IT University of Copenhagen

Applied Statistics - Spring 2022

Useful R Functions

Author
Viggo U. Gascou

May 31, 2022

Comment

The following is a collection of R functions that I found useful for calculating, visualizing all sorts of different things in R.
Made in preparation to Applied Statistics Exam

Contents

1 Useful R stuff **4**
- 1.1 Setup chunk . . . 4
- 1.2 Creating sample spaces . . . 5
  - 1.2.1 Subsets of sample spaces . . . 5
  - 1.2.2 Complements of sets . . . 6
  - 1.2.3 Intersection of sets . . . 6
  - 1.2.4 Union of sets . . . 6

2 Basic probabilities **7**
- 2.1 De Morgan's Law of Union and Intersection . . . 7
- 2.2 Conditional probability . . . 7
  - 2.2.1 Bayes theorem . . . 7
  - 2.2.2 Chain rule . . . 7

3 Venn diagrams **7**

4 Continous random variables & simulation **7**
- 4.1 The normal distribution . . . 7
  - 4.1.1 `dnorm(x, mean, sd)` . . . 7
  - 4.1.2 `pnorm(q, mean, sd)` . . . 9
  - 4.1.3 `qnorm(p, mean, sd)` . . . 10
  - 4.1.4 `rnorm(n, mean, sd)` . . . 10
- 4.2 The binomial (Bernoulli) distribution . . . 11
  - 4.2.1 `dbinom(x, size, prob)` . . . 11
  - 4.2.2 `pbinom(q, size, prob)` . . . 12
  - 4.2.3 `qbinom(q, size, prob)` . . . 12
  - 4.2.4 `rbinom(n, size, prob)` . . . 13
- 4.3 The geometric distribution . . . 13
  - 4.3.1 `dgeom(x, prob)` . . . 13
  - 4.3.2 `pgeom(q, prob)` . . . 14
  - 4.3.3 `qgeom(p, prob)` . . . 14
  - 4.3.4 `rgeom(n, prob)` . . . 14

5 Expectation and variance **15**
- 5.1 Expected value and variance of common distributions . . . 15
  - 5.1.1 Geometric distribution . . . 15
  - 5.1.2 Exponential distribution . . . 15
  - 5.1.3 Normal distribution . . . 15
- 5.2 Expected value . . . 15
  - 5.2.1 Method 1 - using `weighted.mean()` . . . 16
  - 5.2.2 Sample mean . . . 16
- 5.3 Variance . . . 16
  - 5.3.1 Sample variance . . . 17
  - 5.3.2 Population variance . . . 17

6 Joint distributions and independence **17**
- 6.1 Joint probability distribution . . . 18
- 6.2 Marginal probability distribution . . . 18
- 6.3 Independece between two variables . . . 19

7 Covariance and correlation **19**
- 7.1 Covariance . . . 19
  - 7.1.1 Covariance between two variables . . . 19

7.1.2 Covariance between multiple variables . . . 19
7.2 Correlation . . . 20
7.2.1 Correlation between two variables . . . 20
7.2.2 Correlation between multiple variables . . . 20

8 Computations with random variables **21**
8.1 Transforming (scaling and shifting) random variables . . . 21
8.1.1 Change-of-Variable Formula . . . 22
8.1.2 Linearity of Expectation (LOTUS) . . . 22
8.1.3 Jensen Inequality . . . 22

9 Law of large numbers and Central limit theorem **23**
9.1 The law of large numbers . . . 23
9.2 Chebyshev's inequality . . . 25
9.2.1 Example of usage of Chebyshev's Inequality . . . 25
9.3 Central Limit Theorem . . . 26
9.3.1 An example of applying CLT to find a probability . . . 26

10 Exploratory data analysis - Graphical summaries **27**
10.1 Histograms . . . 27
10.1.1 Base R `hist(data)` . . . 28
10.1.2 Plotting multiple histograms on one plot with `hist(data)` . . . 29
10.1.3 ggplot2 `ggplot(data)` . . . 29
10.1.4 Plotting multiple histograms on one plot with `ggplot(data)` . . . 30
10.2 Kernel density estimates . . . 32
10.3 Empirical distribution function . . . 33
10.4 Scatterplot . . . 34

11 Exploratory data analysis - Numerical summaries **35**
11.1 Middle of a dataset (sample mean and median) . . . 35
11.1.1 Sample median . . . 35
11.2 The amount of variability of a dataset . . . 35
11.2.1 Sample standard deviation . . . 35
11.3 Empirical quantiles, quartiles, and the IQR . . . 36
11.3.1 `quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE)` . . . 37
11.3.2 Boxplots . . . 37

12 Basic statistical models **39**
12.1 The linear regression model . . . 39
12.1.1 Simple linear regression model . . . 39
12.1.2 Multiple linear regression model . . . 40
12.1.3 Plotting a regression model . . . 41

13 Bootstrapping **42**
13.1 Empirical bootstrapping . . . 43
13.2 Parametric bootstrapping . . . 45
13.3 Bootstrapping and confidence intervals . . . 48

14 Unbiased estimators **50**
14.1 Unbiased estimator for the mean . . . 51
14.2 Unbiased estimator for the variance . . . 51
14.3 Unbiased for the uniform distribution $U(0, \Theta)$ . . . 51

15 Maximum likelihood **51**
15.1 Normal distribution . . . 51

15.1.1 The maximum likelihood estimators of μ and σ^2 . . . 52
15.2 Exponential distribution . . . 52

List of Tables

1 Goal probability of a football team . . . 16
2 Sample dataset . . . 17
3 Relation between hair color and eye color. . . . 17
4 Joint probability distributions of X and Y . . . 18
5 Marginal probability distributions of X and Y . . . 18
6 Randomly generated normal dataset . . . 28
7 Common parameters with their corresponding sample statistic . . . 43

List of Figures

1 Sixteen things you can say about A and B . . . 8
2 Concave and convex functions . . . 23
3 Normal distribution with marked area . . . 27
4 Examples of well-known kernels K . . . 32
5 Examples of well-known kernels K in `R` . . . 33
6 Some sample statistics and corresponding distribution features . . . 40

```r
# Loading required libraries
library(knitr)
library(kableExtra)
library(RColorBrewer)
library(UsingR)
library(ggplot2)
```

1 Useful R stuff

1.1 Setup chunk

This is a '`setup`' chunk that includes all the libraries allowed in the Applied Statistics exam along with a function I created to display sample spaces as proper LaTeX as long as they have been created using the `expand_grid()` function. See the code chunk for more details on usage of this function. If you want to copy this code chunk remember to add the three backticks before and after the whole thing ```.

```r
{r "setup", include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(kableExtra)
library(dplyr)
library(knitr)
library(UsingR)

# This is a self created function to print sample spaces or events that have
# been created with the expand.grid() function. It can also detect if the event/
# sample space is empty, by printing the symbol for the empty space.
# It works by looping through the data frame
# and adding each row to the cat() function which is able to print and render
# latex and markdown commands properly only if used with the code chunk flag:
# " results = 'asis' ".
# To print a sample space simply call the function with the event/sample space
# you want to print and the name of the event/sample space:
# like this: print_samp_space(event_to_be_printed, name_of_event)
# remember to escape latex commands i.e. if you want to create an event/sample
# space called Omega you have to write " \\Omega " or if you want to write the
# intersection of A and B you have to write " A \\cap B ".

print_samp_space <- function(space, name) {
    if (nrow(space) == 0) return(cat("$$", name, "= \\emptyset", "$$"))

    output <- cat("$$", "\\begin{aligned}", name, "= &\\{")

    for (i in seq_len(nrow(space))) {
        if (i == nrow(space)) {
            output <- output + cat("(", paste(unlist(space[i, ]), collapse = ", "), ")\\}", "\\end{alig
        }
        else if (i == 8 | i == 16 | i == 24| i == 48) {
          output <- output + cat("(", paste(unlist(space[i, ]), collapse = ", "), "),", "\\\\", "&", "\
        }
        else {
            output <- output +  cat("(", paste(unlist(space[i, ]), collapse = ", "), "),")
        }
    }
```

```
}
```

1.2 Creating sample spaces

In R it is sometimes useful to create a sample space of say throwing a coin 3 times. This is kind of tedious to do by hand, but is very easy to do in R:

```
# Create the outcomes that are possible with each throw
coin <- c("H", "T")
# Create the sample space
# Simply repeat the variable name for the amount of throws you require - here: 3
samp_space <- expand.grid(coin, coin, coin)
#Renaming the columns so it becomes more readable
colnames(samp_space) <- c("throw_1", "throw_2", "throw_3")
```

This creates the sample space Ω and using the `print_samp_space` function we get:

```
print_samp_space(samp_space, "\\Omega")
```

$$\Omega = \{(H,H,H),(T,H,H),(H,T,H),(T,T,H),(H,H,T),(T,H,T),(H,T,T),(T,T,T)\}$$

Tip: *If you want to show the sample space right after some text without the weird empty space simply add the '`echo = FALSE`' flag to the chunk flags* Like this:

```
{r, results='asis', echo = FALSE}
print_samp_space(samp_space, "\\Omega")
```

Remember to add the backticks ``` before and after.

1.2.1 Subsets of sample spaces

It is also sometimes useful to create events from sample spaces that live up to specific requirements:

Note: From 2019 exam

(b) (2 pts) Write down the set of outcomes corresponding to each of the following events

- A: "We get at least one tails"
- B: "The coin always comes with the same side up."

Again because we created the sample space in R it become easier to create events that conform to specific requirements:

```
# Checking that either throw_1, throw_2, or throw_3 are tails
event_a <- subset(samp_space, throw_1 == "T" | throw_2 == "T" | throw_3 == "T")
```

Which creates the following event A:

$$A = \{(T,H,H),(H,T,H),(T,T,H),(H,H,T),(T,H,T),(H,T,T),(T,T,T)\}$$

And for event B:

```
# Checking that throw_1 is equal to throw_2 and that throw_2 is equal to throw_3
# If throw_1 is equal to throw_2 and throw_3 is equal to throw_2, then throw_3
# has to be equal to throw_1
event_b <- subset(samp_space, throw_1 == throw_2 & throw_2 == throw_3)
```

Which becomes the event B:

$$B = \{(H,H,H),(T,T,T)\}$$

1.2.2 Complements of sets

We can easily create the complements of events A and B respectively by using the `anti_join` function from the `dplyr` library:

```
comp_a <- anti_join(samp_space, event_a)
comp_b <- anti_join(samp_space, event_b)
```

Which creates A^c:

$$A^c = \{(H, H, H)\}$$

And B^c:

$$B^c = \{(T, H, H), (H, T, H), (T, T, H), (H, H, T), (T, H, T), (H, T, T)\}$$

1.2.3 Intersection of sets

And with the `inner_join` (also from the `dplyr` library) function we can get the intersection of two sets:

```
event_c <- inner_join(event_a, event_b)
```

$$A \cap B = \{(T, T, T)\}$$

1.2.4 Union of sets

And with the `full_join` (from the `dplyr` library) function we can get the union of two sets:

```
event_c <- full_join(event_a, event_b)
```

$$A \cup B = \{(T, H, H), (H, T, H), (T, T, H), (H, H, T), (T, H, T), (H, T, T), (T, T, T), (H, H, H)\}$$

2 Basic probabilities

If we want to calculate the probability of the events created using the above methods we can easily do so by using the `nrow()`function and dividing the two numbers:

```
prob_a <- nrow(event_a) / nrow(samp_space)
prob_a
```

```
## [1] 0.875
```

2.1 De Morgan's Law of Union and Intersection

$$(A \cup B)^c = A^c \cap B^c$$

And

$$(A \cap B)^c = A^c \cup B^c$$

2.2 Conditional probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{\text{A and B occurs}}{\text{B occurs}}$$

2.2.1 Bayes theorem

Bayes Theorem, is one of the most useful formulas in conditional probability. It is a way to compute the opposite conditional probability. So, assume we know $P(B|A)$, we can compute $P(A|B)$ as follows:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

I order to find $P(B)$, we use the law of total probability, which results in the extended Bayes Rule:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B|A) \cdot P(A) + P(B|A^c) \cdot P(A^c)}$$

2.2.2 Chain rule

$$P(A \cap B) = P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$

3 Venn diagrams

Image from MIT OpenCourseWare (On next page)

4 Continous random variables & simulation

4.1 The normal distribution

4.1.1 `dnorm(x, mean, sd)`

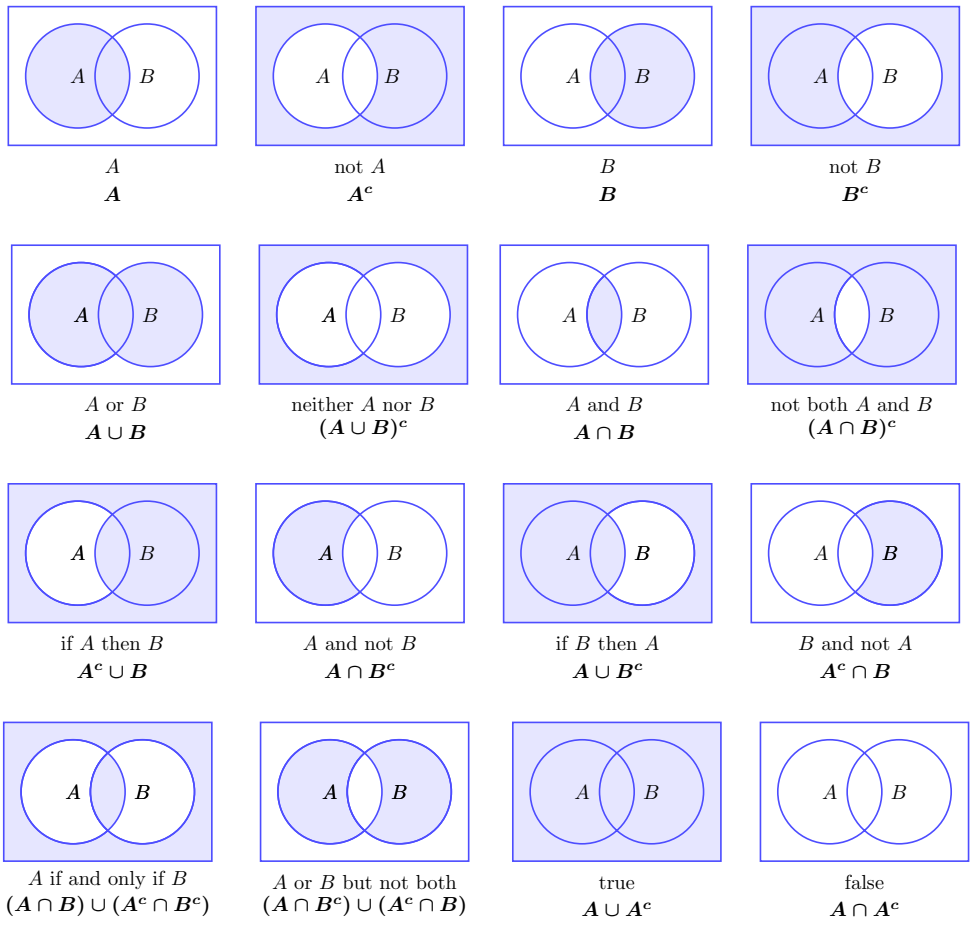
The dnorm function returns the value of the probability density function (pdf) of a normal distribution, given a random variable `x` a population mean μ and a population standard deviation σ

```
#find the value of the standard normal distribution pdf at x=0
dnorm(x = 0, mean = 0, sd = 1)
```

```
## [1] 0.3989423
```

Sixteen things you can say about A and B

Each of the 16 sets below is indicated by a shaded region.



Complementary Laws:	$A \cup A^c = \emptyset^c$	$A \cap A^c = \emptyset$
Associative Laws:	$A \cup (B \cup C) = (A \cup B) \cup C$	$A \cap (B \cap C) = (A \cap B) \cap C$
Commutative Laws:	$A \cup B = B \cup A$	$A \cap B = B \cap A$
Distributive Laws:	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
De Morgan's Laws:	$(A \cup B)^c = A^c \cap B^c$	$(A \cap B)^c = A^c \cup B^c$

Figure 1: Sixteen things you can say about A and B

```r
#by default, R uses mean=0 and sd=1
dnorm(x = 0)
```

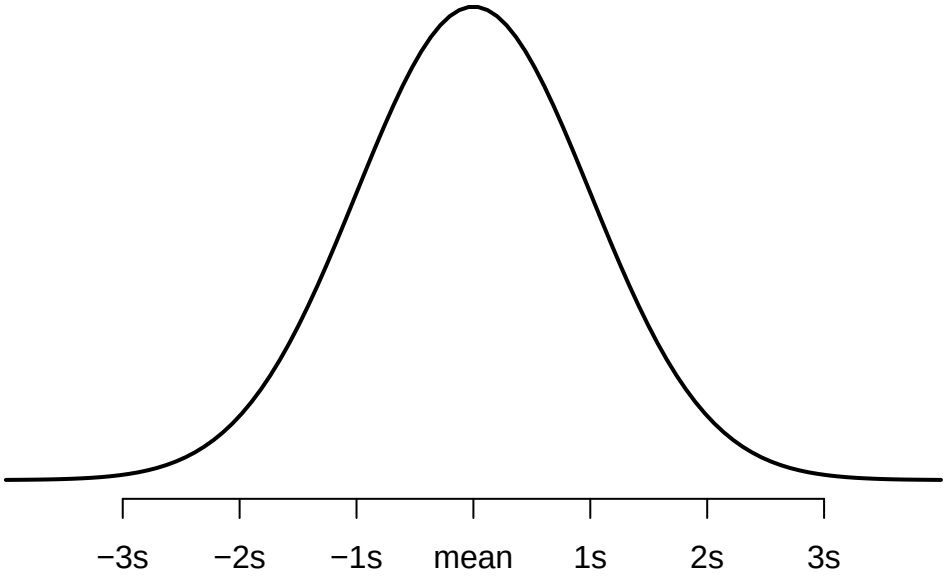
```
## [1] 0.3989423
```

```r
#find the value of the normal distribution pdf at x=10 with mean=20 and sd=5
dnorm(x = 10, mean = 20, sd = 5)
```

```
## [1] 0.01079819
```

`dnorm` is not that useful for solving questions about probability using the normal distribution. It is however useful to create a normal distribution plot:

```r
#Create a sequence of 100 equally spaced numbers between -4 and 4
x <- seq(-4, 4, length = 100)

#create a vector of values that shows the height of the probability distribution
#for each value in x
y <- dnorm(x)

#plot x and y as a scatterplot with connected lines (type = "l") and add
#an x-axis with custom labels
plot(x, y, type = "l", lwd = 2, axes = FALSE, xlab = "", ylab = "")
axis(1, at = -3:3, labels = c("-3s", "-2s", "-1s", "mean", "1s", "2s", "3s"))
```



4.1.2 `pnorm(q, mean, sd)`

The function pnorm returns the value of the cumulative density function (cdf) of the normal distribution given a certain random variable `q`, a population mean μ and population standard deviation σ.

Put simply, pnorm returns the area to the left of a given value x in the normal distribution. If you're interested in the area to the right of a given value `q`, you can simply add the argument `lower.tail = FALSE` i.e., `pnorm(q, size, prob, lower.tail = FALSE)`

Example:
Suppose the height of males at a certain school is normally distributed with a mean of $\mu = 70$ inches and a standard deviation of $\sigma = 2$ inches. Approximately what percentage of males at this school are taller than 74 inches?

```r
#find percentage of males that are taller than 74 inches in a population with
#mean = 70 and sd = 2
pnorm(74, mean = 70, sd = 2, lower.tail = FALSE)
```

```
## [1] 0.02275013
```

At this school, **2.275%** of males are taller than **74 inches**.

Example 2:
Suppose the weight of a certain species of otters is normally distributed with a mean of $\mu = 30$ lbs and a standard deviation of $\sigma = 5$ lbs. Approximately what percentage of this species of otters weight less than 22 lbs?

```r
#find percentage of otters that weight less than 22 lbs in a population with
#mean = 30 and sd = 5
pnorm(22, mean = 30, sd = 5)
```

```
## [1] 0.05479929
```

Approximately** 5.4799%** of this species of otters weigh less than **22 lbs**.

4.1.3 `qnorm(p, mean, sd)`

The function pnorm returns the value of the inverse cumulative density function (cdf) of the normal distribution given a certain random variable `q`, a population mean μ and population standard deviation σ.

Put simply, you can use qnorm to find out what the Z-score is of the p^{th} quantile of the normal distribution.

```r
#find the Z-score of the 99th quantile of the standard normal distribution
qnorm(.99, mean = 0, sd = 1)
```

```
## [1] 2.326348
```

```r
#by default, R uses mean=0 and sd=1
qnorm(.99)
```

```
## [1] 2.326348
```

```r
#find the Z-score of the 95th quantile of the standard normal distribution
qnorm(.95)
```

```
## [1] 1.644854
```

```r
#find the Z-score of the 10th quantile of the standard normal distribution
qnorm(.10)
```

```
## [1] -1.281552
```

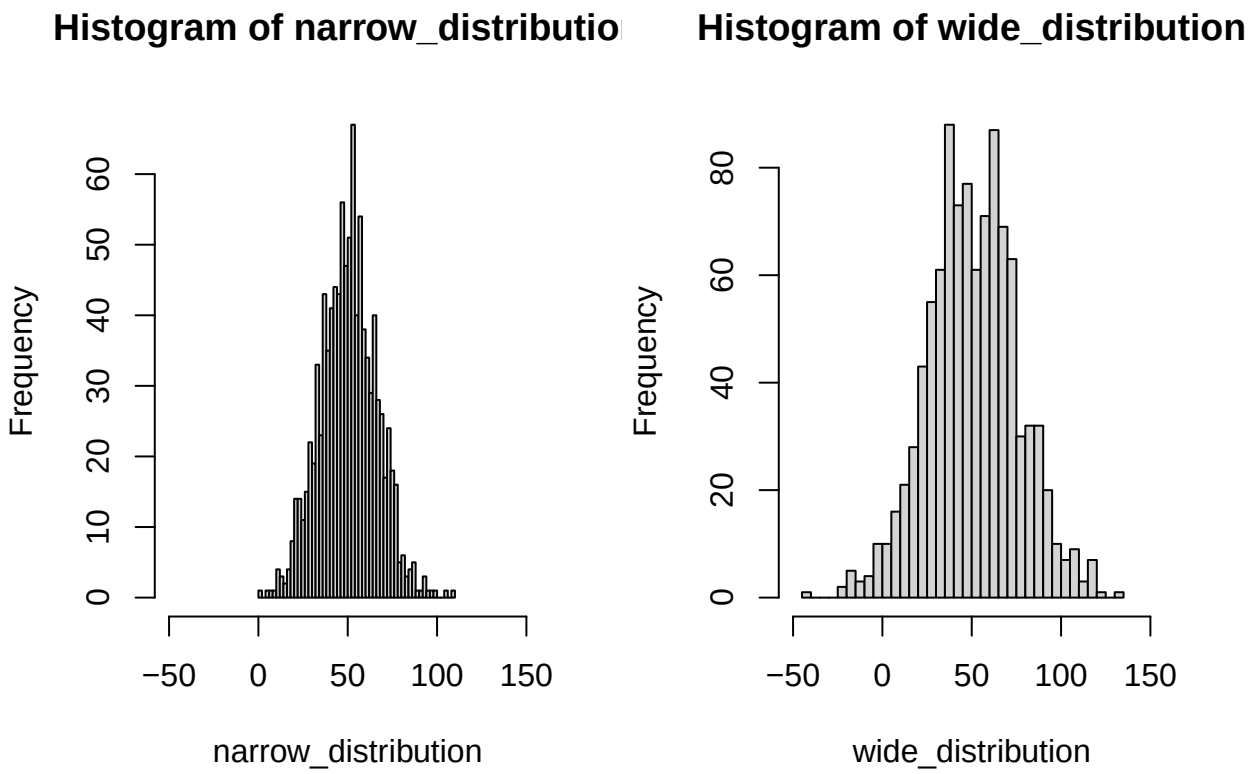
4.1.4 `rnorm(n, mean, sd)`

The function rnorm generates a vector of normally distributed random variables given a vector length `n`, a population mean μ and population standard deviation σ

```r
#generate a vector of 5 normally distributed random variables with mean=10
#and sd=2
five <- rnorm(5, mean = 10, sd = 2)
five
```

```
## [1] 10.660698 13.197321  8.710021 13.485950  7.572564
```

```
# [1] 10.658117 8.613495 10.561760 11.123492 10.802768

#generate a vector of 1000 normally distributed random variables with mean=50
#and sd=5
narrow_distribution <- rnorm(1000, mean = 50, sd = 15)

#generate a vector of 1000 normally distributed random variables with mean=50
# and sd=25
wide_distribution <- rnorm(1000, mean = 50, sd = 25)

#generate two histograms to view these two distributions side by side, specify
#50 bars in histogram and x-axis limits of -50 to 150
par(mfrow = c(1, 2)) #one row, two columns
hist(narrow_distribution, breaks = 50, xlim = c(-50, 150))
hist(wide_distribution, breaks = 50, xlim = c(-50, 150))
```



Notice how the wide distribution is much more spread out compared to the narrow distribution. This is because we specified the standard deviation in the wide distribution to be 25 compared to just 15 in the narrow distribution. Also notice that both histograms are centered around the mean of 50.

4.2 The binomial (Bernoulli) distribution

4.2.1 `dbinom(x, size, prob)`

The function dbinom returns the value of the probability density function (pdf) of the binomial distribution given a certain random variable x, number of trials (size) and probability of success on each trial (prob).

Put simply, dbinom finds the probability of getting a certain number of successes (`x`) in a certain number of trials (`size`) where the probability of success on each trial is fixed (`prob`).

Example:

Bob makes 60% of his free-throw attempts. If he shoots 12 free throws, what is the probability that he makes exactly 10?

```
#find the probability of 10 successes during 12 trials where the probability of
#success on each trial is 0.6
dbinom(x = 10, size = 12, prob = .6)
```

```
## [1] 0.06385228
```

The probability that he makes exactly 10 shots is **0.0639**.

4.2.2 `pbinom(q, size, prob)`

The function pbinom returns the value of the cumulative density function (cdf) of the binomial distribution given a certain random variable q, number of trials (size) and probability of success on each trial (prob).

Put simply, pbinom returns the area to the left of a given value q in the binomial distribution. If you're interested in the area to the right of a given value q, you can simply add the argument `lower.tail = FALSE` i.e, `pbinom(q, size, prob, lower.tail = FALSE)`

Example:
Ando flips a fair coin 5 times. What is the probability that the coin lands on heads more than 2 times?

```
#find the probability of more than 2 successes during 5 trials where the
#probability of success on each trial is 0.5
pbinom(2, size = 5, prob = .5, lower.tail = FALSE)
```

```
## [1] 0.5
```

```
#find the probability of 4 or fewer successes during 10 trials where the
#probability of success on each trial is 0.3
pbinom(4, size = 10, prob = .3)
```

```
## [1] 0.8497317
```

The probability that the coin lands on heads more than **2** times is **0.5**.

Example 2:
Suppose Tyler scores a strike on 30% of his attempts when he bowls. If he bowls 10 times, what is the probability that he scores 4 or fewer strikes?

```
#find the probability of 4 or fewer successes during 10 trials where the
#probability of success on each trial is 0.3
pbinom(4, size = 10, prob = .3)
```

```
## [1] 0.8497317
```

The probability that he scores 4 or fewer strikes is **0.8497**.

4.2.3 `qbinom(q, size, prob)`

The function qbinom returns the value of the inverse cumulative density function (cdf) of the binomial distribution given a certain random variable q, number of trials (size) and probability of success on each trial (prob).

Put simply, you can use qnorm to find out the p^{th} quantile of the binomial distribution.

```
#find the 10th quantile of a binomial distribution with 10 trials and prob
#of success on each trial = 0.4
qbinom(.10, size = 10, prob = .4)
```

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

```
#find the 40th quantile of a binomial distribution with 30 trials and prob
#of success on each trial = 0.25
qbinom(.40, size = 30, prob = .25)
```

```
## [1] 7
```

4.2.4 rbinom(n, size, prob)

The function rbinom generates a vector of binomial distributed random variables given a vector length n, number of trials (size) and probability of success on each trial (prob).

Example:

```
#generate a vector that shows the number of successes of 10 binomial
#experiments with
#100 trials where the probability of success on each trial is 0.3.
results <- rbinom(10, size = 100, prob = .3)
results
```

```
##  [1] 26 30 33 29 30 35 34 20 28 28
```

```
#find mean number of successes in the 10 experiments (compared to expected
#mean of 30)
mean(results)
```

```
## [1] 29.3
```

```
#generate a vector that shows the number of successes of 1000 binomial
#experiments
#with 100 trials where the probability of success on each trial is 0.3.
results <- rbinom(1000, size = 100, prob = .3)

#find mean number of successes in the 100 experiments (compared to expected
#mean of 30)
mean(results)
```

```
## [1] 29.897
```

Notice how the more random variables we create, the closer the mean number of successes is to the expected number of successes.

Note: "Expected number of successes" $= n \cdot p$ where n is the number of trials and p is the probability of success on each trial.

4.3 The geometric distribution

4.3.1 dgeom(x, prob)

The dgeom function finds the probability of experiencing a certain amount of failures before experiencing the first success in a series of Bernoulli trials given (x) the number of failures before the first success and (prob) the probability of success on each trial

A researcher is waiting outside of a library to ask people if they support a certain law. The probability that a given person supports the law is $p = 0.2$. What is the probability that the fourth person the researcher talks to is the first person to support the law?

```
dgeom(x = 3, prob = .2)
```

```
## [1] 0.1024
```

The probability that the researchers experiences 3 "failures" before the first success is 0.1024.

4.3.2 pgeom(q, prob)

The pgeom function finds the probability of experiencing a certain amount of failures or less before experiencing the first success in a series of Bernoulli trials ggiven (q) the number of failures before the first success and (prob) the probability of success on each trial.

A researcher is waiting outside of a library to ask people if they support a certain law. The probability that a given person supports the law is p = 0.2. What is the probability that the researcher will have to talk to more than 5 people to find someone who supports the law?

```
1 - pgeom(q = 5, prob = .2)
```

```
## [1] 0.262144
```

```
# same as
 pgeom(q = 5, prob = .2, lower.tail = FALSE)
```

```
## [1] 0.262144
```

The probability that the researcher will have to talk to more than 5 people to find someone who supports the law is **0.262144**.

4.3.3 qgeom(p, prob)

The qgeom function finds the number of failures that corresponds to a certain percentile given (p) the percentile and (prob) the probability of success on each trial.

A researcher is waiting outside of a library to ask people if they support a certain law. The probability that a given person supports the law is p = 0.2. We will consider a "failure" to mean that a person does not support the law. How many "failures" would the researcher need to experience to be at the 90th percentile for number of failures before the first success?

```
qgeom(p = .90, prob = 0.2)
```

```
## [1] 10
```

The researcher would need to experience **10** "failures" to be at the 90th percentile for number of failures before the first success.

4.3.4 rgeom(n, prob)

The rgeom function generates a list of random values that represent the number of failures before the first success given (n) the number of values to generate and (prob) the probability of success on each trial.

A researcher is waiting outside of a library to ask people if they support a certain law. The probability that a given person supports the law is $p = 0.2$. We will consider a "failure" to mean that a person does not support the law. Simulate 10 scenarios for how many "failures" the researcher will experience until she finds someone who supports the law.

```
set.seed(0) #make this example reproducible

rgeom(10, 0.2)
```

```
##  [1]  1  2  1 10  7  4  1  7  4  1
```

The way to interpret this is as follows:

- During the first simulation, the researcher experienced 1 failure before finding someone who supported the law.
- During the second simulation, the researcher experienced 2 failures before finding someone who supported the law.

- During the third simulation, the researcher experienced 1 failure before finding someone who supported the law.
- During the fourth simulation, the researcher experienced 10 failures before finding someone who supported the law.
- and so on..

5 Expectation and variance

5.1 Expected value and variance of common distributions

5.1.1 Geometric distribution

Let X be geometrically distributed with the parameter p. The expectation is then

$$E[X] = \frac{1}{p}$$

where p is the probability of success.

Let X be geometrically distributed with the parameter p. The variance is then

$$Var[X] = \frac{1-p}{p^2}$$

where p is the probability of success.

5.1.2 Exponential distribution

Let X be exponentially distributed with the parameter λ. The expectation is then

$$E[X] = \frac{1}{\lambda}$$

Let X be exponentially distributed with the parameter λ. The variance is then

$$Var[X] = \frac{1}{\lambda^2}$$

5.1.3 Normal distribution

Let X be a normally distributed random variable, or $X \sim N(\mu, \sigma 2)$. The expectation is then

$$E[X] = \mu$$

Let X be a normally distributed random variable, or $X \sim N(\mu, \sigma 2)$. The variance is then

$$Var[X] = \sigma^2$$

5.2 Expected value

To find the expected value of a probability distribution, we can use the following formula:

$$\mu \text{ or } E[X] = \sum_i a_i \cdot P(X = a_i) = \sum_i a_i \cdot p(a_i)$$

where a_i is the data value and $P(a_i)$ is the probability of that value.

Example:
Here we have a sample data set of the goal probability of a football team:

Table 1: Goal probability of a football team

Goals (X)	Probability P(X)
0	0.18
1	0.34
2	0.35
3	0.11
4	0.02

5.2.1 Method 1 - using `weighted.mean()`

The first method to calculate the expected value is using the `weighted.mean()` function as shown:

```
#define values
vals <- c(0, 1, 2, 3, 4)

#define probabilities
probs <- c(.18, .34, .35, .11, .02)

#calculate expected value
weighted.mean(vals, probs)
```

```
## [1] 1.45
```

5.2.2 Sample mean

We can calculate the sample mean using the `mean()` function like this:

```
#define values
vals <- c(3, -2, -5, 2, 5, 2, 5, -1, -3, 4, 2)

#calculate sample mean
mean(vals)
```

```
## [1] 1.090909
```

5.3 Variance

The variance is a way to measure how spread out data values are around the mean. The formula to find the variance of a population is as follows:

$$\sigma^2 \text{ or } Var(X) = E[X^2] - E[X]^2$$

If the generator of the random variable X is discrete with a probability mass function $(x_1 \rightarrow p_1, x_2 -> p_2 ... x_n -> p_n)$ then the variance can be calculated as such:

$$Var(X) = \sum_{i=1}^{n} p_i \cdot (x_i - \mu)^2$$

Where μ is the expected value i.e.:

$$\mu \text{ or } E[X] = \sum_i a_i \cdot P(X = a_i) = \sum_i a_i \cdot p(a_i)$$

where a_i is the data value and $P(a_i)$ is the probability of that value.

5.3.1 Sample variance

Suppose we have the following dataset:

Table 2: Sample dataset

2	4	4	7	8	12	14	15	19	22

We can then calculate the sample variance using the `var()` function:

```
#define dataset
data <- c(2, 4, 4, 7, 8, 12, 14, 15, 19, 22)

#calculate sample variance
var(data)
```

```
## [1] 46.01111
```

5.3.2 Population variance

Suppose we have the same dataset as previously we can then calculate the population variance by simply multiplying the sample variance by `(n-1)/n`:

```
data <- c(2, 4, 4, 7, 8, 12, 14, 15, 19, 22)

#determine length of data
n <- length(data)

#calculate population variance
var(data) * (n - 1) / n
```

```
## [1] 41.41
```

6 Joint distributions and independence

A joint probability is as the name implies the probability of 'joining' the probability of two events i.e., $P(A \cap B)$ (the intersection of A and B). Take for example this table below:

Table 3: Relation between hair color and eye color.

	Hair color		
	Fair/Red	Medium	Dark/black
Light	1168	825	305
Dark	573	1312	1200

To investigate the relations between hair color and eye colour, the hair color and eye color of 5383 was recorded. The data are given in Table 3. Eye color is encoded by the values 1 (Light) and 2 (Dark), and hair color by 1 (Fair/red), 2 (Medium), and 3 (Dark/black). By dividing the numbers in the table by 5383, the table is turned into a joint probability distribution for random variables X (hair color) taking values 1 to 3 and Y (eye color) taking values 1 and 2.

6.1 Joint probability distribution

We determine the joint probability density distribution by dividing the eye and hair color combinations by the total amount of records (5383) giving us this table (Table 4):

```
pmf <- matrix(round(c(1168, 825, 305, 573, 1312, 1200) / 5383, 3),
  byrow = TRUE, nrow = 2)
rownames(pmf) <- c("1 ", "2 ")
colnames(pmf) <- c("1", "2", "3")
dimnames(pmf) <- list("Y" = rownames(pmf), "X" = colnames(pmf))


kable(pmf,
  caption = "Joint probability distributions of X and Y", booktabs = T) %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 4: Joint probability distributions of X and Y

	1	2	3
1	0.217	0.153	0.057
2	0.106	0.244	0.223

6.2 Marginal probability distribution

In order to determine the marginal probability distribution of X and Y, we have to marginalize for Y and X respectively which we do by finding $P(X = x)$ and $P(Y = y)$ which is done by summing all the values that Y and X can take. See table 5

```
marg <- rbind(pmf, colSums(pmf))
marg2 <- cbind(marg, rowSums(pmf))
```

```
## Warning in cbind(marg, rowSums(pmf)): number of rows of result is not a multiple
## of vector length (arg 2)
```

```
rownames(marg2) <- c("1 ", "2 ", "Px ")
colnames(marg2) <- c("1", "2", "3", "Py ")
marg2[3, 4] <- 1

kable(marg2, format = "latex",
  caption = "Marginal probability distributions of X and Y", booktabs = T) %>%
  row_spec(2, extra_latex_after = "\\cline{1-5}") %>%
  column_spec(5, border_left = T) %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 5: Marginal probability distributions of X and Y

	1	2	3	Py
1	0.217	0.153	0.057	0.427
2	0.106	0.244	0.223	0.573
Px	0.323	0.397	0.280	1.000

6.3 Independece between two variables

In order to find out whether X and Y are independent, we have to check that $P(X = x, Y = y) = P(X = x)P(Y = y)$ for all x and y.
We'll start by checking for $x = 1$ and $y = 1$:

$$P(X = 1, Y = 1) = 0.217$$
$$P(X = 1)P(Y = 1) = 0.057 \cdot 0.323 = 0.018411$$

Because $P(X = x, Y = y) \neq P(X = x)P(Y = y)$. We can say that X and Y are dependent

7 Covariance and correlation

7.1 Covariance

Covariance is a measure of how changes in one variable are associated with changes in a second variable. Specifically, it's a measure of the degree to which two variables are linearly associated.

A covariance matrix is a square matrix that shows the covariance between many different variables. This can be a useful way to understand how different variables are related in a dataset.

7.1.1 Covariance between two variables

```
data <- data.frame(math = c(84, 82, 81, 89, 73, 94, 92, 70, 88, 95),
                   science = c(85, 82, 72, 77, 75, 89, 95, 84, 77, 94),
                   history = c(97, 94, 93, 95, 88, 82, 78, 84, 69, 78))

# create the covariance matrix for the two varibles "math" and "science"
cov(data[, c("math", "science")])
```

```
##             math  science
## math    72.17778 36.88889
## science 36.88889 62.66667
```

Explanation of the covariance matrix in the following section.

7.1.2 Covariance between multiple variables

```
# First, we'll create a data frame that contains the test scores of 10 different
# students for three subjects: math, science, and history.
data <- data.frame(math = c(84, 82, 81, 89, 73, 94, 92, 70, 88, 95),
                   science = c(85, 82, 72, 77, 75, 89, 95, 84, 77, 94),
                   history = c(97, 94, 93, 95, 88, 82, 78, 84, 69, 78))

# create the covariance matrix
cov(data)
```

```
##              math   science   history
## math     72.17778  36.88889 -27.15556
## science  36.88889  62.66667 -26.77778
## history -27.15556 -26.77778  83.95556
```

The numbers on the diagonal of the covariance matrix are the variances of the different variables in the data frame. Meaning that:

- The math scores have a variance of 72.177
- The science scores have a variance of 62.666

- The history scores have a variance of 83.955

The other values in the matrix represent the covariances between the various subjects. For example:

- The covariance between the math and science scores is 36.89
- The covariance between the math and history scores is −27.16
- The covariance between the science and history scores is −26.78

A positive number for covariance indicates that two variables tend to increase or decrease in tandem. For example, math and science have a positive covariance (36.89), which indicates that students who score high on math also tend to score high on science. Conversely, students who score low on math also tend to score low on science.

A negative number for covariance indicates that as one variable increases, a second variable tends to decrease. For example, math and history have a negative covariance (−27.16), which indicates that students who score high on math tend to score low on history. Conversely, students who score low on math tend to score high on history.

```
# if you want to make sure that you only calculate the correlation numerical
# variables you can use this function:
cor(data[, unlist(lapply(data, is.numeric))])
```

```
##               math    science    history
## math     1.0000000  0.5484986 -0.3488448
## science  0.5484986  1.0000000 -0.3691743
## history -0.3488448 -0.3691743  1.0000000
```

7.2 Correlation

One way to quantify the relationship between two variables is to use the Pearson correlation coefficient, which is a measure of the linear association between two variables. It always takes on a value between −1 and 1 where:

−1 indicates a perfectly negative linear correlation between two variables 0 indicates no linear correlation between two variables 1 indicates a perfectly positive linear correlation between two variables

7.2.1 Correlation between two variables

To calculate the correlation between two variables you can use the `cor(x, y)` function.

```
#create data frame
df <- data.frame(A = c(2, 3, 3, 5, 6, 9, 14, 15, 19, 21, 22, 23),
                 B = c(23, 24, 24, 23, 17, 28, 38, 34, 35, 39, 41, 43),
                 C = c(13, 14, 14, 14, 15, 17, 18, 19, 22, 20, 24, 26),
                 D = c(6, 6, 7, 8, 8, 8, 7, 6, 5, 3, 3, 2))
cor(df$A, df$B)
```

```
## [1] 0.9279869
```

This means that there is a strong positive linear correlation between the two variables, because it is almost 1 (1 means that there is a perfect positive linear correlation between two variables) 0.93

7.2.2 Correlation between multiple variables

You can calculate the correlation between a subset of the variables in your dataset e.g.:

```
cor(df[c("A", "B", "D")])
```

```
##            A          B          D
## A  1.0000000  0.9279869 -0.7915488
```

```
## B  0.9279869  1.0000000 -0.7917973
## D -0.7915488 -0.7917973  1.0000000
```

The way to interpret the output is as follows:

- The correlation between A and B is 0.9279869.
- The correlation between A and D is −0.7915488.
- The correlation between B and D is −0.7917973.

Meaning that A and D have a somewhat strong negative linear correlation. B and D also have some strong negative linear correlation and lastly A and B have a strong positive linear correlation.

You can also calculate the correlation all of the variables in your dataset e.g.:

```
cor(df)
```

```
##            A          B          C          D
## A  1.0000000  0.9279869  0.9604329 -0.7915488
## B  0.9279869  1.0000000  0.8942139 -0.7917973
## C  0.9604329  0.8942139  1.0000000 -0.8063549
## D -0.7915488 -0.7917973 -0.8063549  1.0000000
```

```
# if you want to make sure that you only calculate the correlation numerical
# variables you can use this function:
cor(df[, unlist(lapply(df, is.numeric))])
```

```
##            A          B          C          D
## A  1.0000000  0.9279869  0.9604329 -0.7915488
## B  0.9279869  1.0000000  0.8942139 -0.7917973
## C  0.9604329  0.8942139  1.0000000 -0.8063549
## D -0.7915488 -0.7917973 -0.8063549  1.0000000
```

8 Computations with random variables

It is not rare, that random variables are transformed through simple arithmetic operations. Consider ie. instead of being interested in the random variable X modelling the sum of two independent rolls with a die, we could be looking at the random variable $Y = 2X + 1$, which for each the values that can be obtained are determined by the sum of the two die rolls and are transformed in a way, that they are multiplied by 2 and added 1 to. Instead of reformulating the entire range Y is defined on and computing the probability distribution through the PMF and CDF, we would like to be able to compute the expectation of this new random variable fast and conveniently.

Luckily, the single-point summaries of distributions, the expectation and variance of random variables have nice properties when it comes to transformations.

8.1 Transforming (scaling and shifting) random variables

How does the mean and the standard deviation get affected if we were to add to the random variable, or if we were to scale the random variable?

X is our random variable. If a constant were added:

$Y = X + k$

Then the mean will the affected by a addition of k:

$\mu_Y = \mu_X + k$

BUT the standard deviation of Y will not be affected:

$\sigma_Y = \sigma_X$

If X is scaled by some constant k:

$Z = k \cdot X$

The both the mean and the standard deviation will be scaled by k:

$\mu_Z = k \cdot \mu_X$ and $\sigma_Z = k \cdot \sigma_X$

8.1.1 Change-of-Variable Formula

For any discrete random variable X and some defined function we can obtain the expected value of X through the following formula:

$$E[g(X)] = \sum_{x_k \in R_X} g(x_i) P(X = x_k)$$

And for the continuous case, we similarly observe:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx$$

The change-of-variable formula is the most generic way of recomputing the expectation for a transformed random variable, since it works with any kind of transformation function g. This is sensible, since all it does is applying the function to the former range of the random variable and incorporating this new range into the regular formulas for computing expectations.

8.1.2 Linearity of Expectation (LOTUS)

However, for linear transformations of the type $f(x) = ax + b$, we can use the LOTUS-formula to show that:

$$E[ax + b] = a \cdot E[X] + b$$

This allows us to recompute the expectation of a linearly transformed random variable simply by transforming the computed value of the expectation itself. Note that since $E[aX] = E[X_1] + E[X_2] + ... + E[X_a]$, then the total expectation of independent trials of the same random phenomena modelled through X is simply the sum of each single expectation. Adding a constant to some random variable $E[X + b]$ simply shifts the range $R_X$by the constant, but leaves the probability distribution untouched. Then, the expectation also simply shifts by that constant $E[X] + b$.

8.1.3 Jensen Inequality

The inequality says, that if you are given a convex function and a random variable X then this inequality holds:

$$f(E[X]) \leq E[f(x)]$$

the LHS of the formula is telling you how to calculate this value. You draw n values of X and you calculate their average (of n), then you take that average and you pass it into our convex function, which is given us back a number. That number is the LHS (output of the average input). On the RHS, you draw n samples of X, but this time we pass each into the function, giving us many outputs, and then we take the average of those outputs, giving us the RHS (the average output).

So Jensen Inequality states, that the output of the average input is less than (or equal to) the average output.

A convex function

When a function is a convex function, then the epigraph (the space of points above the funciton) is a convex set. Which means if you pick any two points in that epigraph and then draw a line between them, then every point along that line is also within that epigraph.

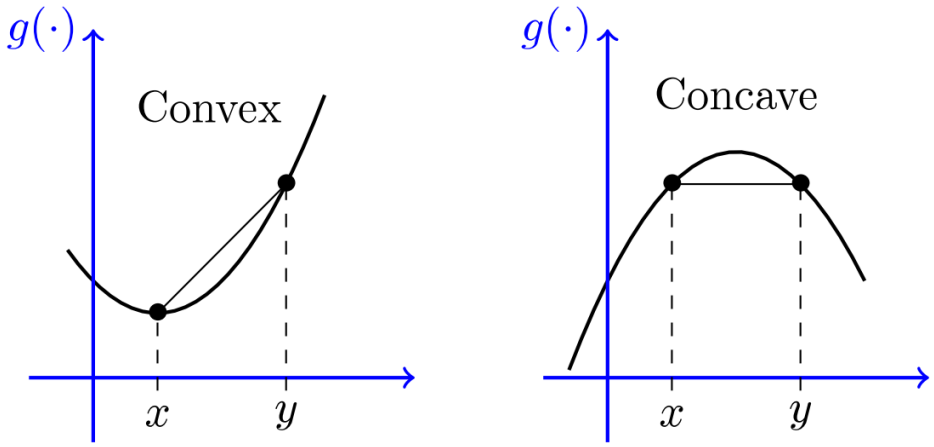


Figure 2: Concave and convex functions

9 Law of large numbers and Central limit theorem

9.1 The law of large numbers

We will now turn our attention to one of the most central - and likewise most intuitive - laws in statistics that relates probability theory (so the theoretical, mathematical background of randomness and chance) to real-world phenomena. The law of large numbers gives us the statistical foundation to be able to relate real-world data to properties of theoretical distributions we have observed.

In that sense, the **law or large numbers** is our way of coping with random variation in random phenomenon, which we can eliminate from our measurements by repetition.

In probability theory, the law of large numbers (LLN) is a theorem that describes the result of performing the same experiment a large number of times. According to the law, the **average of the results** obtained from a large number of trials should be close to the **expected value** and will tend to become closer to the expected value as more trials are performed.

As the number of trials/observations increases, the actual/observed probability approaches the theoretical/expected probability.

Example: Let's say that you flip a fair coin. The theoretical probability is 50% that you get heads. But if you flip a coin 10 times you are not guaranteed to get 5 heads. But is you flip a coin 10 times, and repeat that 1000 times, the observed proportion of heads will get closer and closer to 50%.

Gambler's Fallacy: A common misconception is that you expect a certain value to come. If you flip a coin 10 times, and the first 9 flips were tails, you would think that the next flip should be a heads (because the theoretical probability is 50/50) but that is not the case. Each flip of the coin is an independent event, and the outcome is unaffected by the previous event.

> In probability theory, the law of large numbers (LLN) is a theorem that describes the result of performing the same experiment a large number of times. According to the law, the **average of the results** obtained from a large number of trials should be close to the **expected value** and will tend to become closer to the expected value as more trials are performed.

Proof of correctness: To provide statistical reasoning for why this rule - the *law of large numbers (LLN)* - holds, we consider the experiments as a sequence of random variables $X_1, X_2, ..., X_n$, where X_i is the

specific result or outcome of the ith repetition of our experiment. Each of the random variables X_i has some expectation and standard deviation, which we denote as μ and σ. We confine ourselves to the situation where the experimental conditions and the subsequent experiments are identical, ie. we don't change the conditions of the experiment, and that the outcomes of some experiment doesn't influence the others, ie. we assume independence.

We call such a sequence an **independent and identically distributed (i.i.d) sequence**.

We can compute the average of this sequence through our standard method of averaging, i.e. we sum over the realisation of all random variables and divide by the number of repetitions:

$$\bar{X_n} = \frac{X_1 + X_2 + ... + X_n}{n}$$

$\bar{X}_n$ in itself is a random variable, since for a next n repetitions, we would assume a different value for $\bar{X}_n$. We can compute the expectation and variance of this new random variable.

Expectation

$$\begin{aligned} E[\bar{X}_n] &= E[\frac{1}{n}(X_1 + X_2 + ... + X_n)] \\ &= \frac{1}{n}E[X_1 + X_2 + ... + X_n] \\ &= \frac{1}{n}(E[X_1] + E[X_2] + ... + E[X_n]) \\ &= \frac{1}{n}(\mu + \mu + ... + \mu) \\ &= \frac{n \cdot \mu}{n} = \mu \end{aligned}$$

We see: When performing identical experiments a large number of times, the expected average value of the random experiment is equal to the expectation of each single random variable. Sometimes there will be values above our expectation, sometimes below - but if we average over it, we assume to obtain the same value again.

Variance

$$\begin{aligned} Var[\bar{X}_n] &= Var[\frac{1}{n}(X_1 + X_2 + ... + X_n)] \\ &= \frac{1}{n^2}Var[X_1 + X_2 + ... + X_n] \\ &= \frac{1}{n^2}(Var[X_1] + Var[X_2] + ... + Var[X_n]) \\ &= \frac{1}{n^2}(\sigma^2 + \sigma^2 + ... + \sigma^2) \\ &= \frac{n \cdot \sigma^2}{n^2} = \frac{\sigma^2}{n} \end{aligned}$$

For the variance we have found an even more interesting property. The variance in the average of the realisation of a sequence of random variables, decreases the bigger the sequence, i.e. its standard deviation is less than that of a single realisation by a factor of $\sqrt{n}$. This property of the variance is what we use to proof the law of large numbers: The more often we repeat some experiment, the smaller the spread of the averaged results. \end{ocg}

9.2 Chebyshev's inequality

We have shown that increasing values of n in our random variable $\bar{X}_n$, which averages over the results of some random variable, result in a decreasing variance. If we were to plot the distribution of our new random variable for increasing values of n, we would observe a contraction of the probability mass around the expectation (less variance $\rightarrow$ less spread out probability mass). We will now learn about a tool, that will provide us with the final step to prove the law of large numbers formally.

Chebyshev's Inequality is a way to quantify the bound of the probability that any random variable Y is outside some symmetric interval of width $2a$ around its expectation. So outside the symmetric interval of: $[E[X] - a, E[X] + a]$. Chebyshev proved the following upper bound:

$$P(|Y - E[Y]| > a) \leq \frac{1}{a^2} \cdot Var[Y]$$

To explain Chebyshev's upper bound, it means the probability that the realisation of some random variable Y deviates more than a from its mean is **ALWAYS** lower than $\frac{1}{a^2} \cdot Var[Y]$. This is intuitive since we would expect the probability to get lower for bigger intervals (bigger values of a) and the probability to get higher for the higher variances.

The Chebyshev Inequality provides us with powerful tools to make statements about the probability mass of any random variable. If we want to say something about the probability that our random variable is within a few standard deviations we can employ Chebyshev's inequality.

9.2.1 Example of usage of Chebyshev's Inequality

Let's say at a factory we automatically fill up boxes with screws. The mean of the number of screws per box is 1000 and the variance is 25.

$\mu = 1000$ and $\sigma^2 = 25$ $\sigma = 5$. This variance is a problem since customers complain if they don't get 1000 screws but maybe get 992 screws and it is also a problem the other way around, since you are giving away free screws if the box has 1020 screws.

a) How many σ-units to the right of μ is 1009? So how many standard deviations away from the mean (to the right) is 1009?

There is no information given on the distribution of the data, so we cant expect a perfect normal distribution.

Useful formulas:

To convert a specific value of X into σ -units: $k = \frac{X-\mu}{\sigma}$ To convert a σ - unit to a specific value of X: $X = \mu + k\sigma$ Here k is the number of σ-units.

For problem a) we use the first formula.

$$k = \frac{1009 - 1000}{5} = 1.8$$

So this tells us that 1009 is 1.8 standard deviations away from the mean.

b) What X value is 2.6 σ-units to the left of μ?

Here we use the second formula:

$$X = 1000 - 2.6 \cdot 5 = 1000 - 13 = 987$$

This means that 2.6 standard deviation to the left of the mean lies the value: 987 screws.

Important: since the exercise asks for standard-deviation to the left, we insert a negative number. If they asked for standard-deviation to the right, we would add it, i.e.:

$$X = 1000 + 2.6 \cdot 5 = 1000 + 13 = 1013$$

c) Use Chebyshev's Inequality to find a bound on $P[994 < X < 1006]$

As we can see, the interval $[994 < X < 1006]$ is symmetric around the mean by 6. This means we can use Chebyshev's Inequality.

So first off we use the first formula to calculate how many standard deviations the two numbers are from the mean:

$$k = \frac{1006 - 1000}{5} = 1.2$$

Now we apply the second formula into Chebyshev's Inequality:

$$P(1000 - (1.2)(5) < X < 1000 + (1.2)(5)) \geq 1 - \frac{1}{(1.2)^2}$$

This is the same as just writing:

$$P(994 < X < 1006) \geq 1 - \frac{1}{(1.2)^2} = 0.3056$$

This means that the probability that X will lie between 994 and 1006 is AT LEAST 0.3056, note that you don't know exactly what the probability is, but Chebyshev's Inequality provides a bound.

9.3 Central Limit Theorem

The central limit theorem (CLT) states that the distribution of sample means approximates a normal distribution as the sample size gets larger, regardless of the population's distribution.

The central limit theory is a refinement of the law of large numbers. Again, we consider the average $\bar{X}_n$ over a series of independent, identically distributed (i.i.d) random variables $X_1, X_2, ..., X_n$ By the law of large numbers, we know that this average converges towards the true mean of the distribution $E[\bar{X}_n] = \mu$ for large n, which we were able to prove by applying the limit on Chebyshev's inequality.

The central limit theorem now extends this argument even further, stating that for a random variable modelling any kind of theoretical distribution, the average over n values, so $\bar{X}_n$ after normalisation converges towards the standard normal distribution.

9.3.1 An example of applying CLT to find a probability

Example: The average age at first marriage is 25 for women and 27.8 for men. If the standard deviation for women is 4 years, what is the probability that **a random selection of 32 women** have an average age at first marriage between 26 and 27?

The first thing we want to look at is the distribution. Since the problem, is a probability problem about an average that is derived from 32 women (so an average $\bar{X}$ derived from a random selection of women), then we can use the Central Limit Theorem. This tells us that $\bar{X}$ is normally distributed when the sample size is large (here we sampled 32 women, so we count that as a large n). So now we now we can use a Normal Distribution to calculate the problem.

We are only looking at a bell curve for the women, so we forget about the information given about the men.

So the μ for our normal distribution is 25.

IT IS IMPORTANT TO NOTE THAT we are not dealing with a normal X value, which represent the individual women's age at marriage. We are dealing with a sample of 32 women, so it not X we are dealing with, it is $\bar{X}$. It is not 1 randomly selected woman, it is 32 randomly selected women.

So, the mean for $\bar{X}$ is the same as μ. BUT the standard deviation is not just 4 we have to adjust it: $\sigma_{\bar{X}} = \frac{\sigma}{\sqrt{n}}$, so for out normal distribution we have:

$$\mu_{\bar{X}} = 25$$
$$\sigma_{\bar{X}} = \frac{4}{\sqrt{32}} = 0.7071068$$

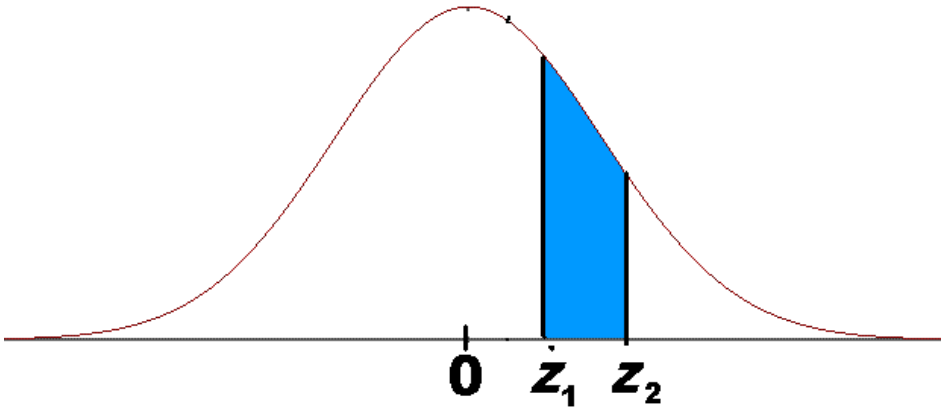


Figure 3: Normal distribution with marked area

So now we want to know the probability that the average lies between 26 and 27. (see drawing to the right, $z_1 = 26$ and $z_2 = 27$)

To calculate this we start off by calculating the z-scores of 26 and 27. recall that $z = \frac{X-\mu}{\sigma}$ and in this case $z = \frac{\bar{X}-\mu_{\bar{X}}}{\sigma_{\bar{X}}}$

$$\text{z value for } 26 = \frac{26-25}{0.7071068} = 1.41 \text{z value for } 27 = \frac{27-25}{0.7071068} = 2.83$$

You can either look it up at a z-table, or simply plug it into R:

```
pnorm(2.83) - pnorm(1.41)
[1] 0.07694244
```

You can also skip the z-score step and plug it into R like this:

```
pnorm(27, mean=25, sd=0.7071068) - pnorm(26, mean=25, sd=0.7071068)
[1] 0.07631074
```

So, $P(26 < \bar{X} < 27) = 0.07631 = 7.6\%$

10 Exploratory data analysis - Graphical summaries

10.1 Histograms

Histograms are useful graphical way of representing a data set and allows you to spot patterns that were otherwise not visible in the raw dataset.

Take for example this dataset of randomly generated numbers sampled from a normal distribution:

Table 6: Randomly generated normal dataset

0.8747092	1.0367287	0.8328743	1.3190562	1.0659016	0.8359063	1.0974858	1.1476649	1.1151563	0.9389223
1.3023562	1.0779686	0.8757519	0.5570600	1.2249862	0.9910133	0.9967619	1.1887672	1.1642442	1.1187803
1.1837955	1.1564273	1.0149130	0.6021297	1.1239651	0.9887743	0.9688409	0.7058495	0.9043700	1.0835883
1.2717359	0.9794425	1.0775343	0.9892390	0.7245881	0.9170011	0.9211420	0.9881373	1.2200051	1.1526351
0.9670953	0.9493277	1.1393927	1.1113326	0.8622489	0.8585010	1.0729164	1.1537066	0.9775308	1.1762215
1.0796212	0.8775947	1.0682239	0.7741274	1.2866047	1.3960800	0.9265557	0.7911731	1.1139439	0.9729891
1.4803236	0.9921520	1.1379479	1.0056004	0.8513454	1.0377585	0.6390083	1.2931110	1.0306507	1.4345223
1.0951019	0.8580107	1.1221453	0.8131805	0.7492733	1.0582892	0.9113416	1.0002211	1.0148683	0.8820958
0.8862663	0.9729643	1.2356174	0.6952866	1.1187892	1.0665901	1.2126200	0.9391632	1.0740038	1.0534198
0.8914960	1.2415736	1.2320805	1.1400427	1.3173667	1.1116973	0.7446816	0.8853469	0.7550775	0.9053199

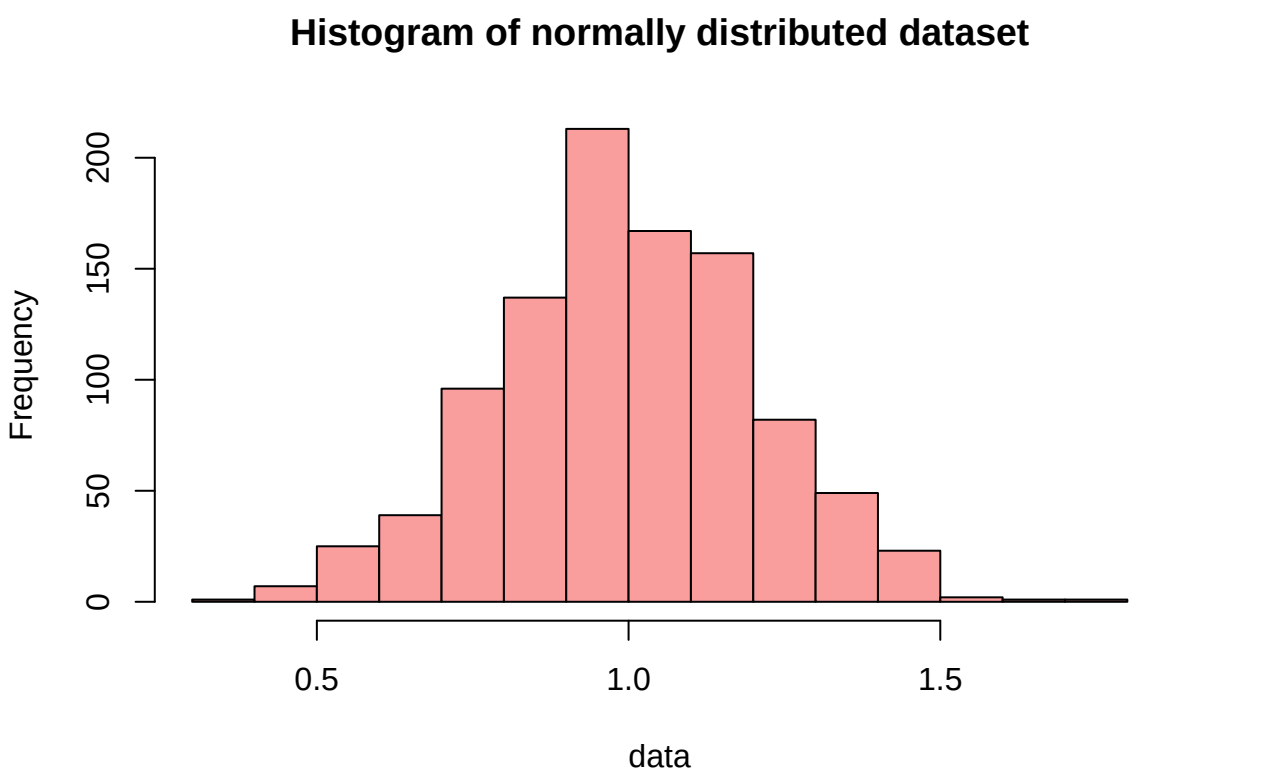
Just staring at the dataset for a while tells us very little. To see something useful, we can plot it as a histogram:

Now this can be done in two ways: 1. Using the **base R** function `hist(data, ...)` 2. Using **ggplot2** to plot the histogram

10.1.1 Base R `hist(data)`

To see all the available arguments that you can pass to the `hist()` function simply write `?hist` in the `R` console

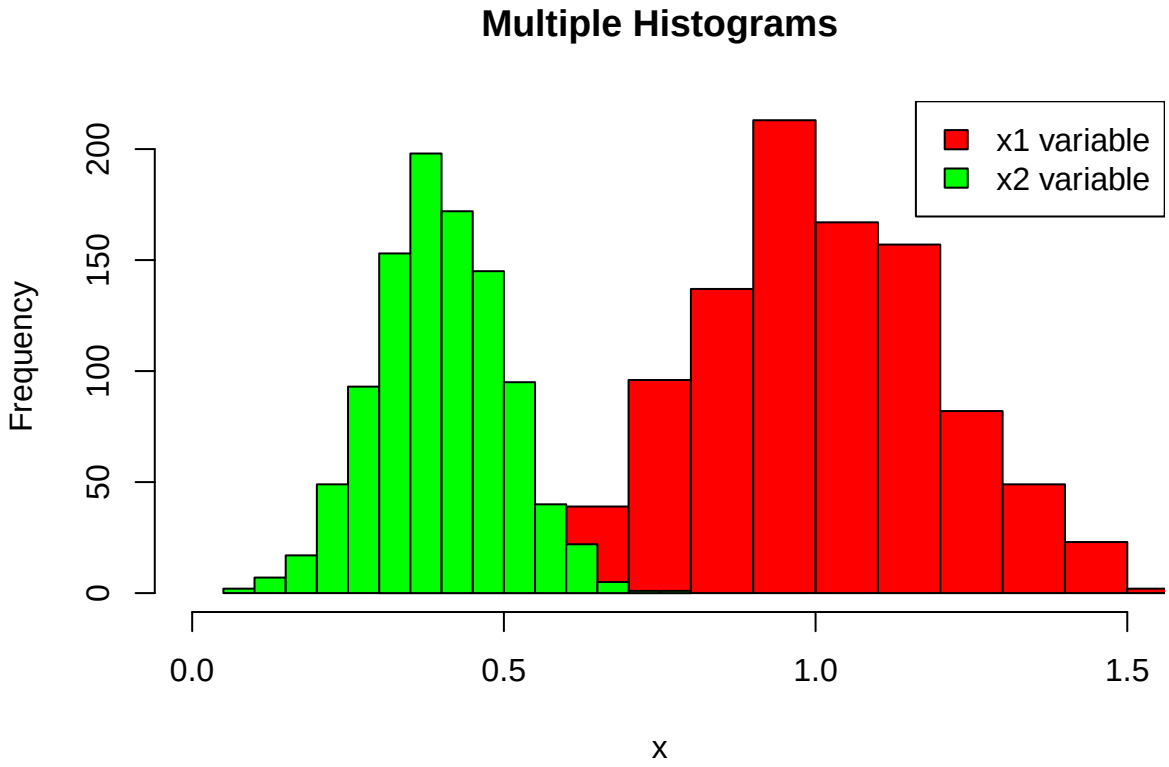
```
set.seed(1)
data <- rnorm(1000, mean = 1, sd = .2)
hist(data, col = "#f99d9d",
  main = "Histogram of normally distributed dataset")
```



10.1.2 Plotting multiple histograms on one plot with `hist(data)`

To plot multiple histograms in with `Base R` one plot one can do as follows:

```
#make this example reproducible
set.seed(1)

#define data
x1 <- rnorm(1000, mean = 1, sd = 0.2)
x2 <- rnorm(1000, mean = 0.4, sd = 0.1)

#plot two histograms in same graph
hist(x1, col = "red", xlim = c(0, 1.5),
    main = "Multiple Histograms", xlab = "x")
hist(x2, col = "green", add = TRUE)

#add legend
legend("topright", c("x1 variable", "x2 variable"), fill = c("red", "green"))
```

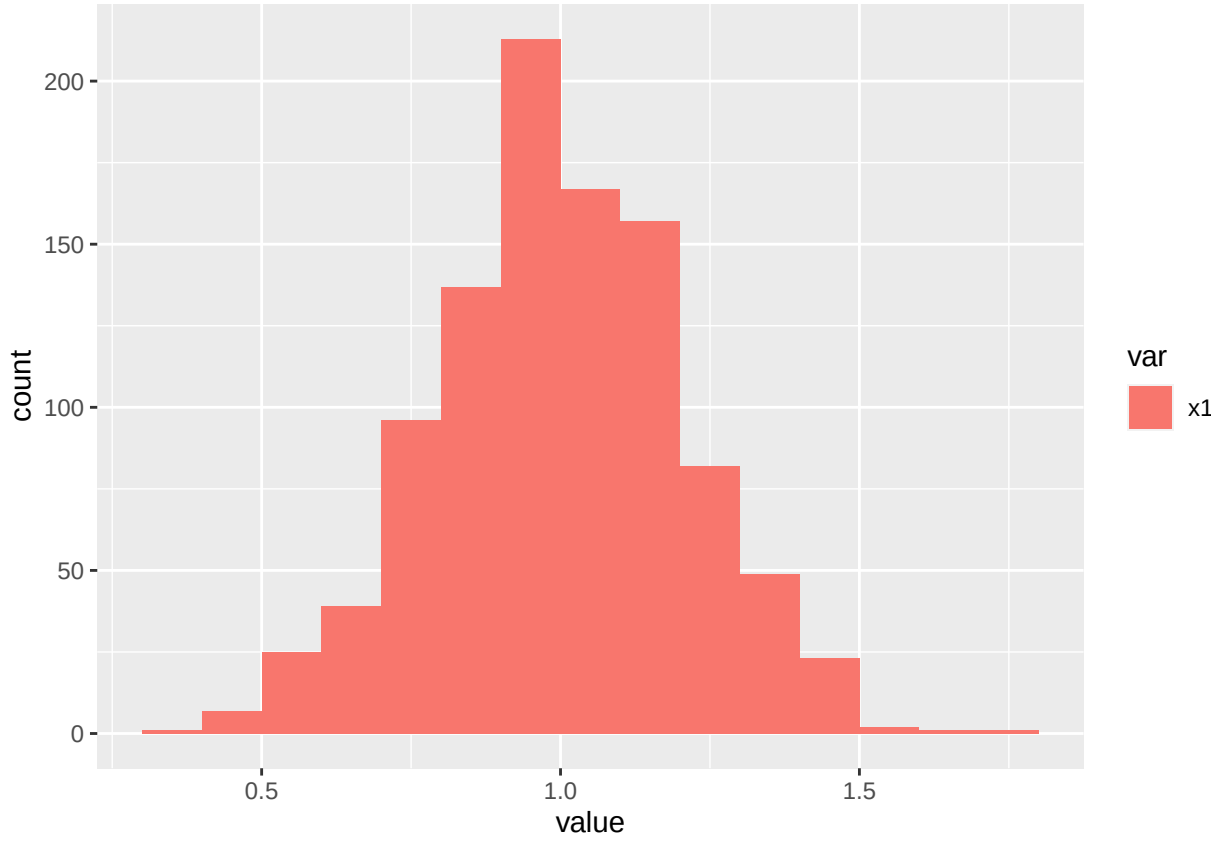


10.1.3 ggplot2 `ggplot(data)`

Now to use `ggplot` to plot your data you have to first make it into a data frame:

```
set.seed(1)
df <- data.frame(var = rep("x1", 1000),
          value = c(rnorm(1000, mean = 1, sd = .2)))
brx <- pretty(range(df$value),
    n = nclass.Sturges(df$value), min.n = 1)
    # the brx function is only used to make it look like the base R
    # hist function which uses 'Sturges rule' to determine binsize i.e, 'breaks'
    # the default binsize for geom_histogram is 30.
```

```
ggplot(df, aes(x = value, fill = var)) + geom_histogram(breaks = brx)
```

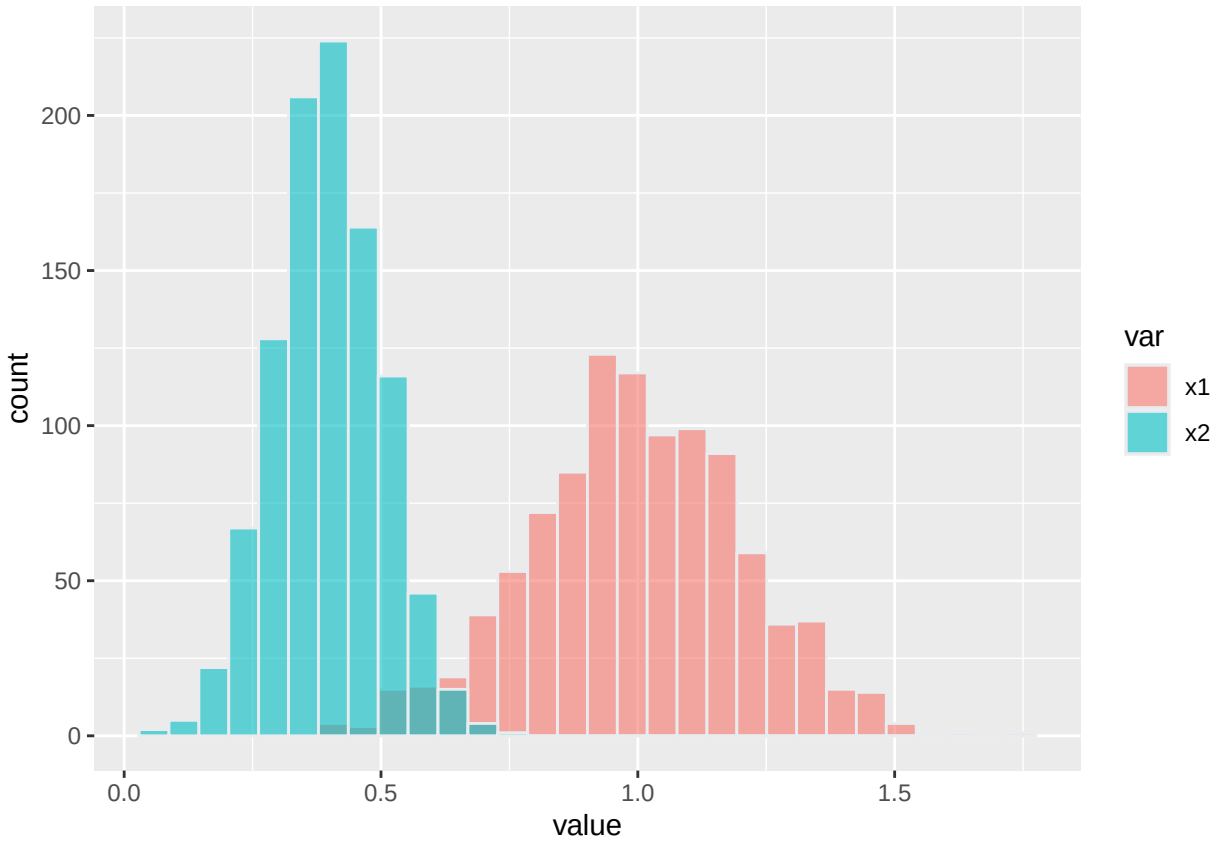


10.1.4 Plotting multiple histograms on one plot with `ggplot(data)`

To plot multiple histograms in with `ggplot2` one plot one can do as follows:

```
#make this example reproducible
set.seed(1)

#create data frame
df <- data.frame(var = c(rep("x1", 1000), rep("x2", 1000)),
                 value = c(rnorm(1000, mean = 1, sd = .2),
                           rnorm(1000, mean = 0.4, sd = 0.1)))

#plot multiple histograms
ggplot(df, aes(x = value, fill = var)) +
  geom_histogram(color = "#e9ecef", alpha = 0.6, position = "identity")
```

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


count
200
150
100
50
0
0.0
0.5
1.0
1.5
value
var
x1
x2

10.2 Kernel density estimates

We can graphically represent data in a more variegated plot by a so-called kernel density estimate.

The idea behind the construction of the plot is to "put a pile of sand" around each element of the dataset. At places where the elements accumulate, the sand will pile up. The actual plot is constructed by choosing a *kernel* K and a *bandwidth* h.

The *kernel* K reflects the shape of the piles of sand, whereas the bandwidth is a tuning parameter that determines how wide the piles of sand will be.

Formally, a kernel K is a function $K : \mathbb{R} \to \mathbb{R}$.

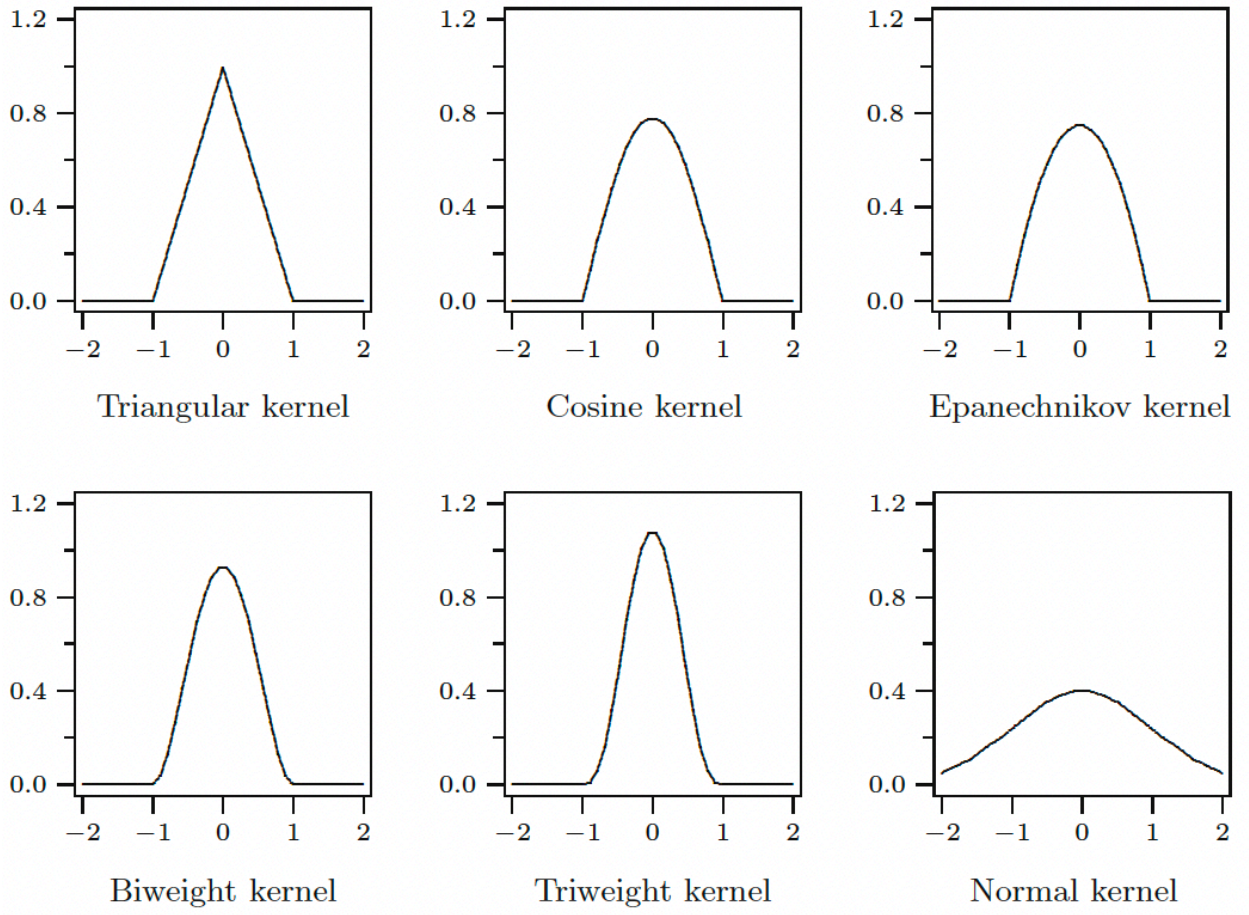


Figure 4: Examples of well-known kernels K

To plot a kernel density estimate you can use either the `densityplot` function from the library `lattice` or you can use the **base R** function `plot` with the parameter `density()`.

For simplicity I will choose to use the **base R** function and using the diamonds dataset from the `UsingR`.

```
library(UsingR)
```

```
kernels <- c("triangular", "cosine", "epanechnikov",
    "biweight", "rectangular", "gaussian")
colors <- brewer.pal(7, "Dark2")
par(mfrow = c(2, 3))
for (kernel in seq_len(length(kernels))) {
```

```
    title <- stringr::str_to_title(kernels[kernel])
    plot(density(diamond$price, bw = 10000, kernel = kernels[kernel]),
    main = paste(title, "kernel"), col = colors[kernel], lwd = 2)
}
```

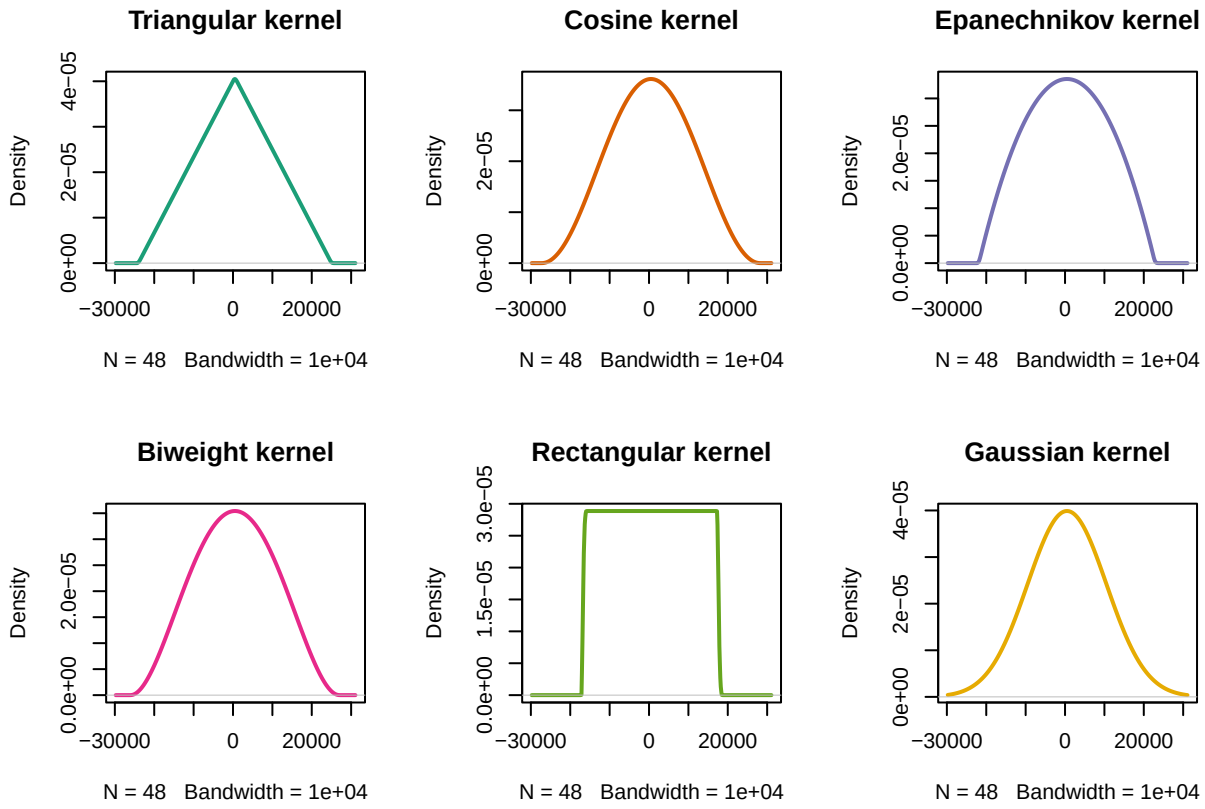


Figure 5: Examples of well-known kernels K in R

Note: R also has the kernel 'optcosine' which is apparently less smooth than 'cosine' see R documentation. It looks almost the same as 'cosine' but with a bit sharper 'hinge' at the beginning of the curve (almost like the 'triangular' kernel). Did not feel it was significant enough to include, but try it for yourself.

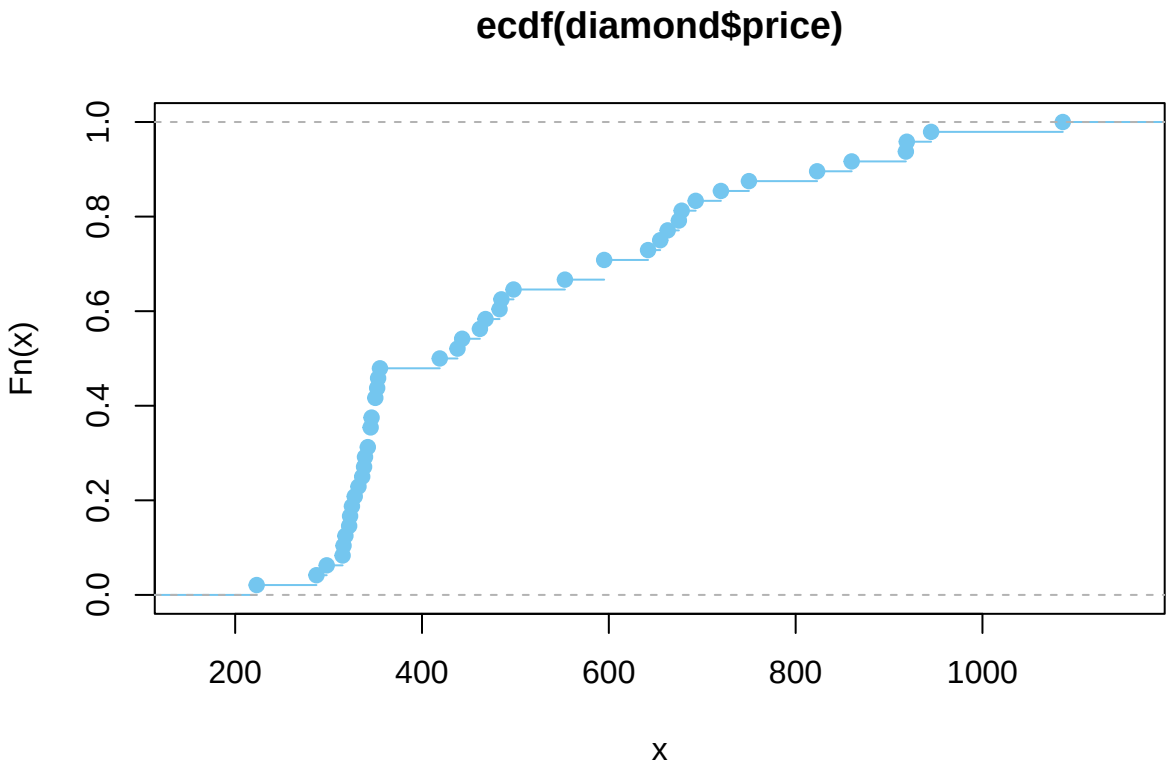
The `bw` argument specifes the bandwith that wil be used to create the kernel density estimate. The `kernel` argument specifies the kernel that will be used

10.3 Empirical distribution function

Another way to graphically represent a dataset is to plot the data in a cumulative manner. This can be done using the empirical cumulative distribution function of the data.

In R this can be done like this:

```
plot(ecdf(diamond$price), col = "#74c6ef")
```



10.4 Scatterplot

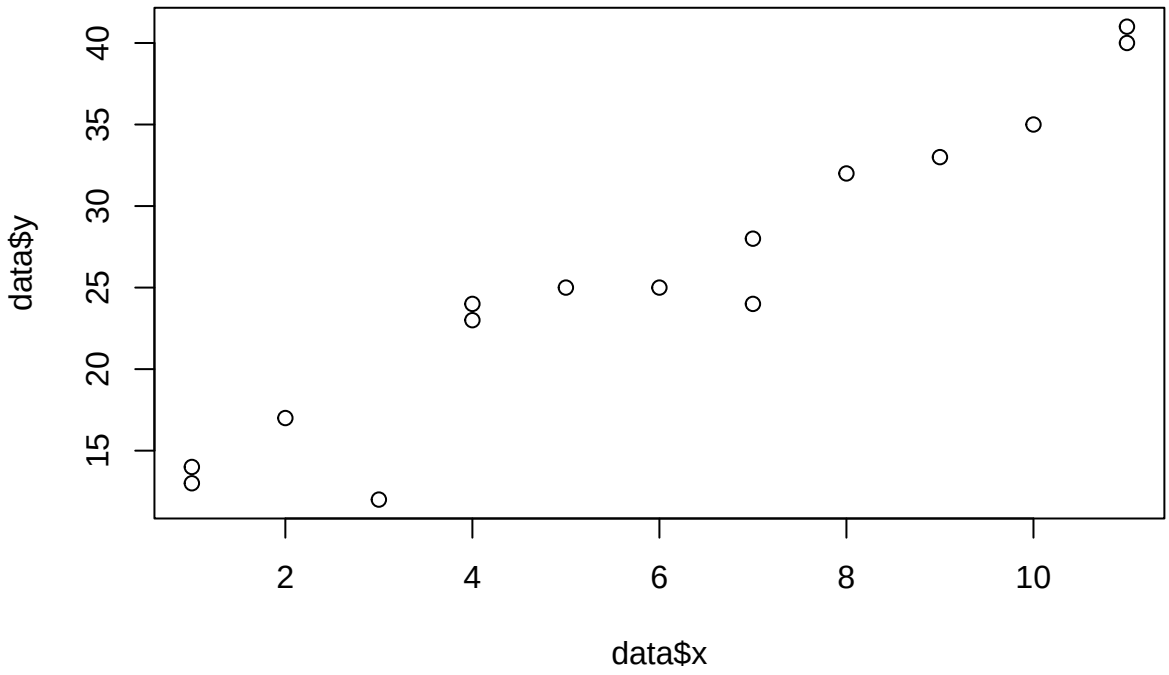
In some situations one wants to investigate the relationship between two or more variables. In the case of two variables x and y, the dataset consists of *pairs of observations*:

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

We call such a dataset a bivariate dataset in contrast to the univariate dataset, which consists of observations of one particular quantity. We often like to investigate whether the value of variable y depends on the value of the variable x, and if so, whether we can describe the relation between the two variables. A first step is to take a look at the data, i.e., to plot the points (x_i, y_i) for $i = 1, 2 \dots, n$. Such a plot is called a scatterplot.

In `R` this can be acheived by simply using the `plot()` function:

```
#create some fake data
data <- data.frame(x = c(1, 1, 2, 3, 4, 4, 5, 6, 7, 7, 8, 9, 10, 11, 11),
                   y = c(13, 14, 17, 12, 23, 24, 25, 25, 24, 28, 32, 33, 35, 40, 41))

#create scatterplot of data
plot(data$x, data$y)
```



11 Exploratory data analysis - Numerical summaries

11.1 Middle of a dataset (sample mean and median)

Both methods have pros and cons. The sample mean is the natural analogue for a dataset of what the expectation is for a probability distribution. However, it is very sensitive to outliers, by which we mean observations in the dataset that deviate a lot from the bulk of the data.

See the section *Sample mean* for how to calculate it in `R`.

11.1.1 Sample median

We can calulate the sample median using the `median()` function like this:

```
# define values
vals <- c(3, -2, -5, 2, 5, 2, 5, -1, -3, 4, 2)

#calculate sample median
median(vals)
```

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

11.2 The amount of variability of a dataset

See the section *Sample variance* for how to calculate it in `R`.

11.2.1 Sample standard deviation

We can calculate the sample standar deviation using the `sd()` function like this:

```
#create dataset
data <- c(1, 3, 4, 6, 11, 14, 17, 20, 22, 23)
```

```r
#find standard deviation
sd(data)
```

```
## [1] 8.279157
```

Note that you must use na.rm = TRUE to calculate the standard deviation if there are missing values in the dataset:

```r
#create dataset with missing values
data <- c(1, 3, 4, 6, NA, 14, NA, 20, 22, 23)

#attempt to find standard deviation
sd(data)
```

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

```r
#find standard deviation and specify to ignore missing values
sd(data, na.rm = TRUE)
```

```
## [1] 9.179753
```

A more robust measure of variability is the median of absolute deviations or MAD. The median absolute deviation measures the spread of observations in a dataset.

It's a particularly useful metric because it's less affected by outliers than other measures of dispersion like standard deviation and variance.

The formula to calculate median absolute deviation, often abbreviated MAD, is as follows:

$$MAD = median(|x_i - x_m|)$$

where:

- x_i: The ith value in the dataset
- x_m: The median value in the dataset

The following examples shows how to calculate the median absolute deviation in R by using the built-in `mad()` function.

The following code shows how to calculate the median absolute deviation for a single vector in R:

```r
#define data
data <- c(1, 4, 4, 7, 12, 13, 16, 19, 22, 24)

#calculate MAD
mad(data)
```

```
## [1] 11.1195
```

11.3 Empirical quantiles, quartiles, and the IQR

Instead of identifying only the center of the dataset, *Tukey* suggested to give a five-number summary of the dataset: the minimum, the maximum, the sample median, and the 25th and 75th empirical percentiles.

The 25th empirical percentile $q_n(0.25)$ is called the *lower quartile* and the 75th empirical percentile $q_n(0.75)$ is called the *upper quartile*.

Together with the median, the lower and upper quartiles divide the dataset in four more or less equal parts consisting of about one quarter of the number of elements.

11.3.1 `quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE)`

In statistics, quantiles are values that divide a ranked dataset into equal groups.

The `quantile()` function in `R` can be used to calculate sample quantiles of a dataset where:

- `x`: Name of vector
- `probs`: Numeric vector of probabilities
- `na.rm`: Whether to remove NA values

```
#define vector of data
data <- c(1, 3, 3, 4, 5, 7, 8, 9, 12, 13, 13, 15, 18, 20, 22, 23, 24, 28)

#calculate quartiles
quantile(data, probs = seq(0, 1, 1 / 4))
```

```
##   0%  25%  50%  75% 100%
##  1.0  5.5 12.5 19.5 28.0
```

```
#calculate quintiles
quantile(data, probs = seq(0, 1, 1 / 5))
```

```
##   0%  20%  40%  60%  80% 100%
##  1.0  4.4  8.8 13.4 21.2 28.0
```

```
#calculate deciles
quantile(data, probs = seq(0, 1, 1 / 10))
```

```
##   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##  1.0  3.0  4.4  7.1  8.8 12.5 13.4 17.7 21.2 23.3 28.0
```

```
#calculate random quantiles of interest
quantile(data, probs = c(.2, .5, .9))
```

```
##  20%  50%  90%
##  4.4 12.5 23.3
```

The distance between the upper and lower quartiles is called the interquartile range, or IQR. You can also get the five number summary in `R` by using the `summary()` function:

```
summary(data)
```

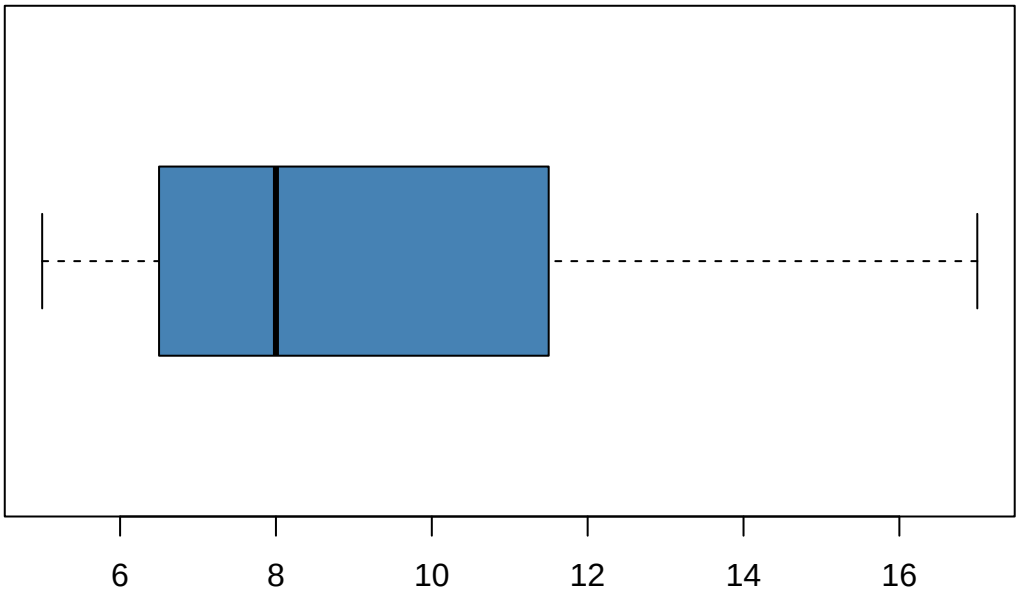
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    5.50   12.50   12.67   19.50   28.00
```

Which as you can see is the same as the quartiles calculated above. This five number summary can also be visualized as a boxplot.

11.3.2 Boxplots

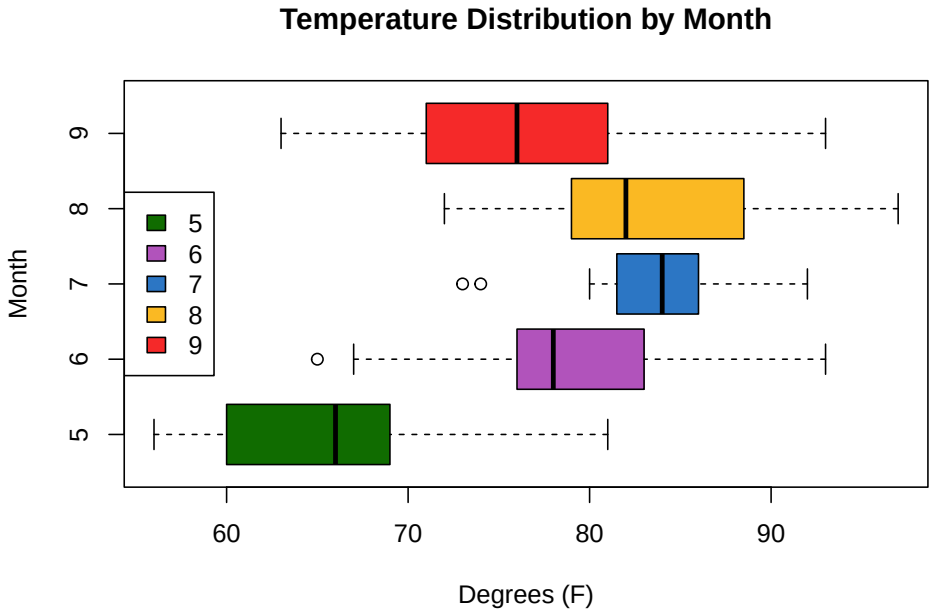
To create a boxplot in `R` you can simply use the `boxplot()` function

```
#create data
df <- data.frame(points = c(7, 8, 9, 12, 12, 5, 6, 6, 8, 11, 6, 8, 9, 13, 17),
                 team = rep(c('A', 'B', 'C'), each = 5))

#create horizontal boxplot for points
#Note: without the 'horizontal = TRUE' the boxplot would simply be vertical
boxplot(df$points, horizontal = TRUE, col = 'steelblue')
```



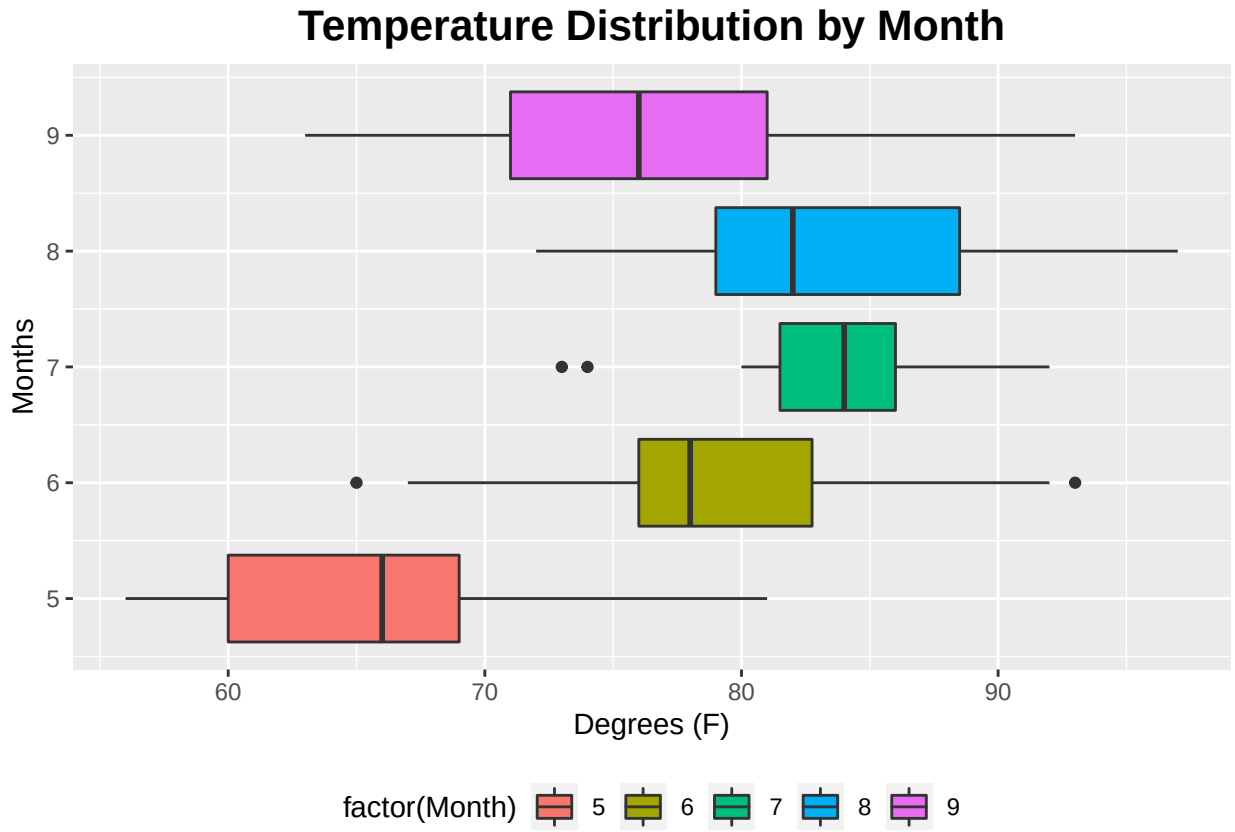
You can also put multiple boxplots inside one plot:

```
#create boxplot that displays temperature distribution for each month in the dataset
colors <- c("#106c00", "#b153bb", "#2c76c4", "#fab923", "#f52929")
months <- unique(airquality$Month)
boxplot(Temp~Month,
data = airquality,
main = "Temperature Distribution by Month",
ylab = "Month",
xlab = "Degrees (F)",
col = colors,
border = "black",
horizontal = TRUE
)
legend("left", legend = months, fill = colors)
```



You can also do this with `ggplot2`:

```
#create horizontal boxplot for points
ggplot(airquality, aes(x = Month, y = Temp, fill = factor(Month), group = Month)) +
  ggtitle("Temperature Distribution by Month") +
  xlab("Months") +
  ylab("Degrees (F)") +
  geom_boxplot() +
  coord_flip() + # this is the argument that flips the boxplots to be horizontal
  #this last line makes the title header bold and in the middle
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = 'bold'),
    legend.position = "bottom")
```



12 Basic statistical models

12.1 The linear regression model

To fit a linear regression model in `R`, we can use the `lm()` function, which uses the following syntax:

```
model <- lm(y ~ x1 + x2, data=df)
```

We can then use the following syntax to use the model to predict a single value:

```
predict(model, newdata = new)
```

The following examples show how to predict a single value using fitted regression models in `R`.

12.1.1 Simple linear regression model

The following code shows how to fit a simple linear regression model in R:

Sample statistic	Distribution feature
Graphical	
Empirical distribution function F_n	Distribution function F
Kernel density estimate $f_{n,h}$ and histogram	Probability density f
(Number of X_i equal to a)/n	Probability mass function $p(a)$
Numerical	
Sample mean $\bar{X}_n$	Expectation μ
Sample median $\text{Med}(X_1, X_2, \ldots, X_n)$	Median $q_{0.5} = F^{\text{inv}}(0.5)$
pth empirical quantile $q_n(p)$	$100p$th percentile $q_p = F^{\text{inv}}(p)$
Sample variance S_n^2	Variance σ^2
Sample standard deviation S_n	Standard deviation σ
$\text{MAD}(X_1, X_2, \ldots, X_n)$	$F^{\text{inv}}(0.75) - F^{\text{inv}}(0.5)$, for symmetric F

Figure 6: Some sample statistics and corresponding distribution features

```
#create data
df <- data.frame(x = c(3, 4, 4, 5, 5, 6, 7, 8, 11, 12),
                 y = c(22, 24, 24, 25, 25, 27, 29, 31, 32, 36))

#fit simple linear regression model
model <- lm(y ~ x, data = df)
```

And we can use the following code to predict the response value for a new observation:

```
#define new observation
new <- data.frame(x = c(5))
# this has to be a dataframe otherwise it will not work.

#use the fitted model to predict the value for the new observation
predict(model, newdata = new)
```

```
##        1
## 25.36364
```

The model predicts that this new observation will have a response value of 25.36364.

12.1.2 Multiple linear regression model

The following code shows how to fit a multiple linear regression model in R:

```
#create data
df <- data.frame(x1 = c(3, 4, 4, 5, 5, 6, 7, 8, 11, 12),
                 x2 = c(6, 6, 7, 7, 8, 9, 11, 13, 14, 14),
                 y = c(22, 24, 24, 25, 25, 27, 29, 31, 32, 36))

#fit multiple linear regression model
model <- lm(y ~ x1 + x2, data = df)
```

And we can use the following code to predict the response value for a new observation:

```r
#define new observation
new <- data.frame(x1 = c(5),
                  x2 = c(10))
# here we can see why the predict function has to use a dataframe - because it
# can also be used to predict values of a multiple linear regression model

#use the fitted model to predict the value for the new observation
predict(model, newdata = new)
```

```
##        1
## 26.17073
```

The model predicts that this new observation will have a response value of 26.17073.

12.1.3 Plotting a regression model

You can use the R visualization library `ggplot2` to plot a fitted linear regression model using the following basic syntax:

```
ggplot(data,aes(x, y)) +

    geom_point() +

    geom_smooth(method='lm')
```

The following example shows how to use this syntax in practice.

Suppose we fit a simple linear regression model to the following dataset:

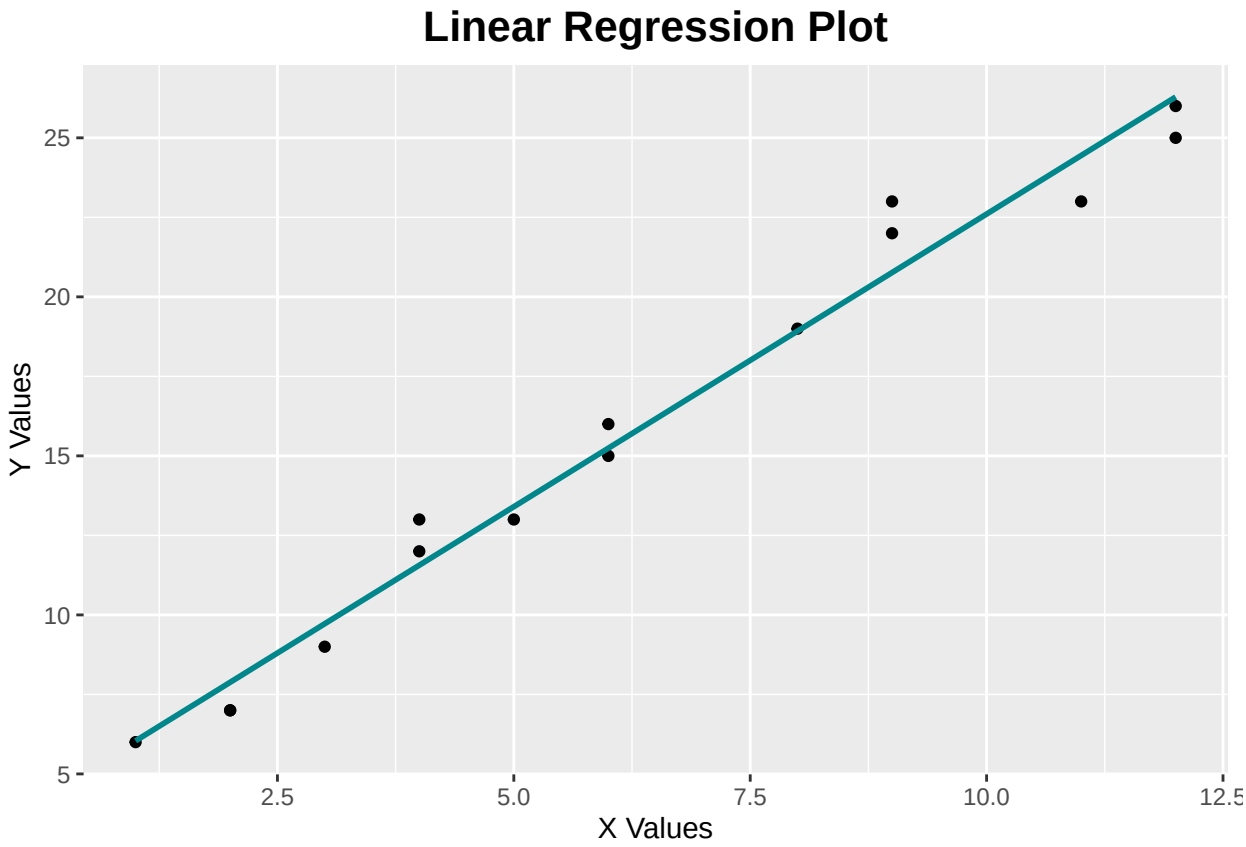
```r
#create dataset
data <- data.frame(y = c(6, 7, 7, 9, 12, 13, 13, 15, 16, 19, 22, 23, 23, 25, 26),
                   x = c(1, 2, 2, 3, 4, 4, 5, 6, 6, 8, 9, 9, 11, 12, 12))

#fit linear regression model to dataset and view model summary
model <- lm(y~x, data = data)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4444 -0.8013 -0.2426  0.5978  2.2363
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.20041    0.56730   7.404 5.16e-06 ***
## x            1.84036    0.07857  23.423 5.13e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.091 on 13 degrees of freedom
## Multiple R-squared:  0.9769, Adjusted R-squared:  0.9751
## F-statistic: 548.7 on 1 and 13 DF,  p-value: 5.13e-12
```

```r
# Make sure to import the ggplot library: library(ggplot2)
```

```r
# Create plot to visualize fitted linear regression model
ggplot(data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = 'lm', se = FALSE, col = "turquoise4") +
  # these last two line makes the title, x and y labels and
  # makes the header bold and in the middle
  labs(x = 'X Values', y = 'Y Values', title = 'Linear Regression Plot') +
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = 'bold'))
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Note the `se` argument to the `geom_smooth()` function removes the standard error from the visualization if it is needed simply remove it or set it to `TRUE`

13 Bootstrapping

The bootstrap method is a simulation procedure that approximates the distribution of things like the sample mean of a finite sample size. The method is generally applicable to other sample statistics than the sample mean as well.

In practice, let's say you have a sample set S of size n. To compute the bootstrapped sample mean of S, one would pick n elements from S, where picking multiple identical elements is allowed. The mean of those n elements is then calculated and stored. This process is done usually 10 000 times. What we end up with is the bootstrapped distribution of the sample mean of S. The expected value or average of that 'new' distribution is called the bootstrapped sample mean.

Put simply, bootstrapping consists of five steps:

Suppose you have a dataset S that conists of n elements.

1. Make a bootstrapped dataset
    - From the original dataset select n random elements with replacements i.e., duplicate element are allowed. This is called *sampling with replacement*
2. Calculate a statistic
    - If you do empirical bootstrapping then this could be: sample mean, sample variance, sample median, standard errors, confidence intervals or even p-values
    - If you do parametric bootstrapping then this could be the Kolmogorov-Smirnov distance between the bootstrapped dataset and the original dataset.
3. Save the calculation somewhere
4. Repeat step 1 through 3 a bunch of times usually 10 000 times.
5. Compare the calculated statistic with the 'real' statistic.
    - In empirical bootstrapping you could e.g., compare the sample mean with the population or 'true' mean
    - In parametric bootstrapping you could e.g., see if a dataset can be approximated by the normal distribution. One could first calculate the mean and the standard deviation and then use that to sample from a normal distribution with those parameters. ESTIMATE WITH MAXIMUM LIKELIHOOD?? FIND SOME MORE INFO ON THIS

Below are some common parameters with their corresponding sample statistic one might be interested in measuring

Measurement	**Sample statistic**	**Population parameter**
Mean	$\bar{x}$	μ (mu)
Standard deviation	s	σ (sigma)
Variance	s^2	σ^2 (sigma squared)
Proportion	p	π (pi)
Correlation	r	ρ (rho)
Regression coefficient	b	β (beta)

Table 7: Common parameters with their corresponding sample statistic

In any problem, we are always interested in measuring the population parameter. However, it's often too time-consuming, too costly, or simply not possible to actually measure every single individual element in the population, which is why we instead calculate a sample statistic and use that statistic to estimate the true population parameter. This can become more 'accurate' with bootstrapping as we know from the law of large numbers.

13.1 Empirical bootstrapping

The method described above can be used to do what is called *empirical bootstrapping* which you do when you have no knowledge about the distribution from which the random sample is from.

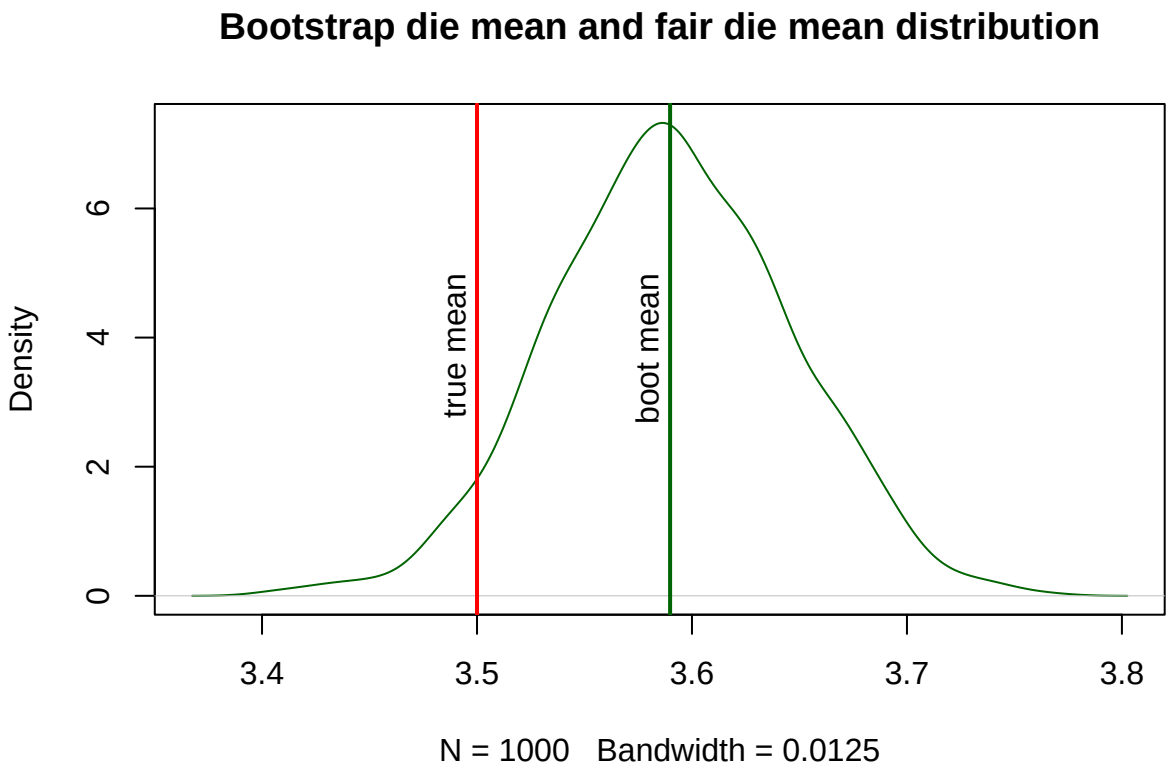
Imagine a situation of purchasing an antique die from the internet for gambling. Before purchasing, you want to make sure that the die is fair. The seller provides 1000 samples of the outcomes available in the file `die_samples.Rdata`

```
load(here::here('Chapters', 'assets', 'die_samples.Rdata'))
```

Determine whether the die is fair or not using bootstrapping

```
boots <- function(data) {
  means <- c()
  for (i in 1:1000) {
    s <- sample(data, 1000, replace = T)
    means <- append(means, mean(s))
    }
    return(means)
```

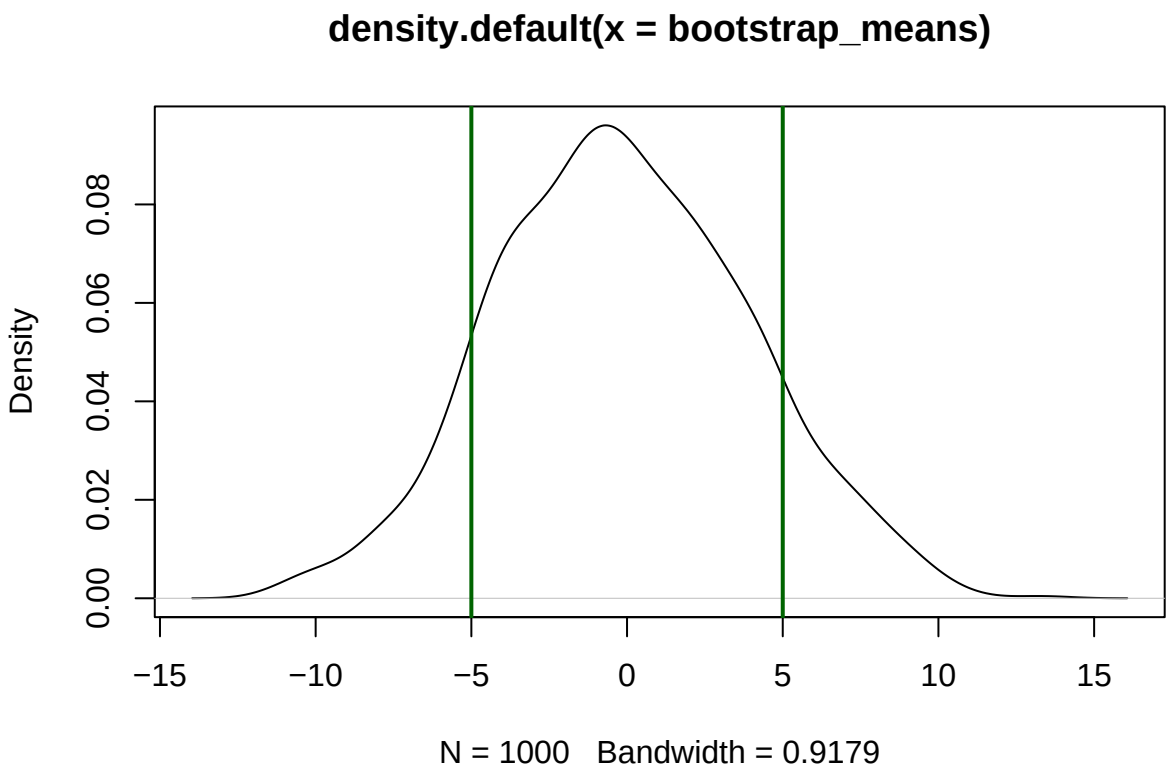
```
    }
bootstrap_mean <- mean(boots(die_samples))
plot(density(boots(die_samples)),
    main = "Bootstrap die mean and fair die mean distribution",
      col = "darkgreen")
abline(v = mean(c(1, 2, 3, 4, 5, 6)), col = "red", lwd = 2)
text(3.5, 5, "true mean", srt = 90, pos = 2)
abline(v = bootstrap_mean, col = "darkgreen", lwd = 2)
text(bootstrap_mean, 5, "boot mean", srt = 90, pos = 2)
```



As is visible from the plot the die is not totally fair as it has a mean of 3.589827 whereas a completely fair die has a mean of 3.5.

We can also use these bootstrapped means to bound the probability that the sample mean is a certain distance away from the norm. If we take the Old Faithful dataset again. Suppose we want to know the probability that the sample mean is more than say 5 seconds away from the norm, we can do as follows

```
eruptions <- faithful$eruptions * 60 # multiplying by 60 because dataset is in minutes
sample_mean <- mean(eruptions)
bootstrap_means <- c()
for (i in 1:1000) {
  bootstrap_sample <- sample(eruptions, size = length(eruptions), replace = TRUE)
  bootstrap_means <- c(bootstrap_means, sample_mean - mean(bootstrap_sample))
}
plot(density(bootstrap_means))
abline(v = 5, lwd = 2, col = "darkgreen")
abline(v = -5, lwd = 2, col = "darkgreen")
```



Since we want to know the probability that the sample mean is more than 5 seconds away from the norm we simply have to subtract the probability that the sample mean is +5 seconds away and −5 seconds away from the norm. This would give us the probability that the sample mean is between −5 and +5 seconds away from the norm so in order to find out the probability that the sample mean is more and 5 seconds away from the norm we have to subtract the whole thing by 1

```
bootstrap_means_diff_dist <- ecdf(bootstrap_means)
probability_outside <- 1 - (bootstrap_means_diff_dist(5) - bootstrap_means_diff_dist(-5))
# calling ecdf(bootstrap_means)(5) tells us probability that it is exactly 5
# seconds away from mean
probability_outside
```

```
## [1] 0.212
```

As such the probability that the sample mean is more than 5 seconds away from the norm is $\approx 23\%$

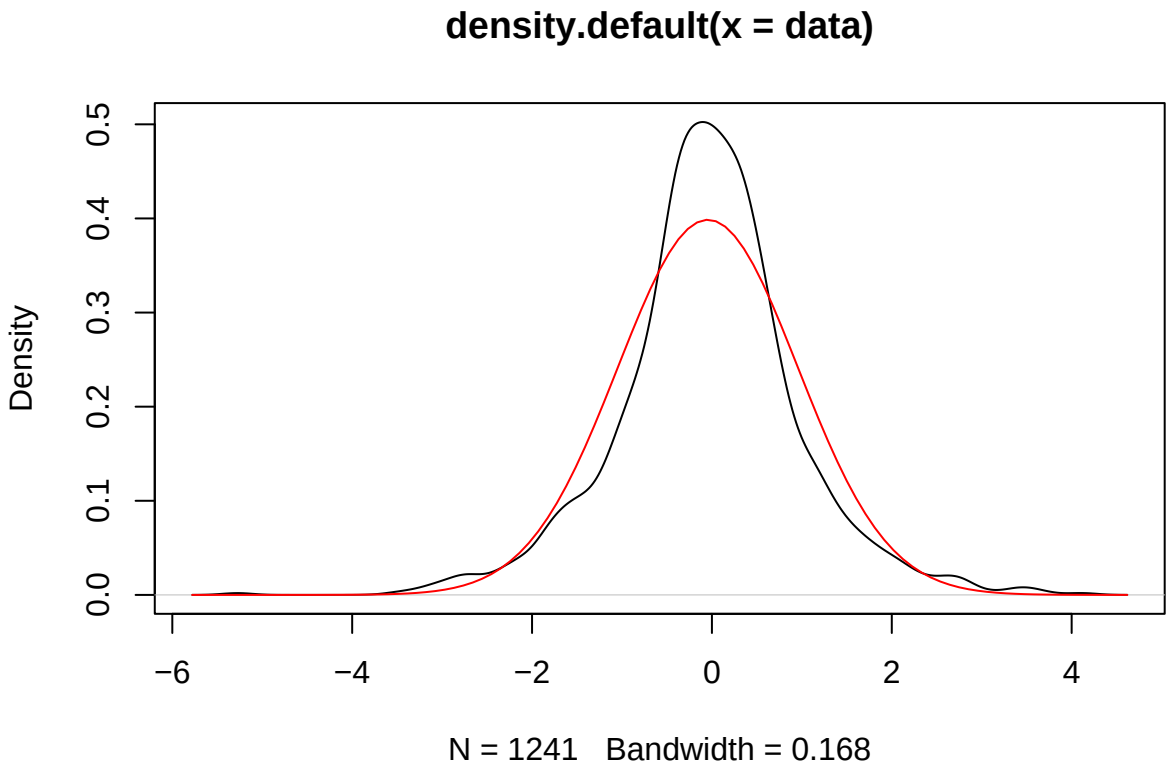
13.2 Parametric bootstrapping

Parametric bootstrapping is on the other hand used when we want to know whether or not it is 'safe' to assume that a dataset can be decribed by a certain distribution. i.e., What is the probability that this collection of samples could have been drawn from that probability distribution?

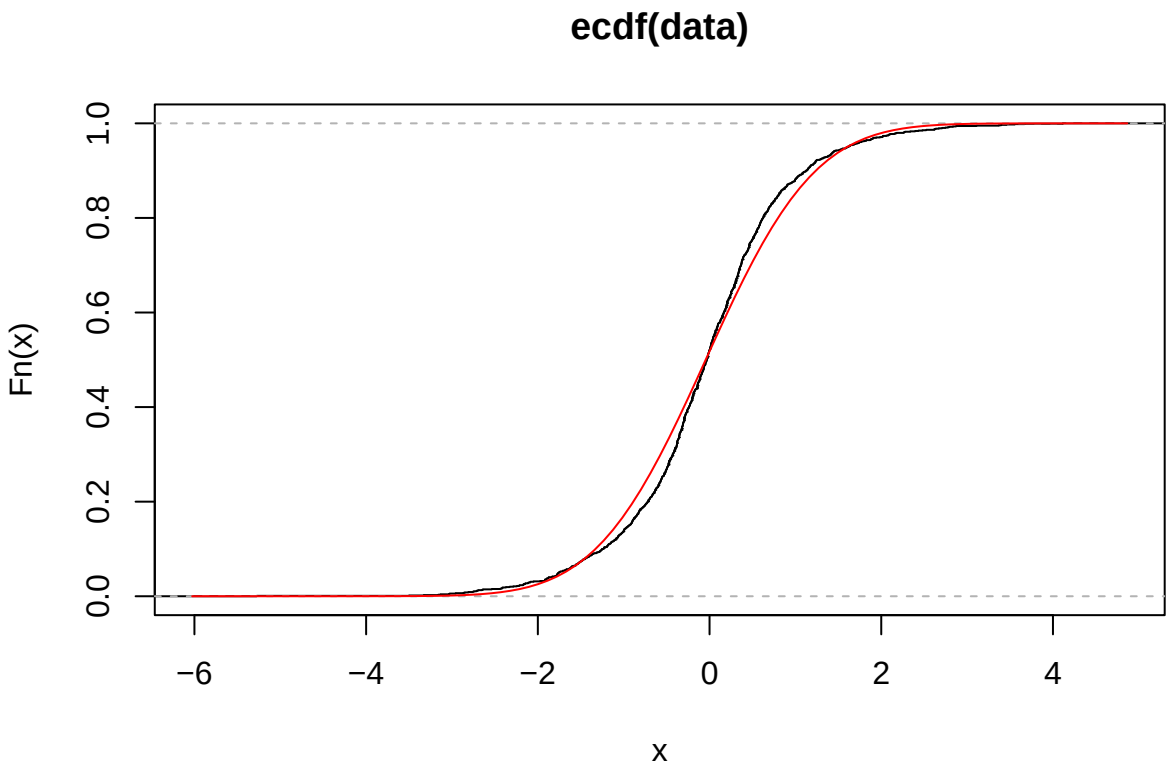
The parametric bootstrap method is differs in the way it draws its samples. If we can make some assumptions about the dataset at hand, we estimate $\widehat{F}$ by the theoretical distribution with the parameters estimated from our data. If we, i.e. assume our data to be normally distributed, we would calculate the sample mean and variance from our dataset and sample n independent samples from the theoretical distribution obtained in that way using the technique obtained to sample from any random variable.

The dataset `arctic.oscillations` (in package UsingR) contains a time series from January to June 2002 of sea-level pressure measurement at the arctic, relative to some base line. Use parametric bootstrap to judge whether it is safe to assume that the measurements are samples from normal distribution or not.

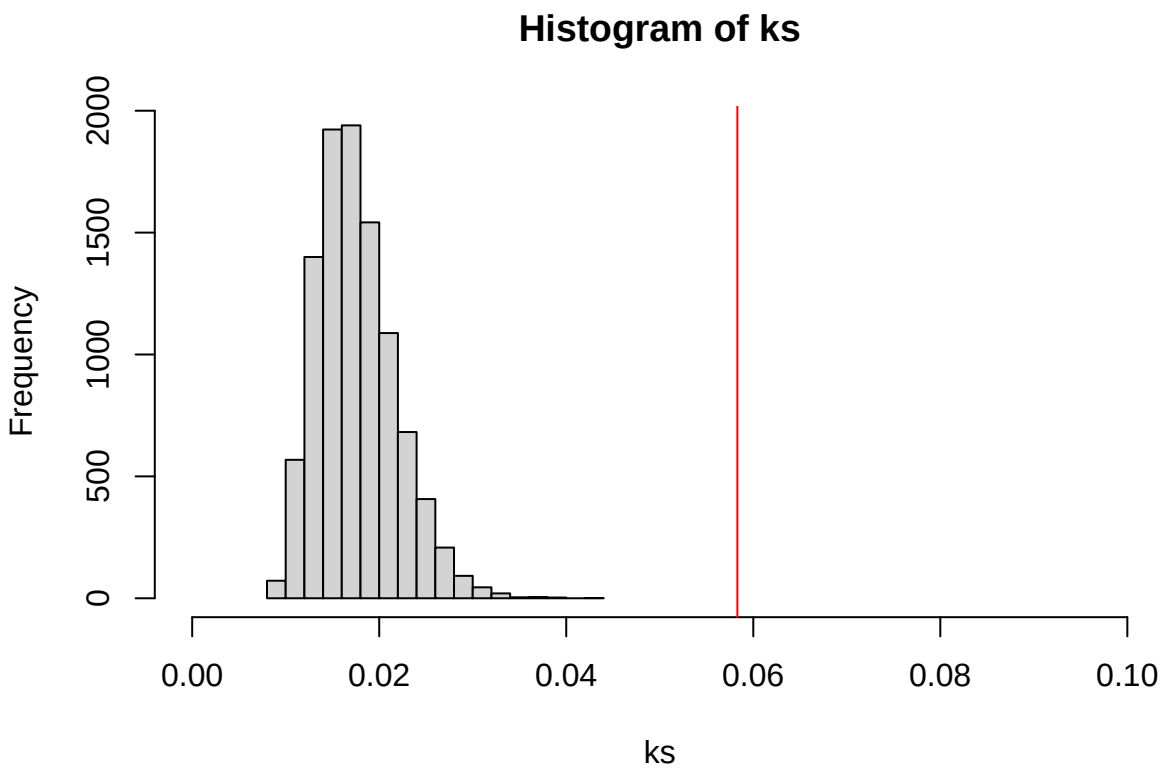
```r
data <- na.omit(arctic.oscillations)

sample_mean <- mean(data)
sample_sd <- sd(data)

xs <- seq(from = min(data), max(data), length.out = 100)
plot(density(data))
curve(dnorm(x, mean = sample_mean, sample_sd), col = "red", add = T)
```




```r
plot(ecdf(data))
curve(pnorm(x, mean = sample_mean, sample_sd), col = "red", add = T)
```




```r
# now we calculate the Kolmogorov-Smirnov distance
ks_dist_norm <- function(data) {
  emp_dist <- ecdf(data)
  max(abs(emp_dist(data) - pnorm(data, mean(data), sd = sd(data))))
}
ks_estimate <- ks_dist_norm(data)
m <- 10 ** 4
n <- length(data)

ks <- c()
for (i in 1:m) {
  bootstrap_sample <- rnorm(n, mean = sample_mean, sd = sample_sd)
  ks <- c(ks, ks_dist_norm(bootstrap_sample))
}

hist(ks, xlim = c(0, 0.1))
abline(v = ks_estimate, col = "red")
```



13.3 Bootstrapping and confidence intervals

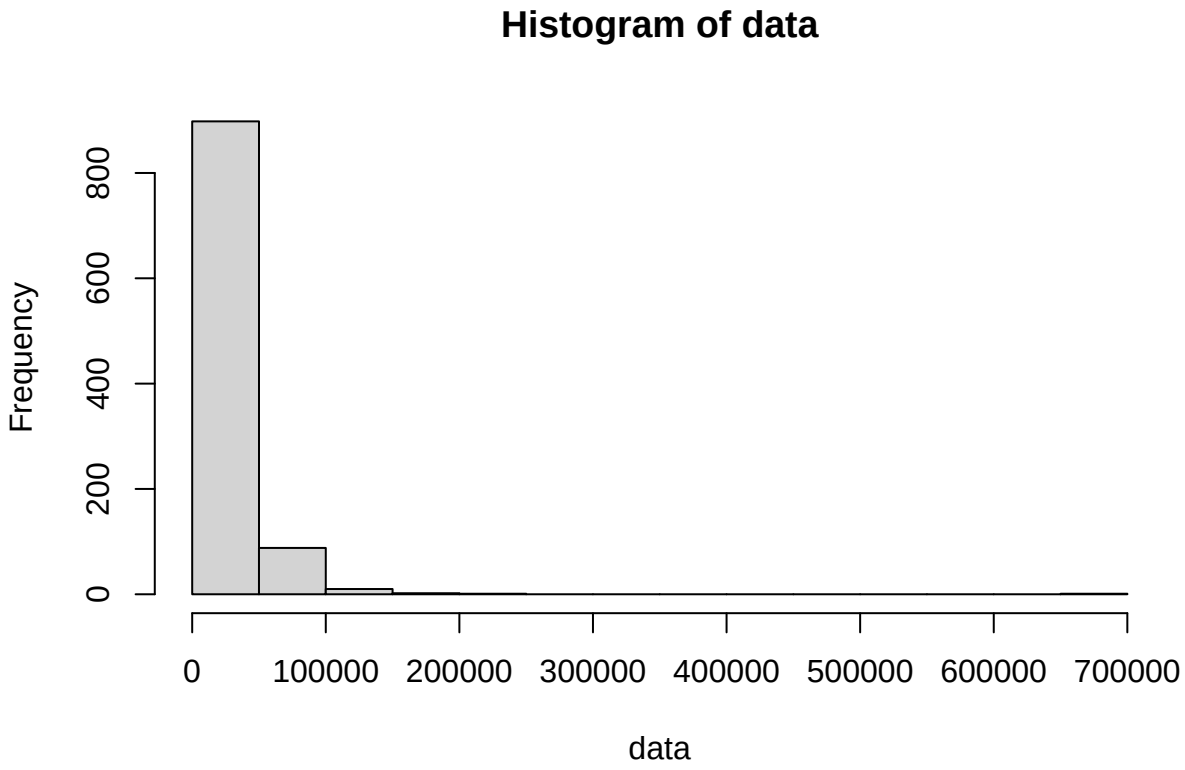
Note: The questions are taken from an applied statistics mock exam. The answers to the questions are mine.

Let us look at the datasets `female.inc (UsingR)` that contain income distribution for females in 2001. You may ignore the information about the race of the individuals. Your goal is to estimate the mean female income together with its 95% confidence intervals.

(a) Why is bootstrapping a good strategy for finding the confidence intervals for the mean female income? (1pt)

Let us look at how the data set is distributed by creating a simple histogram

```
data <- female.inc$income
options(scipen = 999)
# ^ Makes sure that the plot does not use 'scientific' notation
hist(data)
```



Since the data is clearly not normally distributed it is a good strategy to use bootstrapping

(b) Write a computer program that computes the bootstrap estimates for the 95% confidence intervals for the mean income of all the females in the dataset. (5pts)

```
set.seed(2022) # Setting a seed to make the code 100 % reproducible/consistent.
n <- length(data)
boots <- function(data) {
  means <- c()
  for (i in 1:1000) {
    s <- sample(data, n, replace = T)
    means <- append(means, mean(s))
    }
    return(means)
    }
```

(c) Report your numerical results for the mean and confidence interval together with a graphic illustrating the bootstrapped sample. (2pts)

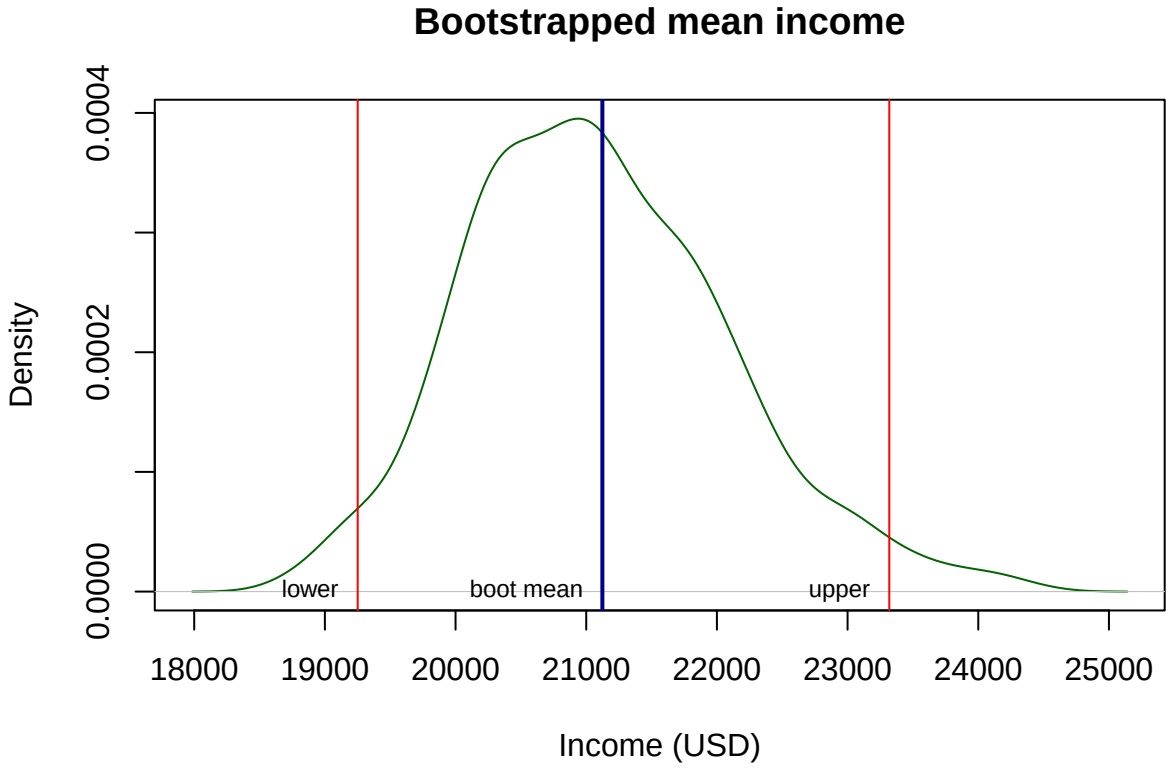
To find the 95 % confidence interval we can use the `quantile` command to determine the upper and lower bound of the confidence interval.

```
boot_data <- boots(data)
lower <- quantile(boot_data, prob = 0.025)
upper <- quantile(boot_data, prob = 0.975)
```

So the lower bound of the confidence interval is 19251.22 and the upper bound of the confidence interval is 23319.37. This means that we with 95 % confidence can say that the true/population mean is somewhere between 19251.22 USD and 23319.37 USD[1].

[1] Currency found via `?female.inc` command

```r
bootstrap_mean <- mean(boots(data))
plot(density(boot_data),
    main = "Bootstrapped mean income",
      col = "darkgreen",
      xlab = "Income (USD)")
abline(v = bootstrap_mean, col = "darkblue", lwd = 2)
text(bootstrap_mean, 0, "boot mean", srt = 90, pos = 2, srt = 0, cex = 0.75)
abline(v = lower, col = "red", lwd = 1)
text(lower, 0, "lower", srt = 90, pos = 2, srt = 0, cex = 0.75)
abline(v = upper, col = "red", lwd = 1)
text(upper, 0, "upper", srt = 90, pos = 2, srt = 0, cex = 0.75)
```



14 Unbiased estimators

An unbiased estimator of a parameter is an estimator whose expected value is equal to the parameter.

That is, if the estimator T is being used to estimate a parameter θ, then T is an unbiased estimator of θ if $E[T] = \theta$

Remember that expectation can be thought of as a long-run average value of a random variable. If an estimator T is unbiased, then on average it is equal to the number it is trying to estimate. Here "on average" involves imagining repeated samples, as follows:

- Draw one random sample; compute the value of T based on that sample.
- Draw another random sample of the same size, independently of the first one; compute the value of T based on this sample.
- Repeat the step above as many times as you can.
- You will now have lots of observed values of T.

If T is an unbiased estimator of θ, then on average, these values will neither be greater than θ nor smaller

than θ. On average in the long run they will be just right: equal to θ.

This seems like a good property for an estimator to have. In many settings, natural estimators turn out to be unbiased. Let's look at some examples.

14.1 Unbiased estimator for the mean

$$\bar{X} = \frac{x_1, \dots, x_n}{n}$$

14.2 Unbiased estimator for the variance

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

14.3 Unbiased for the uniform distribution $U(0, \Theta)$

$$\theta = \frac{2}{n}(X_1 + \cdots + X_n)$$

15 Maximum likelihood

15.1 Normal distribution

Recall the probability density function of a normal distribution is given by:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-(x-\mu)^2/2\sigma^2}$$

Which we denote $N(\mu, \sigma^2)$

The likelihood function is then given by:

$$L(\mu, \sigma | x_1, \dots, x_n) = \prod_{i=1}^{n}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x_i-\mu)^2/2\sigma^2}$$

Since we know that $\ell(\theta) = \ln(L(\theta))$

The loglikelihood function is thus given by:

$$\ell(\mu,\sigma|x_1,\dots,x_n) = \ln(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma^2} e^{-(x_i-\mu)^2/2\sigma^2}) \tag{1}$$

$$= \sum_{i=1}^{n} \ln(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x_i-\mu)^2/2\sigma^2}) \tag{2}$$

$$= \sum_{i=1}^{n} \ln(\frac{1}{\sqrt{2\pi\sigma^2}}) + \ln(e^{-(x_i-\mu)^2/2\sigma^2}) \tag{3}$$

$$= \sum_{i=1}^{n} \ln((2\pi\sigma^2)^{-1/2}) - \frac{(x_i-\mu)^2}{2\sigma^2}\ln(e) \tag{4}$$

$$= \sum_{i=1}^{n} -\frac{1}{2}\ln(2\pi\sigma^2) - \frac{(x_i-\mu)^2}{2\sigma^2} \tag{5}$$

$$= \sum_{i=1}^{n} -\frac{1}{2}\ln(2\pi) - -\frac{1}{2}\ln(\sigma^2) - \frac{(x_i-\mu)^2}{2\sigma^2} \tag{6}$$

$$= \sum_{i=1}^{n} -\frac{1}{2}\ln(2\pi) - 2\cdot\frac{1}{2}\ln(\sigma) - \frac{(x_i-\mu)^2}{2\sigma^2} \tag{7}$$

$$= \sum_{i=1}^{n} -\frac{1}{2}\ln(2\pi) - \ln(\sigma) - \frac{(x_i-\mu)^2}{2\sigma^2} \tag{8}$$

$$= \sum_{i=1}^{n} -\frac{n}{2}\ln(2\pi) - n\cdot\ln(\sigma) - \frac{(x_i-\mu)^2}{2\sigma^2} \tag{9}$$

In step 4 to 5: $\ln(e)$ is equal to 1 so we can just remove it.

In step 7 to 8: $2\cdot\frac{1}{2}$ becomes $\frac{2}{2}$ so we can just remove it.

15.1.1 The maximum likelihood estimators of μ and σ^2

If we take the derivative of the loglikelihood function with respect to μ we get:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

And if we take the derivative of the loglikelihood function with respect to σ we get:

$$\sigma = \frac{1}{n}\sum_{i=1}^{n} \sqrt{(x_i-\mu)^2}$$

15.2 Exponential distribution

Recall the probability density function of an exponential distribution is given by:

$$f(x) = \frac{1}{\lambda}e^{-\lambda x}$$

Which we denote $Exp(\lambda)$

The likelihood function is then given by:

$$L(\lambda|X_1,\dots,X_n) = \lambda^n e^{-\lambda\sum_{i=1}^{n} X_i}$$

Since we know that $\ell(\theta) = \ln(L(\theta))$

The loglikelihood function is thus given by:

$$\ell(\lambda|x_1, \dots, x_n) = \ln(\lambda^n e^{-\lambda \sum_{i=1}^n X_i}) \tag{1}$$

$$= \ln(\lambda^n) + \ln(e^{-\lambda \sum_{i=1}^n X_i}) \tag{2}$$

$$= n \cdot \ln(\lambda) - \lambda \sum_{i=1}^{n} X_i \tag{3}$$

If we take the derivative of the loglikelihood function with respect to λ we get:

$$\lambda = n\frac{1}{\lambda} - \sum_{i=1}^{n} x_i$$