

Lecture: 2pm, Monday 30th September

MAS2906: Computational Probability and Statistics with R

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

Semester 1, 2024/25

2.5 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** of material from stage 1.

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.

2.5 Graphical and numerical summaries

```
1 # To find the mean of the movie budgets but ignoring the missing values
2 mean(movies$Budget, na.rm = TRUE)
3 [1] 41245023
4
5 # And the standard deviation
6 sd(movies$Budget, na.rm = TRUE)
7 [1] 68612118
8
9 # How about a range of numerical summaries with a single function?
10 summary(movies$Budget, na.rm = TRUE)
11 Min.    1st Qu.    Median      Mean   3rd Qu.      Max.     NA
12  0       9e+6     2.5e+7  4.125e+7   5e+7    1.9e+9    998
```

2.5 Graphical and numerical summaries

From the listing above we can make a few observations:

- The mean budget is larger than the median: $\sim$ 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?!)

2.5 Graphical and numerical summaries

From the listing above we can make a few observations:

- The mean budget is larger than the median: $\sim$ 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?!)

2.5 Graphical and numerical summaries

From the listing above we can make a few observations:

- The mean budget is larger than the median: $\sim$ 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?!)

2.5 Graphical and numerical summaries

```
1 # Let's find out the position of the most expensive movie
2 which(movies$Budget == max(movies$Budget, na.rm=TRUE))
3 [1] 1211
4
5 # What's the ▶ name of this movie?
6 movies$Title[1211]
7 [1] "Enthiran"
8 # Never heard of it!
9
10 # Which movies were made for free?!
11 movies$Title[which(movies$Budget==0)]
12 [1] "Homefront"      "Just Friends"    "Knock Knock"
13 [4] "Now You See Me" "Safe House"      "Snitch"
```

2.5 Graphical and numerical summaries

What about some **graphical summaries**?

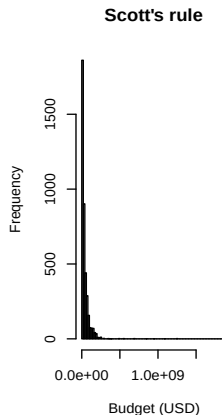
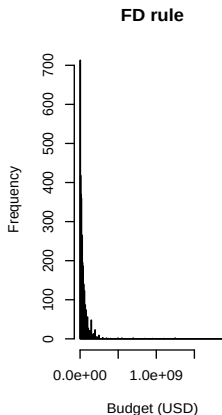
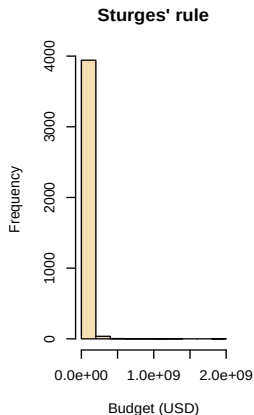
By far the most commonly-used graphics are **histograms** and **boxplots**.

Of course, there are so many more graphical options available in R; you should consult your MAS1608 notes for a more comprehensive coverage.

2.5 Graphical and numerical summaries

```
1 # Partitions the plotting space: 1 row, 3 columns
2 par(mfrow = c(1,3))
3
4 # Now we'll try three different methods for producing histograms
5
6 # 1. Sturges' rule; the default, tries to find an optimal number of bars
7 hist(movies$Budget, col="wheat", breaks="sturges",
8      xlab="Budget (USD)", main="Sturges' rule")
9
10 # 2. Freedman-Diaconis's rule; tries to optimise the class width
11 hist(movies$Budget, col="gold", breaks="fd",
12      xlab="Budget (USD)", main="FD rule")
13
14 # 3. Scott's rule; also tries to optimise the class width
15 hist(movies$Budget, breaks="scott",
16      xlab="Budget (USD)", main="Scott's rule")
```

2.5 Graphical and numerical summaries

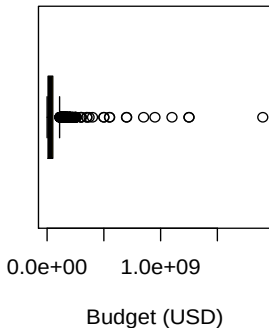


2.5 Graphical and numerical summaries

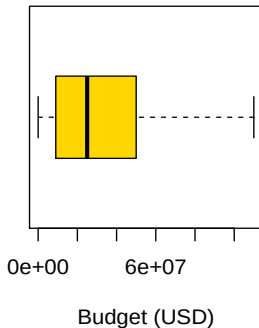
```
1 par(mfrow=c(1,2))
2 # Let's try a boxplot now
3 boxplot(movies$Budget, col="gold",
4         main="Boxplot of movie budgets", horizontal=TRUE, xlab
5         ="Budget (USD)")
6 # Outliers are spoiling things a bit. Let's remove them
7 boxplot(movies$Budget, col="gold",
8         main="Boxplot of movie budgets", horizontal=TRUE, xlab
9         ="Budget (USD)", outline=FALSE)
```

2.5 Graphical and numerical summaries

Boxplot of movie budgets



Boxplot of movie budgets



2.5 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 our data, it might be preferable to produce the boxplot with the outliers removed

2.5 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 our data, it might be preferable to produce the boxplot with the outliers removed

2.5 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 our data, it might be preferable to produce the boxplot with the outliers removed

2.5 Graphical and numerical summaries

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

Example 2.4



Load the `music` dataset into RStudio, just as you did in Example 2.3.

- (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 2.4

- (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):

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

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

2.6 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.

2.6 Functions

```
1 Fun1 = function(x){  
2     return(x*x)  
3 }
```

The key elements here are:

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

2.6 Functions

```
1 Fun1 = function(x){  
2     return(x*x)  
3 }
```

The key elements here are:

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

2.6 Functions

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

```
1 Fun1(5)
2 [1] 25
3
4 y = Fun1(10)
5
6 y
7 [1] 100
8
9 z = c(1, 2, 3, 4)
10 Fun1(z)
11 [1] 1 4 9 16
```

2.6 Functions

Other variations include:

```
1 # Default argument
2 Fun2 = function(x=1){
3     return(x*x)
4 }
5
6 Fun2()
7 [1] 1
8
9 Fun2(4)
10 [1] 16
```

2.6 Functions

```
1 # Two arguments
2 Fun3 = function(x, y){
3     return(x*y)
4 }
5
6 Fun3(3,4)
7 [1] 12
```

2.6 Functions

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

For example, Fun4 uses a global variable:

```
1 blob = 5
2 Fun4 = function() {
3   return(blob)
4 }
5
6 Fun4()
7 [1] 5
```

2.6 Functions

However, Fun5 uses a local variable:

```
1 Fun5 = function(){  
2   blob = 6  
3   return(blob)  
4 }  
5  
6 Fun5()  
7 [1] 6  
8  
9 blob  
10 [1] 5
```

Quiz #1

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

```
1 fun1 = function(x=3){  
2   return(2*x)  
3 }  
4  
5 y = fun1(2)
```



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

Quiz #1

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

```
1 fun1 = function(x=3){  
2   return(2*x)  
3 }  
4  
5 y = fun1(2)
```



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

Quiz #2

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

```
1 fun1 = function(x=3){  
2   x = 2  
3   return(2*x)  
4 }  
5  
6 y = fun1()
```



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

Quiz #2

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

```
1 fun1 = function(x=3){  
2   x = 2  
3   return(2*x)  
4 }  
5  
6 y = fun1()
```



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

Quiz #3

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

```
1 x = 1
2 fun1 = function(x){
3     x = 2
4     return(x)
5 }
6 y = fun1(3)
```



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

Quiz #3

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

```
1 x = 1
2 fun1 = function(x){
3     x = 2
4     return(x)
5 }
6 y = fun1(3)
```



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

Quiz #4

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

```
1 x = 1
2 fun1 = function(x = 3){
3     x = 2
4     return(x)
5 }
6 y = fun1()
```



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

Quiz #4

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

```
1 x = 1
2 fun1 = function(x = 3){
3     x = 2
4     return(x)
5 }
6 y = fun1()
```



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

Quiz #5

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

```
1 x = 1
2 fun1 = function(x = 3){
3     return(2*x)
4 }
5 x = fun1(x)
6 y = fun1(x)
```



Quiz #5

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

```
1 x = 1
2 fun1 = function(x = 3){
3     return(2*x)
4 }
5 x = fun1(x)
6 y = fun1(x)
```



4

2.6 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:

```
1 if (expr) {  
2     ##do something  
3 }
```

where `expr` is evaluated to be either TRUE or FALSE.

2.6 Conditionals: If statements

For example:

```
1 x = 5
2 y = 5
3 if (x < 5){
4     y = 0
5 }
6
7 y
8 [1] 5
```

2.6 Conditionals: If statements

Another example:

```
1 x = 5
2 y = 5
3 if (x > 0){
4     y = 0
5 }
6
7 y
8 [1] 0
```

2.6 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!
```

2.6 Conditionals: If else statements

We can link together a number of if statements:

```
1 x = 0
2 if(x > 0){
3     cat("x is greater than zero")
4 } else if(x < 0){
5     cat("x is less than zero")
6 } else{
7     cat("x must be zero!")
8 }
9 x must be zero!
```

2.6 Conditionals: If statements inside functions

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

```
1 IsNegative = function(value){
2   l = FALSE
3   if(value < 0){
4     l = TRUE
5   }
6   return(l)
7 }
8
9 IsNegative(1)
10 [1] FALSE
11
12 IsNegative(-5.6)
13 [1] TRUE
```

Quiz #6

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

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



Quiz #6

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

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



2

2.6 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
```

2.6 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:

```
1 x = 0
2 for(i in 1:10){
3   x = x + i^2
4 }
```

```
x
[1] 385
```

2.6 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:

```
1 x = 0
2 for(i in 1:10){
3   x = x + i^2
4 }
```

```
1 x
2 [1] 385
```

2.6 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
```

2.6 Conditionals: For loops

Example: Find

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

```
1 total = 0
2 for(j in -5:-1){
3     total = total + exp(j)/j^2
4 }
```

```
total
[1] 0.4086594
```

2.6 Conditionals: For loops

Example: Find

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

```
1 total = 0
2 for(j in -5:-1){
3     total = total + exp(j)/j^2
4 }
```

```
1 total
2 [1] 0.4086594
```

2.6 Conditionals: For loops

Example: Find

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

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

total
[1] 3208939
```

2.6 Conditionals: For loops

Example: Find

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

```
1 total = 0
2 for(i in 1:7){
3     k = i*3
4     total = total + exp(k)/k^2
5 }
6
7 total
8 [1] 3208939
```

Quiz #7

What is the value of x?

```
1 total = 0
2 for(blob in 1:10){
3   if(blob>9){
4     total = total + 1
5   }
6 }
7 x = total
```



Quiz #7

What is the value of x?

```
1 total = 0
2 for(blob in 1:10){
3   if(blob>9){
4     total = total + 1
5   }
6 }
7 x = total
```



1

2.6 Conditionals: The apply function

- We use the `apply` function when we want to **apply** the same function to every row or column of a dataframe
- For example, suppose we have a dataframe with three columns:

```
1 (df4 = data.frame(c1 = 1:4, c2 = 4:7, c3 = 2:5))
2   c1 c2 c3
3   1  4  2
4   2  5  3
5   3  6  4
6   4  7  5
```

2.6 Conditionals: The apply function

- The apply function takes (at least) three arguments
- The first argument is the data frame, the second the number 1 or 2 indicating row or column and the third a function to apply to each row or column
- So:

```
1 apply(df4, 1, mean)
2 [1] 2.333333 3.333333 4.333333 5.333333
```

calculates the mean value of every row.

2.6 Conditionals: The apply function

What about the standard deviation for every column? Easy!

```
apply(df4, 2, sd)
      c1      c2      c3
1.290994 1.290994 1.290994
```

2.6 Conditionals: The apply function

What about the standard deviation for every column? Easy!

```
1 apply(df4, 2, sd)
2   c1      c2      c3
3 1.290994 1.290994 1.290994
```

Things to think about...

- Can you write your own function in R to calculate the **mean** of a vector, without using any of R's built-in functions?
- Can you write your own function in R to calculate the **standard deviation** of a vector, without using any of R's built-in functions? Recall that

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2},$$

where n is the sample size and $\bar{x}$ is the sample mean.