Data Analysis with R and the Tidyverse

Session 1

John Thompson

08 August 2019

Reference Intervals for African Buffalo

Introduction

The aim of this session is to produce a complete data analysis in R, from importing the data through to preparing a report. To illustrate this process we will analyse data described in the paper

Couch, C. E., Movius, M. A., Jolles, A. E., Gorman, M. E., Rigas, J. D., & Beechler, B. R. (2017).
Serum biochemistry panels in African buffalo: Defining reference intervals and assessing variability across season, age and sex.
PloS one, 12(5), e0176830.

The paper can be found at https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0176830 and the authors have kindly made their data available by placing a csv file in the dryad repository at https://datadryad.org/resource/doi:10.5061/dryad.kf6r5

The researchers captured wild buffalo from the Kruger National Park in South Africa and moved them to a 900-hectare enclosure where they could live free but be protected from large preditors. At intervals of about 3 months during 2014 and 2015 buffalo were recaptured and sedated so that blood samples could be taken. The samples were analysed for a range of biochemicals with a view to establishing reference intervals that could be used to determine the health of buffalo captured in the wild.

In this data analysis we will emphasise

- Use of the pipe
- Storing data in a data frame
- Importing data from a csv file
- Saving data in rds format
- Data manipulation functions, select(), rename(), filter(), group_by() and summarise()
- Visualisation with qplot()
- Creating an html report with rmarkdown

Necessary skills

The Pipe

One of the big advantages of the tidyverse is that it makes it easy to write very readable code. This legibility is greatly enhanced by linking R's functions using the pipe operator.

Let's consider different ways of combining functions in R. Suppose that we need to transform data object x into data object y and that this involves using three functions. We could code this as,

```
a <- fun1(x)
b <- fun2(a)
y <- fun3(b)
```

Here the stages are explicit but we are left with unwanted intermediate data objects a and b. If we code in this style we will collect a great number of unwanted intermediates and eventually we will need to declutter our workspace by deleting them.

An alternative approach is to write the code as,

```
y <- fun3(fun2(fun1(x)))
```

Using this method the intermediates are not stored but the code is much more difficult to read and therefore much more error prone.

The pipe, written %>%, enables us to avoid storing intermediates, while retaining legibility. Using this approach the code would be written

```
x %>%
  fun1() %>%
  fun2() %>%
  fun3() -> y
```

We read this code as saying, start with the data object x, pipe or feed x into fun1(), take the result and pipe it into fun2(), then take the new result and pipe it into fun3(). Finally, the result is assigned to the data object named y.

Notice that the code is spread over several lines, this is non-essential but it is good practice because it improves legibity. The assignment arrow on the last line points to the right so that the code mirrors the natural order in which the stages are executed.

In a pipe, the input data object becomes the **first** argument of the next function, so

```
x %>%
  fun1(y, z)  ->  w
```

is exactly equivalent to coding

```
w <- fun1(x, y, z)
```

It is possible to use any of the three coding styles but we will use the pipe in our scripts whenever we need to chain together a series of functions.

Using a package

Since we plan to use the tidyverse in our data analyses, the tidyverse packages will need to be installed from the internet before we start. The packages that we need are all available from the CRAN repository and they can be downloaded in RStudio using the 'Packages' tab. The tab displays the packages that are currently installed and has an install button for adding others. Rather than install the tidyverse packages one by one, there is a short-cut that installs the most commonly used of the tidyverse packages in one step.

Click the install button, a window will open that enables you to specify the package that you want. CRAN is the default repository, so in the Packages box type the word 'tidyverse' and make sure that the install dependencies box is checked. It is often the case that one package will use functions taken from another package and by checking the dependencies box you ensure that everything that you need will be installed. Installation is automatic but may take a minute or so depending on the speed of your internet connection.

Once it is completed you will see that the list of available packages includes some new entries including tidyverse packages such as dplyr and ggplot2.

The packages have been copied from CRAN and stored in a local folder called a library. The path to your R library was automatically identified by RStudio and it was displayed in the window that appeared when you clicked the install button.

When you want to use the functions included in a particular package within one of your R scripts you need to include a line of code such as

```
library(dplyr)
```

This tells R to make the functions of the dplyr package available to this script and R automatically goes to your library and retrieves the functions. This process is called loading the package.

When you want to use the tidyverse packages there is another short-cut that loads the most frequently used packages in one step, just start the script with the line

```
library(tidyverse)
```

In the demonstration we will use a package called **fs**, which you will need to install from CRAN.

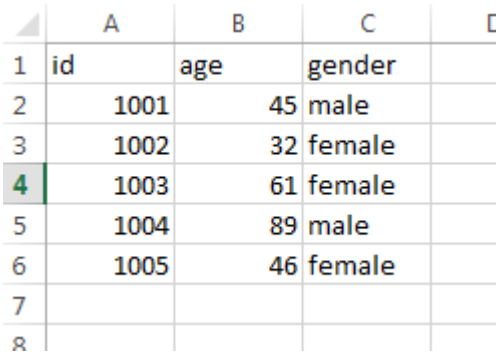
Data frames

To simplify our workflow, we will do most of our analyses using a type of R data object called a data frame. By limiting ourselves to data frames we will make R simpler to learn and we will make our scripts easier to follow. Eventually we will need to make use of other types of data object but it is amazing just how much can be done with data frames.

A data frame is a rectangular array of data in which rows represent subjects and columns represent variables, so it is not unlike the structure of a spreadsheet and if we import data from a spreadsheet we can arrange that it is automatically converted into a data frame in R.

The tidyverse functions use a data structure called a tibble (tidy table) that is essentially identical to a data frame except that when you print it, the output is much neater.

To illustrate the structure of a tibble, let's look at a very small example of data on 5 subjects that has been entered into an Excel worksheet



	A	B	C	D
1	id	age	gender	
2	1001	45	male	
3	1002	32	female	
4	1003	61	female	
5	1004	89	male	
6	1005	46	female	
7				
8				

If we read these data into R, the tibble would print as

```
## # A tibble: 5 x 3
##      id   age gender
##   <dbl> <dbl> <chr>
## 1  1001    45 male
## 2  1002    32 female
## 3  1003    61 female
```

```
## 4  1004    89 male
## 5  1005    46 female
```

In this tibble, id and age are stored as double precision numbers (dbl) and gender is stored as a character string (chr). The numbers on the left are row numbers and are there to enhance the display; they are not stored in the tibble.

I use the convention of ending the names of my data frames and tibbles with DF, as in exampleDF. In that way, it is always clear what type of data object is being referred to. This is not standard practice.

Reading a csv file

A csv file is just a text file containing data separated by commas. It could be viewed using any program that can handle text files including notepad, Word and the RStudio Editor.

The tidyverse includes a function read_csv() that reads the contents of a file and saves them as a data frame.

```
library(tidyverse)

read_csv( "Z:/Projects/buffalo/data/rawData/buffalo_serum.csv")  -> buffaloDF
```

Notice that in the call to read_csv I have not specified the argument name. When the argument name is omitted the arguments are assigned in the order that they are defined, in this case the file path is assigned to the first argument of read_csv. Omitting the argument name is not best practice, but everyone does it.

Many people would code this with the assignment is the other direction, as in

```
buffaloDF <- read_csv( "Z:/Projects/buffalo/data/rawData/buffalo_serum.csv")
```

It makes no difference but it is best to be consistent.

Saving data

There are several different R formats for saving data that enable fast access. The format that I recommend and which we will use on this course is called RDS. To write data to a file in RDS format we use the tidyverse function write_rds() and to read it again we use read_rds(). So we might extend our previous code

```
library(tidyverse)

read_csv( "Z:/Projects/buffalo/data/rawData/buffalo_serum.csv")  -> buffaloDF

write_rds( buffaloDF,
           path   = "Z:/Projects/buffalo/data/rData/buffalo.rds")
```

There is no assignment with the write_rds() function because the data are sent to the file and are not copied to another R data object.

To read the data from the rds file we could use

```
read_rds("Z:/Projects/buffalo/data/rData/buffalo.rds") -> buffaloDF
```

Data manipulation

In almost every project we will need to manipulate the data, perhaps selecting a subset of the variables, filtering a subset of the subjects, sorting the data etc. The tidyverse includes a package called **dplyr** that contains dozens of functions for manipulating data that are stored in tibbles but for this data analysis we will only need six of them,

- **rename()**: change the name of a column (variable)
- **select()**: choose a subset of the columns (variables)
- **filter()**: choose a subset of the rows (subjects)
- **mutate()**: create a new column (variable) by calculation
- **group_by()**: group together the rows (subjects) into subsets
- **summarise()**: calculate summary statistics

From a study of buffalo, let's suppose that we have a tibble called **myBuffaloDF** that contains 494 rows (buffalo) and 5 columns (variables) called

- period: occasion when the sample was taken, coded 1 to 5
- sex: sex of the buffalo coded "M" or "F"
- age: estimated age of the buffalo in years
- alb: serum albumin level
- ca: serum calcium level

```
print(myBuffaloDF)
```

```
## # A tibble: 494 x 5
##    period sex     age   alb    ca
##     <int> <chr> <dbl> <dbl> <dbl>
##  1      1 F      0.02   4.3  10.1
##  2      1 F      0.02   4.1  11.4
##  3      1 M      0.26   4.7   8.9
##  4      1 M      0.26   3.9   7.5
##  5      1 F      0.5    5.1  10.3
##  6      1 M      0.5    5.2  10.2
##  7      1 M      0.67   5     9.6
##  8      1 F      0.67   4.8   9.1
##  9      1 F      1.01   4.9   9.7
## 10      1 M      1.01   4     9.1
## # ... with 484 more rows
```

Here are some examples of data manipulation using dplyr. First we will filter the rows to obtain the data on female buffalo. Notice the use of the double equals sign for testing equality and the quotes for character strings. In R you can choose whether to use double quotes or single quotes, but as always, it is best to be consistent.

```
myBuffaloDF %>%
  filter( sex == "F" ) %>%
  print()
```

```
## # A tibble: 333 x 5
##    period sex     age   alb    ca
##     <int> <chr> <dbl> <dbl> <dbl>
##  1      1 F      0.02   4.3  10.1
##  2      1 F      0.02   4.1  11.4
##  3      1 F      0.5    5.1  10.3
```

```
##  4      1 F      0.67   4.8   9.1
##  5      1 F      1.01   4.9   9.7
##  6      1 F      1.01   4.5   9
##  7      1 F      1.01   4     9.6
##  8      1 F      1.51   4.2   9.1
##  9      1 F      1.51   4.4   9.6
## 10      1 F      2.01   4.5   9.1
## # ... with 323 more rows
```

Next we will filter the male buffalo but only display the albumin and calcium levels.

```
myBuffaloDF %>%
  filter( sex == "M" ) %>%
  select( alb, ca ) %>%
  print()
```

```
## # A tibble: 161 x 2
##      alb    ca
##    <dbl> <dbl>
##  1   4.7   8.9
##  2   3.9   7.5
##  3   5.2  10.2
##  4   5     9.6
##  5   4     9.1
##  6   4.2   9.6
##  7   4.2   9.1
##  8   4.5   9.6
##  9   4.2   9.3
## 10   4.1   9.2
## # ... with 151 more rows
```

In this example we rename the biochemical variables.

```
myBuffaloDF %>%
  rename( albumin = alb,
          calcium = ca ) %>%
  print()
```

```
## # A tibble: 494 x 5
##    period sex     age albumin calcium
##     <int> <chr> <dbl>   <dbl>   <dbl>
##  1      1 F      0.02     4.3    10.1
##  2      1 F      0.02     4.1    11.4
##  3      1 M      0.26     4.7     8.9
##  4      1 M      0.26     3.9     7.5
##  5      1 F      0.5      5.1    10.3
##  6      1 M      0.5      5.2    10.2
##  7      1 M      0.67     5       9.6
##  8      1 F      0.67     4.8     9.1
##  9      1 F      1.01     4.9     9.7
## 10      1 M      1.01     4       9.1
## # ... with 484 more rows
```

I could have coded this as

```
myBuffaloDF%>%rename(albumen=alb,calcium=ca)%>%print()
```

It would still have worked correctly but this type of illegible coding is bad practice because it is difficult to

read and more likely to contain errors.

Now we calculate some summary statistics for the entire sample of buffalo.

```
myBuffaloDF %>%
  summarise( n       = n(),
             meanAlb = mean(alb),
             sdAlb   = sd(alb) ) %>%
  print()
```

```
## # A tibble: 1 x 3
##       n meanAlb sdAlb
##   <int>   <dbl> <dbl>
## 1   494    4.03 0.416
```

In the next example we group the rows by sex and then calculate the same summary statistics.

```
myBuffaloDF %>%
  group_by( sex ) %>%
  summarise( n       = n(),
             meanAlb = mean(alb),
             sdAlb   = sd(alb) ) %>%
  print()
```

```
## # A tibble: 2 x 4
##   sex       n meanAlb sdAlb
##   <chr> <int>   <dbl> <dbl>
## 1 F       333    4.05 0.416
## 2 M       161    3.99 0.415
```

Now we divide albumin by calcium and save the result in a column called ratio

```
myBuffaloDF %>%
  mutate( ratio = alb/ca  ) %>%
  print()
```

```
## # A tibble: 494 x 6
##    period sex     age   alb    ca ratio
##     <int> <chr> <dbl> <dbl> <dbl> <dbl>
##  1      1 F      0.02   4.3  10.1 0.426
##  2      1 F      0.02   4.1  11.4 0.360
##  3      1 M      0.26   4.7   8.9 0.528
##  4      1 M      0.26   3.9   7.5 0.52
##  5      1 F      0.5    5.1  10.3 0.495
##  6      1 M      0.5    5.2  10.2 0.510
##  7      1 M      0.67   5     9.6 0.521
##  8      1 F      0.67   4.8   9.1 0.527
##  9      1 F      1.01   4.9   9.7 0.505
## 10      1 M      1.01   4     9.1 0.440
## # ... with 484 more rows
```

Visualisation

The tidyverse uses a package called ggplot2 for plotting data. ggplot2 is very powerful and we will develop our skills in using it as we progress through the sessions. For this analysis we will use a short-cut function called qplot() to create a basic plot.

```
myBuffaloDF %>%
  qplot( x = age, y = alb, data = .
         )
```

The full stop for the data argument in qplot() needs an explanation. It is needed because of the way that the pipe works. You will recall that the pipe assigns the input data object to the **first** argument of the function. A problem arises if we want to assign the input to an argument that is not first in the list and the solution is to write a full stop where we want the assignment to be made. In the case of qplot(), we want myBuffaloDF to be assigned to the data argument.

Report writing

If you want to control the format of your statistical report then you will need to create a file that combines text with the formatting information. The language in which the formatting is specified is called a **markup language**; perhaps the most commonly used markup languages are latex and html. These markup languages are designed to be all-embrassing, so more or less whatever format you want, there is a way of producing it. This is a great quality but these markup languages have the disadvantage that they are complex to use.

markdown was created in response to the complexity of complete markup languages. Markdown can only create the most commonly used formatting but as a consequence it can be kept relatively simple. In markdown a typical piece of formatted text would be

```
## Methods

The data were analysed by **analysis of variance**.
```

requests a medium sized header and ** surrounds text that we want to be printed in bold.

When Yihui Xie wanted to write a report writing package for R, he decided to base it on the markdown markup language and he called the package **rmarkdown**. With this package you can create a markdown

document and then insert chunks of R code within it. When the document is created, in a process that has become known as knitting, the R code is executed and the results are inserted into the document.

The blocks of code that are inserted into the markdown file are referred to as **chunks** and each chunk starts with three backticks and r in cruly brackets and ends with another three backticks

```
```r
```

Between these markers you insert standard R code.

RStudio has developed a number of tools to simply the use of rmarkdown. You can use the menus **File - New File - R Markdown...** to create a dummy rmarkdown script. RStudio will ask you to specify a title and author before creating the script for a short report. This short script can be edited to create the script needed for your own report. There is also a "Knit" button that automatically appears above any rmarkdown script that is open in RStudio and clicking this button causes the report to be created.

Here is the dummy rmarkdown file that RStudio creates

```
---
title: "R Markdown Example"
author: "John Thompson"
date: "August 26, 2018"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring
HTML, PDF, and MS Word documents. For more details on using R Markdown see
<http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both
content as well as the output of any embedded R code chunks within the document.
You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent
printing of the R code that generated the plot.
```

The script begins with a header that starts and ends with lines containing three dashes. The header is written in a language called YAML (yet another markup language). It includes information about the document and instructions on how to process it. In this case rmarkdown is instructed to create an html document.
```

Next comes the text for the document organised into sections with headers “R Markdown” and “Including Plots”. Interspersed within the text are the chunks of R code.

A common way of working is start with this dummy file but to remove all of the text and just leave the YAML header. Then you add your own text and R chunks.

rmarkdown has been developed to the point where it can produce, pdfs, webpages, Word documents, slides, articles for journals, books and conference posters. We will see some of these extensions in future data analyses.

For this first data analysis we will

- develop the code for our data analysis using R scripts
- prepare the text of our report as an rmarkdown file
- copy the R code from the scripts into the report
- knit the report in RStudio to create an html document that can be read in a browser

Demonstration: Buffalo reference intervals

Study design

The researchers captured wild buffalo from the Kruger National Park in South Africa and moved them to a 900-hectare enclosure where they could live free but be protected from large preditors. At intervals of about 3 months during 2014 and 2015 buffalo were captured and sedated so that blood samples could be taken. The samples were analysed for a range of biochemicals with a view to establishing reference intervals that could be used to determine the health of buffalo captured in the wild.

Some buffalos were sampled more than once. Calcium was measured in units of mg/dL.

Aim

The aim of this analysis is to obtain a 90% reference interval for calcium and to report the findings in an html document.

Project set-up

I worked though the checklist for creating a new project given in the course introduction.

- named the project ‘buffalo’
- created a project folder called “Z:/Projects/buffalo”.
- generated the standard subfolders
- copied the csv file from the dryad respository into data/rawData
- created an RStudio project based on the project folder
- created a text file diary.txt in the docs folder and opened it in RStudio’s Editor

Here are the first few lines that I wrote in diary.txt

```
26 June 2018     Project: buffalo

Study to obtain reference ranges for biochemicals in African Buffalo used as the first
example in the "Data Analysis with the tidyverse" course

Couch, C. E., Movius, M. A., Jolles, A. E., Gorman, M. E.,
Rigas, J. D., & Beechler, B. R. (2017).
Serum biochemistry panels in African buffalo: Defining reference intervals and assessing
variability across season, age and sex.
PloS one, 12(5), e0176830.

The paper https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0176830 and
data in https://datadryad.org/resource/doi:10.5061/dryad.kf6r5

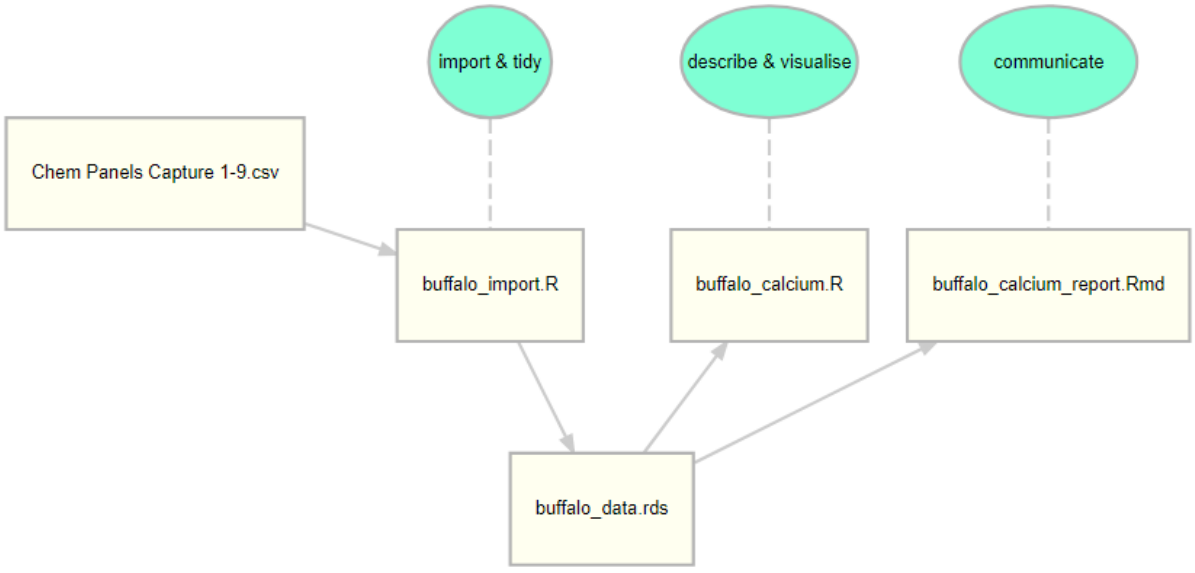
Downloaded data file "Chem Panels Captures 1-9.csv"

Aim: Create 90% reference limits for calcium
```

Other notes would be added as the analysis progressed.

Workflow

The workflow for this data analysis will consist of three stages



Stage 1: Import and tidy
Use the script buffalo_import.R to import the data from the downloaded file "Chem Panels Captures 1-9.csv", rename the variables and save the resulting data frame in buffalo_data.rds.

Stage 2: Describe and visualise
Use the script buffalo_calcium.R to read the tidy data from buffalo_data.rds, calculate summary statistics and plot the data.

Stage 3: Communicate
Use the script buffalo_calcium_report.Rmd to prepare an html report including selected results from stage 2

Import the Buffalo data

Here is an R script file for importing the data. Notice that I have chosen to save the paths to the files as character strings in named data objects. home is the main project folder and the relative addresses of the files are stored in buffalo_csv and buffalo_rds. The full paths are constructed by combining these data objects using the function path() from the fs package.

In the csv file many of the variables have names that contain spaces. R copes with such variable names by placing them in single back-sloping quotes (ticks) as in

`` `Month of Capture` ``

Names with spaces are awkward to handle in R so the script renames those variables and at the same time converted upper case names to lower case.

The variable bun (blood urea nitrogen) contains one entry of "<2", because of this R reads the whole variable as characters rather than numbers. The script converts the characters to numbers using the R function as.numeric(); this function cannot convert "<2" to a number and so it replaces it with a missing value.

```
#============================================================
# File:    buffalo_import.R
# Project: buffalo
# Author:  john thompson
# Date:    7 July 2018
# Description: import from csv & tidy the data
#============================================================
library(tidyverse)
library(fs)

home        <- "Z:/Projects/buffalo"
buffalo_csv <- "data/rawData/Chem Panels Captures 1-9.csv"
buffalo_rds <- "data/rData/buffalo.rds"
#------------------------------------------------
# Read the csv file
#
read_csv( file = path(home, buffalo_csv) ) -> rawDF

#------------------------------------------------
# Rename columns
# bun contains an entry "<2" so is read as characters
# convert bun to numeric
#
rawDF %>%
  rename( id       = `Unique ID`,
          animal   = `Animal ID`,
          period   = Capture,
          first    = `Animals First Capture`,
          capture  = `Animals Capture`,
          month    = `Month of Capture`,
```

```
          alb      = ALB,
          alp      = ALP,
          ast      = AST,
          ca       = CA,
          ggt      = GGT,
          tp       = TP,
          glob     = GLOB,
          bun      = BUN,
          ck       = CK,
          phos     = PHOS,
          mg       = MG,
          ageFirst = `Age at First Capture`,
          sex      = Sex,
          age      = `Actual Age`,
          ageCat   = `Age Category`) %>%
     mutate( bun = as.numeric(bun) ) -> tidyDF

#-----------------------------------------------------
# Save as rds file
#
write_rds( tidyDF, path(home, buffalo_rds))
```

Analyse the Buffalo data

The script 'buffalo_calcium.R' creates a description of the calcium data. The reference intervals are calculated using R's quantile() function.

```
#=============================================================
# File:    buffalo_calcium.R
# Project: buffalo
# Author:  john thompson
# Date:    7 July 2018
# Description: reference intervals for calcium
#=============================================================
library(tidyverse)
library(fs)

home         <- "Z:/Projects/buffalo"
buffalo_rds  <- "data/rData/buffalo.rds"

#-----------------------------------------------------
# Read the csv file
#
read_rds( path(home, buffalo_rds) ) -> buffaloDF

#-----------------------------------------------------
# Histogram of the serum calcium levels
#
buffaloDF %>%
  qplot( x = ca, data = . )

#-----------------------------------------------------
# 90% reference interval (and median)
```

```
#
buffaloDF %>%
  summarise( low  = quantile(ca, prob=0.05),
             mid  = quantile(ca, prob=0.50),
             high = quantile(ca, prob=0.95))

#-------------------------------------------------
# 90% reference interval using only the first
# measurement for each buffalo
#
buffaloDF %>%
  filter( capture == 1 ) %>%
  summarise( low  = quantile(ca, prob=0.05),
             high = quantile(ca, prob=0.95))

#-------------------------------------------------
# 90% reference limits by sex, month and age category
#
buffaloDF %>%
  group_by( sex) %>%
  summarise( low  = quantile(ca, prob=0.05),
             high = quantile(ca, prob=0.95))

buffaloDF %>%
  group_by( month) %>%
  summarise( low  = quantile(ca, prob=0.05),
             high = quantile(ca, prob=0.95))

buffaloDF %>%
  group_by( ageCat) %>%
  summarise( low  = quantile(ca, prob=0.05),
             high = quantile(ca, prob=0.95))

#-------------------------------------------------
# Plot calcium by age
#
buffaloDF %>%
  qplot( x = age, y = ca, data = . )
```

Report the findings

The report was constructed in stages

1. Create the subfolder rmd/calcium and then create a dummy rmarkdown file using the RStudio menus **File - New File - R Markdown**. I opted for an html document, gave it the title "Buffalo reference interval for calcium" and entered my own name as the author. I deleted all of the contents after the YAML header. I saved the empty rmarkdown file as Rmd/calcium/buffalo_calcium_report.Rmd.
2. Create level one headers (#) for Background, Results and Conclusions and I entered a few lines of text in Background that describes the study.
3. Create blank code chunks using **insert - R** and into them copy sections of code from the script buffalo_calcium.R. Decide that I would not show the code in the report so I set the chunk option, echo=FALSE.

4. I always use the first chunk of R code to load packages and read the data. Loading packages often produces a lot of messages and warnings so I set the chunk options, message=FALSE and warning=FALSE in order to stop the messages from being shown in the final report.
5. Before and after each code chunk, insert short pieces of explanatory text.
6. Add a bland conclusion.
7. I clicked the Knit button and the html document was created. The report was saved in the same subfolder as the Rmd code and also displayed on the screen.

Here is my completed rmarkdown file

````
---
title: "Buffalo reference interval for calcium"
author: "John Thompson"
date: "17 January 2019"
output: html_document
---

# Introduction

The data for this analysis come from

Couch, C. E., Movius, M. A., Jolles, A. E., Gorman, M. E., Rigas, J. D., &
Beechler, B. R. (2017).
**Serum biochemistry panels in African buffalo: Defining reference intervals
and assessing variability across season, age and sex.**
PloS one, 12(5), e0176830.

The paper can be found at https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0176830
and the authors have kindly made their data available by placing a csv file in the dryad repository
(https://datadryad.org/) at https://datadryad.org/resource/doi:10.5061/dryad.kf6r5

The researchers captured wild buffalo from the Kruger National Park in South Africa and moved
them to a 900-hectare enclosure where they could live free but be protected from large preditors.
At intervals of about 3 months during 2014 and 2015 buffalo were captured and sedated so that
blood samples could be taken. The samples were analysed for a range of biochemicals with a view
to establishing reference ranges that could be used to determine the health of buffalo captured
in the wild.


# Results

```{r echo=FALSE, warning=FALSE, message=FALSE}
library(tidyverse)
library(fs)

home         <- "Z:/RCourse/buffalo"
buffalo_rds  <- "data/rData/buffalo.rds"

read_rds( path(home, buffalo_rds) ) -> buffaloDF
```

###Calcium reference level across all buffalo
````

```
The calcium levels are reasonably symmetrically distributed

```{r echo=FALSE, warning=FALSE, message=FALSE}
buffaloDF %>%
  qplot( x=ca, data=.)
```

The 90% reference interval is

```{r echo=FALSE}
buffaloDF %>%
  summarise( low  = quantile(ca, prob=0.05),
             high = quantile(ca, prob=0.95))
```

Reference intervals by gender

The reference intervals for male and female buffalo are similar.
```{r echo=FALSE}
buffaloDF %>%
  group_by(sex) %>%
  summarise( low  = quantile(ca, prob=0.05),
             high = quantile(ca, prob=0.95))
```

Conclusions

The study provides a 90% reference interval for use when studying buffalo
captured in the wild, but further investigation is needed to determine whether
or not the practice of resampling of the same buffalo has impacted on the
calculated reference interval.
```

Exercise: Reference interval for Phosphorus

Background

In this exercise you will analyse the serum phosphorus measurements. Phosphorus was measured in units of mg/dL.

Nutrient Requirements of Dairy Cattle Seventh Revised Edition, https://profsite.um.ac.ir/~kalidari/software/NRC/HELP/NRC%202001.pdf says that the level of serum phosphorus in healthy cattle is between 4 and 8 mg/dL, although they do not say what percentage of measurements on healthy cattle would be expected to lie this range. They also say that the levels are higher in young cattle (6-8 mg/dL) compared with adult cattle (4-6 mg/dL).

Aim

The aim of this analysis is to obtain a reference interval for **phosphorus** in African buffalo and to see whether or not it is similar to that for dairy cattle.

Tasks

Complete the following tasks and be sure to make your R scripts legible by including helpful comments, by choosing descriptive names and by paying attention to the layout.

(a) Set up a folder for this project and within it create the standard folder structure. Download the data from Dryad and save it in the rawData folder.

(b) Adapt the script buffalo_import.R from the demonstration to read the downloaded data, rename the variables and save the data frame in rds format in your own folder structure.

(c) Write a script file called buffalo_phosphorus.R that carries out the following eight tasks

(i) Plot a histogram of the serum phosphorus levels for the whole dataset. (for some examples of histograms produced by qplot() see https://www.r-bloggers.com/how-to-make-a-histogram-with-ggplot2/)

(ii) Use the quantile() function to calculate 90% and 80% reference intervals for serum phosphorus in African Buffalo. Are these intervals consistent with the interval for dairy cattle?

(iii) Calculate the mean, m, and standard deviation, s, of serum phosphorus in African Buffalo. If the distribution of serum phosphorus were normal then a 90% reference interval would be (m-1.64s, m+1.64s) and an 80% interval would be (m-1.28s, m+1.28s). Write R code to calculate 90% and 80% reference intervals using this method. Do they agree with intervals obtained using the quantile method? Which method would you expect to be more accurate?

(iv) Plot separate histograms for male and female buffalo and use the quantile() method to calculate 80% reference limits for males and females. Do they differ?

(v) It would be convenient to have the male and female histograms together for easier comparison. In qplot() sets of matched plots are created by the facets argument. Plot a histogram of phosphorus with the added argument, **facets= sex ~ .**. Then try **facets= . ~ sex** and finally **facets=sex ~ ageCat**. The qplot manual (https://ggplot2.tidyverse.org/reference/qplot.html) includes examples of the use of facets.

(vi) Plot a graph of age (x-axis) vs phosphorus (y-axis). Calculate 80% reference intervals separately for each age category. Are the plot and calculations consistent with the finding in dairy cattle that the levels of phosphorus are higher in young animals?

(vii) Some buffalo were sampled on multiple occasions and so appear several times in the dataset. Restrict the analysis to the first capture so that each animal is only sampled once and use the quantile() function to calculate 90% and 80% reference intervals for serum phosphorus in African Buffalo. In theory, is it better to use all of the data or to restrict the analysis to the first capture? Does it matter in practice?

(viii) In R, the function cor() calculates the correlation, for example, **cor(phos, mg)** would calculate the correlation between those two measurments. Write a piped command that calculates the correlation between serum phosphorus and serum magnesium separately for each age category. Plot phosphorus (x-axis) vs magnesium (y-axis) using facets to show the four age categories in different panels. Does the analysis support the idea that phosphorus and magnesium levels are correlated?

(d) Prepare a rmarkdown file called buffalo_phosphorus_report.rmd that contains a section called **introduction**, a section called **results** and a section called **conclusions**. In the introduction write a very brief description of the study. You can leave the other two section blank. knit your report into an html document.

(e) In the results section add three chunks of R code. In the first chunk, plot a histogram of all serum phosporus measurements, in the second, calculate the 80% reference interval for phosphorus in buffalo and in the third, calculate 80% reference intervals for phosphorus separately for each age category.

(f) After each of the R chunks, add a sentence of explanation and then add a one sentence conclusion to your report. Knit your final report into an html document.