SANBI-GBIF training workshop in Data Management and Cleaning:

Intro to R

Vernon Visser

SEEC - Statistics in Ecology, Environment and Conservation
SANBI
Biodiversity for Life
South African National Biodiversity Institute

What is R?

R is a free software environment for statistical computing and graphics. It is probably the most commonly used software used for data analysis in academia and is also widely used in the private sector.

To download R, go to CRAN.

Choose your operating system (Linux / MacOS / Windows). The steps outlined below are for Windows.

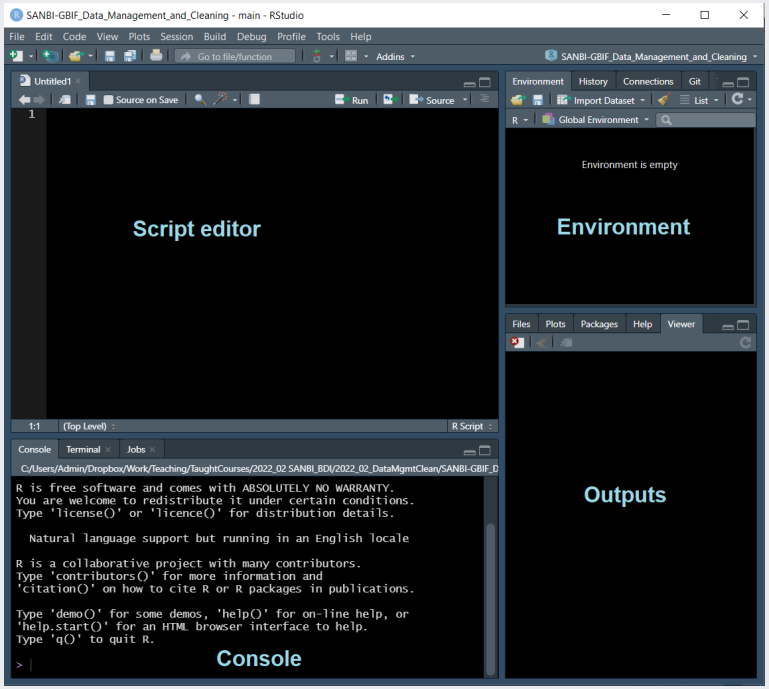
Select "install R for the first time" and then "Download R..."

New versions of R come out regularly and it is a good idea to update regularly. Upgrading can be a pain because you need to reinstall all your packages, but putting it off only makes it worse.

RStudio

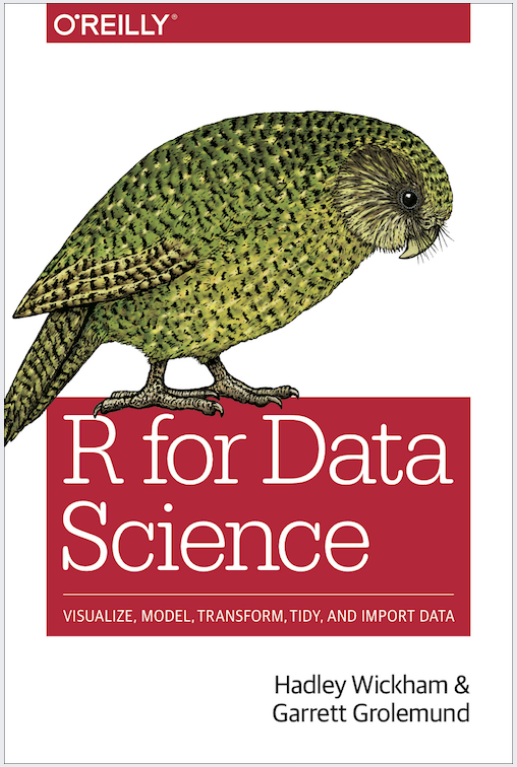
RStudio is an integrated development environment, or IDE, for R programming.

RStudio has four main panels. The **script editor** is where we will be writing out and saving our code. The **console** is the "brain", which shows the calculations and code outputs. The **environment** shows stored items in the "brain". The **"outputs"** panel can be used for a number of things, including installing packages and viewing plots and help documentation.



R4DS

We are going to be using material from the online book, R for Data Science, to introduce you to R. This is a really exceptional book for learning about R and I strongly encourage you to go through it in more detail in your own time. However, we will just touch on a few key concepts to get you started with using R.



Some basic concepts

Before we learn some basic functioning of R, such as using it as a calculator, it is useful to know how to do some simple tasks, such as running code and identifying errors.

By placing the cursor on a line OR highlighting a section of text, you can run code by either using `Cmd/Ctrl + Enter` on the keyboard or pressing the Run button:

You can pick up errors either from some diagnostics RStudio provides or interrogating the error messages provided by R in the console after running a line of code. For example:

```
x y = 10
```

```
## Error: <text>:2:3: unexpected symbol
## 1:
## 2: x y
##      ^
```

Comments are prefaced with the # symbol and are not read by R. Make copious use of commenting throughout your script. This is important for yourself to remember what you have done as well as for other people to understand what you did.

R as a calculator

```
# Basic calculations as on a calculator
1 + 2
```

```
## [1] 3
```

```
1 / 200 * 30
```

```
## [1] 0.15
```

```
# Create objects
x = 2 + 3
x <- 2 + 3 #This is equivalent to the line above. I prefer the = sign - it's one less key to press, but most R users use the <- because apparently it's "confusing"to use =, although I don't think so.
x
```

```
## [1] 5
```

Variable naming styles

i_use_snake_case

otherPeopleUseCamelCase

some.people.use.periods

And_aFew.People_RENOUNCEconvention

R functions and getting help

```
# R has a large collection of built-in functions that are called like this:
# function_name(arg1 = val1, arg2 = val2, ...)
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# This is telling R to produce a sequence of integers from 1 to 10. It's not apparent from the code above what the arguments for the function are. To find this out, use the ? to find help on a function
?seq

# So the code above is actually using the arguments below:
seq(from=1, to=10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Data and data formats

```
# Load some packages that we will need
library(nycflights13) #Package with data
library(tidyverse) #R tidyverse group of packages with lots of useful data wrangling and plotting functions

# Let's look at the flights dataset from the nycflights13 package
flights
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_de…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      544        545      -1    1004    1022     -18 B6
##  5  2013     1     1      554        600      -6     812     837     -25 DL
##  6  2013     1     1      554        558      -4     740     728      12 UA
##  7  2013     1     1      555        600      -5     913     854      19 B6
##  8  2013     1     1      557        600      -3     709     723     -14 EV
##  9  2013     1     1      557        600      -3     838     846      -8 B6
## 10  2013     1     1      558        600      -2     753     745       8 AA
## # … with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   ¹sched_dep_time, ²dep_delay, ³arr_time, ⁴sched_arr_time, ⁵arr_delay
```

R stores data in different formats, which the "tibble" above highlights, e.g., <int> is an integer, <chr> is a character object.

dplyr and data wrangling

`dplyr` is the key data wrangling package within the `tidyverse` group of packages.

There are five key `dplyr` functions that allow you to solve the vast majority of your data manipulation challenges:

- Pick observations by their values: `filter()`.
- Reorder the rows: `arrange()`.
- Pick variables by their names: `select()`.
- Create new variables with functions of existing variables: `mutate()`.
- Collapse many values down to a single summary: `summarise()`.

Filter rows with filter()

The first argument in `filter()` is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame.

```
# Filter to get all flights on 1 Jan
filter(flights, month == 1, day == 1)
```

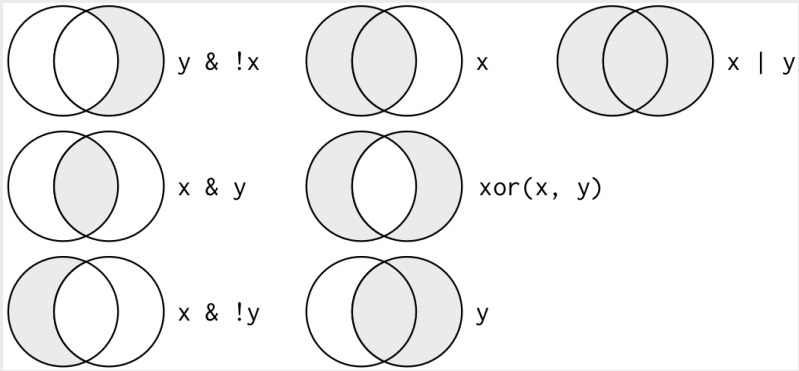
```
## # A tibble: 842 × 19
##     year month   day dep_time sched_de…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      544        545      -1    1004    1022     -18 B6
##  5  2013     1     1      554        600      -6     812     837     -25 DL
##  6  2013     1     1      554        558      -4     740     728      12 UA
##  7  2013     1     1      555        600      -5     913     854      19 B6
##  8  2013     1     1      557        600      -3     709     723     -14 EV
##  9  2013     1     1      557        600      -3     838     846      -8 B6
## 10  2013     1     1      558        600      -2     753     745       8 AA
## # … with 832 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   ¹sched_dep_time, ²dep_delay, ³arr_time, ⁴sched_arr_time, ⁵arr_delay
```

Logical operators

In the previous example of `filter()` we used == to select rows. Note the double equals sign, which denotes the "logical operator" equivalent of =. R has a number of other logical operators:

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
!=	not equal to
!x	not x
x|y	x OR y
x&y	x AND y

Boolean operations as detailed in R4DS



Arrange rows with arrange()

The first argument in `arrange()` is the name of the data frame. The second and subsequent arguments are the expressions that order/arrange the data frame.

```
# Filter to get all flights on 1 Jan
arrange(flights, year, month, day) %>%
  slice(1:3) #Selects only first 3 rows
```

```
## # A tibble: 3 × 19
##    year month   day dep_time sched_dep…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier
##   <int> <int> <int>    <int>       <int>   <dbl>   <int>   <int>   <dbl> <chr>
## 1  2013     1     1      517         515       2     830     819      11 UA
## 2  2013     1     1      533         529       4     850     830      20 UA
## 3  2013     1     1      542         540       2     923     850      33 AA
## # … with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>, and abbreviated variable names ¹sched_dep_time,
## #   ²dep_delay, ³arr_time, ⁴sched_arr_time, ⁵arr_delay
```

You can use `desc()` to arrange in descending order.

```
arrange(flights, desc(dep_delay)) %>%
  slice(1:3) #Selects only first 3 rows
```

```
## # A tibble: 3 × 19
##    year month   day dep_time sched_dep…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier
##   <int> <int> <int>    <int>       <int>   <dbl>   <int>   <int>   <dbl> <chr>
## 1  2013     1     9      641         900    1301    1242    1530    1272 HA
## 2  2013     6    15     1432        1935    1137    1607    2120    1127 MQ
## 3  2013     1    10     1121        1635    1126    1239    1810    1109 MQ
## # … with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minu…
## #   time_hour <dttm>, and abbreviated variable names ¹sched_dep…
## #   ²dep_delay, ³arr_time, ⁴sched_arr_time, ⁵arr_delay
```

Select columns with select()

`select()` allows you to narrow a dataset down to a few select columns.

```
names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

```
#Select columns by name
select(flights, year, month, day) %>%
  slice(1:3) #Selects only first 3 rows
```

```
## # A tibble: 3 × 3
##    year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
```

```
#Select all columns except those from dep_time to time_hour
select(flights, -(dep_time:time_hour)) %>%
  slice(1:3) #Selects only first 3 rows
```

```
## # A tibble: 3 × 3
##    year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
```

A quick aside...

Later on when we use the raster package you may encounter conflicts between raster and dplyr with the use of select. To prevent this use $dplyr :: select()$ in R to tell R to use the dplyr package $select()$

Add new variables with mutate()

```
#Create a smaller select dataset
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
#Create new variables: gain and speed
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

```
## # A tibble: 336,776 × 9
##     year month   day dep_delay arr_delay distance air_time  gain speed
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227    -9  370.
##  2  2013     1     1         4        20     1416      227   -16  374.
##  3  2013     1     1         2        33     1089      160   -31  408.
##  4  2013     1     1        -1       -18     1576      183    17  517.
##  5  2013     1     1        -6       -25      762      116    19  394.
##  6  2013     1     1        -4        12      719      150   -16  288.
##  7  2013     1     1        -5        19     1065      158   -24  404.
##  8  2013     1     1        -3       -14      229       53    11  259.
##  9  2013     1     1        -3        -8      944      140     5  405.
## 10  2013     1     1        -2         8      733      138   -10  319.
## # … with 336,766 more rows
```

Grouped summaries with summarise()

`summarise()` provides a one-row summary based on a specified function.

```
#Get the mean departure delay
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##   delay
##   <dbl>
## 1  12.6
```

```
#Use the pipe (%>%) to do multiple things in one calculation
flights %>%
  group_by(dest) %>% #This groups the data by "dest" - "summarise()" then provides a value for each "dest" level
  summarise(
    count = n(), #Get a count for each "dest" level
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL") #Filter to rows that have counts>20 and the destination is not "HNL"
```

```
## # A tibble: 96 × 4
##    dest  count  dist delay
##    <chr> <int> <dbl> <dbl>
##  1 ABQ     254 1826   4.38
##  2 ACK     265  199   4.85
##  3 ALB     439  143  14.4
##  4 ATL   17215  757. 11.3
##  5 AUS    2439 1514.  6.02
##  6 AVL     275  584.  8.00
##  7 BDL     443  116   7.05
##  8 BGR     375  378   8.03
##  9 BHM     297  866. 16.9
## 10 BNA    6333  758. 11.8
## # … with 86 more rows
```

tibble vs data.frame

The `tidyverse` ecosystem uses tibbles instead of dataframes. `data.frame` is the native R format of storing datasets, but lacks certain functionality and information that a tibble provides. The best way to understand this is through an example:

```
head(iris) #iris is the most commonly used R example dataset and is stored as a data.frame. head is a function that gives a snapshot view of a dataset (first six lines)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# Let's compare the above view of "iris" as a data.frame with it as a tibble
as_tibble(iris)
```

```
## # A tibble: 150 × 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # … with 140 more rows
```

RStudio projects

What we are about to talk about next is **VERY important**. The reason I say this is because by learning how to use RStudio projects from the beginning will make your experience using R going forward so much more pleasant.

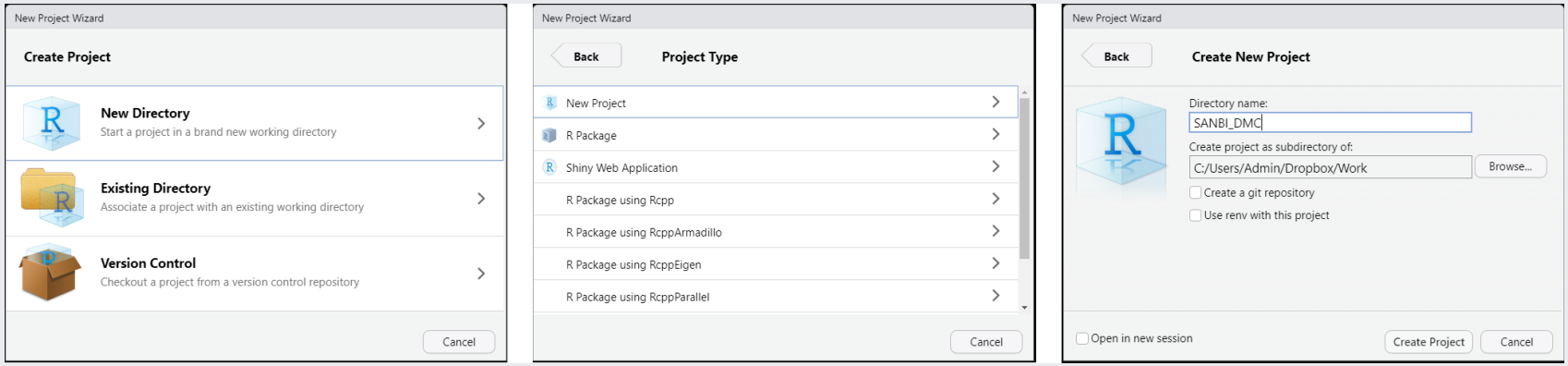
RStudio when opened normally will be set to a default working directory. You can see what this is by typing:

```
getwd()
```

Your working directory will vary depending on your computer's administrator name and whether you have edited the default working directory in RStudio's global options. We can set the working directory using the function `setwd()`, but this is not recommended! You should rather create a RStudio project for each bit of work you are doing. RStudio will open to the predetermined working directory when you open your project, removing any fuss of setting working directories. It also allows other people to replicate your setup on their own computers, thereby facilitating collaboration. Within a project you can create multiple scripts.

Let's try it out...

RStudio projects



The first step is to click File -> New Project

You can choose to either create a new directory or put the project in an existing directory. We will create a new one.

Select "New Project".

Specify a directory name, i.e. the folder in which your project will be stored. We will create one called "SANBI_DMC". Choose the subdirectory where you want to create your new directory.

That's it! As simple as that!

Project management

Before we move on, just a quick aside about best practice for managing your code, data, outputs, etc. for each project you do.

Within the SANBI_BDI folder that you created for this RStudio Project, it's a good idea to create a number of sub-folders that will keep things tidy, but more importantly, keep your raw data "untouched" and ensure that you send all manipulated data and outputs to separate directories. I'd suggest creating the following folders:

- Code
- Data
- Outputs

Within these sub-directories you can obviously create more folders to separate stuff and keep things neat, e.g., folders for figures, cleaned data...

You can now copy the data you will need for the course into a "Data" sub-directory in the "SANBI_DMC" directory.

Reading data into R

The `readr` package is the `tidyverse` way of reading data into R. However, R has some base functions that do the same thing, which we will also look at.

`read_csv()` reads comma delimited files, `read_csv2()` reads semicolon separated files (common in countries where , is used as the decimal place), `read_tsv()` reads tab delimited files, and `read_delim()` reads in files with any delimiter.

Base R uses `read.csv` or `read.table` to do the same sort of thing as the functions above. Let's take a look:

```
#Let's try reading in an example dataset using the readr package
datReadr = read_csv("../Data/Example_data_Pentameris_barbata.csv")
datReadr %>% slice(1:3)
```

```
## # A tibble: 3 × 4
##       gbifID species            decimalLongitude decimalLatitude
##        <dbl> <chr>                         <dbl>           <dbl>
## 1 1096614794 Pentameris barbata               NA              NA
## 2 1095980470 Pentameris barbata               NA              NA
## 3 3346432018 Pentameris barbata               NA              NA
```

```
#Now let's try using the base R function, read.csv()
datBase = read.csv("../Data/Example_data_Pentameris_barbata.csv")
datBase[1:3,]
```

```
##       gbifID            species decimalLongitude decimalLatitude
## 1 1096614794 Pentameris barbata               NA              NA
## 2 1095980470 Pentameris barbata               NA              NA
## 3 3346432018 Pentameris barbata               NA              NA
```

Do you notice any differences in the outputs?

You're set to go!

We have now covered enough basic R functionality for you to progress with the course. It may take you a while to get your head around the way R "thinks", but just persevere and you will get it. The more you use R in your day to day work, the easier it will become.