Introduction to R, Session 1

Rockefeller University, Bioinformatics Resource Centre
http://rockefelleruniversity.github.io/Intro_To_R/

Overview

Set up

Materials.

All prerequisites, links to material and slides for this course can be found on github.

- Intro_To_R_1

Or can be downloaded as a zip archive from here.

- Download zip

Materials. - Presentations, source code and

Once the zip file in unarchived. All presentations as HTML slides and pages, their R code and HTML practical sheets will be available in the directories underneath.

- **presentations/slides/** Presentations as an HTML slide show.
- **presentations/singlepage/** Presentations as an HTML single page.
- **presentations/rcode/** R code in presentations.
- **exercises/** Practicals as HTML pages.
- **answers/** Practicals with answers as HTML pages and R code solutions.

Set the Working directory

Before running any of the code in the practicals or slides we need to set the working directory to the folder we unarchived.

You may navigate to the unarchived Intro_To_R_1Day folder in the Rstudio menu

Session -> Set Working Directory -> Choose Directory

or in the console.

Background to R

What is R?

R is a scripting language and environment for **statistical computing**.

Developed by Robert Gentleman and Ross Ihaka.

Inheriting much from **S** (Bell labs).

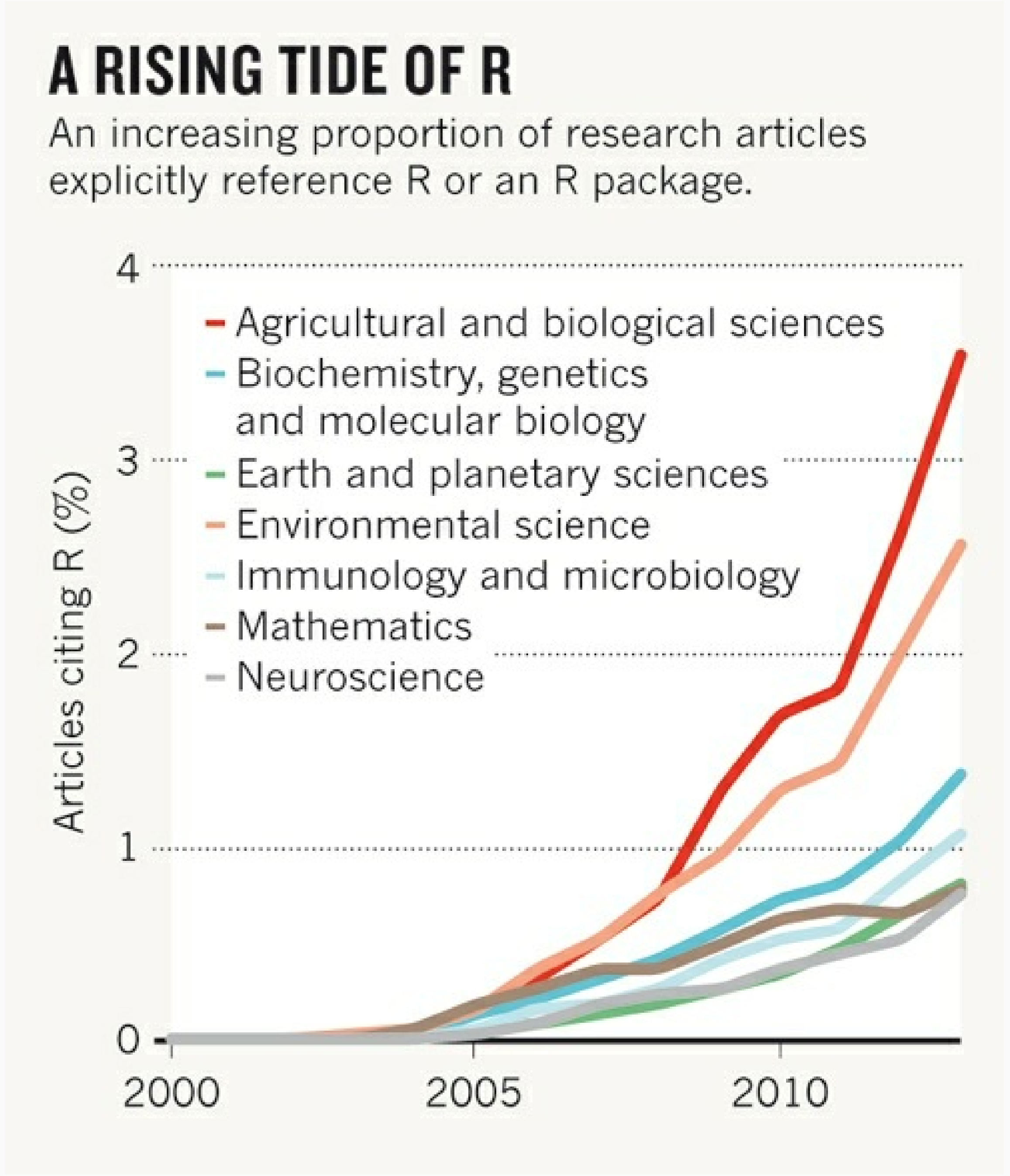
- Suited to high level data analysis
- Open source & cross platform
- Extensive graphics capabilities
- Diverse range of add-on packages
- Active community of developers
- Thorough documentation

What is R to you?

R comes with excellent "out-of-the-box" statistical and plotting capabilities.

R provides access to 1000s of packages (CRAN/MRAN/R-forge) which extend the basic functionality of R while maintaining high quality documentation.

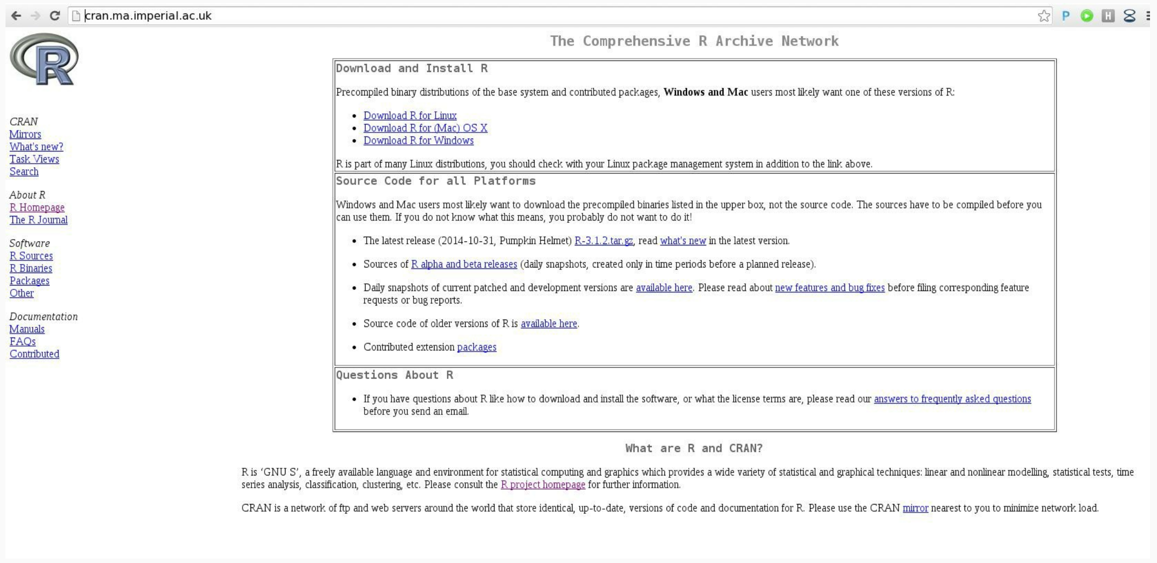
In particular, Robert Gentleman developed the **Bioconductor** project where 100's of packages are directly related to computational biology and analysis of associated high-throughput experiments.



How to get R?

Freely available from R-project website.

RStudio provides an integrated development environment (IDE) which is freely available from RStudio site



cran.ma.imperial.ac.uk

The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2014-10-31, Pumpkin Helmet) R-3.1.2.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

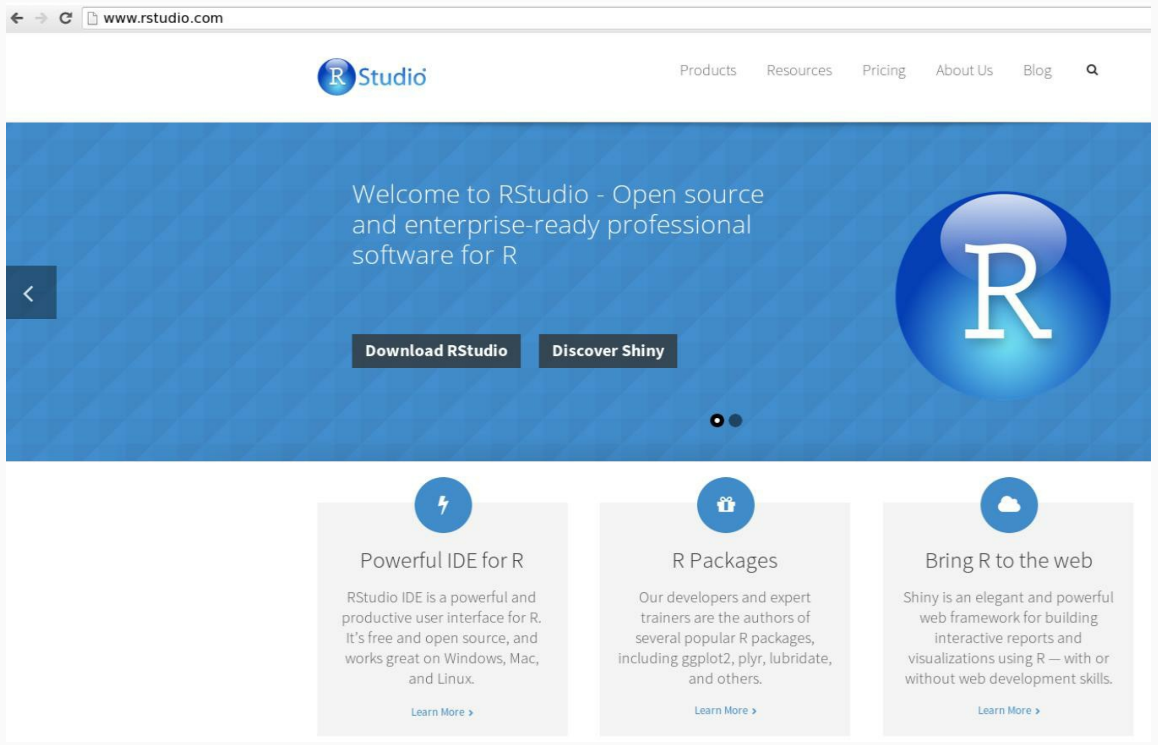
Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

What are R and CRAN?

R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.



www.rstudio.com

Welcome to RStudio - Open source and enterprise-ready professional software for R

Powerful IDE for R

RStudio IDE is a powerful and productive user interface for R. It's free and open source, and works great on Windows, Mac, and Linux.

R Packages

Our developers and expert trainers are the authors of several popular R packages, including ggplot2, plyr, lubridate, and others.

Bring R to the web

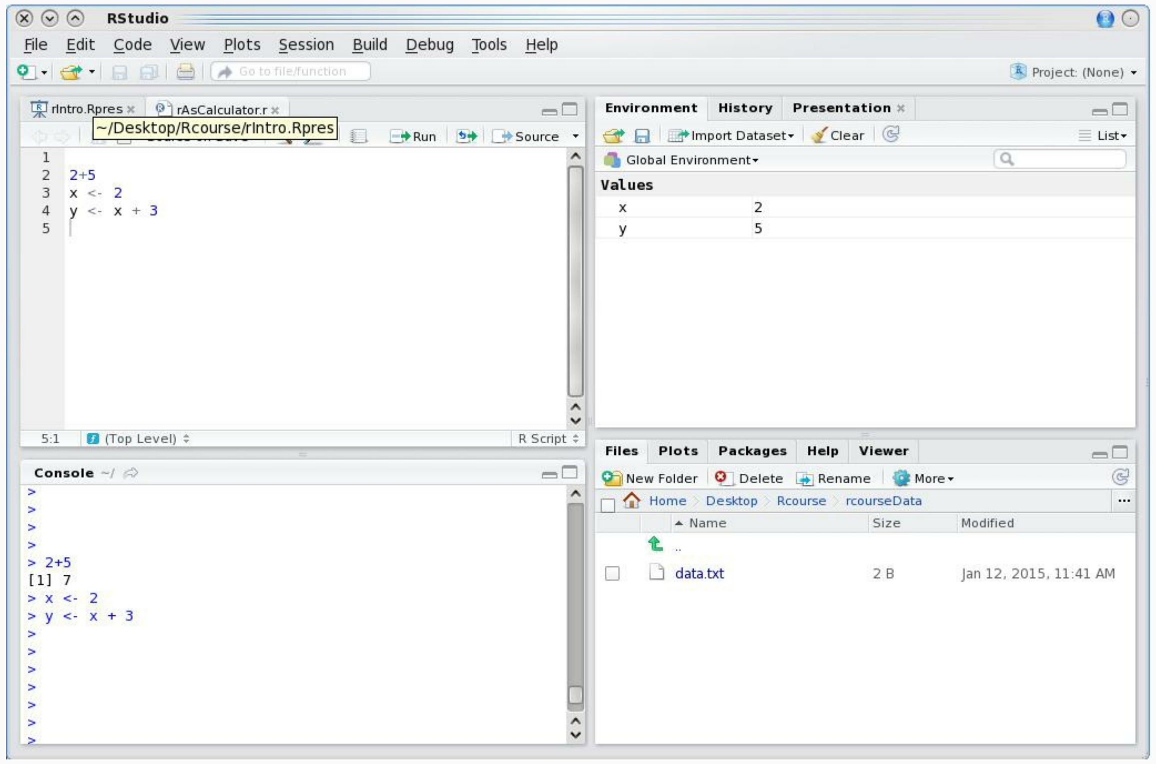
Shiny is an elegant and powerful web framework for building interactive reports and visualizations using R — with or without web development skills.

A quick tour of RStudio

Four main panels

- Scripting panel
- R interface
- Environment and history
- Files, directories and help

Let's load RStudio and take a look


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Project: (None)
rIntro.Rpres
rAsCalculator.r
~/Desktop/Rcourse/rIntro.Rpres
Run
Source
Environment History Presentation
Import Dataset
Clear
List
Global Environment
Console ~/
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home Desktop Rcourse rcourseData
Name Size Modified
5:1 (Top Level) R Script

```
2+5
x <- 2
y <- x + 3
```

```
> 2+5
[1] 7
> x <- 2
> y <- x + 3
```

```
Values
  x   2
  y   5
```

```
data.txt   2 B   Jan 12, 2015, 11:41 AM
```

Data types in R

Different Data types in R

- Simple calculations
- Variables
- Vectors
- Lists
- Matrices
- Data frames

Simple Calculations

At its most basic, **R** can be used as a simple calculator.

Using functions.

The **sqrt(25)** demonstrates the use of functions in R. A function performs a complex operation on it's arguments and returns the result.

In R, arguments are provided to a function within the parenthesis -- **( )** -- that follows the function name. So **sqrt(   )** will provide the square root of the value of   .

Other examples of functions include **min()**, **sum()**, **max()**.

Note multiple arguments are separated by a comma.

Using functions.

R has many useful functions "built in" and ready to use as soon as R is loaded.

An incomplete, illustrative list can be seen here

In addition to R standard functions, additional functionality can be loaded into R using libraries. These include specialised tools for areas such as sequence alignment, read counting etc.

If you need to see how a function works try **?** in front of the function name.

Lets run **?sqrt** in RStudio and look at the help.

Using functions (Arguments have names and

With functions such as min() and sqrt(), the arguments to be provided are obvious and the order of these arguments doesnt matter.

Many functions however have an order to their arguments. Try and look at the arguments for the dir() function using `?dir`.

Using functions (Setting names for

Often we know the names of arguments but not necessarily their order. In cases where we want to be sure we specify the right argument, we provide names for the arguments used.

This also means we don't have to copy out all the defaults for arguments preceeding it.

Variables

As with other programming languages and even graphical calculators, **R** makes use of **variables**.

A **variable** stores a value as a letter or word.

In **R**, we make use of the assignment operator **<-**

Now **x** holds the value of 10

Variables.

Variables can be altered in place

Variables.

Variables can be used just as the values they contain.

Variables can be used to create new variables

Vectors

In **R** the most basic variable or data type is a **vector**. A vector is an ordered collection of values. The x and y variables we have previously assigned are examples of a vector of length 1.

Vectors

To create a multiple value vector we use the function **c()** to the supplied arguments into one vector.

Vectors

Vectors of continuous stretches of values can be created by the shortcut - **:**

Other useful function to create stretchs of numeric vectors are **seq()** and **rep()**. The **seq()** function creates a sequence of numeric values from a specified start and end value, incrementing by a user defined amount. The **rep()** function repeats a variable a user-defined number of times.

Vectors - Indexing

Square brackets **[]** identify the position within a vector (the **index**). These indices can be used to extract relevant values from vectors.

Vectors - Indexing

Indices can be used to extract values from multiple positions within a vector.

Negative indices can be used to extract all positions except that specified

Vectors - Indexing and replacement

We can use indices to modify a specific position in a vector

Vectors - Indexing and replacement

Indices can be specified using other vectors.

Remember!

Square brackets **[]** for indexing

Parentheses **()** for function argments.

Vectors - Arithmetic operations

Vectors in R can be used in arithmetic operations as seen with variables earlier. When a standard arithmetic operation is applied to vector, the operation is applied to each position in a vector.

Multiple vectors can be used within arithmetic operations.

Vectors - Arithmetic operations

When applying an arithmetic operation between two vectors of unequal length, the shorter will be recycled.

Vectors - Character vectors.

So far we have only looked at numeric vectors or variables.

In R we can also create character vectors again using **c()** function. These vectors can be indexed just the same.

Character vectors can be used to assign names to other vectors.

Vectors - Character vectors as names.

These named vectors maybe indexed by a position's "name".

Index names missing from vectors will return special value "NA"

A note on NA values

In R, like many languages, when a value in a variable is missing, the value is assigned a **NA** value.

Similarly, when a calculation can not be perfomed, R will input a **NaN** value.

- **NA** - Not Available.
- **NaN** - Not A Number.

NA values allow for R to handle missing data correctly but requires different handling than standard numeric or character values. We will illustrate an example handling **NA** values later.

Vectors - The unique() function

The unique() function can be used to retrieve all unique values from a vector.

Vectors - Logical vectors

Logical vectors are a class of vector made up of TRUE/T or FALSE/F boolean values.

Logical vectors can be used like an index to specify postions in a vector. TRUE values will return the corresponding position in the vector being indexed.

Vectors - The %in% operator

A common task in R is to subset one vector by the values in another vector.

The **%in%** operator in the context **A %in% B** creates a logical vector of whether values in **A** matches any values in of **B**.

This can be then used to subset the values within one character vector by a those in a second.

Vectors . Logical vectors from operators

Vectors may be evaluated to produce logical vectors. This can be very useful when using a logical to index.

Common examples are:

- **==** evaluates as equal.
- **>** and **<** evaluates as greater or less than respectively.
- **>=** and **<=** evaluates as greater than or equal or less than or equal respectively.

Vectors - Combining logical vectors.

Logical vectors can be used in combination in order to index vectors. To combine logical vectors we can use some common R operators.

- **&** - Requires both logical operators to be TRUE
- **|** - Requires either logical operator to be TRUE.
- **!** - Reverses the logical operator, so TRUE is FALSE and FALSE is TRUE.

Vectors . Logical vectors continued.

Such combinations can allow for complex selection of a vector's values.

Time for an exercise!

Exercise on vectors can be found here

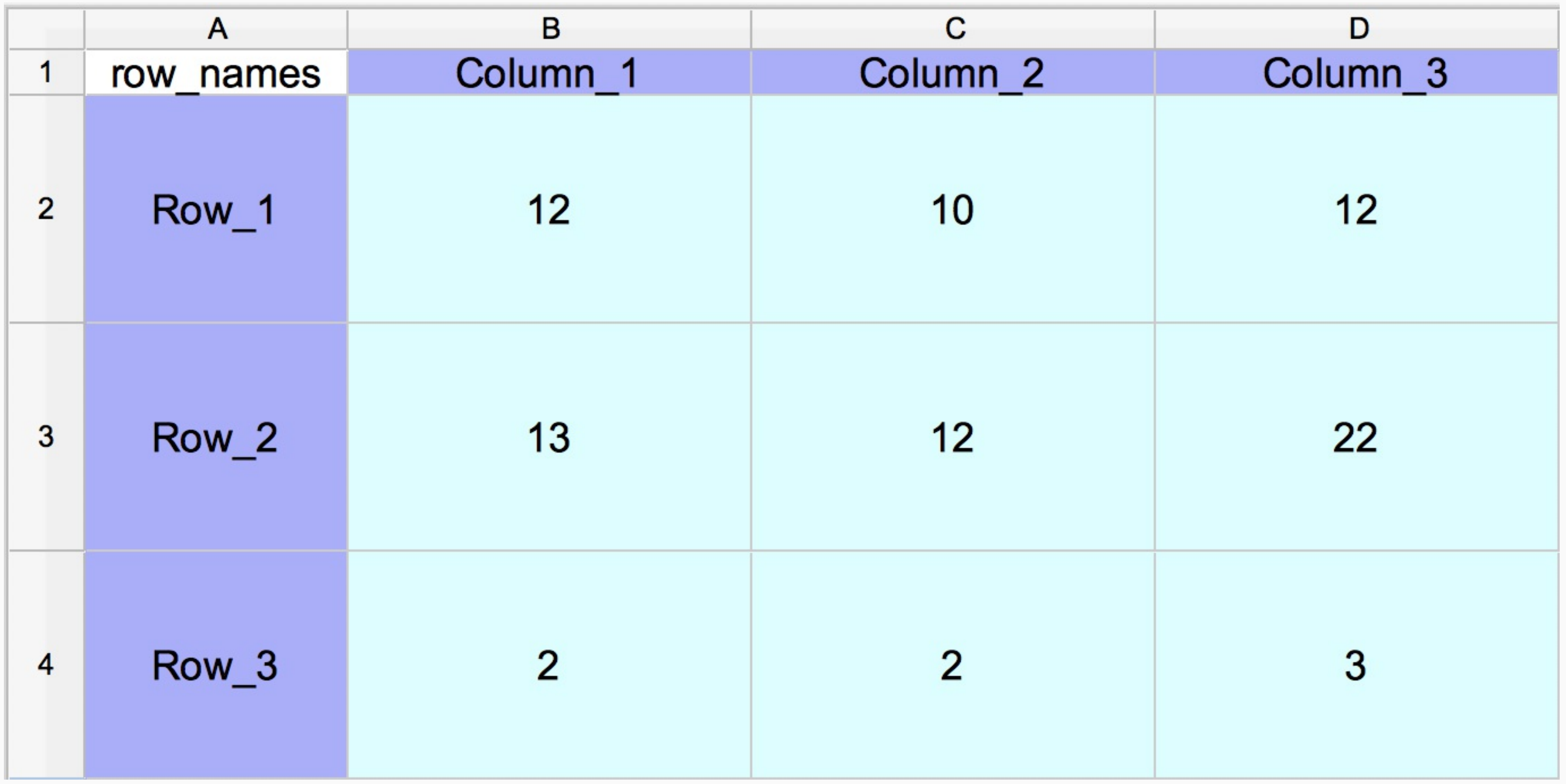
Answers to exercise.

Answers can be found here here

R code for solutions can be found here here

Matrices - Creating matrices

In programs such as Excel, we are used to tables.



	A	B	C	D
1	row_names	Column_1	Column_2	Column_3
2	Row_1	12	10	12
3	Row_2	13	12	22
4	Row_3	2	2	3

Matrices - Creating matrices

All progamming languages have a concept of a table. In **R**, the most basic table type is a **matrix**.

A **matrix** can be created using the   function with the arguments of **nrow** and **ncol** specifying the number of rows and columns respectively.

Matrices - Creating matrices

By default when creating a matrix using the **matrix** function, the values fill the matrix by columns. To fill a matrix by rows the **byrow** argument must be set to TRUE.

Matrices - Finding dimensions

To find dimensions of a matrix, the **dim()** function will provide dimensions as the row then column number while **nrow()** and **ncol()** will return just row number and column number respectively.

Matrices (Joining vectors and matrices)

A matrix can be created from multiple vectors or other matrices.

cbind() can be used to attach data to a matrix as columns.

Matrices - (Joining vectors and matrices)

rbind() functions to bind to a matrix as rows.

Matrices - Joining incompatable vectors and

When creating a matrix using **cbind()** or **matrix()** from incompatable vectors then the shorter vector is recycled.

Matrices - Joining incompatable vectors and

For **rbind()** function, the longer vector is clipped.

Matrices - Column and row names

As we have seen with vectors, matrices can be named. For matrices the naming is done by columns and rows using **colnames()** and **rownames()** functions.

Matrices - Column and row names

Information on matrix names can also be retreived using the same functions.

Matrices - Indexing

Selecting and replacing portions of a matrix can be done by **indexing** using square brackets **[]** much like for vectors.

When indexing matrices, two values may be provided within the square brackets separated by a comma to retrieve information on a matrix position.

The first value(s) corresponds to row(s) and the second to column(s).

-

Matrices - Indexing

Value of first column, second row

Matrices - Indexing

Similarly, whole rows or columns can be extracted. Single rows and columns will return a vector.

Values of second column (row index is empty!)

Values of third row (column index is empty!)

Matrices - Indexing

When multiple columns or row indices are specified, a matrix is returned.

Values of second and third row (column index is empty!)

Matrices - Indexing by name

As with vectors, names can be used for indexing when present

Matrices - Advanced indexing

As with vectors, matrices can be subset by logical vectors

Matrices - Arithmetic operations.

As with vectors, matrices can have arithmetic operations applied to cells,rows, columns or the whole matrix

Matrices - Replacement

As with vectors, matrices can have their elements replaced

Matrices -Matrices can contain only one

Matrices must be all one type (i.e. numeric or character).

Here replacing one value with character will turn numeric matrix to character matrix.

Time for an exercise!

Exercise on matrices can be found here

Answers to exercise.

Answers can be found here here

R code for solutions can be found here

Factors - Creating factors

A special case of a vector is a **factor**.

Factors are used to store data which may be grouped in categories (categorical data). Specifying data as categorical allows R to properly handle the data and make use of functions specific to categorical data.

To create a factor from a vector we use the **factor()** function. Note that the factor now has an additional component called **"levels"** which identifies all categories within the vector.

Factors - Summary() function

An example of the use of levels can be seen from applying the **summary()** function to the vector and factor examples

Factors - Display order of levels

In our factor example, the levels have been displayed in an alphabetical order. To adjust the display order of levels in a factor, we can supply the desired display order to **levels** argument in the **factor()** function call.

Factors - Nominal factors

In some cases there is no natural order to the categories such that one category is greater than the other (nominal data). In this case we can see that R is gender neutral.

Factors - Ordinal factors

In other cases there will be a natural ordering to the categories (ordinal data). A factor can be specified to be ordered using the **ordered** argument in combination with specified levels argument.

Factors - Replacement

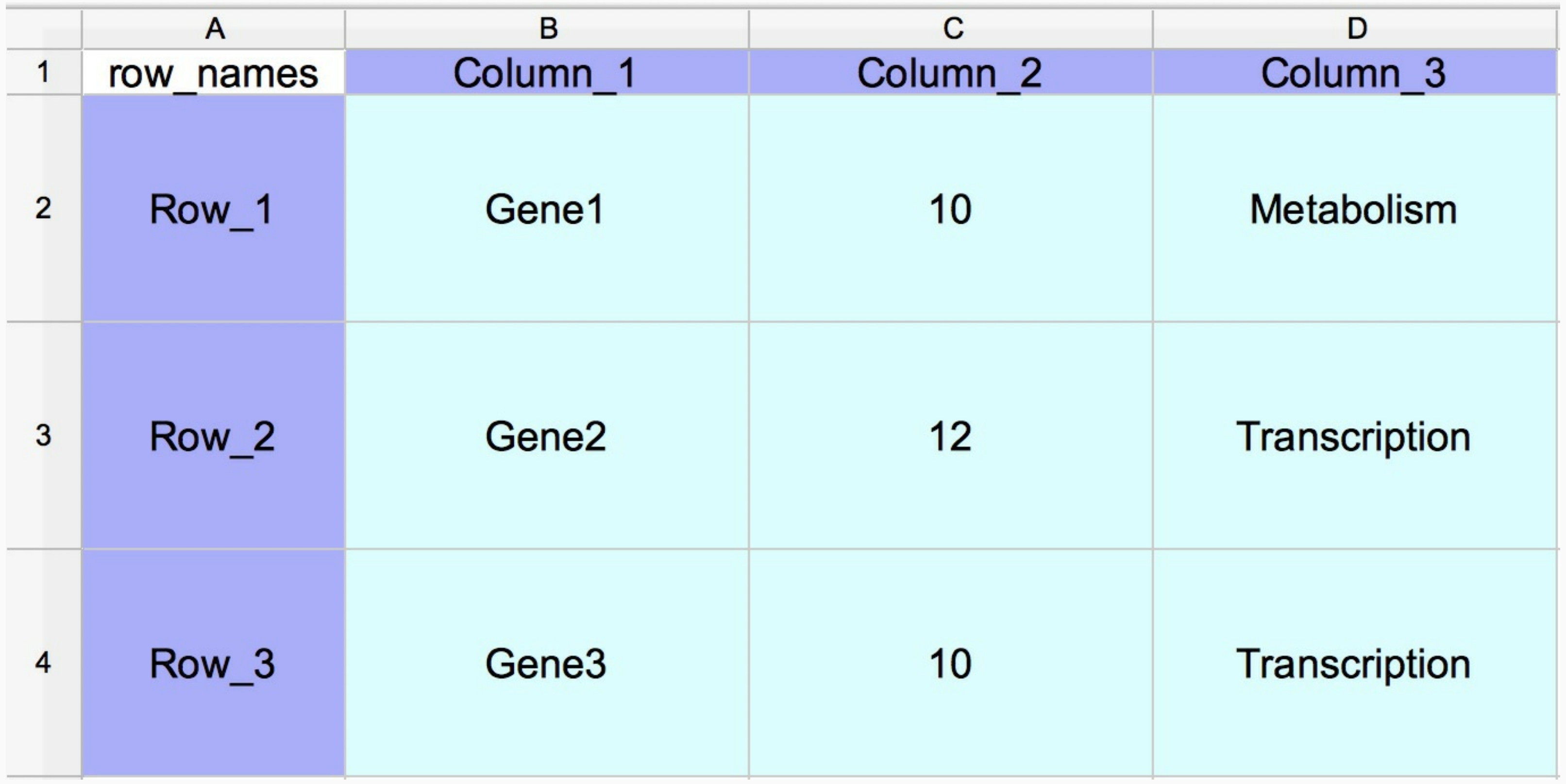
Unlike vectors, replacing elements within a factor isn't so easy. While replacing one element with an established level is possible, replacing with a novel element will result in a warning.

To add a new level we can use the levels argument.

Data frames - Creating data frames

We saw that with matrices you can only have one type of data. We tried to create a matrix with a character element and the entire matrix became a character.

In practice, we would want to have a table which is a mixture of types (e.g a table with sample names (character), sample type (factor) and survival time (numeric))



	A	B	C	D
1	row_names	Column_1	Column_2	Column_3
2	Row_1	Gene1	10	Metabolism
3	Row_2	Gene2	12	Transcription
4	Row_3	Gene3	10	Transcription

Data frames - Creating data frames

In R, we make use of the **data frame** object which allows us to store tables with columns of different data types. To create a data frame we can simply use the **data.frame()** function.

Data frames - Indexing and replacement

Data frames may be indexed just as matrices.

Data frames - Using $ to specify columns

Unlike matrices, it is possible to index a column by using the **$** symbol.

Data frames - Creating data frames

Using the **$** allows for R to autocomplete your selection and so can speed up coding.

But this will not work..

Data frames - Creating new columns with $

The **$** operator also allows for the creation of new columns for a data frame on the fly.

Data frames - Indexing and replacement

Certain columns can not be replaced in data frames. Numeric columns may have their values replaced but columns with character values may not by default.

Data frames - Indexing and replacement

This occurs because character vectors are treated as factors by default.

Data frames - Factors in data frames

It is possible to update factors in data frames just as with standard factors.

Data frames - Creating data frames without

If you wish to avoid using factors in data frames then the **stringsAsFactors** argument to **data.frame()** function should be set to **FALSE**

Data frames - Ordering with order() function

A useful function in R is **order()**

For numeric vectors, **order()** by default returns the indices of a vector in that vector's increasing order. This behaviour can be altered by using the "decreasing" argument passed to order.

Data frames - Ordering with NA values

When a vector contains NA values, these NA values will, by default, be placed last in ordering indices. This can be controlled by **na.last** argument.

Data frames - Ordering data frames

Since the order argument returns an index of intended order for a vector, we can use the order() function to order data frames by certain columns

Data frames - Ordering data frames

We can also use order to arrange multiple columns in a data frame by providing multiple vectors to order() function. Ordering will be performed in order of arguments.

Data frames - Merging data frames

A common operation is to join two data frames by a column of common values.

Data frames - Merging data frames with

To do this we can use the **merge()** function with the data frames as the first two arguments. We can then specify the columns to merge by with the **by** argument. To keep only data pertaining to values common to both data frames the **all** argument is set to TRUE.

Time for an exercise!

Exercise on data frames can be found here

Answers to exercise.

Answers can be found here here

R code solutions can be found here

Lists - Creating lists

Lists are the final data-type we will look at.

In R, lists provide a general container which may hold any data types of unequal lengths as part of its elements.

Lists - lists

To create a list we can simply use the **list()** function with arguments specifying the data we wish to include in the list.

Lists - Named lists

Just as with vectors, list elements can be assigned names.

Lists - Indexing

List, as with other data types in R can be indexed. In contrast to other types, using **[]** on a list will subset the list to another list of selected indices. To retrieve an element from a list in R , two square brackets **[[]]** must be used.

As with data.frames, the $ sign may be used to extract named elements from a list

Lists - Joining lists

Again, similar to vectors, lists can be joined together in R using the c() function

Lists - Joining vectors to lists

Note that on last slide we are joining two lists. If we joined a vector to a list, all elements of the vector would become list elements.

Lists - Flattening lists

Sometimes you will wish to "flatten" out a list. When a list contains compatable objects, i.e. list of all one type, the **unlist()** function can be used. Note the maintenance of names with their additional sufficies.

Lists - Flattening lists to matrices

A common step is to turn a list of standard results into matrix. This can be done in a few steps in R.