Intro to R part 1

Jeremy Jenrette

1/19/2023

This is a brief intro to R and RStudio. Here are some good resources – VT library: The R Book, Michael J. Crawley, 2007  
– [bookdown](https://bookdown.org/josiesmith/labbook/basics-of-r.html#objects) R basics

Use an .Rmd file within RStudio. This way, you can experiment with code in the “code blocks” provided. When you are done, use the “Knit” option to view the final Word document.

## R as a calculator

1+1

## [1] 2

7\*4

## [1] 28

1753/18

## [1] 97.38889

2+5\*3

## [1] 17

(2+5)\*3

## [1] 21

Simple math functions

log(100)

## [1] 4.60517

sqrt(100)

## [1] 10

### Objects

You can assign values to objects easily and perform calculations.

x <- 6  
2\*x + 3

## [1] 15

y <- 2  
y^2

## [1] 4

Create a new object HERE

object1 <- 5

### Functions

Functions simplify complex calculations

object2 = c(5, 10, 15, 20, 25, 30) # 'c()' stands for combine   
mean(object2)

## [1] 17.5

sum(object2)

## [1] 105

mean(cars$dist)

## [1] 42.98

sum(cars$dist)

## [1] 2149

Note: Sometimes na.rm = TRUE is necessary within the paranetheses to instruct R to ignore missing data.

mean(cars$dist, na.rm = TRUE)

## [1] 42.98

### Vectors

You can create vectors using the combine function c() A vector is a sequence of data that are of the same type (numbers, characters)

x <- c(1,2,3,4,5,6,7) # integers or numerals  
x

## [1] 1 2 3 4 5 6 7

class(x)

## [1] "numeric"

y <- c("a", "b", "c", "d") # characters  
y

## [1] "a" "b" "c" "d"

class(y)

## [1] "character"

### Indexing vectors

Within a vector, you can extract individual values based on their index, for example:

x[1] # first value  
x[3] # third value  
  
y[2]  
y[4]

## [1] 1  
## [1] 3  
## [1] "b"  
## [1] "d"

To get multiple values, you can either extract a range of values using the colon, or by combining indices with c()

x

## [1] 1 2 3 4 5 6 7

x[2:5] # 2nd, 3rd, 4th, and 5th values

## [1] 2 3 4 5

x[c(2,4,7)] # 2nd, 4th, and 7th values

## [1] 2 4 7

Index a value from a vector with an object

object3 = 3  
x[object3]

## [1] 3

### Lists

A list is defined as a collection of multiple data types.

listObject <- list("Virginia Tech", 1, F)  
listObject

## [[1]]  
## [1] "Virginia Tech"  
##   
## [[2]]  
## [1] 1  
##   
## [[3]]  
## [1] FALSE

str(listObject)

## List of 3  
## $ : chr "Virginia Tech"  
## $ : num 1  
## $ : logi FALSE

### Matrix

Create a matrix from using matrix()

matrix(1:9, nrow = 4, ncol = 4) # range, number of rows, number of columns

## Warning in matrix(1:9, nrow = 4, ncol = 4): data length [9] is not a sub-  
## multiple or multiple of the number of rows [4]

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

## Basic Statistics

You can run a wide range of basic statistics in R - here we examine vector x we just created:

x

## [1] 1 2 3 4 5 6 7

mean(x) # arithmetic mean

## [1] 4

var(x) # variance

## [1] 4.666667

sd(x) #standard deviation

## [1] 2.160247

min(x) # minimum

## [1] 1

max(x) # maximum

## [1] 7

sum(x) # sum of all values

## [1] 28

prod(x) # product of all values

## [1] 5040

median(x) # median

## [1] 4

## View summary

summary(x) # summary statistics

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.0 2.5 4.0 4.0 5.5 7.0

summary(cars)

## speed dist   
## Min. : 4.0 Min. : 2.00   
## 1st Qu.:12.0 1st Qu.: 26.00   
## Median :15.0 Median : 36.00   
## Mean :15.4 Mean : 42.98   
## 3rd Qu.:19.0 3rd Qu.: 56.00   
## Max. :25.0 Max. :120.00

### Operators

Operators create conditions and are very useful for analyzing data

x <- 5  
y <- 10  
x < y # less than  
x <= y # less than or equal to  
x > y # greater than  
x >= y # greater than or equal to  
x == y # is equal to  
x != y #is not equal to  
!x # is not x  
x|y # x OR y  
x&y # x AND y  
isTRUE(x) # test if X is TRUE  
isTRUE(TRUE)

## Data Frames

When multiple columns or vectors of data are combined, this is called a dataframe (think of data in a spreadsheet). Dataframes can be created manually or loaded into R using a spreadsheet in .csv format.

### Writing Dataframes

First - We will write a simple dataset with mock catch data for a hypothetical fish species for the years 2010-2014 using data.frame()

catch1 <- data.frame("year" = 2010:2014, "catch" = c(5000,5200,7100,4900,5150))  
catch1

## year catch  
## 1 2010 5000  
## 2 2011 5200  
## 3 2012 7100  
## 4 2013 4900  
## 5 2014 5150

### Indexing Dataframes

Select value from row and column

catch1[4,1] # row, column

## [1] 2013

Select row

catch1[4,]

## year catch  
## 4 2013 4900

Select column

newcatch <- catch1[1]  
newcatch

## year  
## 1 2010  
## 2 2011  
## 3 2012  
## 4 2013  
## 5 2014

Use the $ to select columns as well

catch1$year # selects all values in the 'year' column

## [1] 2010 2011 2012 2013 2014

catch1$catch # selects all values in the 'catch' column

## [1] 5000 5200 7100 4900 5150

### Combining and merging dataframes

Dataframes can be merged using the row bind rbind() function.

# write another dataframe with same column names  
catch2 <- data.frame("year"=c(2015:2019), "catch"=c(4100,5700,3850,5500,4900)) # new catch dataframe  
allcatch <- rbind(catch1, catch2) # binds x & y dataframes together  
allcatch

## year catch  
## 1 2010 5000  
## 2 2011 5200  
## 3 2012 7100  
## 4 2013 4900  
## 5 2014 5150  
## 6 2015 4100  
## 7 2016 5700  
## 8 2017 3850  
## 9 2018 5500  
## 10 2019 4900

Add a new column of **hooks deployed** with cbind() or $

# these two lines do the same thing  
cbind(allcatch, hooks = c(20000, 25000, 23000, 31000, 19000, 21000, 21000, 30000, 18000, 34000))

## year catch hooks  
## 1 2010 5000 20000  
## 2 2011 5200 25000  
## 3 2012 7100 23000  
## 4 2013 4900 31000  
## 5 2014 5150 19000  
## 6 2015 4100 21000  
## 7 2016 5700 21000  
## 8 2017 3850 30000  
## 9 2018 5500 18000  
## 10 2019 4900 34000

allcatch$hooks = c(20000, 25000, 23000, 31000, 19000, 21000, 21000, 30000, 18000, 34000)

Perform operations on columns

allcatch$catch\_per\_hook = allcatch$catch / allcatch$hooks # divide catch by hooks deployed  
allcatch

## year catch hooks catch\_per\_hook  
## 1 2010 5000 20000 0.2500000  
## 2 2011 5200 25000 0.2080000  
## 3 2012 7100 23000 0.3086957  
## 4 2013 4900 31000 0.1580645  
## 5 2014 5150 19000 0.2710526  
## 6 2015 4100 21000 0.1952381  
## 7 2016 5700 21000 0.2714286  
## 8 2017 3850 30000 0.1283333  
## 9 2018 5500 18000 0.3055556  
## 10 2019 4900 34000 0.1441176

New data for the same years is presented

more\_catch = data.frame(year=c(2012, 2014), catch=c(4000, 2000))  
more\_catch

## year catch  
## 1 2012 4000  
## 2 2014 2000

Use the merge() function to join the two dataframes

merged\_allcatch = merge(allcatch, more\_catch, by="year", all.x=T) # merge more\_catch to allcatch by the year column  
# all.x=T means an inner join to the first dataframe specified  
merged\_allcatch

## year catch.x hooks catch\_per\_hook catch.y  
## 1 2010 5000 20000 0.2500000 NA  
## 2 2011 5200 25000 0.2080000 NA  
## 3 2012 7100 23000 0.3086957 4000  
## 4 2013 4900 31000 0.1580645 NA  
## 5 2014 5150 19000 0.2710526 2000  
## 6 2015 4100 21000 0.1952381 NA  
## 7 2016 5700 21000 0.2714286 NA  
## 8 2017 3850 30000 0.1283333 NA  
## 9 2018 5500 18000 0.3055556 NA  
## 10 2019 4900 34000 0.1441176 NA

### Subsetting Dataframes

subset() works by selecting a specific value or values based on a given condition

allcatch\_sub <- subset(allcatch, year>2016) # select all catch data recorded after 2016  
allcatch\_sub

## year catch hooks catch\_per\_hook  
## 8 2017 3850 30000 0.1283333  
## 9 2018 5500 18000 0.3055556  
## 10 2019 4900 34000 0.1441176

# Tip: run these functions one at a time instead of the whole chunk at once   
subset(allcatch, year==2012) # only 2012 data  
subset(allcatch, year!=2012) # select all EXCEPT the year 2012  
subset(allcatch, year>2012 | catch>4000)  
subset(allcatch, year>2012 & catch>4000 & hooks>30000)

## Setting a working directory and reading .csv files

### Setting a working directory

The working directory is the directory on your computer where R will retrieve and store data as well as save outputs such as plots and spreadsheets.

For this class, you should maintain a consistent file structure such that you can always code in it.

For example, on your local desktop, you might start a folder called “FisheriesManagement”, with subfolders such as “Lab1 - Intro to R part1”, “Lab2 - Intro to R part2”, “Lab3 - Fisheries Data Visualization” etc., so that you can easily tell R where to save and retrieve files. There are three ways to set your working directory.

|-- FisheriesManagement/ <- root directory  
 |-- Lab\_1/   
 |-- Lab1.Rmd   
 |-- Lab1.R  
 |-- allcatch.csv  
 |-- lab1\_testdata.csv  
 |-- Lab\_2/

1. In the menu bar, click on “Session” and then select “Set Working Directory” from the drop-down menu. Select “Choose Directory…” and navigate to the desired folder.
2. In your pane with the following tabs: Files, Plots, Packages, Help, and Viewer (Default position is bottom right). After selecting “Files”, navigate to the desired folder and click “More” (the gear icon) and then “Set as Working Directory”.
3. The command to set your working directory is setwd(). Everyone’s home directory will not be the same.

I highly recommend to use option 3 because it is faster and does not require clicking. So, create a file structure like the one outlined above.

getwd()

## [1] "/Users/jjeremy1/Dropbox/FisheriesManagement2023/Lab1 - Intro to R part1"

Here is a trick to get the directory of the file you are working on. This should typically go at the top of your Rmarkdown file

lab\_path = dirname(rstudioapi::getSourceEditorContext()$path)  
knitr::opts\_knit$set(root.dir = lab\_path)  
getwd()

## [1] "/Users/jjeremy1/Dropbox/FisheriesManagement2023/Lab1 - Intro to R part1"

### Reading a .csv file

Once you have set your working directory, you can read in the data using the read.csv() function.

list.files()

## [1] "allcatch.csv" "Lab1\_files" "lab1\_testdata.csv"  
## [4] "Lab1.docx" "Lab1.html" "Lab1.Rmd"

usa\_landings <- read.csv("lab1\_testdata.csv")

### Saving dataframes

We can now save our new master dataframe as a .csv file as well, using the write.csv() command. Files will be saved to your working directory.

write.csv(allcatch, "allcatch.csv", row.names=FALSE) # writes the dataframe that we created earlier  
list.files()

### Exploring the new dataframe

This dataset describes the total catch reported to the Food and Agriculture Organization (FAO) by the USA for the years 1950-2018.

paste("Catch average:",mean(usa\_landings$catch))

## [1] "Catch average: 3967149.96028986"

paste("Catch standard deviation:",sd(usa\_landings$catch))

## [1] "Catch standard deviation: 1165877.26111905"

summary(usa\_landings)

## year catch country   
## Min. :1950 Min. :2311726 Length:69   
## 1st Qu.:1967 1st Qu.:2772595 Class :character   
## Median :1984 Median :4229471 Mode :character   
## Mean :1984 Mean :3967150   
## 3rd Qu.:2001 3rd Qu.:4998490   
## Max. :2018 Max. :5694242

## Scripts

Scripts allow you to save and modify your code. You can open a script under File - New File - New R Script.

Type your code directly into the script, and run individual lines of code either by clicking “Run” at the top of the script, or using the shortcut Cmd-Enter (Mac) / Ctrl-Enter (Windows).

You can annotate your code with information that helps you to understand each line by commenting after the symbol # in a code line.

# anything written after '#' will not be run as code  
1+1 # A simple calculation

## [1] 2

## Using Packages

Packages offer different objects, functions, and data Get the list of all loaded packages

search()

## [1] ".GlobalEnv" "package:stats" "package:graphics"   
## [4] "package:grDevices" "package:utils" "package:datasets"   
## [7] "package:methods" "Autoloads" "package:base"

We are going to install tidyverse, which is a collection of several frequently used packages including ggplot, dplyr, and tidyr.

# install.packages("tidyverse")

# load the package  
library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
## ✔ ggplot2 3.3.6 ✔ purrr 0.3.4   
## ✔ tibble 3.1.8 ✔ dplyr 1.0.10  
## ✔ tidyr 1.2.0 ✔ stringr 1.4.1   
## ✔ readr 2.1.2 ✔ forcats 0.5.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

With tidyverse we can use more functions, such as left\_join()

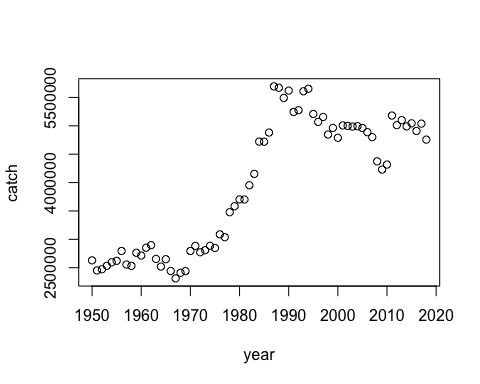
more\_usa\_landings = data.frame(year=c(1952, 1956), catch=c(40000, 20000), country="USA")  
merge(usa\_landings, more\_usa\_landings, by="year")

## year catch.x country.x catch.y country.y  
## 1 1952 2472565 USA 40000 USA  
## 2 1956 2795334 USA 20000 USA

## Plotting in Base R

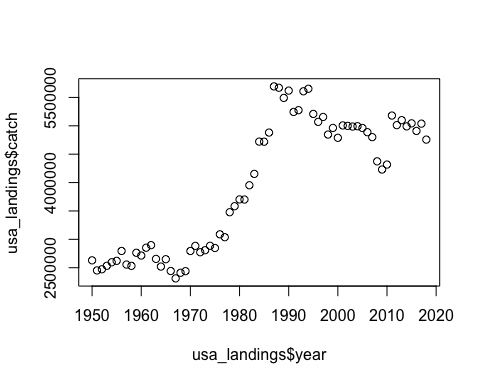
The Base R packages allow for some basic plotting through the plot() function. The default plot type will be a scatter plot. To plot the catches from our usa\_landings dataframe, we can use the following code:

plot(data=usa\_landings, catch~year)



We can also use the $ operator

plot(usa\_landings$catch~usa\_landings$year)



Now, see what else the plot function can do with ?plot

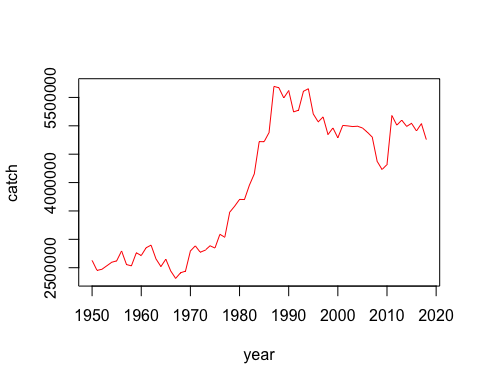
?plot

## Help on topic 'plot' was found in the following packages:  
##   
## Package Library  
## graphics /Library/Frameworks/R.framework/Versions/4.1/Resources/library  
## base /Library/Frameworks/R.framework/Resources/library  
##   
##   
## Using the first match ...

# Select Generic X-Y Plotting and scroll down to see different arguments the function accepts

This works for all functions. We can see in this case that we can change the plot style; for example, we can create the same plot as a line plot

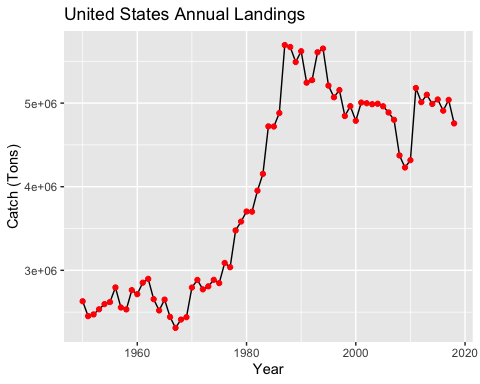
plot(catch~year, data=usa\_landings, type="l", col='red')



## Plotting in ggplot

ggplot is a commonly used package for plotting as it is highly customizable. Plots are created by defining geom objects, such as points, lines, or polygons. There are numerous ggplot guides available which discuss how different plot components can be customized.

ggplot(usa\_landings) + # ggplot loads in the data and uses '+' to add properties  
 geom\_line(aes(x=year, y=catch)) + # creates the line plot  
 geom\_point(aes(x=year, y=catch), color="red") + # adds red points for each value  
 ylab("Catch (Tons)") + # y axis title  
 xlab("Year") + # x axis title  
 ggtitle("United States Annual Landings") # main plot title



ggplot allows you to customize axes, labels, legends, line and shape styles, and most any other aspect of the plot you may want to modify. Look online at the available guides for additional information on plotting in ggplot.

[bookdown\_ggplot](https://bookdown.org/rdpeng/RProgDA/basic-plotting-with-ggplot2.html)

## Using packages and functions to access data

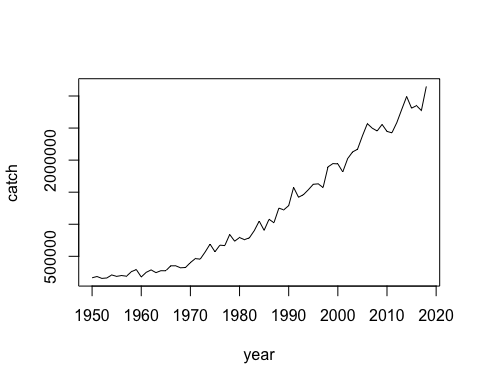
# install.packages("rfisheries")   
# install.packages("FSAdata")

library(rfisheries)  
library(FSAdata)

## ## FSAdata v0.3.9. See ?FSAdata to find data for specific fisheries analyses.

The rfisheries package provides access to the Open Fisheries global fisheries data - including the total annual number of *landings* for each year. The FSA package records biological fisheries data from the Global Fisheries Stock Assesment organization.

annual.landings = of\_landings(species="SKJ") # Global landings of skipkack tuna  
plot(data=annual.landings, catch~year, type='l')

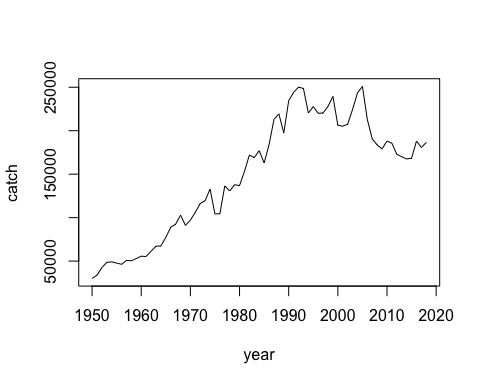


What does the function of\_landings() do?

?of\_landings

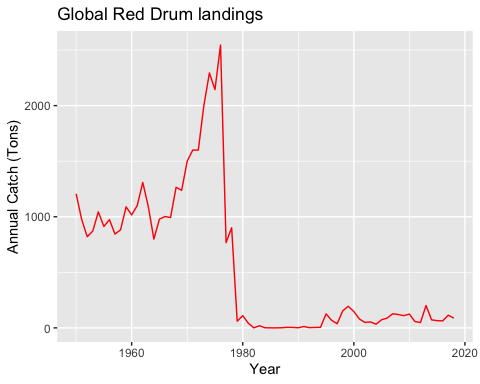
Plot by country with a country-specific code

annual.landings.usa = of\_landings(country="AUS") # Total landings from Australia  
plot(data=annual.landings.usa, catch~year, type='l')



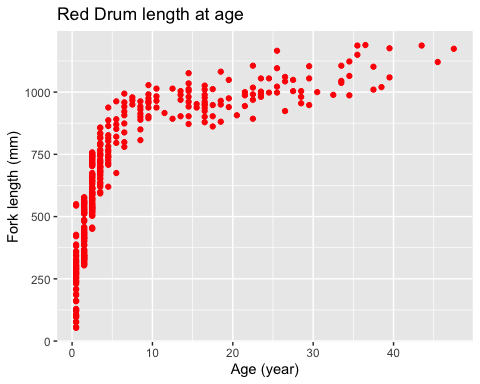
Use ggplot to plot landings of Red Drum and annotate the axes and title.

RDM\_landings = of\_landings(species="RDM")  
ggplot(RDM\_landings) +  
 geom\_line(aes(x=year, y=catch), col="red") +  
 ylab("Annual Catch (Tons)")+  
 xlab("Year")+  
 ggtitle("Global Red Drum landings")



We can plot biological data using the FSA package

data(RedDrum) # loads FSA data for Red Drum biological entries  
ggplot(RedDrum) +  
 geom\_point(aes(x=age, y=fl), col="red") + # using points instead of a line  
 ylab("Fork length (mm)")+  
 xlab("Age (year)")+  
 ggtitle("Red Drum length at age")



# Assignment instructions

Choose *1 country* and *1 species* from rfisheries, and *2 species* from FSAdata. Explore the data with statistical functions and summaries, and visualize the data with plots. Keep track of your report with RMarkdown and when you are finished, Knit the file to Word (you can knit to HTML first to see if it compiles without error).

Find all countries from rfisheries

head(of\_country\_codes()) # `head()` prints the top of the output, and leaves out the rest

## country iso3c  
## 1 Afghanistan AFG  
## 2 Albania ALB  
## 3 Algeria DZA  
## 4 American Samoa ASM  
## 5 Andorra AND  
## 6 Angola AGO

Find all species from rfisheries

head(of\_species\_codes())

## scientific\_name taxocode a3\_code isscaap english\_name  
## 1 Petromyzon marinus 1020100101 LAU 25 Sea lamprey  
## 2 Lampetra fluviatilis 1020100201 LAR 25 River lamprey  
## 3 Lampetra tridentata 1020100202 LAO 25 Pacific lamprey  
## 4 Ichthyomyzon unicuspis 1020100401 LAY 25 Silver lamprey  
## 5 Eudontomyzon mariae 1020100501 LAF 25 Ukrainian brook lamprey  
## 6 Geotria australis 1020100701 LAE 25 Pouched lamprey

Use of\_landings() with country OR species arguments

Find all species from FSA [FSAdata](http://derekogle.com/fishR/data/byPackage) Load species data with

# data([insert species name])  
# example  
data(Bonito)  
head(Bonito)

## sex age fl  
## 1 Female 0.1666667 30.3  
## 2 Female 0.1666667 30.5  
## 3 Female 0.1666667 28.8  
## 4 Female 0.1666667 26.3  
## 5 Female 0.1666667 31.9  
## 6 Female 0.1666667 32.4

## Assignment 1