



UiT The Arctic University of Norway

# FSK-2053 Data science & bioinformatics for fisheries and aquaculture

## *Lecture 2 – Data Wrangling*

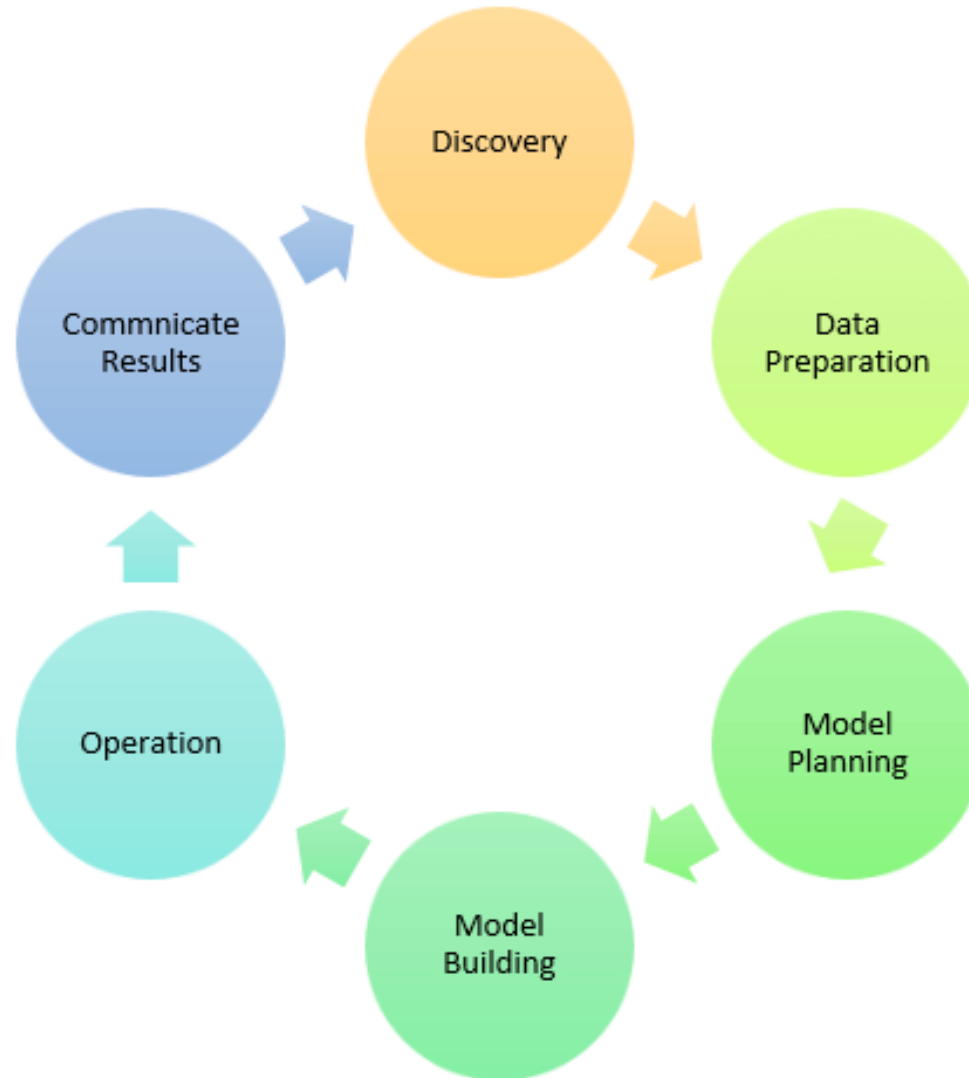
Daniel Kumazawa Morais

*daniel.morais@uit.no*

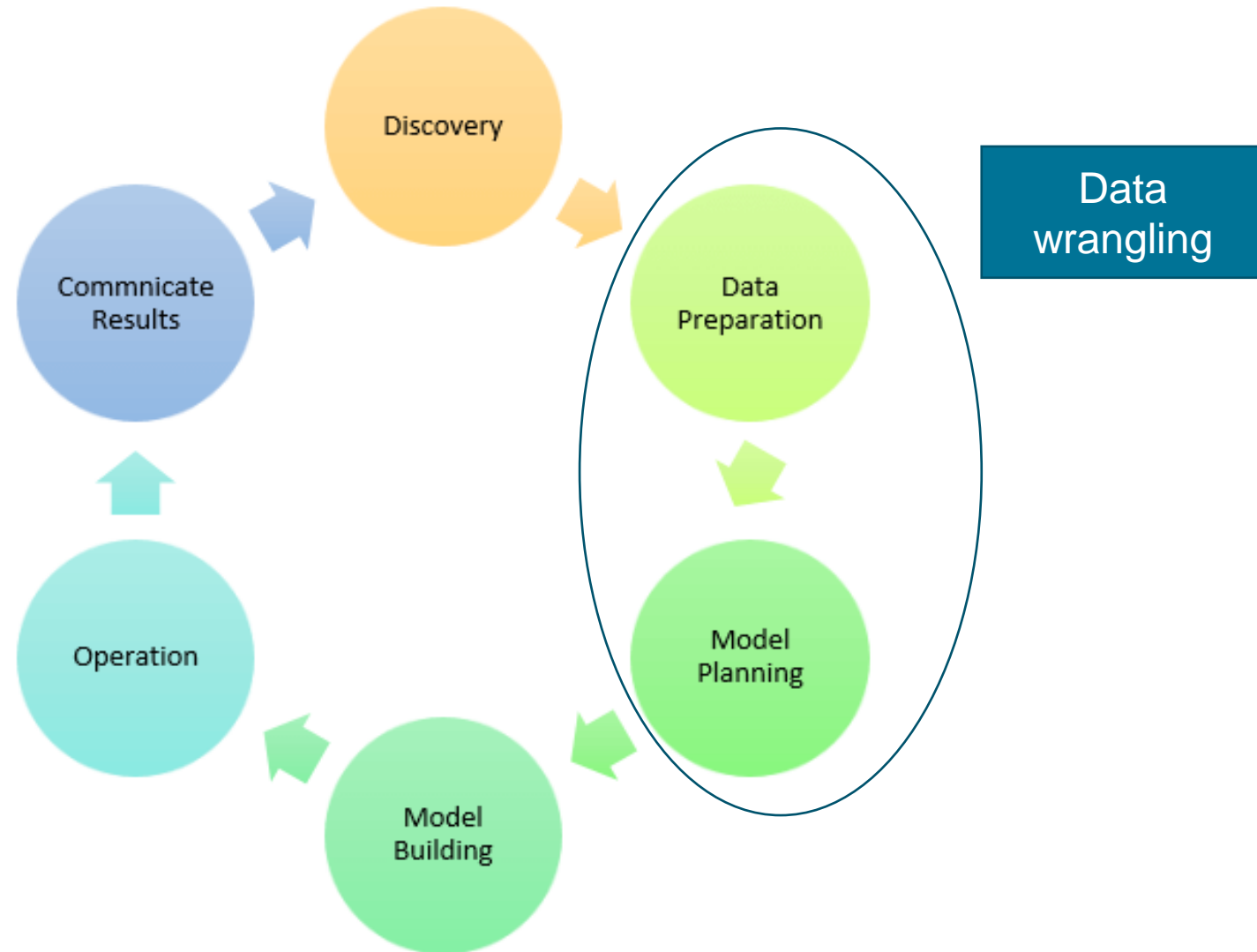


[https://en.wikipedia.org/wiki/Wrangler\\_\(profession\)#/media/File:07350u\\_The\\_horse\\_wrangler.tif](https://en.wikipedia.org/wiki/Wrangler_(profession)#/media/File:07350u_The_horse_wrangler.tif)

# The Data Science Workflow



# The Data Science Workflow



## 1. Discovery:

Discovery step involves acquiring data from all the identified internal & external sources which helps you to shape and describe the problem.

The data can be:

- Retrieved from online public databases (using direct queries or automated APIs)
- Census and governmental databases
- Other... e.g. gathered from social media

# Data Import : : CHEAT SHEET

high

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.



The front side of this sheet shows how to read text files into R with **readr**.



The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

## OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

## Save Data

Save **x**, an R object, to **path**, a file path, as:

### Comma delimited file

**write\_csv**(x, path, na = "NA", append = FALSE, col\_names = !append)

### File with arbitrary delimiter

**write\_delim**(x, path, delim = " ", na = "NA")

## Read Tabular Data - These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
n_max), progress = interactive())
```

a,b,c  
1,2,3  
4,5,NA

A	B	C
1	2	3
4	5	NA

### Comma Delimited Files

**read\_csv**("file.csv")

To make file.csv run:

**write\_file**(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

a;b;c  
1;2;3  
4;5;NA

A	B	C
1	2	3
4	5	NA

### Semi-colon Delimited Files

**read\_csv2**("file2.csv")

**write\_file**(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

a|b|c  
1|2|3  
4|5|NA

A	B	C
1	2	3
4	5	NA

### Files with Any Delimiter

**read\_delim**("file.txt", delim = "|")

**write\_file**(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

a b c  
1 2 3  
4 5 NA

A	B	C
1	2	3
4	5	NA

### Fixed Width Files

**read\_fwf**("file.fwf", col\_positions = c(1, 3, 5))

**write\_file**(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

### Tab Delimited Files

**read\_tsv**("file.tsv") Also **read\_table**()

**write\_file**(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

## USEFUL ARGUMENTS

a,b,c  
1,2,3  
4,5,NA

### Example file

**write\_file**("a,b,c\n1,2,3\n4,5,NA","file.csv")  
f <- "file.csv"

1	2	3
4	5	NA

### Skip lines

**read\_csv**(f, **skip** = 1)



## 2. Data preparation (data wrangling):

Data can have lots of inconsistencies like missing values, blank columns, incorrect data format which needs to be cleaned. You need to process, explore, filter, and condition data before modelling. The cleaner your data, the better are your predictions.

### Data Transformation with dplyr : : CHEAT SHEET



**dplyr** functions work with pipes and expect **tidy data**. In tidy data:



Each **variable** is in its own **column**



Each **observation**, or **case**, is in its own **row**



**pipes**

**x %>% f(y)** becomes **f(x, y)**

#### Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



**summary function**



**summarise(.data, ...)**  
Compute table of summaries.  
*summarise(mtcars, avg = mean(mpg))*



**count(x, ..., wt = NULL, sort = FALSE)**  
Count number of rows in each group defined by the variables in ... Also **tally()**.  
*count(iris, Species)*

#### VARIATIONS

**summarise\_all()** - Apply funs to every column.  
**summarise\_at()** - Apply funs to specific columns.  
**summarise\_if()** - Apply funs to all cols of one type.

#### Manipulate Cases

##### EXTRACT CASES

Row functions return a subset of rows as a new table.



**filter(.data, ...)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*



**distinct(.data, ..., keep\_all = FALSE)** Remove rows with duplicate values.  
*distinct(iris, Species)*



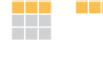
**sample\_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame())** Randomly select fraction of rows.  
*sample\_frac(iris, 0.5, replace = TRUE)*



**sample\_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame())** Randomly select size rows. *sample\_n(iris, 10, replace = TRUE)*



**slice(.data, ...)** Select rows by position.  
*slice(iris, 10:15)*



**top\_n(x, n, wt)** Select and order top n entries (by group if grouped data). *top\_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with **filter()**

#### Manipulate Variables

##### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



**pull(.data, var = -1)** Extract column values as a vector. Choose by name or index.  
*pull(iris, Sepal.Length)*



**select(.data, ...)**  
Extract columns as a table. Also **select\_if()**.  
*select(iris, Sepal.Length, Species)*

Use these helpers with **select()**,  
e.g. *select(iris, starts\_with("Sepal"))*

<b>contains(match)</b>	<b>num_range(prefix, range)</b> : e.g. <i>mpg:cyl</i>
<b>ends_with(match)</b>	<b>one_of(...)</b> : e.g. <i>-Species</i>
<b>matches(match)</b>	<b>starts_with(match)</b>

##### MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).



**mutate(.data, ...)**  
Compute new column(s).  
*mutate(mtcars, rpm = 1/mpg)*

### **3. Model planning:**

In this stage, you need to determine the method and technique to draw the relation between input variables. Planning for a model is performed by using different summarizing statistical tools and visualization tools.

### **4. Model building:**

In this step, the actual model building process starts. Here, data scientist typically distributes datasets for training and testing. Techniques like association, classification, and clustering are applied to the training data set. The model once prepared is tested against the "testing" dataset.

### 3. Model planning:

In this stage, you need to determine the method and technique to draw the relation between input variables. Planning for a model is performed by using different summarizing statistical tools and visualization tools.

### 4. Model building:

In this step, the actual model building process starts. Here, data scientist typically distributes datasets for training and testing. Techniques like association, classification, and clustering are applied to the training data set. The model once prepared is tested against the "testing" dataset.

$$\bar{X} = \frac{\sum_{i=1}^n x_i}{n} \qquad \mu = \frac{\sum_{i=1}^N x_i}{N}$$

The simplest of the  
models



## 5. Operationalize:

In this stage, you deliver the final baselined model with reports, code, and technical documents. Model can be deployed into a real-time production environment after thorough testing.

## 6. Communicate results

In this stage, the key findings are communicated to all stakeholders. This helps you to decide if the results of the project are a success or a failure based on information from the model.

```
X Mean: 54.26  
Y Mean: 47.83  
X SD   : 16.76  
Y SD   : 26.93  
Corr.  : -0.06
```

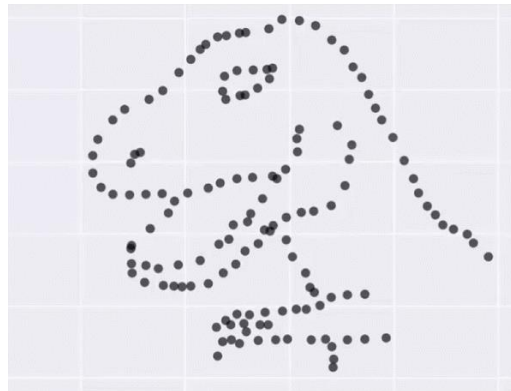
## 5. Operationalize:

In this stage, you deliver the final baselined model with reports, code, and technical documents. Model can be deployed into a real-time production environment after thorough testing.

## 6. Communicate results

In this stage, the key findings are communicated to all stakeholders. This helps you to decide if the results of the project are a success or a failure based on information from the model.

```
X Mean: 54.26  
Y Mean: 47.83  
X SD   : 16.76  
Y SD   : 26.93  
Corr.  : -0.06
```



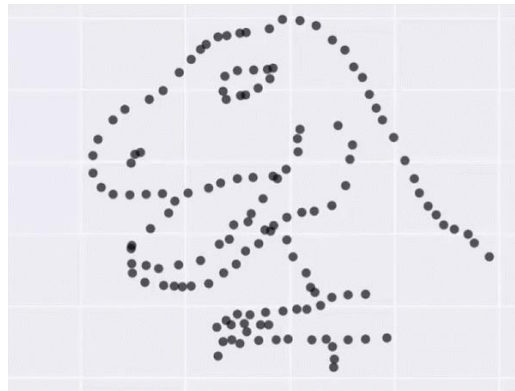
## 5. Operationalize:

In this stage, you deliver the final baselined model with reports, code, and technical documents. Model can be deployed into a real-time production environment after thorough testing.

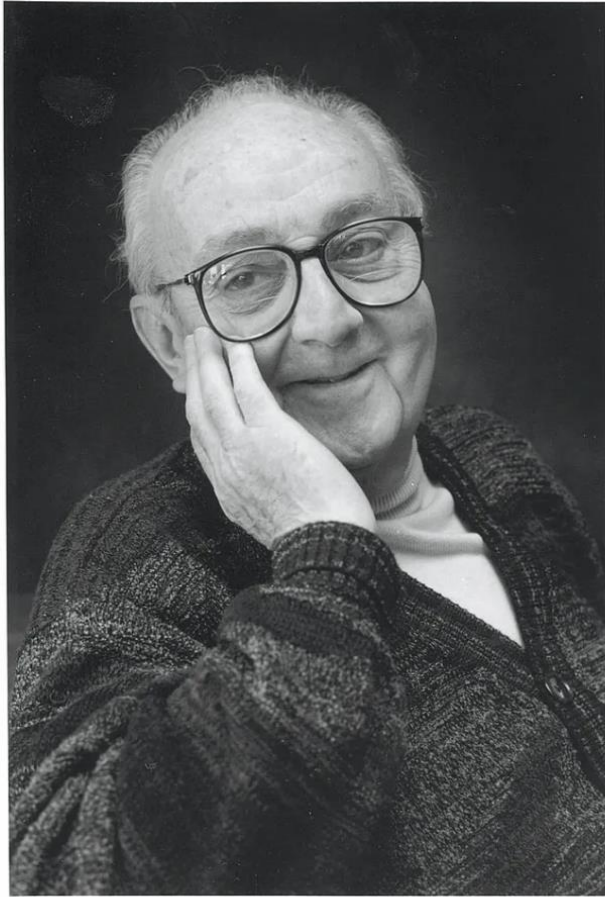
## 6. Communicate results

In this stage, the key findings are communicated to all stakeholders. This helps you to decide if the results of the project are a success or a failure based on information from the model.

```
X Mean: 54.26  
Y Mean: 47.83  
X SD : 16.76  
Y SD : 26.93  
Corr. : -0.06
```



**“All models are wrong, but some are useful”**



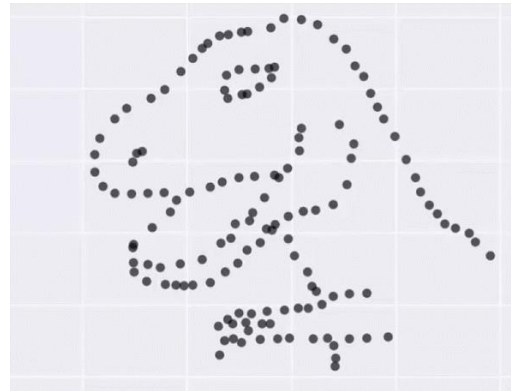
George E. P. Box.  
British mathematician and professor of statistics

<https://medium.com/@chasecottle/all-models-are-wrong-but-some-are-useful-c97d8f169a8e>

Simple models to describe the data

```
X Mean: 54.26  
Y Mean: 47.83  
X SD   : 16.76  
Y SD   : 26.93  
Corr.  : -0.06
```

The actual data



How can we translate it to our fields  
of expertise?

# Communicate results

## Critical step for evidence-base management



# Sources of Data:

## Online databases and resources for biology, fisheries & aquaculture

- BarentsWatch: <https://www.barentswatch.no/>
- Global Fishing Watch: <https://globalfishingwatch.org/>
- FAO Global Fishery Databases: <http://www.fao.org/fishery/topic/16054>
- FishBase: <https://www.fishbase.se/>
- Global Biodiversity Information Facility: <https://www.gbif.org/>
- WORMS (World Register Of Marine Species): <http://marinespecies.org/>
- Ocean Data Platform: <https://www.oceandata.earth/>
  
- NCBI/Genbank: <https://www.ncbi.nlm.nih.gov/>
- BOLD database: <https://boldsystems.org/>
  
- DRYAD: <https://datadryad.org/stash>
- Mendeley Data: <https://data.mendeley.com/>
- UiT Open Research Data: <https://dataverse.no/dataverse/uit>



# Sources of Data:

## Online databases and resources for biology, fisheries & aquaculture

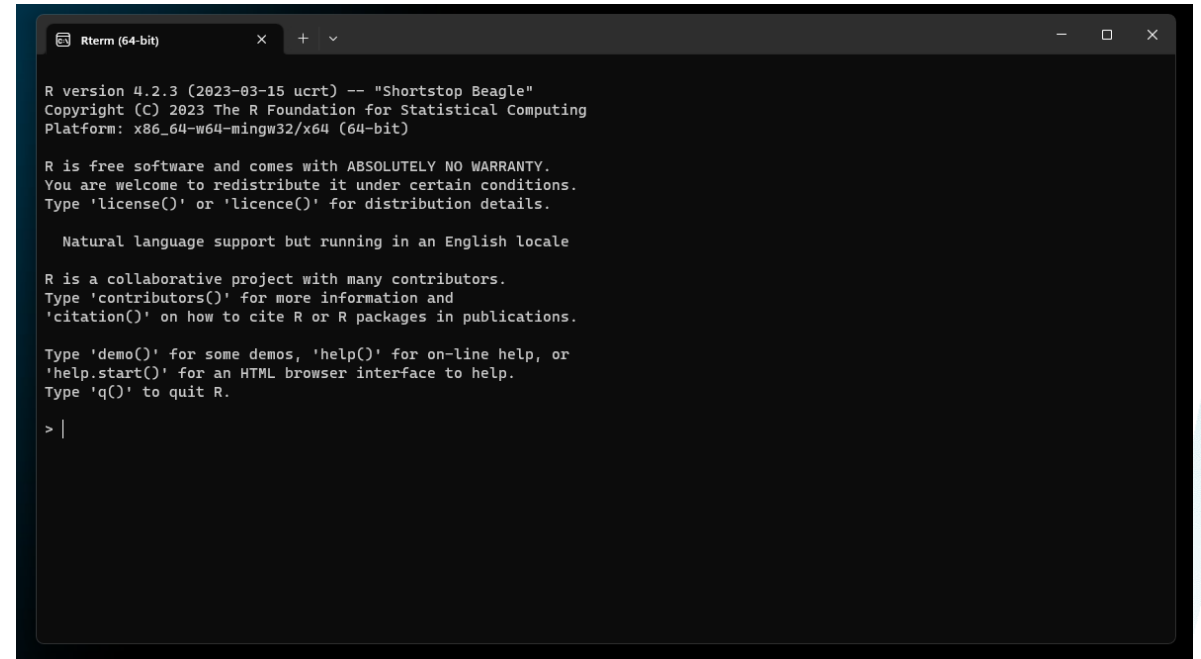
- BarentsWatch: <https://www.barentswatch.no/>
- Global Fishing Watch: <https://globalfishingwatch.org/>
- FAO Global Fishery Databases: <https://www.fao.org/fishery/statistics-query/en/capture>
- FishBase: <https://www.fishbase.se/>
- Global Biodiversity Information Facility: <https://www.gbif.org/>
- WORMS (World Register Of Marine Species): <http://marinespecies.org/>
- Ocean Data Platform: <https://www.oceandata.earth/>
  
- NCBI/Genbank: <https://www.ncbi.nlm.nih.gov/>
- BOLD database: <https://boldsystems.org/>
  
- DRYAD: <https://datadryad.org/stash>
- Mendeley Data: <https://data.mendeley.com/>
- UiT Open Research Data: <https://dataverse.no/dataverse/uit>

# The R prompt

The R programming language was officially released in the early 2000.

It was created by Ross Ihaka and Robert Gentleman as a tool for teaching statistics classes.

Therefore, it is considered that R doesn't behave as most general-purpose programming languages. Being quite unique in many aspects.

A screenshot of the R console window. The window title is "Rterm (64-bit)". The text inside shows the R version (4.2.3), copyright (2023), and platform (x86\_64-w64-mingw32/x64). It also displays the R license information, stating that R is free software with absolutely no warranty, and provides instructions on how to use the console, including commands like 'license()', 'contributors()', 'citation()', 'demo()', 'help()', 'help.start()', and 'q()'. The prompt ">|" is visible at the bottom.

```
R version 4.2.3 (2023-03-15 ucrt) -- "Shortstop Beagle"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

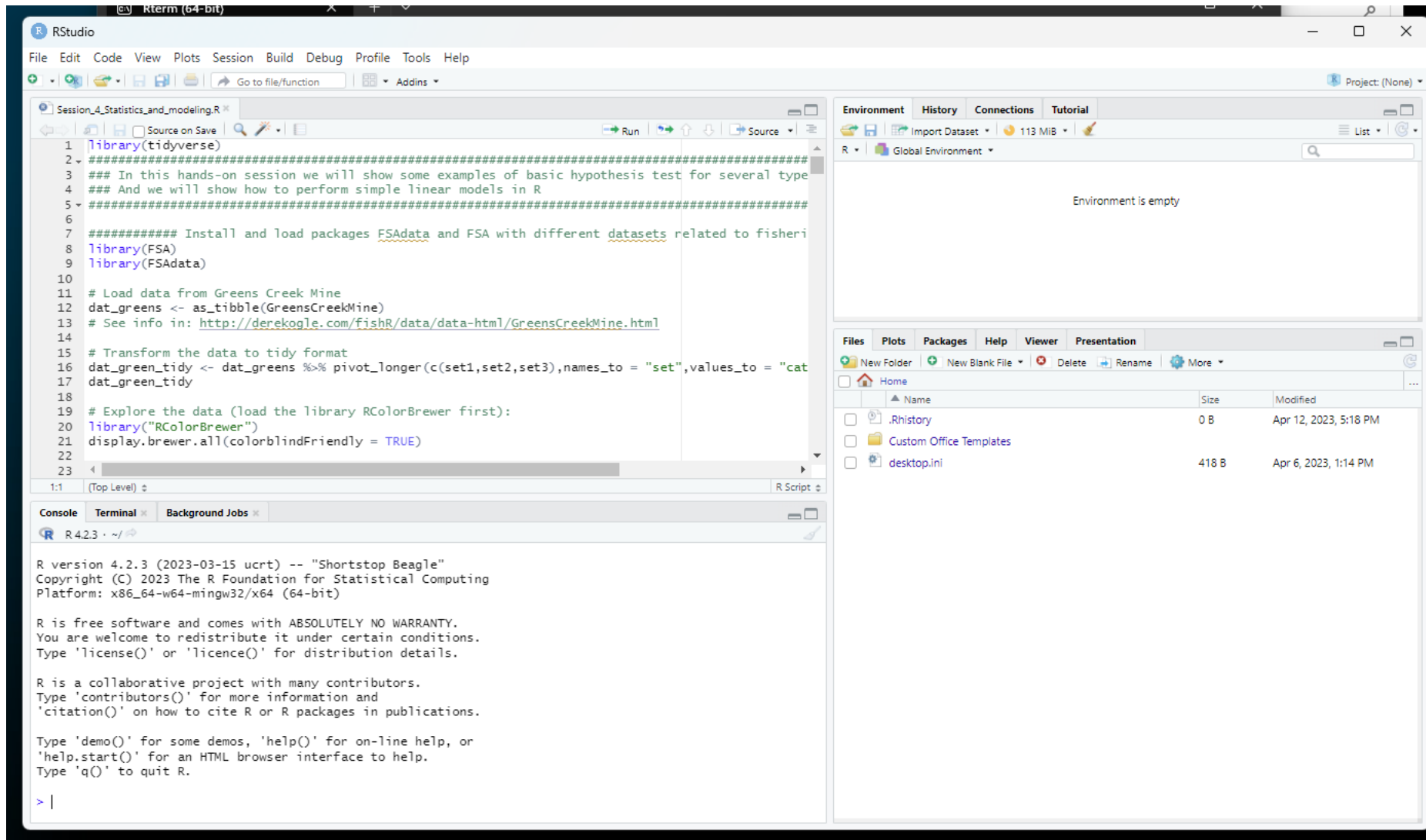
  Natural language support but running in an English locale

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.

>|
```

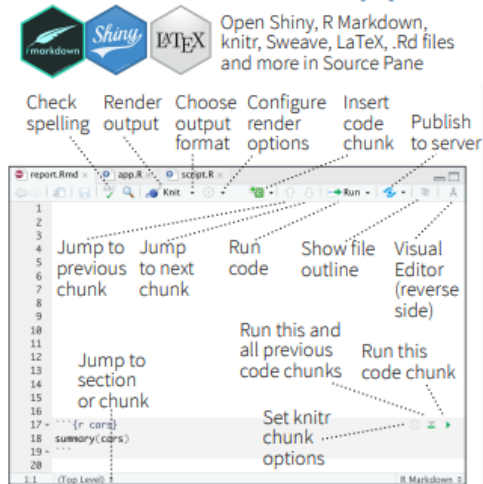
# IDE - Integrated Development Environment



# IDE - Integrated Development Environment

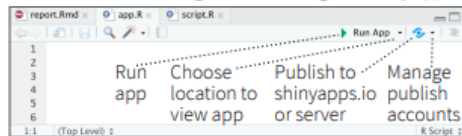
## RStudio IDE : : CHEAT SHEET

### Documents and Apps

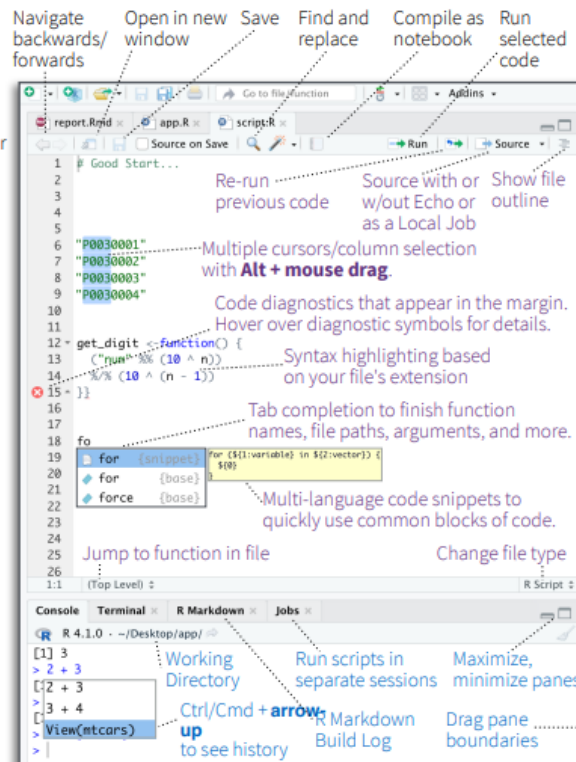


Access markdown guide at **Help > Markdown Quick Reference**  
See reverse side for more on **Visual Editor**

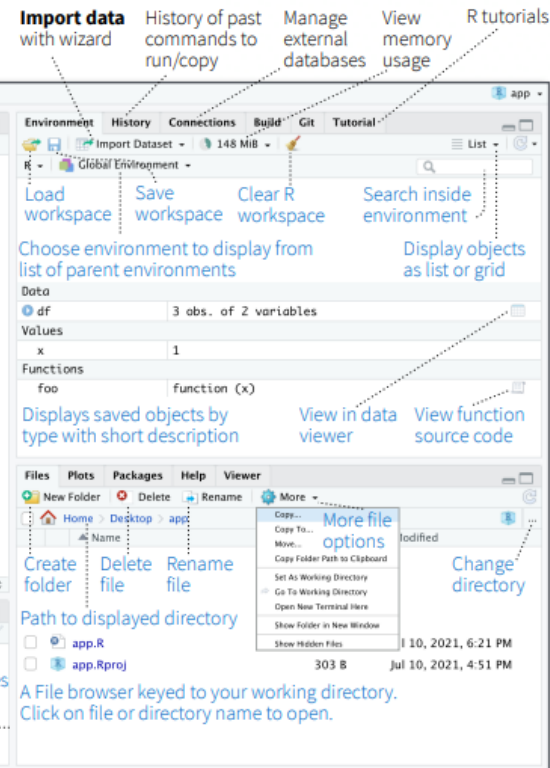
RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app



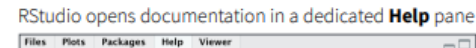
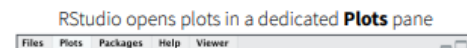
### Source Editor



### Tab Panes



### Package Development



# Data types in R

## Base R Cheat Sheet

### Getting Help

#### Accessing the help files

##### ?mean

Get help of a particular function.

**help.search('weighted mean')**

Search the help files for a word or phrase.

**help(package = 'dplyr')**

Find help for a package.

#### More about an object

##### str(iris)

Get a summary of an object's structure.

**class(iris)**

Find the class an object belongs to.

### Using Libraries

**install.packages('dplyr')**

Download and install a package from CRAN.

**library(dplyr)**

Load the package into the session, making all its functions available to use.

**dplyr::select**

Use a particular function from a package.

**data(iris)**

Load a built-in dataset into the environment.

### Working Directory

**getwd()**

Find the current working directory (where inputs are found and outputs are sent).

**setwd('C://file/path')**

Change the current working directory.

**Use projects in RStudio to set the working directory to the folder you are working in.**

RStudio® is a trademark of RStudio, Inc. • [CC BY](#) Mhairi McNeill • mhairihmcneill@gmail.com

### Vectors

#### Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

#### Vector Functions

**sort(x)**

Return x sorted.

**table(x)**

See counts of values.

**rev(x)**

Return x reversed.

**unique(x)**

See unique values.

#### Selecting Vector Elements

##### By Position

<b>x[4]</b>	The fourth element.
<b>x[-4]</b>	All but the fourth.
<b>x[2:4]</b>	Elements two to four.
<b>x[-(2:4)]</b>	All elements except two to four.
<b>x[c(1, 5)]</b>	Elements one and five.

##### By Value

<b>x[x == 10]</b>	Elements which are equal to 10.
<b>x[x &lt; 0]</b>	All elements less than zero.
<b>x[x %in% c(1, 2, 5)]</b>	Elements in the set 1, 2, 5.

##### Named Vectors

<b>x['apple']</b>	Element with name 'apple'.
-------------------	----------------------------

### Matrixes

**m <- matrix(x, nrow = 3, ncol = 3)**

Create a matrix from x.



**m[2, ]** - Select a row

**m[, 1]** - Select a column

**m[2, 3]** - Select an element

**t(m)**

Transpose

**m %\*% n**

Matrix Multiplication

**solve(m, n)**

Find x in:  $m \cdot x = n$

### Lists

**l <- list(x = 1:5, y = c('a', 'b'))**

A list is collection of elements which can be of different types.

**l[[2]]**

Second element of l.

**l[1]**

New list with only the first element.

**l\$x**

Element named x.

**l['y']**

New list with only element named y.

Also see the **dplyr** library.

### Data Frames

**df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))**

A special case of a list where all elements are the same length.

#### List subsetting

x	y
1	a
2	b
3	c

**df\$x**

**df[[2]]**

*Understanding a data frame*

**View(df)** See the full data frame.

**head(df)** See the first 6 rows.

#### Matrix subsetting

<b>df[, 2]</b>		<b>nrow(df)</b> Number of rows.	<b>cbind</b> - Bind columns.
<b>df[2, ]</b>		<b>ncol(df)</b> Number of columns.	<b>rbind</b> - Bind rows.
<b>df[2, 2]</b>		<b>dim(df)</b> Number of columns and rows.	



# Introduction to the Tidyverse



readr

tidyr

dplyr

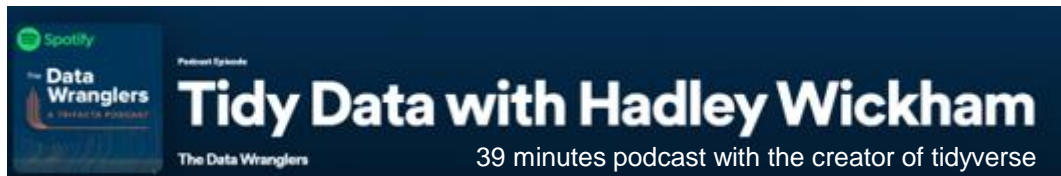
tibble

ggplot2

stringr

forcats

purrr



<https://www.tidyverse.org/packages/>



# Tidy Data

*“Happy families are all alike; every unhappy family is unhappy in its own way.”*

— Leo Tolstoy

*“Tidy datasets are all alike, but every messy dataset is messy in its own way.”*

— Hadley Wickham

There are three interrelated rules which make a dataset tidy:

- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	1280425583

variables

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	1280425583

observations

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	1280425583

values

# Tidyverse is based on using Tibbles

```
> 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
# i 140 more rows
# i Use `print(n = ...)` to see more rows
> |
```

```
> iris
  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
7      4.6         3.4         1.4         0.3 setosa
8      5.0         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
11     5.4         3.7         1.5         0.2 setosa
12     4.8         3.4         1.6         0.2 setosa
13     4.8         3.0         1.4         0.1 setosa
14     4.3         3.0         1.1         0.1 setosa
15     5.8         4.0         1.2         0.2 setosa
16     5.7         4.4         1.5         0.4 setosa
17     5.4         3.9         1.3         0.4 setosa
18     5.1         3.5         1.4         0.3 setosa
19     5.7         3.8         1.7         0.3 setosa
20     5.1         3.8         1.5         0.3 setosa
21     5.4         3.4         1.7         0.2 setosa
22     5.1         3.7         1.5         0.4 setosa
23     4.6         3.6         1.0         0.2 setosa
24     5.1         3.3         1.7         0.5 setosa
25     4.8         3.4         1.9         0.2 setosa
26     5.0         3.0         1.6         0.2 setosa
27     5.0         3.4         1.6         0.4 setosa
```

# Tidyverse is based on using Tibbles

*Tibbles* are data frames, but with a series of advantages:

- Printing is much tidier for large data
- Creating a tibble never changes the type of the inputs  
(e.g. it never converts strings to factors!)
- Creating a tibble never changes the names of variables
- Creating a tibble never creates row names
- Non-syntactic names are allowed for columns (delimited by backticks `` )
- Subsetting of a tibble always gives another tibble
- You can still use [ ] for subsetting a tibble. However, functions `dplyr::filter()` and `dplyr::select()` allow you to solve the same problems with clearer code

DOWNLOAD

# RStudio IDE

The most popular coding environment for R, built with love by Posit.

Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python. It includes a console, syntax-highlighting editor that supports direct code execution. It also features tools for plotting, viewing history, debugging and managing your workspace.

**RStudio Desktop**

RStudio Server

## RStudio Desktop

Find out more about RStudio Desktop and RStudio Desktop Pro below.

DOWNLOAD RSTUDIO

<https://posit.co/downloads/>

