Introduction to R

Kim Dill-McFarland, Kris Hong, and Yue Liu
in collaboration with the Applied Statistics and Data science group
U. of British Columbia

version March 12, 2019

Contents

Overview **1**

A Tour of RStudio **1**
RStudio Projects . . . 3
R Scripts . . . 4
R packages . . . 4

Getting started **5**
Loading data into an R data frame . . . 5
Help function . . . 7
Data types . . . 7
Working with vectors and data frames . . . 7
Data visualization with quickplot . . . 8

Exercises **9**

Overview

In this workshop, we introduce you to R and RStudio at the beginner level. This condensed 2-hour workshop is meant to get you started in R and acts as a pre-requisite for our more advanced workshops. In it, we cover:

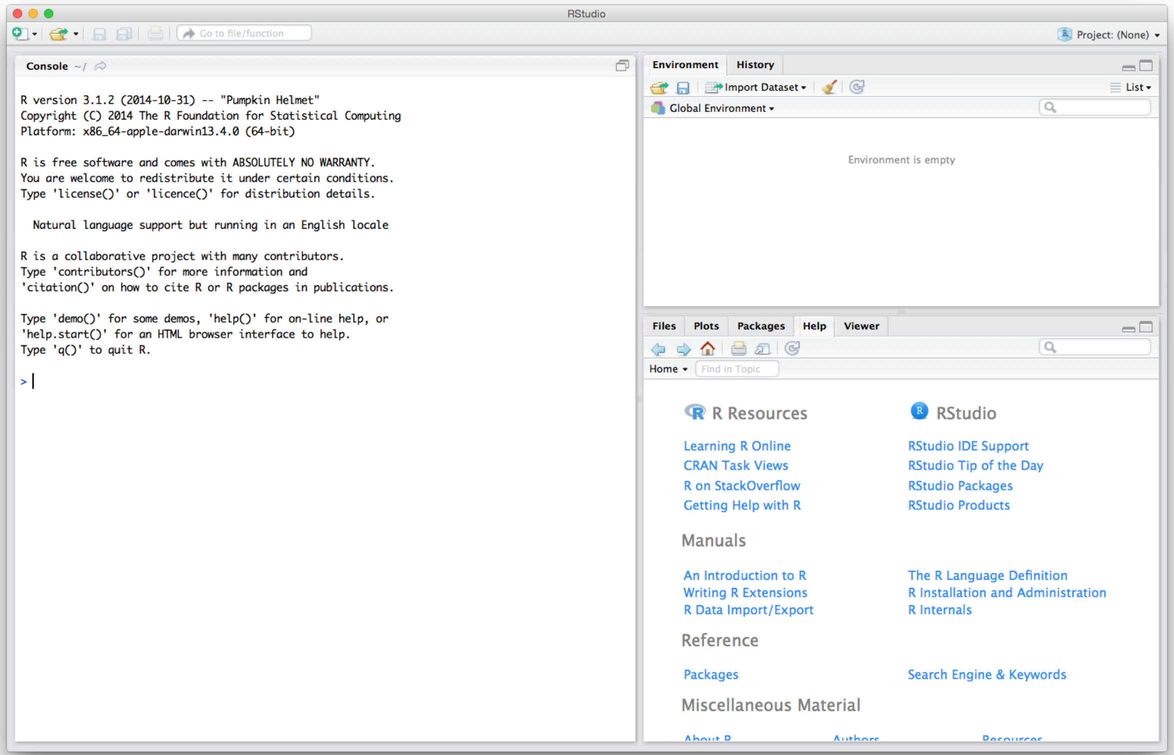
- R and RStudio
- RStudio projects
- R scripts
- Installing packages
- Reading in data as a data frame
- Vectors, single values, and data types
- Basic data visualization
- The help function

We will do all of our work in RStudio. RStudio is an integrated development and analysis environment for R that brings a number of conveniences over using R in a terminal or other editing environments. You should have both R and RStudio installed, as described in the setup instructions.

A Tour of RStudio

When you start RStudio, you will see something like the following window appear:

Notice that the window is divided into three "panes":



```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Figure 1:

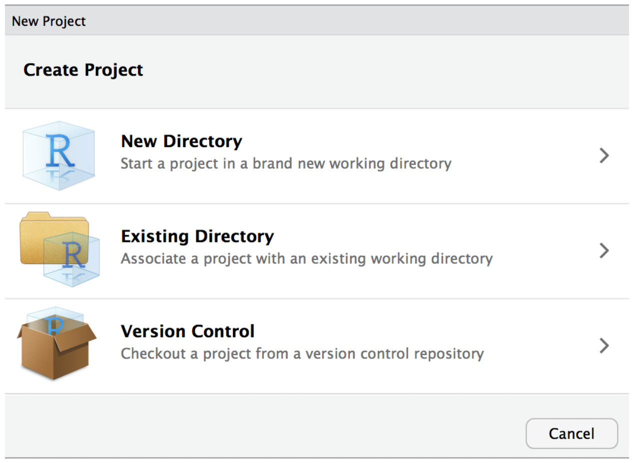

New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

Figure 2:

- Console (the entire left side): this is your view into the R engine. You can type in R commands here and see the output printed by R. (To make it easier to tell them apart, your input is printed in blue, while the output is black.) There are several editing conveniences available: use up and down arrow keys to go back to previously entered commands, which can then be edited and re-run; TAB for completing the name before the cursor; see more in online docs.
- Environment/History (tabbed in upper right): view current user-defined objects and previously-entered commands, respectively.
- Files/Plots/Packages/Help (tabbed in lower right): as their names suggest, these are used to view the contents of the current directory, graphics created by the user, install packages, and view the built-in help pages.

To change the look of RStudio, you can go to Tools -> Global Options -> Appearance and select colors, font size, etc. If you plan to be working for longer periods, we suggest choosing a dark background color scheme to save your computer battery and your eyes.

RStudio Projects

Projects are a great feature of RStudio. When you create a project, RStudio creates an `.Rproj` file that links all of your files and outputs to the project directory. When you import data, R automatically looks for the file in the project directory instead of you having to specify a full file path on your computer like `/Users/username/Desktop/`. R also automatically saves any output to the project directory. Finally, projects allow you to save your R environment in `.RData` so that when you close RStudio and then re-open it, you can start right where you left off without re-importing any data or re-calculating any intermediate steps.

RStudio has a simple interface to create and switch between projects, accessed from the button in the top-right corner of the RStudio window. (Labelled "Project: (None)", initially.)

Create a Project

Let's create a project to work in for this workshop. Start by click the "Project" button in the upper right or going to the "File" menu. Select "New Project" and the following will appear.

You can either create a project in an existing directory or make a new directory on your computer - just be sure you know where it is.

After your project is created, navigate to its directory using your Finder/File explorer. You will see the `.RProj` file has been created.

Place data in the project directory

Next, move or save the `data.csv` file from the workshop materials into the project directory so that we can use it in this workshop.

To access this project in the future, simply double-click the `RProj` and RStudio will open the project or choose File > Open Project from within an already open RStudio window.

R Scripts

R script files are the primary way in which R facilitates reproducible research. They contain the code that loads your raw data, cleans it, performs the analyses, and creates and saves visualizations. R scripts maintain a record of everything that is done to the raw data to reach the final result. That way, it is very easy to write up and communicate your methods because you have a document listing the precise steps you used to conduct your analyses. This is one of R's primary advantages compared to traditional tools like Excel, where it may be unclear how to reproduce the results.

Generally, if you are testing an operation (*e.g.* what would my data look like if I applied a log-transformation to it?), you should do it in the console (left pane of RStudio). If you are committing a step to your analysis (*e.g.* I want to apply a log-transformation to my data and then conduct the rest of my analyses on the log-transformed data), you should add it to your R script so that it is saved for future use.

Additionally, you should annotate your R scripts with comments. In each line of code, any text preceeded by the `#` symbol will not execute. Comments can be useful to remind yourself and to tell other readers what a specific chunk of code does.

Let's create an R script (File > New File > R Script) and save it as `live_notes.R` in your main project directory. If you again look to the project directory on your computer, you will see `live_notes.R` is now saved there.

We will work together to create and populate the `live_notes.R` script throughout this workshop.

R packages

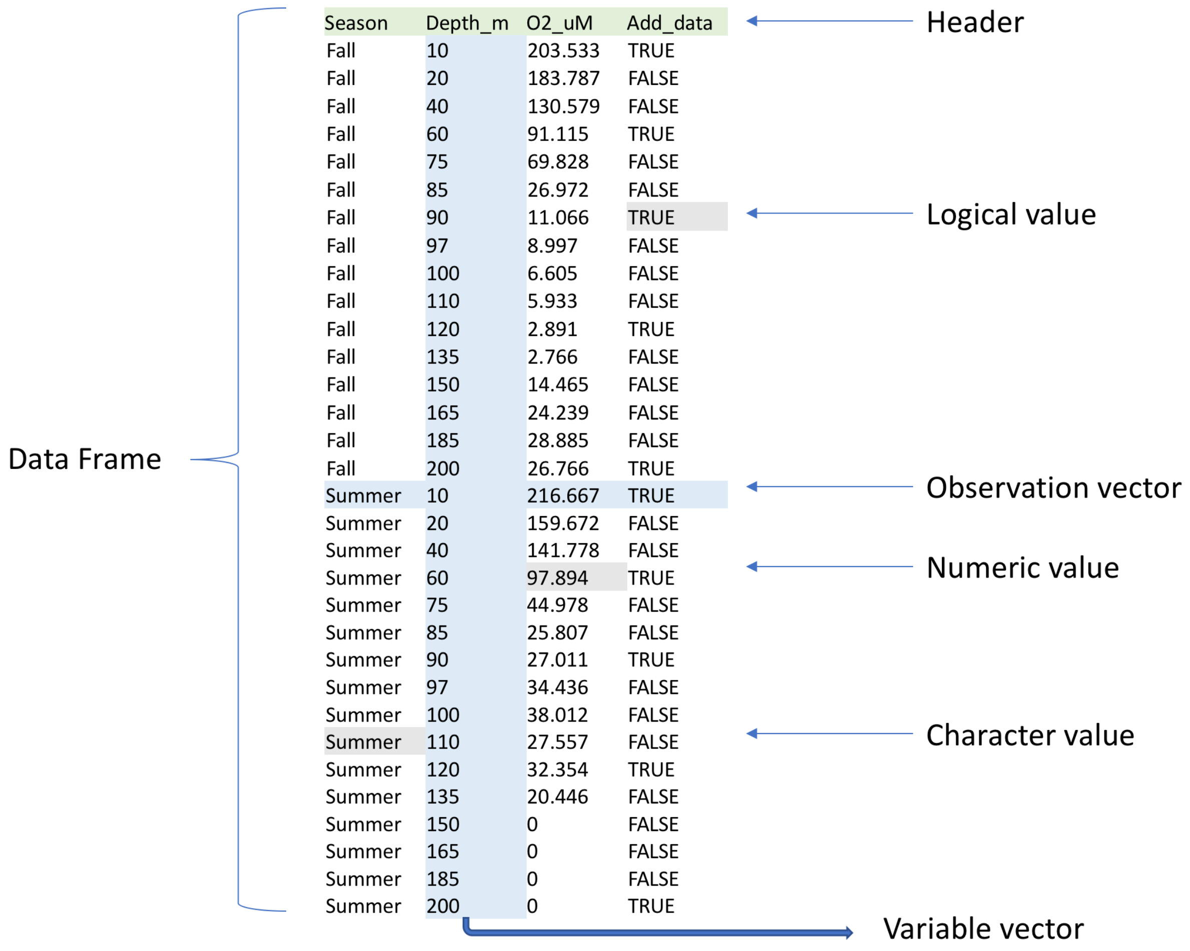
R packages are units of shareable code, containing functions that facilitate and enhance analyses. Let's install `ggplot2`, a very popular data visualization package in R that we will use later in the workshop. Packages are typically installed from CRAN (The Comprehensive R Archive Network), which is a database containing R itself as well as many R packages. Any package can be installed from CRAN using the `install.packages` function. You can input this into your console (as opposed to `live_notes.R`) since once a package is installed on your computer, you won't need to reinstall it again.

```r
install.packages("ggplot2")
```

After installing a package, and *everytime* you open a new RStudio session, the packages you want to use need to be loaded into the R workspace with the `library` function. This tells R to access the package's functions and prevents RStudio from lags that would occur if it automatically loaded every downloaded package everytime you opened it.

Since this is something you would need to repeat in future, it should go in your `live_notes.R` script.

```r
library(ggplot2)
```



Season	Depth_m	O2_uM	Add_data
Fall	10	203.533	TRUE
Fall	20	183.787	FALSE
Fall	40	130.579	FALSE
Fall	60	91.115	TRUE
Fall	75	69.828	FALSE
Fall	85	26.972	FALSE
Fall	90	11.066	TRUE
Fall	97	8.997	FALSE
Fall	100	6.605	FALSE
Fall	110	5.933	FALSE
Fall	120	2.891	TRUE
Fall	135	2.766	FALSE
Fall	150	14.465	FALSE
Fall	165	24.239	FALSE
Fall	185	28.885	FALSE
Fall	200	26.766	TRUE
Summer	10	216.667	TRUE
Summer	20	159.672	FALSE
Summer	40	141.778	FALSE
Summer	60	97.894	TRUE
Summer	75	44.978	FALSE
Summer	85	25.807	FALSE
Summer	90	27.011	TRUE
Summer	97	34.436	FALSE
Summer	100	38.012	FALSE
Summer	110	27.557	FALSE
Summer	120	32.354	TRUE
Summer	135	20.446	FALSE
Summer	150	0	FALSE
Summer	165	0	FALSE
Summer	185	0	FALSE
Summer	200	0	TRUE

Figure 3: Figure 1. Introduction to R data frame

Getting started

Loading data into an R data frame

One of R's most essential data structures is the data frame, which is simply a table of `m` columns by `n` rows of data. We will read the data shown in Figure 1 into R using the `read.table` function.

Each R function follows the following basic syntax, where `Function` is the name of the function.

```
Function(argument1=..., argument2=..., ...)
```

The read.table has many arguments; however, we only need to specify four arguments to correctly read in our data as a data frame. For our data, we will need to specify:

- `file` - gives the path to the file that we want to load from our working directory (current project directory).
- `sep` - tells R that our data are comma-separated
- `header` - tells R that the first row in our data contains the names of the variables (columns).
- `stringsAsFactors` - tells R not to read in character columns as a factor variable. You can learn more about factor variables in our Tidyverse workshop.

We will store the data as an *object* named `dat` using the assignment operator `<-`, so that we can re-use it in our analysis.

```
# read the data and save it as an object
dat <- read.table(file="data.csv", sep=",", header=TRUE, stringsAsFactors=FALSE)
```

Now we can refer to the data frame as `dat` like so.

```
dat
```

```
##    Season Depth_m   O2_uM Add_data
## 1    Fall      10 203.533     TRUE
## 2    Fall      20 183.787    FALSE
## 3    Fall      40 130.579    FALSE
## 4    Fall      60  91.115     TRUE
## 5    Fall      75  69.828    FALSE
## 6    Fall      85  26.972    FALSE
## 7    Fall      90  11.066     TRUE
## 8    Fall      97   8.997    FALSE
## 9    Fall     100   6.605    FALSE
## 10   Fall     110   5.933    FALSE
## 11   Fall     120   2.891     TRUE
## 12   Fall     135   2.766    FALSE
## 13   Fall     150  14.465    FALSE
## 14   Fall     165  24.239    FALSE
## 15   Fall     185  28.885    FALSE
## 16   Fall     200  26.766     TRUE
## 17 Summer      10 216.667     TRUE
## 18 Summer      20 159.672    FALSE
## 19 Summer      40 141.778    FALSE
## 20 Summer      60  97.894     TRUE
## 21 Summer      75  44.978    FALSE
## 22 Summer      85  25.807    FALSE
## 23 Summer      90  27.011     TRUE
## 24 Summer      97  34.436    FALSE
## 25 Summer     100  38.012    FALSE
## 26 Summer     110  27.557    FALSE
## 27 Summer     120  32.354     TRUE
## 28 Summer     135  20.446    FALSE
## 29 Summer     150   0.000    FALSE
## 30 Summer     165   0.000    FALSE
## 31 Summer     185   0.000    FALSE
## 32 Summer     200   0.000     TRUE
```

These data contain information on oxygen concentrations sampled at Saanich Inlet, British Columbia at several depths and during various seasons. It contains:

- Season - season in which measurement was obtained; Fall or Summer
- Depth_m - depth in meters (m) at which measurement as obtained
- O2_uM - oxygen (O_2) concentration in micromolar (μM)
- Add_data - whether additional microbiological data was collected; TRUE or FALSE

For a brief introduction to the data used in our workshops, see Hallam SJ *et al.* 2017. Sci Data 4: 170158 "Monitoring microbial responses to ocean deoxygenation in a model oxygen minimum zone". More detailed information on the environmental context and time series data can be found in Torres-Beltrán M *et al.* 2017. Sci Data 4: 170159. "A compendium of geochemical information from the Saanich Inlet water column".

Help function

You can read up about the different arguments of a specific function by typing `?Function` or `help(Function)` in your R console.

```
?read.table
```

Data types

This data frame consists of 32 rows (observations) and 4 columns (variables), where each row and column is itself an R vector. R vectors are one-dimensional arrays of data. For example, we can extract column vectors from data frames using the `$` operator.

```
# Extract the oxygen concentrations
dat$O2_uM
```

```
##  [1] 203.533 183.787 130.579  91.115  69.828  26.972  11.066   8.997
##  [9]   6.605   5.933   2.891   2.766  14.465  24.239  28.885  26.766
## [17] 216.667 159.672 141.778  97.894  44.978  25.807  27.011  34.436
## [25]  38.012  27.557  32.354  20.446   0.000   0.000   0.000   0.000
```

R objects have several different classes (types). Our data frame contains the three most basic of R data types: numeric, character, and logical. The built-in `class` function will tell you what data type an object is.

```
class(dat$Season)
```

```
## [1] "character"
```

```
class(dat$O2_uM)
```

```
## [1] "numeric"
```

```
class(dat$Add_data)
```

```
## [1] "logical"
```

Working with vectors and data frames

Functions executed on an object in R may respond exclusively to one or more data types or may respond differently depending on the data type. A large proportion of R functions operate on vectors to perform quick computations over their values. Here are some examples:

```
# Compute the variance of the Oxygen values
var(dat$O2_uM)
```

```
## [1] 4140.459
```

```
# Find whether an Oxygen concentration is greater than 50 uM
dat$O2_uM > 50
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# Convert the depth variable from metres to kilometres
dat$Depth_m / 1000
```

```
##  [1] 0.010 0.020 0.040 0.060 0.075 0.085 0.090 0.097 0.100 0.110 0.120
## [12] 0.135 0.150 0.165 0.185 0.200 0.010 0.020 0.040 0.060 0.075 0.085
## [23] 0.090 0.097 0.100 0.110 0.120 0.135 0.150 0.165 0.185 0.200
```

```r
# Find the unique character values of the Season variable
unique(dat$Season)
```

```
## [1] "Fall"   "Summer"
```

Since vectors are 1D arrays of a defined length, their individual values can be retrieved using vector indices. R uses 1-based indexing, meaning the first value in an R vector corresponds to the index 1. Each subsequent element increases the index by 1. For example, we can extract the value of the 5th element of the Oxygen vector using the square bracket operator `[]` like so.

```r
dat$O2_uM[5]
```

```
## [1] 69.828
```

In contrast, data frames are 2D arrays so indexing is done across both dimensions as `[rows, columns]`. So, we can extract the same oxygen value directly from the data frame knowing it is in the 5th row and 3rd column.

```r
dat[5, 3]
```

```
## [1] 69.828
```

The square bracket operator is most often used with logical vectors (TRUE/FALSE) to subset data. For example, we can subset our data frame to all observations (rows) with an Oxygen concentration of 0.

```r
# Create logical vector for which oxygen values are 0
logical.vector <- dat$O2_uM == 0
#View vector
logical.vector
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

```r
#Apply vector to data frame to select only observations where the logical vector is TRUE (i.e. the oxyg
dat[logical.vector, ]
```

```
##    Season Depth_m O2_uM Add_data
## 29 Summer     150     0    FALSE
## 30 Summer     165     0    FALSE
## 31 Summer     185     0    FALSE
## 32 Summer     200     0     TRUE
```

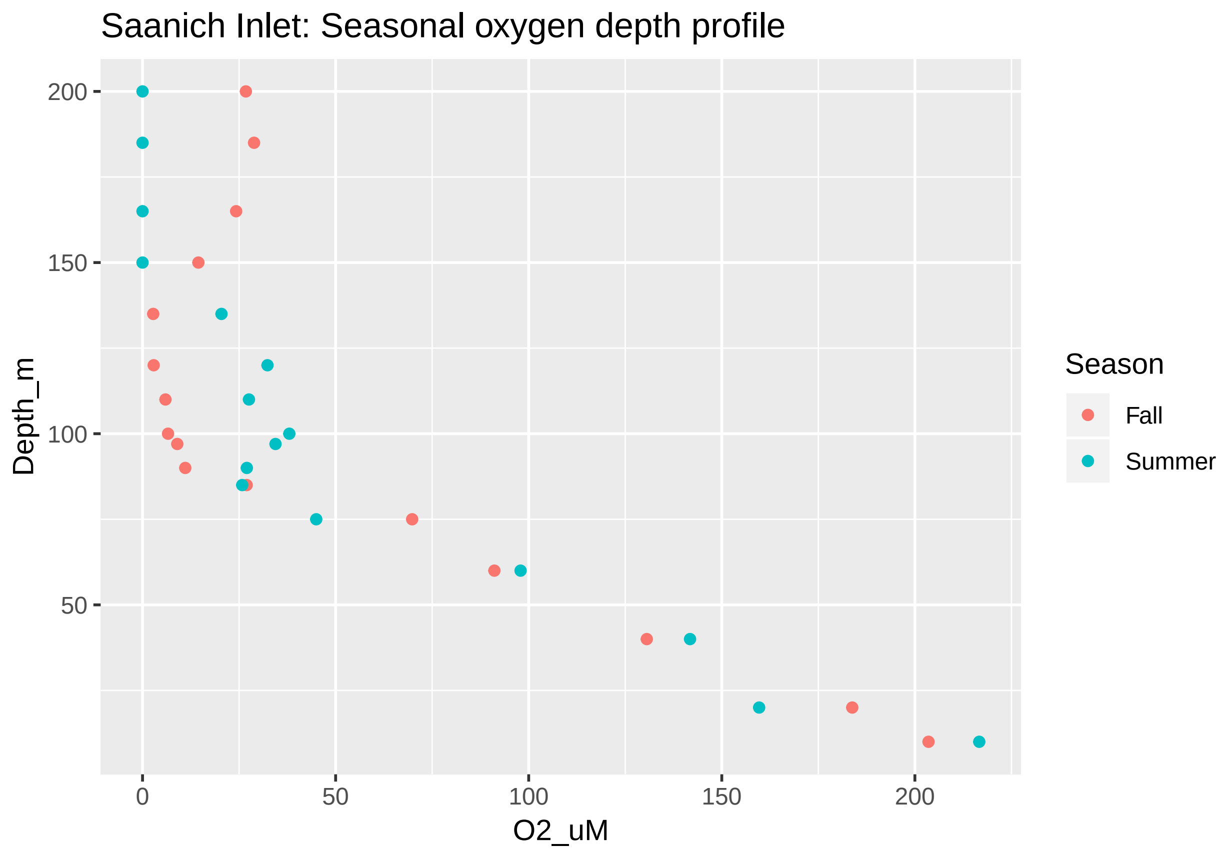
Subsetting is extremely useful when working with large data. You can learn more complex subsets in our Tidyverse workshop using the `tidyverse` packages, which are a popular and widely used suite of R package for working with and visualizing data.

Data visualization with quickplot

There are many data visualization options in R. The most popular plotting package is `ggplot2`, which is capable of creating publication quality figures.

Let's create a simple depth profile (scatterplot) of the Oxygen concentrations with colors indicating the Season that the sample was collected. We will use the `quickplot` function that we loaded earlier when we installed `ggplot2`.

```r
quickplot(data=dat,
     x=O2_uM,
     y=Depth_m,
     colour=Season,
     main="Saanich Inlet: Seasonal oxygen depth profile")
```



Exercises

To practice skills covered in this workshop and prepare for your future R endeavors, please complete the following exercises. Happy coding!

1. Install the packages used in the next workshop you plan to attend.
    - The R tidyverse: `tidyverse`, `lubridate`, `cowplot`
    - Intermediate R programming: `tidyverse`, `lmerTest`, `devtools`, `roxygen2`
    - Statistical models: `tidyverse`, `nycflights13`, `plyr`,`readxl`, `broom`, `gapminder`, `lme4`, `lmerTest`, `lsmeans`, `car`, `MASS`, `HSAUR3`
    - Reproducible Research: `tidyverse`, `packrat`
    - None (but to practice package install anyway): `tidyverse`

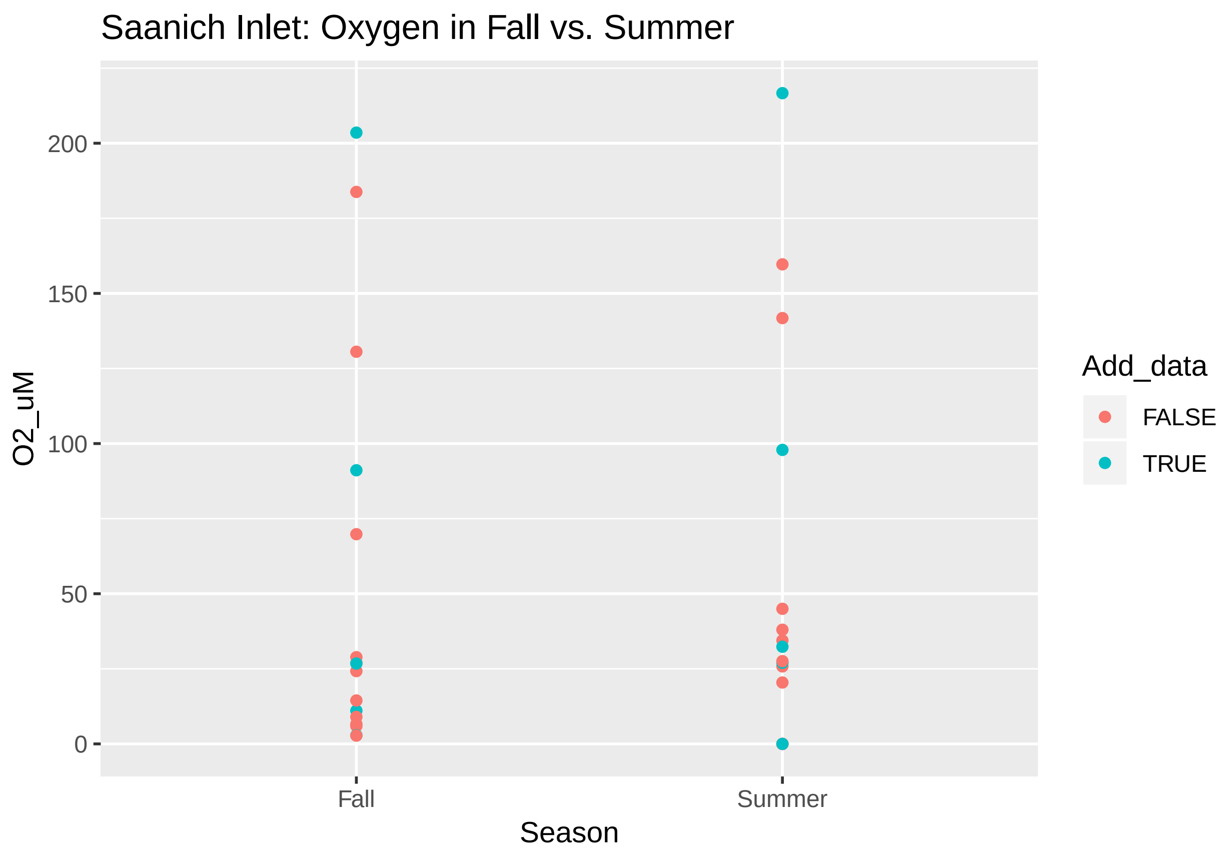
*Please note that if you have **R v3.3 or older**, you may not be able to install `tidyverse`. In this case, you need to separately install each package within the tidyverse. This includes:* `readr`, `tibble`, `dplyr`, `tidyr`, `stringr`, `ggplot2`, `purr`, `forcats`

2. Using help to identify the necessary arguments for the log function compute the natural logarithm of 4, base 2 logarithm of 4, and base 4 logarithm of 4.
3. Using an R function, determine what data type the Depth_m variable is.
4. Using indexing and the square bracket operator `[]`:

- determine what depth value occurs in the 20th row
- return the cell where oxygen equals 91.115

5. Subset the data to observations where depth equals 100 m. *Hint*: Use a logical vector.
6. Complete the following code to create a stacked scatterplot of oxygen concentrations within the two different seasons, colored by whether or not microbial data are available (example below).

```r
quickplot(data= ,
     x= ,
     y= ,
     colour= ,
     main="Saanich Inlet: Oxygen in Fall vs. Summer")
```


Saanich Inlet: Oxygen in Fall vs. Summer
O2_uM
Fall
Summer
Season
Add_data
FALSE
TRUE