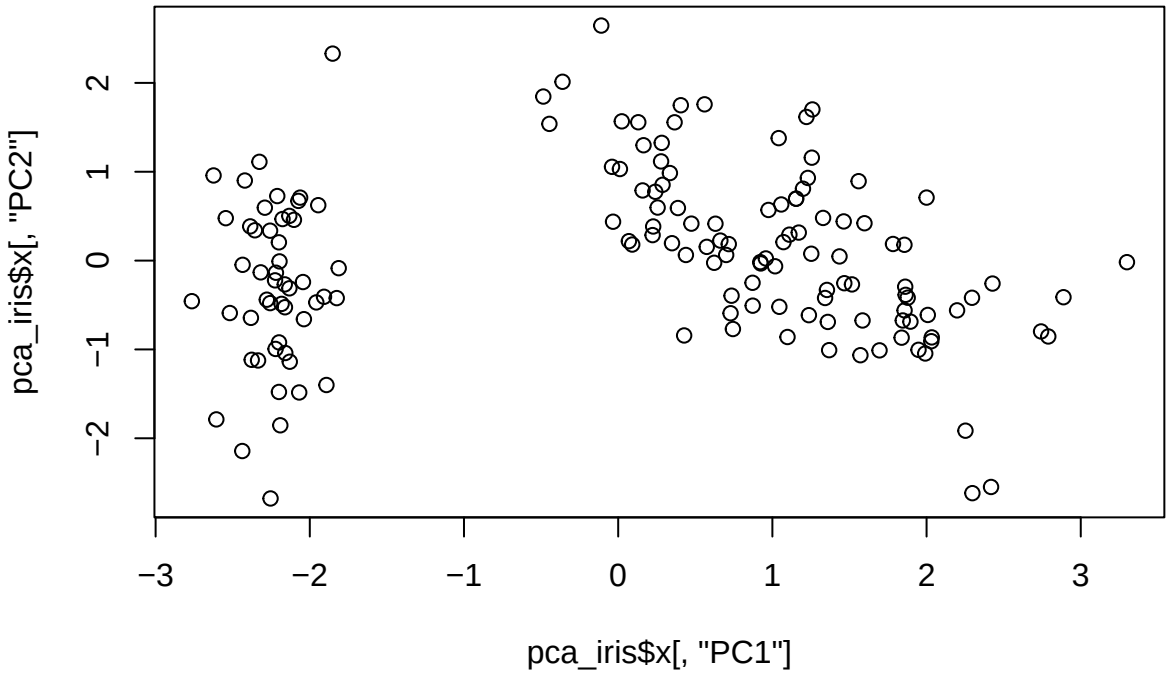
```r
pca_iris$rotation
```

```
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
```

```
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

- Visualise the PCA projection using `plot()`.

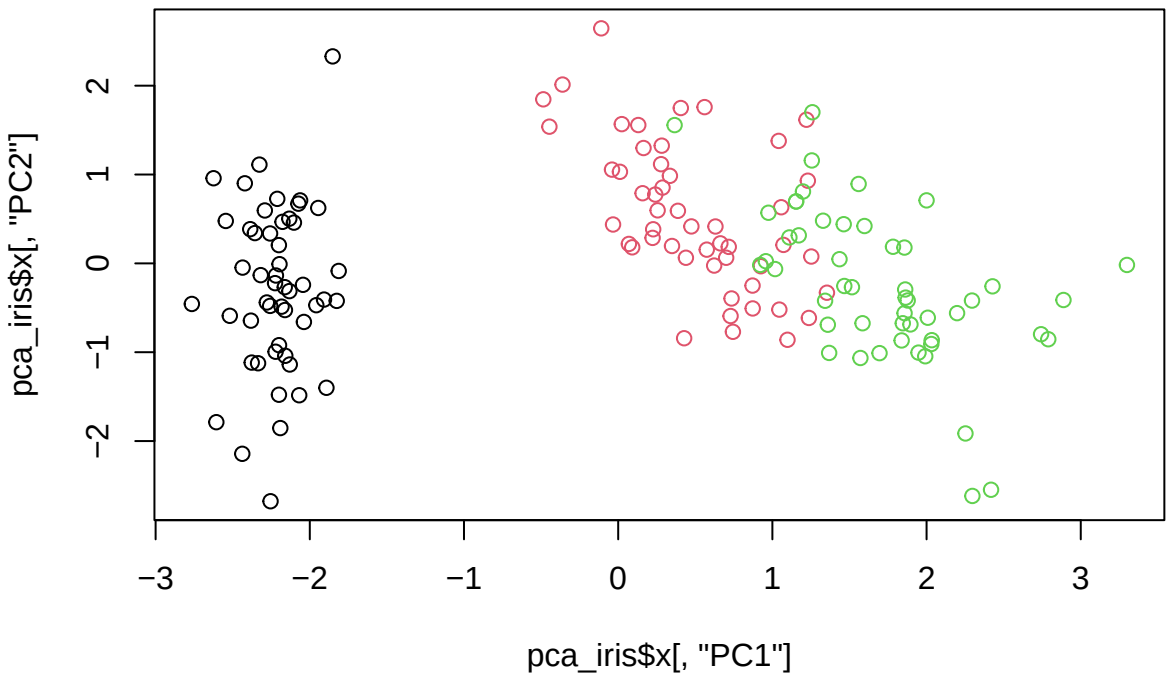
```
plot(pca_iris$x[, "PC1"], pca_iris$x[, "PC2"])
```




**Bonus point**

- Edit the plot above, coloring data points according to their class label.

```
plot(pca_iris$x[, "PC1"], pca_iris$x[, "PC2"], col = iris$Species)
```




## Exercise

### Variance explained

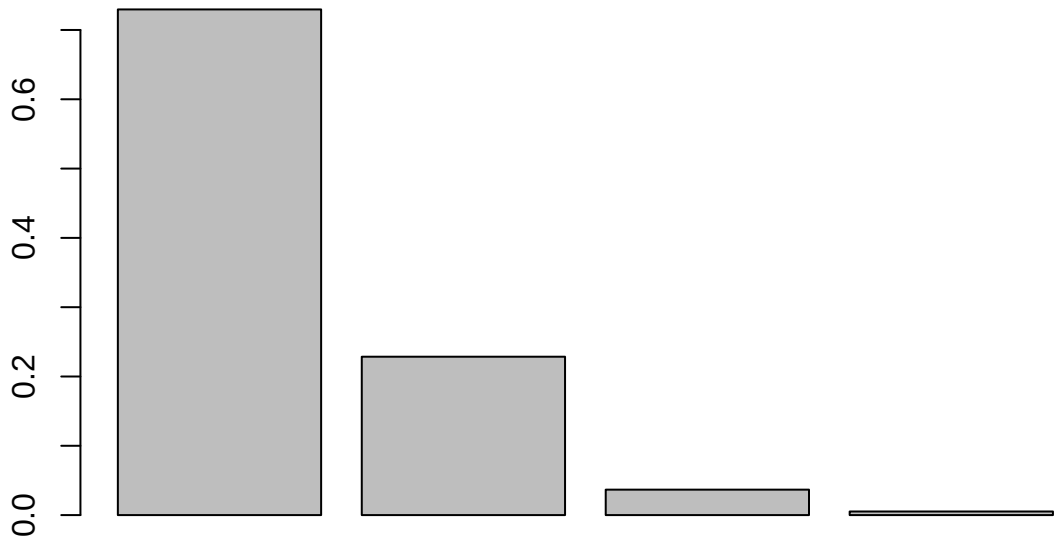
- Compute the variance explained by principal components, using information present in the return value of the `prcomp()` function.

```
explained_variance_ratio <- pca_iris$sdev ^ 2 / sum(pca_iris$sdev ^ 2)
explained_variance_ratio
```

```
## [1] 0.729624454 0.228507618 0.036689219 0.005178709
```

- Visualise the variance explained by each principal component using `barplot()`.

```
barplot(explained_variance_ratio)
```




# Exercise

## Hierarchical clustering

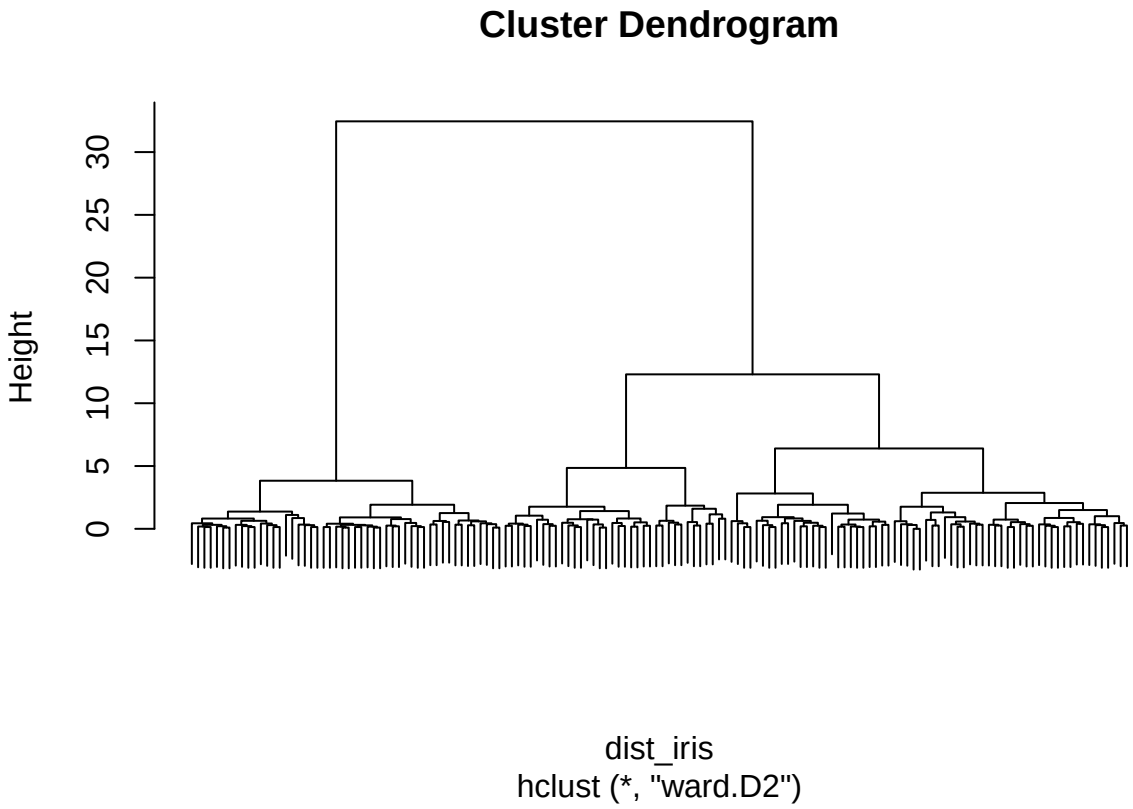
- Perform hierarchical clustering on the `iris_features` data set, using the `euclidean` distance and method `ward.D2`. Use the functions `dist()` and `hclust()`.

```
dist_iris <- dist(iris_features, method = "euclidean")
hclust_iris_ward <- hclust(dist_iris, method = "ward.D2")
hclust_iris_ward
```

```
##
## Call:
## hclust(d = dist_iris, method = "ward.D2")
##
## Cluster method   : ward.D2
## Distance         : euclidean
## Number of objects: 150
```

- Plot the clustering tree using `plot()`.

```
plot(hclust_iris_ward, labels = FALSE)
```




How many clusters would you call from a visual inspection of the tree?

> Answer: - Visually: two major clusters. - One could argue for three - maybe four - clusters, depending on follow-up analyses.

- Cut the tree in 3 clusters and extract the cluster label for each flower. Use the function `cutree()`.

```
iris_hclust_ward_labels <- cutree(hclust_iris_ward, k = 3)
iris_hclust_ward_labels
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 2 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
```

- Repeat clustering using 3 other agglomeration methods:
  - `complete`
  - `average`
  - `single`

```
# complete
hclust_iris_complete <- hclust(dist_iris, method = "complete")
iris_hclust_complete_labels <- cutree(hclust_iris_complete, k = 3)
iris_hclust_complete_labels
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3 2 3 2 2
##  [75] 2 2 2 2 2 3 3 3 3 2 3 2 2 2 3 3 3 2 3 3 3 3 2 3 3 2 2 2 2 2 2 3 2 2 2 2
```

```
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2
```

```
# average
hclust_iris_average <- hclust(dist_iris, method = "average")
iris_hclust_average_labels <- cutree(hclust_iris_average, k = 3)
iris_hclust_average_labels
```

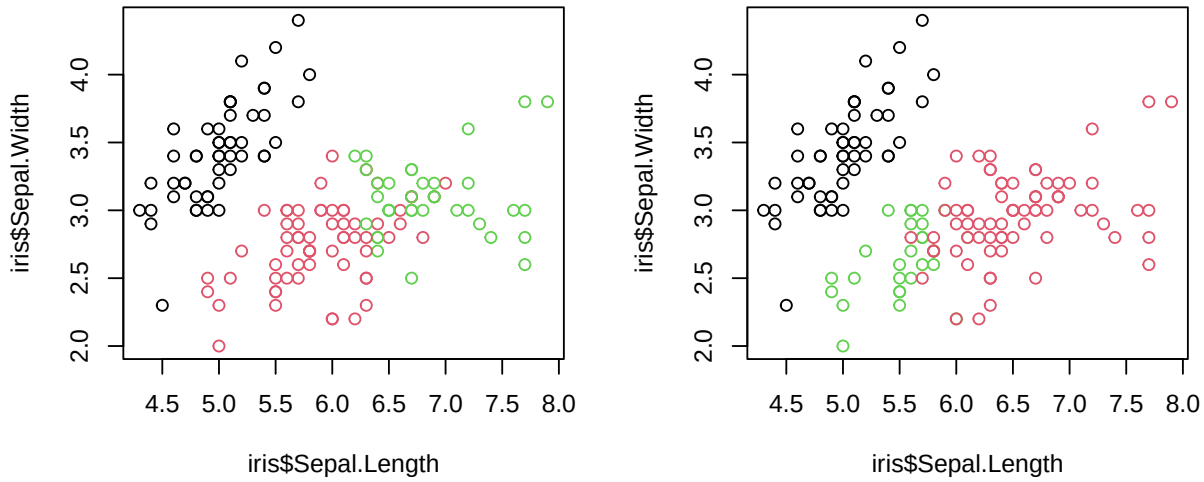
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
```

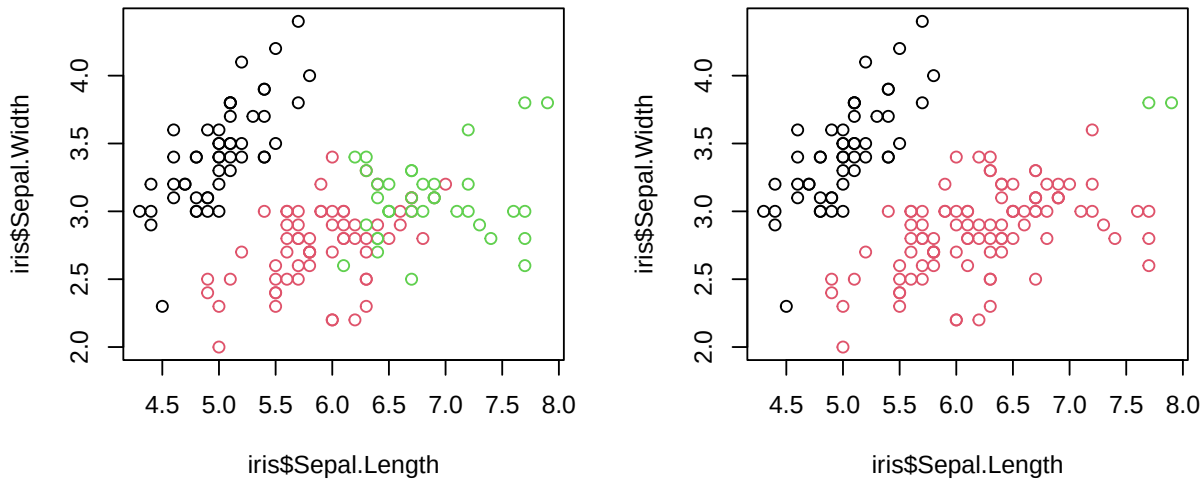
```
# single
hclust_iris_single <- hclust(dist_iris, method = "single")
iris_hclust_single_labels <- cutree(hclust_iris_single, k = 3)
iris_hclust_single_labels
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2
```

- Compare clustering results on scatter plots of the data using `plot()`.

```
par(mfrow = c(2, 2))
plot(iris$Sepal.Length, iris$Sepal.Width, col = iris_hclust_ward_labels)
plot(iris$Sepal.Length, iris$Sepal.Width, col = iris_hclust_complete_labels)
plot(iris$Sepal.Length, iris$Sepal.Width, col = iris_hclust_average_labels)
plot(iris$Sepal.Length, iris$Sepal.Width, col = iris_hclust_single_labels)
```




```
par(mfrow = c(1, 1))
```