Computational Statistics with R

Newcastle University
`lee.fawcett@ncl.ac.uk`

Semester 1, 2023/24

Books and other resources

Some useful books include:

- W.N. Venables and D.M. Smith: An Introduction To R. Free to download at:

  `https://cran.r-project.org/doc/manuals/R-intro.pdf`
- D. Braun and D.J. Murdoch: A First Course In Statistical Programming With R
- S. Ross: A First Course in Probability
- N.A. Weiss, P.T. Holmes and H. Michael: A Course In Probability

Books and other resources

Some useful books include:

- W.N. Venables and D.M. Smith: An Introduction To R. Free to download at:

  https://cran.r-project.org/doc/manuals/R-intro.pdf
- D. Braun and D.J. Murdoch: A First Course In Statistical Programming With R
- S. Ross: A First Course in Probability
- N.A. Weiss, P.T. Holmes and H. Michael: A Course In Probability

General background

In this class we will learn about computational tools for performing probability calculations and statistical analysis.

We will use the R programming language.

R is very widely used in universities, businesses and other organizations due to its many advantages over other similar software packages.

- It is available for **free**
- It is '**open source**' and can be used for a very wide range of statistical and scientific applications
- **Libraries** of R functions and data-structures have been created and are readily available on the web

General background

Learning to use R is important because:

(i) It is the foremost tool in **modern computational statistics**

(ii) It is a great way to learn general concepts in **programming** which can be used with other languages or in other contexts

(iii) It can be used to illustrate ideas in **probability and statistics**

Preliminaries

R is installed on all University machines.

In this module we will be using **RStudio**, which is an R *Integrated Development Environment*, or IDE.

This just makes R much easier to use and navigate, through the four windows that you see when you open RStudio (text editor, console window, file explorer and graphics display).

Preliminaries

R is installed on all University machines.

In this module we will be using **RStudio**, which is an R *Integrated Development Environment*, or IDE.

This just makes R much easier to use and navigate, through the four windows that you see when you open RStudio (text editor, console window, file explorer and graphics display).

Preliminaries

R is installed on all University machines.

In this module we will be using **RStudio**, which is an R *Integrated Development Environment*, or IDE.

This just makes R much easier to use and navigate, through the four windows that you see when you open RStudio (text editor, console window, file explorer and graphics display).

Preliminaries

In this brief intro, we will quickly review:

(i) Data types and vectors in R
(ii) Dataframe manipulation
(iii) Data summaries
(iv) Functions and control statements

Preliminaries

I like watching movies – most people do – so I often use the IMDB movies database to demonstrate key concepts in RStudio.

IMDB is a website devoted to collecting movie data supplied by studios and fans.

More information can be found online:

`http://imdb.com/help/show_leaf?about`

Preliminaries

I like watching movies – most people do – so I often use the IMDB movies database to demonstrate key concepts in RStudio.

IMDB is a website devoted to collecting movie data supplied by studios and fans.

More information can be found online:

`http://imdb.com/help/show_leaf?about`

Preliminaries

				Voting statistics				.	Movie genre							
Title	Year	Length	Budget	Rating	Votes	r1	...	r10	mpaa	Action	Animation	Comedy	Drama	Documentary	Romance	Short
A.k.a. Cassius	1970	85	NA	5.7	43	4.5	...	14.5	PG	0	0	0	0	1	0	0
AKA	2002	123	NA	6.0	335	24.5	...	1.5	R	0	0	0	1	0	0	0
Alien Vs. Pred	2004	102	45000000	5.4	14651	4.5	...	4.5	PG-13	1	0	0	0	0	0	0
Abandon	2002	99	25000000	4.7	2364	4.5	...	4.5	PG-13	0	0	0	1	0	0	0
Abendland	1999	146	NA	5.0	46	14.5	...	24.5	R	0	0	0	0	0	0	0
Aberration	1997	93	NA	4.8	149	14.5	...	4.5	R	0	0	0	0	0	0	0
Abilene	1999	104	NA	4.9	42	0.0	...	24.5	PG	0	0	0	1	0	0	0
Ablaze	2001	97	NA	3.6	98	24.5	...	14.5	R	1	0	0	1	0	0	0
Abominable Dr	1971	94	NA	6.7	1547	4.5	...	14.5	PG-13	0	0	0	0	0	0	0
About Adam	2000	105	NA	6.4	1303	4.5	...	4.5	R	0	0	1	0	0	1	0

Preliminaries

I have written an add-on R package which will allows easy access to many interesting datasets (including the movies dataset).

Installing the package is straightforward:

```
install.packages("ncldata", repos = "http://R-Forge.R-
    project.org", type = "source")
```

To load the package, type:

```
library(ncldata)
```

Then, to specifically load the movies dataset, type:

```
data(movies)
```

Preliminaries

I have written an add-on R package which will allows easy access to many interesting datasets (including the movies dataset).

Installing the package is straightforward:

```
install.packages("ncldata", repos = "http://R-Forge.R-
    project.org", type = "source")
```

To load the package, type:

```
library(ncldata)
```

Then, to specifically load the movies dataset, type:

```
data(movies)
```

Preliminaries

Alternatively, you can by-pass the pacakge altogether as I've posted the dataset to my personal webpage:

```
movies = read.table("https://www.mas.ncl.ac.uk/~nlf8/
    basicR/movies.txt", header=TRUE)
attach(movies)
```

Data types in R

R has a variety of data types. For example:

```
v = TRUE                    # 'Logical' or 'boolean' - TRUE or FALSE
w = "fred"                  # 'Character' or 'string'
x = 5.0                     # 'Double' or 'numerical'
```

There are also some 'special' data types:

```
y = 5/0                     # Returns Inf
z = y-y                     # Returns NaN
```

Data types in R

Another important data type is `NA`, short for 'Not Available' and used for missing values.

A summary of the data types in R is given in the Table below.

Type	Example 1	Example 2	Example 3	Example 4
Doubles	2	3.1242	-45.6	4e-10
Logicals/Boolean	TRUE	FALSE		
Characters	"Fred"	"TRUE"	"Male"	"2"
Infinity	Inf	5/0		
Other	NaN	NA		

Quiz #1

In the code below, what is the value of x?

```
x = 5
x = 2*x + 1
x + 5
```

1. 5
2. 11
3. 16
4. 22

Quiz #1

In the code below, what is the value of x?

```
x = 5
x = 2*x + 1
x + 5
```

1. 5
2. **11**
3. 16
4. 22

Quiz #2

In the code below, what type of variable is x?

```
x = 5
x = 2*x + 1
x + 5
```

1. Character
2. Double
3. Logical
4. NA

Quiz #2

In the code below, what type of variable is x?

```
x = 5
x = 2*x + 1
x + 5
```

1. Character
2. **Double**
3. Logical
4. NA

Quiz #3

In the code below, what type of variable is y?

```
x = 0
y = 2/0
y = "Fred"
```

1. NaN
2. Double
3. NA
4. Character

Quiz #3

In the code below, what type of variable is y?

```
x = 0
y = 2/0
y = "Fred"
```

1. NaN
2. Double
3. NA
4. **Character**

Vectors

Vectors are the most basic of all data structures, but are used in almost all R code.

An R vector contains *n* values of the same type, where *n* can be zero.

For example:

```
c(0, 1, 2, 3, 4, 5)
 [1] 0 1 2 3 4 5
(first_vec = c(0, 1, 2, 3, 4, 5))
 [1] 0 1 2 3 4 5
(second_vec = c("Male", "Female", "Male"))
 [1] "Male"    "Female"    "Male"
x = 5.2
y = "Fred"
```

Vectors

There are special functions in R that can be used to determine the variable type. For example:

```
is.double(x)
 [1] TRUE
is.character(x)
 [1] FALSE
is.character(y)
 [1] TRUE
is.vector(first_vec)
 [1] TRUE
```

Vectors

And of course, we can use the `length` command to determine the length of a vector:

```
length(first_vec)
 [1] 6
length(second_vec)
 [1] 3
```

And you can use the seq command to generate sequence vectors:

```
(x1 = seq(1, 6))
 [1] 1 2 3 4 5 6
(x2 = seq(-4, 4, by=2))
 [1] -4 -2 0 2 4
(x3 = seq(3, -2, by=-0.5))
 [1]  3.0  2.5  2.0  1.5  1.0  0.5  0.0 -0.5 -1.0 -1.5
[11] -2.0
```

Vectors

And of course, we can use the `length` command to determine the length of a vector:

```
length(first_vec)
 [1] 6
length(second_vec)
 [1] 3
```

And you can use the `seq` command to generate sequence vectors:

```
(x1 = seq(1, 6))
 [1] 1 2 3 4 5 6
(x2 = seq(-4, 4, by=2))
 [1] -4 -2 0 2 4
(x3 = seq(3, -2, by=-0.5))
 [1]  3.0  2.5  2.0  1.5  1.0  0.5  0.0 -0.5 -1.0 -1.5
[11] -2.0
```

Vectors

Other useful commands for vectors:

```
rev(x1)                  # Reverses the order of the vector
 [1] 6 5 4 3 2 1
sort(x3)                 # Sorts the vector into ascending order
 [1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5
     3.0
sort(second_vec)         # Can be applied to characters: alphabetical order
[1] "Female" "Male"    "Male"
sum(x2); sum(x3); sum(second_vec)
 [1] 0
 [1] 5.5
 Error in sum(second_vec) : invalid 'type' (character)
    of argument
unique(second_vec)       # Lists all unique elements of the vector
 [1] "Male"    "Female"
which(x2==2)             # Gives index/indices of particular elements
 [1] 4
```

Vector operations

When our data are in a vector structure, we can apply standard operations to the entire vector. For example:

```
(x = seq(-4, 4))
 [1] -4 -3 -2 -1 0 1 2 3 4
x*x
 [1] 16 9 4 2 1 0 1 2 4 9 16
x-5
 [1] -9 -8 -7 -6 -5 -4 -3 -2 -1
x + x
 [1] -8 -6 -4 -2 0 2 4 6 8
```

Extracting elements from vectors

R has a number of useful methods that we can use to extract subsets of our data.

For example, square brackets:

```
first_vec[2]
 [1] 1
second_vec[2:3]
 [1] "Female"   "Male"
first_vec[4:2]
 [1] 3 2 1
(z = 1:10)
 [1]  1  2  3  4  5  6  7  8  9 10
z[z<3 | z>=7]          # Values of z that are less than 3 or at least 7
[1]  1  2  7  8  9 10
```

We will look at extraction from two-dimensional R objects shortly.

Extracting elements from vectors

R has a number of useful methods that we can use to extract subsets of our data.

For example, square brackets:

```
first_vec[2]
 [1] 1
second_vec[2:3]
 [1] "Female"   "Male"
first_vec[4:2]
 [1] 3 2 1
(z = 1:10)
 [1]  1  2  3  4  5  6  7  8  9 10
z[z<3 | z>=7]          # Values of z that are less than 3 or at least 7
[1]  1  2  7  8  9 10
```

We will look at extraction from two-dimensional R objects shortly.

Extracting elements from vectors

Another useful way of extracting elements from vectors is to use logical vectors.

Recall that the logical data type in R can take only the values `TRUE` and `FALSE`. Hence:

```
A = TRUE
B = FALSE
!A          # Read as "Not A"
  [1] FALSE
!B
  [1] TRUE
A & B       # Need "both A and B"
  [1] FALSE
A | B       # Need "either A or B"
  [1] TRUE
```

Extracting elements from vectors

Then we could declare a vector of such logicals to help us extract from another vector:

```
(logic1 = c(TRUE, FALSE, TRUE, FALSE))
 [1] TRUE  FALSE  TRUE  FALSE
(vec1 = seq(3, 6))
 [1] 3 4 5 6
vec1[logic1]      # Will extract the elements in vec1 where logic1 is true
 [1] 3 5
```

We can also sum a logical vector. For example:

```
sum(logic1)
 [1] 2
sum(vec1[logic1])
 [1] 8
```

Extracting elements from vectors

Then we could declare a vector of such logicals to help us extract from another vector:

```
(logic1 = c(TRUE, FALSE, TRUE, FALSE))
 [1] TRUE  FALSE  TRUE  FALSE
(vec1 = seq(3, 6))
 [1] 3 4 5 6
vec1[logic1]      # Will extract the elements in vec1 where logic1 is true
 [1] 3 5
```

We can also sum a logical vector. For example:

```
sum(logic1)
 [1] 2
sum(vec1[logic1])
 [1] 8
```

Example

In RStudio, commit the following lines of code. Both lines read in files stored on a webpage. **Do not copy-and-paste this code – you must type it in carefully!**

```
x = scan("http://www.mas.ncl.ac.uk/~nlf8/basicR/x.txt")
y = as.logical(scan("http://www.mas.ncl.ac.uk/~nlf8/
    basicR/y.txt", what="logical"))
```

Questions about x	Questions about y
(a) What type of data is in the vector x?	(a) What type of data is in the vector y?
(b) How many values are in the vector x?	(b) What is the first value of y?
(c) What is the smallest value in the vector x?	(c) What is the last value of y?
(d) What is the value of x in position 3500?	(d) How many times does TRUE occur?
(e) What is the 97th smallest value in x?	(e) When is the 472nd occurrence of TRUE?
(f) Find $\sum_{i=5612}^{8664} x_i$.	(f) How many occurrences of TRUE are there between positions 504 and 593 (inclusive)?

Dataframes

A dataframe is a special kind of object in R.

We use dataframes for storing and managing datasets that have a rectangular structure.

Typically the rows correspond to **cases** and the columns **variables**.

The crucial difference between a dataframe and a matrix is that all values in a matrix must be of the same type, whereas in a dataframe we can have responses for cases on variables of different types.

Dataframes

For example, we might have survey results on 10 individuals, who have each been asked 5 questions.

- This would give a dataframe with 10 rows and 5 columns
- We call this a "**10 by 5 dataframe**" (generally, we refer to $r \times c$ dataframes)
- The variable types need not all be the same:
  - The first column could be a vector of doubles recording the individuals' height in metres
  - The second column could be a vector of logicals taking the value `TRUE` if they like Marmite
  - The third column could be a vector of characters taking the name of their favourite book, etc.

Creating and querying your own dataframe

You can construct your own dataframe using R's `data.frame` command. For example:

```
height_m = c(1.81, 1.75, 1.58)
weight_kg = c(NA, NA, 70.3)
marmite = c(TRUE, FALSE, FALSE)
book = c("Midnight in Chernobyl", "To Kill a
    Mockingbird", "Life of Pi")

(small.survey = data.frame(height_m, weight_kg, marmite
    , book))
  height_m weight_kg marmite                 book
1     1.81        NA    TRUE Midnight in Chernobyl
2     1.75        NA   FALSE To Kill a Mockingbird
3     1.58      70.3   FALSE            Life of Pi
```

Creating and querying your own dataframe

If you had a large dataframe and wanted to quickly know its size, you could use the `dim` command. For example:

```
dim(small.survey)
 [1] 3  4
```

Creating and querying your own dataframe

We can extract elements from a dataframe using square brackets (just as we demonstrated earlier for vector extraction).

If the columns have labels, we can also use the $ command.

```
 1 small.survey[2, 4]        # Extract row 2, column 4
 2 [1] To Kill a Mockingbird
 3
 4 small.survey[ , 4]        # Extract all rows from column 4
 5 [1] Midnight in Chernobyl    To Kill a Mockingbird
       Life of Pi
 6
 7 small.survey$book         # Does exactly the same as line 4!
 8 [1] Midnight in Chernobyl    To Kill a Mockingbird
       Life of Pi
 9
10 small.survey[3, ]          # Extract all columns from row 3
11   height_m weight_kg marmite       book
12 3     1.58      70.3   FALSE Life of Pi
```

Creating and querying your own dataframe

Though used less, the `subset` command can be applied to extract information from selected columns only. For example:

```
(small.survey2 = subset(small.survey, select = 3:4))
  marmite                  book
1    TRUE  Midnight in Chernobyl
2   FALSE To Kill a Mockingbird
3   FALSE            Life of Pi
```

And what if you wanted the height and favourite book of respondents who didn't like Marmite?

```
subset(small.survey, marmite == FALSE, select=c(1,4))
  height_m                  book
2     1.75 To Kill a Mockingbird
3     1.58            Life of Pi
```

Creating and querying your own dataframe

Though used less, the `subset` command can be applied to extract information from selected columns only. For example:

```
(small.survey2 = subset(small.survey, select = 3:4))
  marmite                    book
1    TRUE  Midnight in Chernobyl
2   FALSE To Kill a Mockingbird
3   FALSE             Life of Pi
```

And what if you wanted the height and favourite book of respondents who didn't like Marmite?

```
subset(small.survey,marmite == FALSE,select=c(1,4))
  height_m                    book
2     1.75 To Kill a Mockingbird
3     1.58             Life of Pi
```

The movies dataframe

The Movies dataset is available from within the `ncldata` package.

Recall that you can access this in the following way:

```
install.packages("ncldata", repos="http://R-Forge.R-
    Project.org", type="source")
library(ncldata)
data(movies)
```

Or:

```
movies = read.table("https://www.mas.ncl.ac.uk/~nlf8/
    basicR/movies.txt", header=TRUE)
```

We will now explore the movies dataset in RStudio.

The movies dataframe

The Movies dataset is available from within the `ncldata` package.

Recall that you can access this in the following way:

```r
install.packages("ncldata", repos="http://R-Forge.R-Project.org", type="source")
library(ncldata)
data(movies)
```

Or:

```r
movies = read.table("https://www.mas.ncl.ac.uk/~nlf8/basicR/movies.txt", header=TRUE)
```

We will now explore the movies dataset in RStudio.

The movies dataframe

The Movies dataset is available from within the `ncldata` package.

Recall that you can access this in the following way:

```
install.packages("ncldata", repos="http://R-Forge.R-
    Project.org", type="source")
library(ncldata)
data(movies)
```

Or:

```
movies = read.table("https://www.mas.ncl.ac.uk/~nlf8/
    basicR/movies.txt", header=TRUE)
```

We will now explore the movies dataset in RStudio.

Example

Load the `ncldata` package and read in the `movies` dataframe. Then answer the following questions.

(a) How many people submitted votes for the films "*10 Things I Hate About You*", "*Midnight in the Garden of Good and Evil*" and "*Zoolander*"?

(b) How many films are classified as "Action"? How many are classified as "Romance"? How many "Short" films are there in the database?

(c) How many films were made between the Years 1970 and 2000 (inclusive)?

(d) How many movies are longer than two hours in duration?

(e) How many movies made since the year 2005 (not including 2005) are longer than three hours in duration?

Example

Load and explore the `music` dataset from the `ncldata` package by typing the following lines of code:[1]

```
data(music)
# What's this dataset all about? Let's find out
?music
```

[1]This will only work if you have correctly installed the `ncldata` package and read the package into your working RStudio space using the `library` command. Or, read in directly from my webpage, replacing `movies.txt` with `music.txt`.

Example

(a) How many tracks do we have data for, and on how many variables?

(b) How many tracks do we have from *The Beatles* in this dataframe? What about *Lady GaGa*?

(c) How many tracks have a tempo of at least 130 beats per minute?

(d) Of the tracks with a tempo of at least 130 beats per minute, how many have been released since the year 2000 (including the year 2000)?

(e) What is the most popular key for tracks in this database?

Graphical and numerical summaries

R has very powerful built-in functions for summarising datasets both **graphically** and **numerically**.

Most of these are fairly intuitive, so in this section we provide only a **brief review**.

For example, we might want to explore the distribution of budgets in the `movies` dataframe.

As we know, there are some missing values in the dataset, but most of the built-in functions for data analysis in R have an argument we can invoke to ignore these.

Graphical and numerical summaries

```
# To find the mean of the movie budgets but ignoring the missing values
mean(movies$Budget, na.rm = TRUE)
 [1] 41245023

# And the standard deviation
sd(movies$Budget, na.rm = TRUE)
 [1] 68612118

# How about a range of numerical summaries with a single function?
summary(movies$Budget, na.rm = TRUE)
Min.   1st Qu.   Median       Mean  3rd Qu.      Max.   NA
  0      9e+6   2.5e+7  4.125e+7    5e+7   1.9e+9   998
```

Graphical and numerical summaries

From the listing above we can make a few observations:

- The mean budget is larger than the median: ∼US$ 41M, compared to US$ 25M. This indicates a **positively skewed** dataset, as the mean is being 'pulled up' by some very large budgets (of around USD$1.9Bn?! Which movie is this?! Must be amazing...)
- The standard deviation seems extremely large; again, this could be due to a small number of extremely large budgets...
- ... or perhaps some unrealistically *small* budgets (of US$ 0?!)

Graphical and numerical summaries

From the listing above we can make a few observations:

- The mean budget is larger than the median: ∼US$ 41M, compared to US$ 25M. This indicates a **positively skewed** dataset, as the mean is being 'pulled up' by some very large budgets (of around USD$1.9Bn?! Which movie is this?! Must be amazing...)
- The standard deviation seems extremely large; again, this could be due to a small number of extremely large budgets...
- ... or perhaps some unrealistically *small* budgets (of US$ 0?!)

Graphical and numerical summaries

From the listing above we can make a few observations:

- The mean budget is larger than the median: ∼US$ 41M, compared to US$ 25M. This indicates a **positively skewed** dataset, as the mean is being 'pulled up' by some very large budgets (of around USD$1.9Bn?! Which movie is this?! Must be amazing...)
- The standard deviation seems extremely large; again, this could be due to a small number of extremely large budgets...
- ... or perhaps some unrealistically *small* budgets (of US$ 0?!)

Graphical and numerical summaries

```
# Let's find out the position of the most expensive movie
which(movies$Budget == max(movies$Budget, na.rm=TRUE))
 [1] 1211

# What's the ▸ name of this movie?
movies$Title[1211]
 [1] "Enthiran"
# Never heard of it!

# Which movies were made for free?!
movies$Title[which(movies$Budget==0)]
 [1] "Homefront"        "Just Friends"    "Knock Knock"
 [4] "Now You See Me"   "Safe House"        "Snitch"
```

Graphical and numerical summaries

What about some **graphical summaries**?

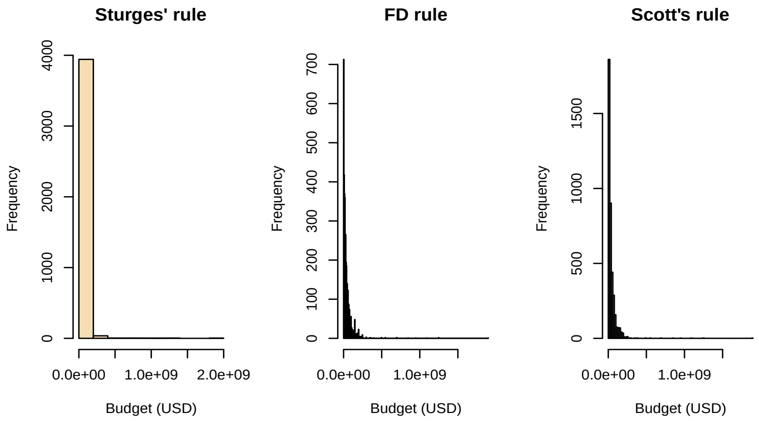
By far the most commonly-used graphics are **histograms** and **boxplots**.

Of course, there are so many more graphical options available in R; you should consult one of the books/texts mentioned at the start for a more comprehensive coverage.

Graphical and numerical summaries

```r
# Partitions the plotting space: 1 row, 3 columns
par(mfrow = c(1,3))

# Now we'll try three different methods for producing histograms

# 1. Sturges' rule; the default, tries to find an optimal number of bars
hist(movies$Budget, col="wheat",breaks="sturges",
 xlab="Budget (USD)", main="Sturges' rule")

# 2. Freedman-Diaconis's rule; tries to optimise the class width
hist(movies$Budget, col="gold",breaks="fd",
 xlab="Budget (USD)", main="FD rule")

# 3. Scott's rule; also tries to optimise the class width
hist(movies$Budget ,breaks="scott",
 xlab="Budget (USD)", main="Scott's rule")
```

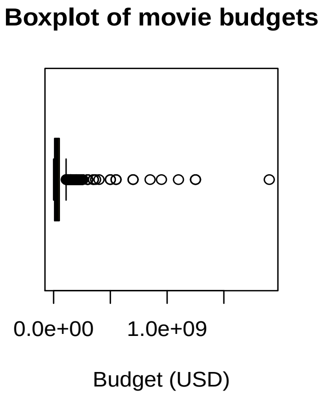
Graphical and numerical summaries

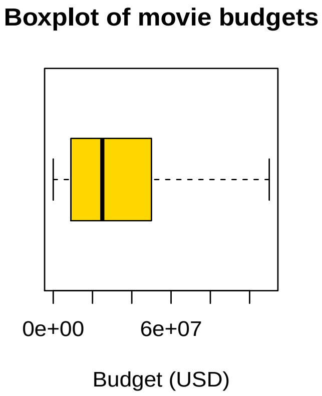


Graphical and numerical summaries

```
par(mfrow=c(1,2))
# Let's try a boxplot now
boxplot(movies$Budget, col="gold",
 main="Boxplot of movie budgets", horizontal=TRUE, xlab
    ="Budget (USD)")

# Outliers are spoiling things a bit. Let's remove them
boxplot(movies$Budget, col="gold",
 main="Boxplot of movie budgets", horizontal=TRUE, xlab
    ="Budget (USD)", outline=FALSE)
```

Graphical and numerical summaries





Graphical and numerical summaries

- The histograms reveal just how positively skewed the movie budget data are
- They also show how care should be taken with the method chosen to produce the histogram – of course, this is a subjective choice, and some experimentation might be necessary
- The boxplots also indicate just how skewed our dataset is – to get a handle on the distribution of most of of our data, it might be preferable to produce the boxplot with the outliers removed

Graphical and numerical summaries

- The histograms reveal just how positively skewed the movie budget data are
- They also show how care should be taken with the method chosen to produce the histogram – of course, this is a subjective choice, and some experimentation might be necessary
- The boxplots also indicate just how skewed our dataset is – to get a handle on the distribution of most of of our data, it might be preferable to produce the boxplot with the outliers removed

Graphical and numerical summaries

- The histograms reveal just how positively skewed the movie budget data are
- They also show how care should be taken with the method chosen to produce the histogram – of course, this is a subjective choice, and some experimentation might be necessary
- The boxplots also indicate just how skewed our dataset is – to get a handle on the distribution of most of of our data, it might be preferable to produce the boxplot with the outliers removed

Graphical and numerical summaries

Command	Comment	Example
`table`	Contingency table	`table(x)`
`barplot`	Generate a bar chart	`barplot(table(x))`
`hist`	Histogram	`hist(x)`
`plot`	Scatterplot	`plot(x, y)`
`points`	Add points to a plot	`points(x, y)`
`lines`	Add lines to a plot	`lines(x, y)`
`boxplot`	Box and whisker plot	`boxplot`

Example

Load the `music` dataset into RStudio, just as you did earlier.

(a) Find the mean and standard deviation for the duration of tracks in this dataset.

(b) Produce histograms for track duration and danceability; put these histograms on the same panel of plots, use different colours for the histograms (you can choose whichever colours you like!) and make sure you provide suitable titles and labels on the axes.

Example

(c) You are asked to investigate whether or not the duration of tracks has changed over time.

(i) Use the `plot` command in R to produce a scatterplot of the year each track was released against track duration.

(ii) Try out the following code, and comment on what you see in the plots in part (c):

```
boxplot(music$duration~music$decade)
```

(d) On average, how do energy scores for “Hip Hop” tracks compare to “Electronic” tracks?

Functions, for loops and if statements

A very powerful aspect of R is that it is relatively easy to write your own functions.

Functions take **inputs** and return a **single value**.

Let's look at some simple functions.

Functions

```
Fun1 = function(x){
  return(x*x)
 }
```

The key elements here are:

- The word **function**
- The brackets () which enclose the **argument list**
- A sequence of statements inside braces {}
- A **return** statement

Functions

```
Fun1 = function(x){
  return(x*x)
 }
```

The key elements here are:

- The word **function**
- The brackets () which enclose the **argument list**
- A sequence of statements inside braces {}
- A **return** statement

Functions

We **call** `Fun1` in the following way:

```
Fun1(5)
 [1] 25

y = Fun1(10)

y
 [1] 100

z = c(1, 2, 3, 4)
Fun1(z)
 [1] 1  4  9  16
```

Functions

Other variations include:

```
# Default argument
Fun2 = function(x=1){
   return(x*x)
   }

Fun2()
 [1] 1

Fun2(4)
 [1] 16
```

Functions

```
# Two arguments
Fun3 = function(x, y){
   return(x*y)
   }

Fun3(3,4)
 [1] 12
```

Functions

When we call a function, R first looks for **local** variables and then **global** variables.

For example, `Fun4` uses a global variable:

```
blob = 5
Fun4 = function(){
  return(blob)
  }

Fun4()
 [1] 5
```

Functions

However, `Fun5` uses a local variable:

```
Fun5 = function(){
  blob = 6
  return(blob)
   }

Fun5()
 [1] 6

blob
 [1] 5
```

Quiz #4

In the code below, what is the value of y?

```
fun1 = function(x=3){
  return(2*x)
  }

y = fun1(2)
```

1. 3
2. 4
3. 6
4. 10

Quiz #4

In the code below, what is the value of y?

```
fun1 = function(x=3){
  return(2*x)
  }

y = fun1(2)
```

1. 3
2. **4**
3. 6
4. 10

Quiz #5

In the code below, what is the value of y?

```
fun1 = function(x=3){
  x = 2
  return(2*x)
  }

y = fun1()
```

1. 3
2. 4
3. 6
4. 10

Quiz #5

In the code below, what is the value of y?

```
fun1 = function(x=3){
  x = 2
  return(2*x)
  }

y = fun1()
```

1. 3
2. **4**
3. 6
4. 10

Quiz #6

In the code below, what is the value of y?

```
x = 1
fun1 = function(x){
  x = 2
  return(x)
}
y = fun1(3)
```

1. 1
2. 2
3. 3
4. 4

Quiz #6

In the code below, what is the value of y?

```
x = 1
fun1 = function(x){
  x = 2
  return(x)
}
y = fun1(3)
```

1. 1
2. **2**
3. 3
4. 4

Quiz #7

In the code below, what is the value of y?

```r
x = 1
fun1 = function(x = 3){
  x = 2
  return(x)
}
y = fun1()
```

1. 1
2. 2
3. 3
4. 4

Quiz #7

In the code below, what is the value of y?

```
x = 1
fun1 = function(x = 3){
  x = 2
  return(x)
}
y = fun1()
```

1. 1
2. **2**
3. 3
4. 4

Quiz #8

In the code below, what is the value of y?

```
x = 1
fun1 = function(x = 3){
  return(2*x)
}
x = fun1(x)
y = fun1(x)
```

Quiz #8

In the code below, what is the value of y?

```
x = 1
fun1 = function(x = 3){
  return(2*x)
}
x = fun1(x)
y = fun1(x)
```

4

Conditionals: If statements

Conditional statements are features of a programming language which perform different computations or actions depending on whether a condition evaluates TRUE or FALSE.

The basic structure of an **if statement** is:

```
if(expr) {
   ##do something
 }
```

where expr is evaulated to be either TRUE or FALSE.

Conditionals: If statements

For example:

```
x = 5
y = 5
if(x < 5){
  y = 0
 }

y
 [1] 5
```

Conditionals: If statements

Another example:

```
x = 5
y = 5
if(x > 0){
    y = 0
 }

y
 [1] 0
```

Conditionals: If else statements

We can link together a number of if statements:

```
x = 0
if(x > 0){
   cat("x is greater than zero")
 } else if(x < 0){
   cat("x is less than zero")
 } else{
   cat("x must be zero!")
 }
x must be zero!
```

Conditionals: If else statements

We can link together a number of if statements:

```
x = 0
if(x > 0){
    cat("x is greater than zero")
 } else if(x < 0){
    cat("x is less than zero")
 } else{
    cat("x must be zero!")
 }
x must be zero!
```

Conditionals: If statements inside functions

We can also use if statements inside functions. For example:

```
IsNegative = function(value){
  l = FALSE
  if(value < 0){
    l = TRUE
  }
  return(l)
   }

IsNegative(1)
 [1] FALSE

IsNegative(-5.6)
 [1] TRUE
```

Quiz #9

In the code below, what is the value of z?

```
x = 5
y = "male"
if(x > 6 & y == "Female"){
  z = 0
  } else if(x < 5 & y =="Male"){
  z = 1
  } else{
    z = 2}
```

2

Quiz #9

In the code below, what is the value of z?

```
x = 5
y = "male"
if(x > 6 & y == "Female"){
  z = 0
  } else if(x < 5 & y =="Male"){
  z = 1
  } else{
     z = 2}
```

2

Conditionals: For loops

- At times we would like to perform some operation on a vector or a data frame
- Often R has built-in functions that will do this for you, e.g. `mean`, `sd`; other times we have to write our own functions
- For example, suppose we want to calculate $\sum_{i=1}^{10} i^2$
- In R we can use a `for` loop:

```
x = 0
for(i in 1:10){
  x = x + i^2
}
```

```
x
 [1] 385
```

Conditionals: For loops

- At times we would like to perform some operation on a vector or a data frame
- Often R has built-in functions that will do this for you, e.g. `mean`, `sd`; other times we have to write our own functions
- For example, suppose we want to calculate $\sum_{i=1}^{10} i^2$
- In R we can use a `for` loop:

```r
x = 0
for(i in 1:10){
  x = x + i^2
}
```

```
x
  [1] 385
```

Conditionals: For loops

- At times we would like to perform some operation on a vector or a data frame
- Often R has built-in functions that will do this for you, e.g. `mean`, `sd`; other times we have to write our own functions
- For example, suppose we want to calculate $\sum_{i=1}^{10} i^2$
- In R we can use a `for` loop:

```
x = 0
for(i in 1:10){
  x = x + i^2
}
```

```
x
 [1] 385
```

Conditionals: For loops

Example: Find

$$\sum_{j=-5}^{-1} e^j/j^2.$$

```r
total = 0
for(j in -5:-1){
  total = total + exp(j)/j^2
}
```

```
total
 [1] 0.4086594
```

Conditionals: For loops

Example: Find

$$\sum_{j=-5}^{-1} e^j/j^2.$$

```
total = 0
for(j in -5:-1){
  total = total + exp(j)/j^2
}
```

```
total
 [1] 0.4086594
```

Conditionals: For loops

Example: Find

$$\sum_{j=-5}^{-1} e^j/j^2.$$

```
total = 0
for(j in -5:-1){
  total = total + exp(j)/j^2
}
```

```
total
 [1] 0.4086594
```

Conditionals: For loops

Example: Find

$$\sum e^k/k^2, \quad k = 3, 6, 9, \ldots, 21.$$

```
total = 0
for(i in 1:7){
  k = i*3
  total = total + exp(k)/k^2
}

total
 [1] 3208939
```

Conditionals: For loops

Example: Find

$$\sum e^k/k^2, \quad k = 3, 6, 9, \ldots, 21.$$

```
total = 0
for(i in 1:7){
  k = i*3
  total = total + exp(k)/k^2
}

total
 [1] 3208939
```

Quiz #10

What is the value of x?

```
total = 0
 for(blob in 1:10){
  if(blob>9){
     total = total + 1
  }
}
x = total
```

1

Quiz #10

What is the value of x?

```
total = 0
 for(blob in 1:10){
  if(blob>9){
    total = total + 1
  }
}
x = total
```

1

Comparison of mean survival times

The table below contains the survival times (in weeks) of patients with leukaemia given 6-MP (group 1) or control (group 2):

Group 1	6	6	6	6	7	9	10
	10	11	13	16	17	19	20
	22	23	25	32	32	34	35
Group 2	1	3	4	4	5	5	8
	8	8	8	11	11	12	15
	17	22	23	27			

- Are survival times for patients in group 1 significantly higher than those in group 2?
- What sort of hypothesis test might we use here?
- Assumptions?

Comparison of mean survival times

The table below contains the survival times (in weeks) of patients with leukaemia given 6-MP (group 1) or control (group 2):

Group 1	6	6	6	6	7	9	10
	10	11	13	16	17	19	20
	22	23	25	32	32	34	35
Group 2	1	3	4	4	5	5	8
	8	8	8	11	11	12	15
	17	22	23	27			

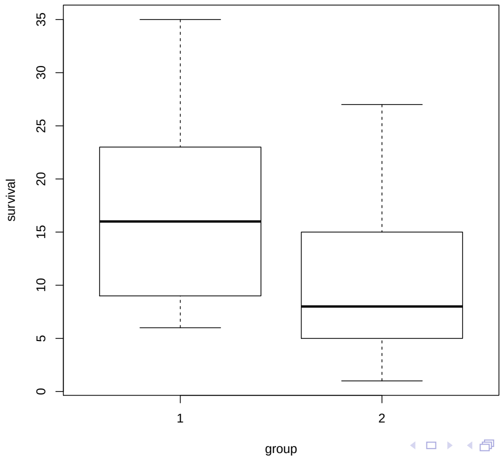
- Are survival times for patients in group 1 significantly higher than those in group 2?
- What sort of hypothesis test might we use here?
- Assumptions?

Comparisons

```r
survival = c(6, 6, 6, 6, 7, ...)
n1 = 21
n2 = 18
group = c(rep(1,n1), rep(2,n2))

mean(survival[group==1]); mean(survival[group==2])
[1] 17.09524
[1] 10.66667

(diff = mean(survival[group==1]) - mean(survival[group
    ==2]))
[1] 6.428571
```

Comparisons

```
boxplot(survival~group)
```



t-test

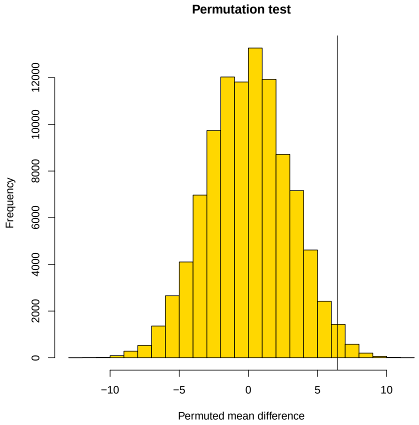
```
t.test(survival~group, alternative="greater", var.equal
    =TRUE)

  Two Sample t-test

data:  survival by group
t = 2.2445, df = 37, p-value = 0.01543
alternative hypothesis: true difference in means is
    greater than 0
95 percent confidence interval:
 1.596515      Inf
sample estimates:
mean in group 1 mean in group 2
       17.09524        10.66667
```

Permutation test

```
N = length(survival)
ind = 1:N

K = 100000
perm.diff = vector("numeric",K)
for(i in 1:K){
  ind.A = sample(ind, size = n1, replace=FALSE)
  ind.B = ind[-ind.A]

  new.A = survival[ind.A]
  new.B = survival[ind.B]

  perm.diff[i] = mean(new.A) - mean(new.B)}

(p = length(perm.diff[perm.diff>diff])/K)
[1] 0.01458
```

Permutation test

```r
hist(perm.diff, col="gold", main="Permutation test",
    xlab="Permuted mean difference")
abline(v = diff)
```


Permutation test
Frequency
Permuted mean difference