MATHS 7107
Data Taming
Practical 4

Overview

To investigate the effect of fitness, sex, country of origin on red blood cell count (RBC), you have been given an Excel spreadsheet (`rbc2024.xlsx`) with observations[1] made on 500 subjects.

The variables are

- `sex`: male or female, recorded as `M/F`.
- `fitness`: a measure of fitness lying between 0 and 100. Zero corresponds to completely unfit, while 100 is the maximum possible fitness.
- `country`: there are three countries denoted by 1, 2, or 3.
- `RBC`: red blood cell count in millions of red blood cells in one μL.

What to do with missing data:

- any values of `NA` or `XX` denotes that the data missing, and the observation should be deleted.

In this practical, the main thing we'll do is clean the dataset `rbc2024.xlsx`. We'll do it by going through a set of "Rules" that you can try to follow when presented with a new data set (although this will probably depend on the data set).

IMPORTANT!

In the prac we will use Excel to "pre-clean" our data, but in your assignments make sure you do all the cleaning in `R`, otherwise your code won't run properly on the client's data file.

Rule 0: Setup your code and data structure

- Setup an `RStudio` project for this prac.
- Put the data in a `/data` subdirectory.
- Open a new script.
- Are there any standard packages that you normally load? (Hint: yes there are!)
- At the top of your script write the code to load these packages.
- Save the script.

[1]All data appearing in this work is fictitious. Any resemblance to real data, living or dead, is purely coincidental.

Rule 1: Look at the data

Let's start with pre-cleaning of the `.xlsx` file.

- It is best practice to keep a copy of your original data so save a new file to clean, `originalname_clean.xlsx`.
- Remove `useless_figure`.
- Move the data so it starts at cell `A1`.
- Find and remove any extra comments in the data
- Delete the mean calculations at the bottom of the data.
- Count up how many missing values there are, ie. `NA`, `XX` or completely missing.

IMPORTANT!

When you are finished with Excel, make sure you close it before you go on to the next step.

Eat your veggies!

Make sure you type all these commands **BY HAND!** (Don't just copy and paste.) You will learn a lot faster if you type the code!!

Rule 2: Import your data

Since we have data in an Excel `.xlsx` file, we could convert it to `.csv` first and then use `read_csv()`. But for this prac we'll try the `readxl` package.

The `read_excel` function takes Excel data and turns it into a tibble. It also has a useful option `na =` allowing us to automatically find missing values and give them a value of `NA`. If we give the option a vector of character strings, then when `read_excel` encounters any of them (or a value that is missing completely) it will replace it with `NA`. So our command is:

```r
rbc <- read_excel("./data/rbc2024_clean.xlsx", na = c("NA", "XX"))
rbc
```

```
# A tibble: 500 × 5
   name  sex   fitness country   RBC
   <chr> <chr>   <dbl>   <dbl> <dbl>
 1 132   male     76.2       1  5.27
 2 309   male     92.3       2  6.42
 3 234   M        65.2       1  3.61
 4 400   F        89.4        3  6.15
 5 316   F        53.9        2  3.16
 6 483   F        60.8        3  4.24
 7 28    M        95.5        1  5.36
 8 255   F        62.8        2  4.43
 9 209   M        77.0        2  4.36
10 34    M        86.0        3  4.25
# i 490 more rows
# i Use `print(n = ...)` to see more rows
```

(If you got an error, then check to make sure you closed Excel.)

2.1 Deal with missing data `NA`

In general, you need to be careful about missing values. Often you can just ignore the missing value and use the rest of the data for that observation, but other times it can indicate that the data is unreliable.

- A good rule is don't throw away data unless:
  - it's **definitely** useless;
  - it's **definitely** unreliable;
  - the client or data provider tells you to.

In this case, we were told that if the observation contained a value of `NA` or `XX` then the whole observation should be deleted. So that's what we'll do. First let's find the missing data. For that, we'll use `inspect_na` from the `inspectdf` package.

```
inspect_na(rbc)
```

```
# A tibble: 5 × 3
  col_name   cnt  pcnt
  <chr>    <int> <dbl>
1 fitness     10   2
2 RBC          2   0.4
3 name         0   0
4 sex          0   0
5 country      0   0
```

So we can see that there are 10 `NA` values in `fitness` and 2 in `RBC`. Confirm that this matches what you found in **Rule 1**, when you were looking at the file in Excel. Now we just have get rid of them. For this we can use the `is.na` function, and we'll need some logical operators:

- `&`: AND
- `|`: OR
- `!`: NOT

We now type:

```
rbc <- filter(rbc, !(is.na(fitness) | is.na(RBC)))
```

```
# A tibble: 488 × 5
   name  sex   fitness country   RBC
   <chr> <chr>   <dbl>   <dbl> <dbl>
 1 132   male     76.2       1  5.27
 2 309   male     92.3       2  6.42
 3 234   M        65.2       1  3.61
 4 400   F        89.4       3  6.15
 5 316   F        53.9       2  3.16
 6 483   F        60.8       3  4.24
 7 28    M        95.5       1  5.36
 8 255   F        62.8       2  4.43
 9 209   M        77.0       2  4.36
10 34    M        86.0       3  4.25
# i 478 more rows
# i Use `print(n = ...)` to see more rows
```

Rule 3: Get it into a tibble

Luckily for us, `readxl` does this automatically. In general, the `tidyverse` should have enough tools to get data straight into a tibble, but there may be cases where you need to convert it yourself. We'll see that in Module 5.

Rule 4: Convert from wide form to long form

Not needed in this case as already in long form, ie. every variable is already its own column.

Rule 5: Look at each column

5.2 `name` column

The first column is not mentioned in the variables description, and so we should remove it.

- **Be extremely careful here**: it may be that the client wants to keep the `name` data, and they just forgot to include it in the variable list. You should check with them before you do any final reports.

In this case we'll assume it's deliberate, and the `name` column is superfluous, so we'll delete it.

```r
rbc <-
  rbc %>%
  select(-name)
rbc
```

```
# A tibble: 488 × 4
   sex   fitness country   RBC
   <chr>   <dbl>   <dbl> <dbl>
 1 male     76.2       1  5.27
 2 male     92.3       2  6.42
 3 M        65.2       1  3.61
 4 F        89.4       3  6.15
 5 F        53.9       2  3.16
 6 F        60.8       3  4.24
 7 M        95.5       1  5.36
 8 F        62.8       2  4.43
 9 M        77.0       2  4.36
10 M        86.0       3  4.25
# i 478 more rows
# i Use `print(n = ...)` to see more rows
```

Question:

1. Repeat this operation using the `mutate` command.

5.3 sex

The only values in this columns should be just `F` and `M`, but a quick check gives the extra value `male`.

```
rbc %>%
  count(sex)
```

```
## # A tibble: 3 x 2
##   sex      n
##   <chr> <int>
## 1 F       226
## 2 M       243
## 3 male     19
```

So we will convert to `M`.

```
rbc <-
  rbc %>%
  mutate(
    sex = fct_recode(sex, "M" = "male")
  )
rbc %>% count(sex)
```

```
## # A tibble: 2 x 2
##   sex      n
##   <chr> <int>
## 1 F       226
## 2 M       262
```

In the code above, we used the `fct_recode()` function, which takes a string or factor type and always returns a factor. Lastly, we need to Tame this tibble, so the `sex` column should be a `factor` data type, but as we just said, `fct_recode()` converts it to a `<fct>` type for us. But in general, we would have to do this step ourselves.

Question:

2. In our data categorisation, what sort of variable is `sex`?

If we did need to convert then we type:

```
rbc$sex <- rbc$sex %>%
  as.factor()
rbc
```

```
# A tibble: 488 × 4
   sex   fitness country   RBC
   <fct>   <dbl>   <dbl> <dbl>
 1 M        76.2       1  5.27
 2 M        92.3       2  6.42
 3 M        65.2       1  3.61
 4 F        89.4       3  6.15
 5 F        53.9       2  3.16
 6 F         60.8       3  4.24
 7 M         95.5       1  5.36
 8 F         62.8       2  4.43
 9 M         77.0       2  4.36
10 M         86.0       3  4.25
# i 478 more rows
# i Use ‘print(n = ...)‘ to see more rows
```

5.4 Fitness

We are told that fitness should lie between 0 and 100, so lets check, using the `NOT` operator.

```
rbc %>%
  filter(!between(fitness, 0, 100))
```

```
## # A tibble: 1 x 4
##   sex fitness country   RBC
##   <fct>    <dbl>   <dbl> <dbl>
## 1 M          105       3  4.56
```

So this is probably a data entry error, and it’s not clear what the true value should be. In this case, we really need to check with the data provider and we’ll assume they said that the entire observation is an error, and so it should be removed.

```
rbc <-
  rbc %>%
  filter(between(fitness, 0, 100))
```

It’s always good to check that your manipulation worked, and did what you expected. So now we check for errors again and we find

```
rbc %>%
  filter(!between(fitness, 0, 100))
```

```
# A tibble: 0 × 4
# i 4 variables: sex <fct>, fitness <dbl>, country <dbl>, RBC <dbl>
```

So the erroneous observation has now gone.

Lastly we check the data type: we see that this is a continuous quantitative continuous variable, and so `dbl` is the correct data type.

5.5 Country

We are told that there should be three levels: 1,2 and 3. So let’s check:

```
rbc %>%
  count(country)
```

```
## # A tibble: 4 x 2
##   country     n
##     <dbl> <int>
## 1       1   181
## 2       2   164
## 3       3   141
## 4       4     1
```

That row with country equal to `4` must be wrong.

```
rbc %>%
  filter(
    country == 4
  )
```

```
## # A tibble: 1 x 4
##   sex fitness country   RBC
##   <chr>    <dbl>   <dbl> <dbl>
## 1 M         59.5       4  3.61
```

Again we need to check with the data provider, and they told us that this should be country `3`. So we'll change it:

```
rbc <-
  rbc %>%
  mutate(country = ifelse(country == 4, 3, country ))

rbc %>%
  count(country)
```

```
# A tibble: 3 × 2
  country     n
    <dbl> <int>
1       1   181
2       2   164
3       3   142
```

The `ifelse` command has syntax

`ifelse(condition, return if true, return if false)`

Experiment with changing the arguments of the code above to make sure you know how this works.

Lastly, we just have to tame the data, so the `country` variable should be a factor.

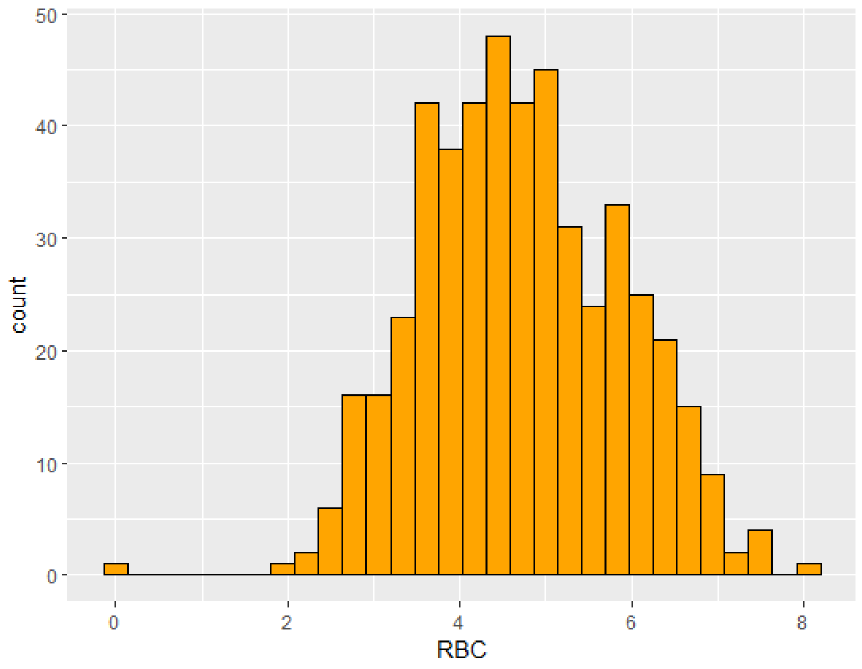
```
rbc <-
  rbc %>%
  mutate(
    country = as.factor(country)
  )
```

5.6 RBC

For this variable, we have not been given a range, so lets have a look to see if the data makes sense.

```
rbc %>%
  ggplot(aes(RBC)) +
  geom_histogram(colour = "black", fill = "orange")
```

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



So generally looks good except for that very low one. Let's check it

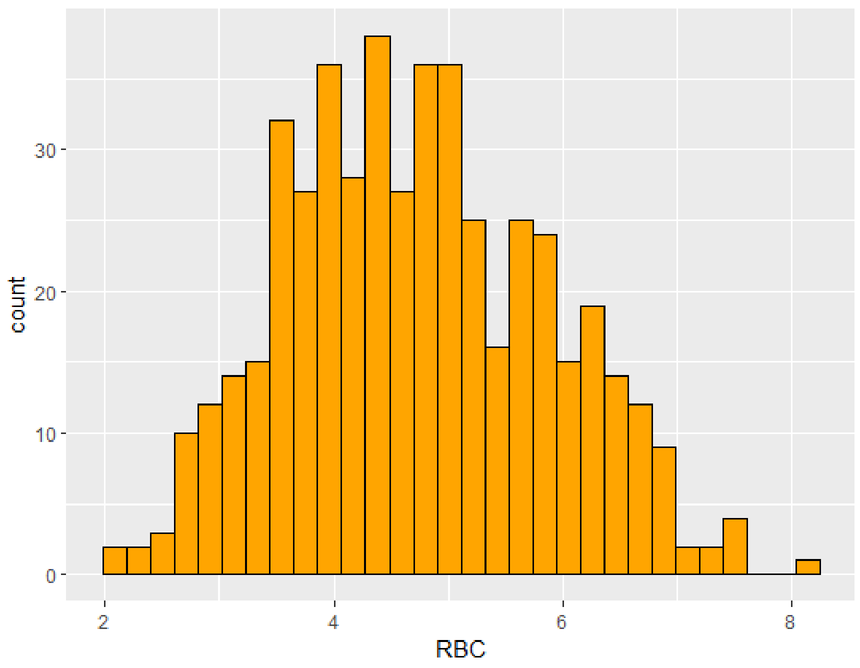
```
rbc %>%
  filter(RBC < 1)
```

```
## # A tibble: 1 x 4
##   sex fitness country   RBC
##   <chr>     <dbl> <fct>   <dbl>
## 1 M          72.0 1           0
```

A red blood cell count of zero is not good (for a living person). Check with data provider who says remove.

```
rbc <-
  rbc %>%
  filter(
    RBC > 1
  )
```

Now we check the histogram again, and it looks much better:



Rule 6: create new columns (variables)?

No new columns needed.

Rule 7: Save the data and write it up

Save this data and make sure you date it. Keep this Rmd as information on the cleaning.

Your turn

Questions:

3. How many observations are there in the final dataset?
4. How many males are there in the dataset? (Hint: try the `count` function.)
5. How many observations are there in country 2?
6. What is the mean `RBC` to 1 decimal place?
7. Produce a scatter plot of `fitness` against `RBC` and colour the points differently for males and females. Describe the relationship between `RBC` and `fitness`. Also, what do you notice about males and females?
8. Produce a boxplot of `fitness` and `country` with separate boxes for males and females. Write some text to describe the distribution. (See page 12 of Module 4 for how to describe the distribution.)

[The code for generating the answers in this section is contained in `DT_prac4_solns.R`.]