Universität Rostock Traditio et Innovatio

Universitätsmedizin Rostock

Integrative Bioinformatics and Systems Biology

Dr. Mohamed Hamed

Agenda

- Get to know each others

Agenda

- Get to know each others
- Your expectations

Agenda

- Get to know each others
- Your expectations
- Quick overview on the contents

LECTURE PLANNING

Lecture: 12 lectures, 3 hrs each.

Total workload: 42 hrs : 36 hrs of lectures and tutorials and 6 hrs of self studies.

Entrance requirements: basic knowledge of biology and computer science.

Literature: Lecture slides, tutorial handouts and scripts will be provided.

INTEGRATIVE BIOINFORMATICS

COURSE ABBREVIATION: INT-BIO

LANGUAGE: ENGLISH
USED MEDIA: POWERPOINT PRESENTATION

Module: Lecture and tutorial.

By: Dr. Mohamed Hamed

Head of the Integrative OMICs Analysis Group in Rostock University medical center, Rostock University, Germany.

MOTIVATION AND COURSE OBJECTIVES

The main challenge of modern systems biology is unraveling the holistic picture of the complex molecular interactions that occur on different molecular levels (genomic, transcriptomic, epigenomic, proteomic, etc). Therefore, the needs to integrate/jointly analyze biological data from different high-throughput technologies emerged in order to identify biomarkers for early diagnosis and prognosis of complex diseases and facilitating the development of novel treatment approaches.

This course aims at teaching students how to perform data-specific computational analyses as well as integrative analysis approaches, combining knowledge from different OMICs-based datasets. Both, theoretical and practical aspects will be covered. Students will have the opportunity to work both independently during tutorials and in teams during the research project.

COMPETENCES TO BE DEVELOPED

Students will get practical and extensive hands-on experience on:

-R scripting language and bioconductor packages.

-Data-specific computaional analysis and pipelines for the vast amounts of biological data produced using high-throughput technologies.

-Developing and applying integrative bioinformatics methods that could be utilized in all biology-related areas of interest.

-Basics of machine learning methods as tools for integrating biological features from heterogeneous Omics data.

-Students will be developing their own research projects, interpreting the obtained results, writing a manuscript, scientifically discussing the results.

ASSESSMENT

-Students need to finalize a research project applying all/ most of the learned methods and skills during the course. Novelty and extending the learned methods is highly encouraged and will be well graded.

-The outcomes of each research project should be compiled in a high scientific quality research article that is ready for submission in a peer-review journal.

-All projects will be presented, discussed and scientifically reviewed in the last lecture.

R language mini-course 1	R language mini-course 2	R language mini-course 3	R language mini-course 4
-Introduction to the course -Basics of R language and -R studio IDE	-R statistics -Advanced R statistics	-Bio-conductor packages	-Case study: Microarray analysis using R
Transcriptomic analysis I	**Transcriptomic analysis I I**	**Non-coding RNAs**	**Introduction to integrative bioinformatics**
-From microarrays to RNA-seq -RNA-seq analysis	-Linking to Ontologies and pathways -Data visualization	-Small and long non-coding RNAs -miRNA sequencing analysis -miRNA databases	-Importance of data integration -Different methods for biological data integration -OMICs data types, and TCGA repository -Databases of diseases-related genes and miRNAs
Network- based integrative methods	**Epigenetics**	**Integrative analysis based on machine learning**	**PROJECTS DISCUSSION AND CLOSURE**
-TFmiR analysis -Network motif analysis -Central hubs identifications	-Introduction to the epigenetic landscapes of normal and tumor cells -DNA methylation, Co-methylation analysis -DMRs identifications	-Introduction to machine learning in bioinformatics. -Unsupervised methods: Clustering biological data -PCA analysis, MOFA method -Outlook at deep neural networks applications in bioinformatics	-Projects presentation. -Reviews of the potential manuscripts

LECTURE PLANNING

Lecture: 12 lectures, 3 hrs each.

Total workload: 42 hrs : 36 hrs of lectures and tutorials and 6 hrs of self studies.

Entrance requirements: basic knowledge of biology and computer science.

Literature: Lecture slides, tutorial handouts and scripts will be provided.

INTEGRATIVE BIOINFORMATICS

COURSE ABBREVIATION: INT-BIO

LANGUAGE: ENGLISH
USED MEDIA: POWERPOINT PRESENTATION

Module: Lecture and tutorial.

By: Dr. Mohamed Hamed

Head of the Integrative OMICs Analysis Group in Rostock University medical center, Rostock University, Germany.

MOTIVATION AND COURSE OBJECTIVES

The main challenge of modern systems biology is unraveling the holistic picture of the complex molecular interactions that occur on different molecular levels (genomic, transcriptomic, epigenomic, proteomic, etc). Therefore, the needs to integrate/jointly analyze biological data from different high-throughput technologies emerged in order to identify biomarkers for early diagnosis and prognosis of complex diseases and facilitating the development of novel treatment approaches.

This course aims at teaching students how to perform data-specific computational analyses as well as integrative analysis approaches, combining knowledge from different OMICs-based datasets. Both, theoretical and practical aspects will be covered. Students will have the opportunity to work both independently during tutorials and in teams during the research project.

COMPETENCES TO BE DEVELOPED

Students will get practical and extensive hands-on experience on:

-R scripting language and bioconductor packages.

-Data-specific computaional analysis and pipelines for the vast amounts of biological data produced using high-throughput technologies.

-Developing and applying integrative bioinformatics methods that could be utilized in all biology-related areas of interest.

-Basics of machine learning methods as tools for integrating biological features from heterogeneous Omics data.

-Students will be developing their own research projects, interpreting the obtained results, writing a manuscript, scientifically discussing the results.

ASSESSMENT

-Students need to finalize a research project applying all/ most of the learned methods and skills during the course. Novelty and extending the learned methods is highly encouraged and will be well graded.

-The outcomes of each research project should be compiled in a high scientific quality research article that is ready for submission in a peer-review journal.

-All projects will be presented, discussed and scientifically reviewed in the last lecture.

R language mini-course 1	R language mini-course 2	R language mini-course 3	R language mini-course 4
-Introduction to the course -Basics of R language and -R studio IDE	-R statistics -Advanced R statistics	-Bio-conductor packages	-Case study: Microarray analysis using R
Transcriptomic analysis I	**Transcriptomic analysis I I**	**Non-coding RNAs**	**Introduction to integrative bioinformatics**
-From microarrays to RNA-seq -RNA-seq analysis	-Linking to Ontologies and pathways -Data visualization	-Small and long non-coding RNAs -miRNA sequencing analysis -miRNA databases	-Importance of data integration -Different methods for biological data integration -OMICs data types, and TCGA repository -Databases of diseases-related genes and miRNAs
Network- based integrative methods	**Epigenetics**	**Integrative analysis based on machine learning**	**PROJECTS DISCUSSION AND CLOSURE**
-TFmiR analysis -Network motif analysis -Central hubs identifications	-Introduction to the epigenetic landscapes of normal and tumor cells -DNA methylation, Co-methylation analysis -DMRs identifications	-Introduction to machine learning in bioinformatics. -Unsupervised methods: Clustering biological data -PCA analysis, MOFA method -Outlook at deep neural networks applications in bioinformatics	Projects presentation. Reviews of the potential manuscripts

Schedule

Saturdays	12:00-15:00
* Next Saturday (14.11.2020)	off

WGS/WES (Optional)

Whole Genome/Exome sequencing (WGS/WES)

- Introduction to WGS and WES
- WGS/WES analysis workflow
- Quality control
- Alignment and Mapping
- PCR duplicates removal
- SNP calling
- SNP annotations
- Variant analysis and mutation types
- Copy number variation analysis (CNV) (optional)
- Calculating tumor mutational burden (TMB)
- Data visualization

Agenda

- Get to know each others
- Your expectations
- Quick overview on the contents
- Course outcomes

Agenda

- Get to know each others
- Your expectations
- Quick overview on the contents
- Course outcomes
- Grading the course (student groups)

NATIONAL CANCER INSTITUTE
GDC Data Portal

Home | Projects | Exploration | Analysis | Repository | Quick Search | Manage Sets | Login | Cart 472 | GDC Apps

Harmonized Cancer Datasets

Genomic Data Commons Data Portal

Get Started by Exploring:

Projects | Exploration | Analysis | Repository

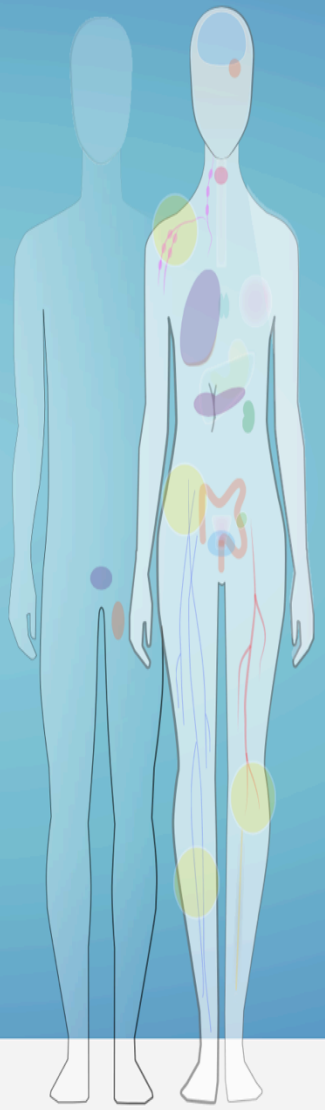
e.g. BRAF, Breast, TCGA-BLCA, TCGA-A5-A0G2

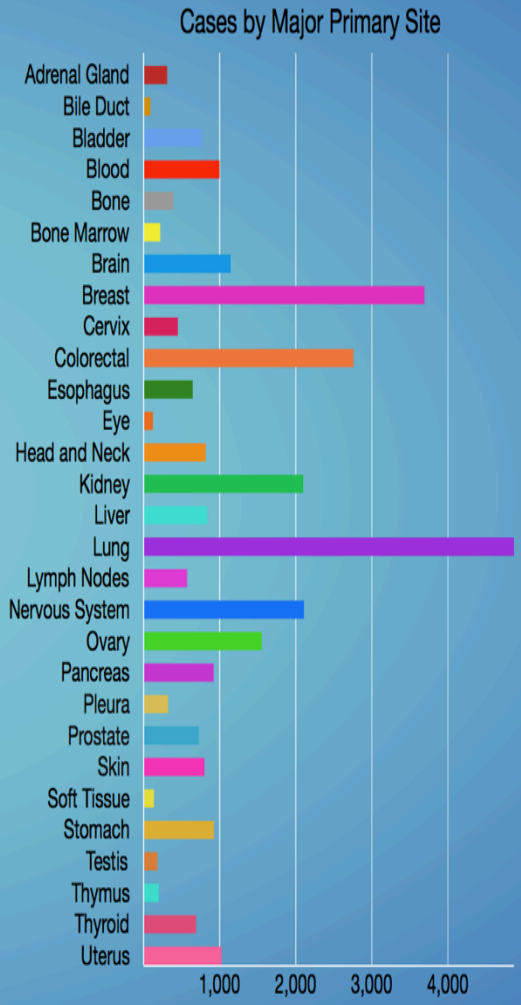
Data Portal Summary

Data Release 13.0 - September 27, 2018

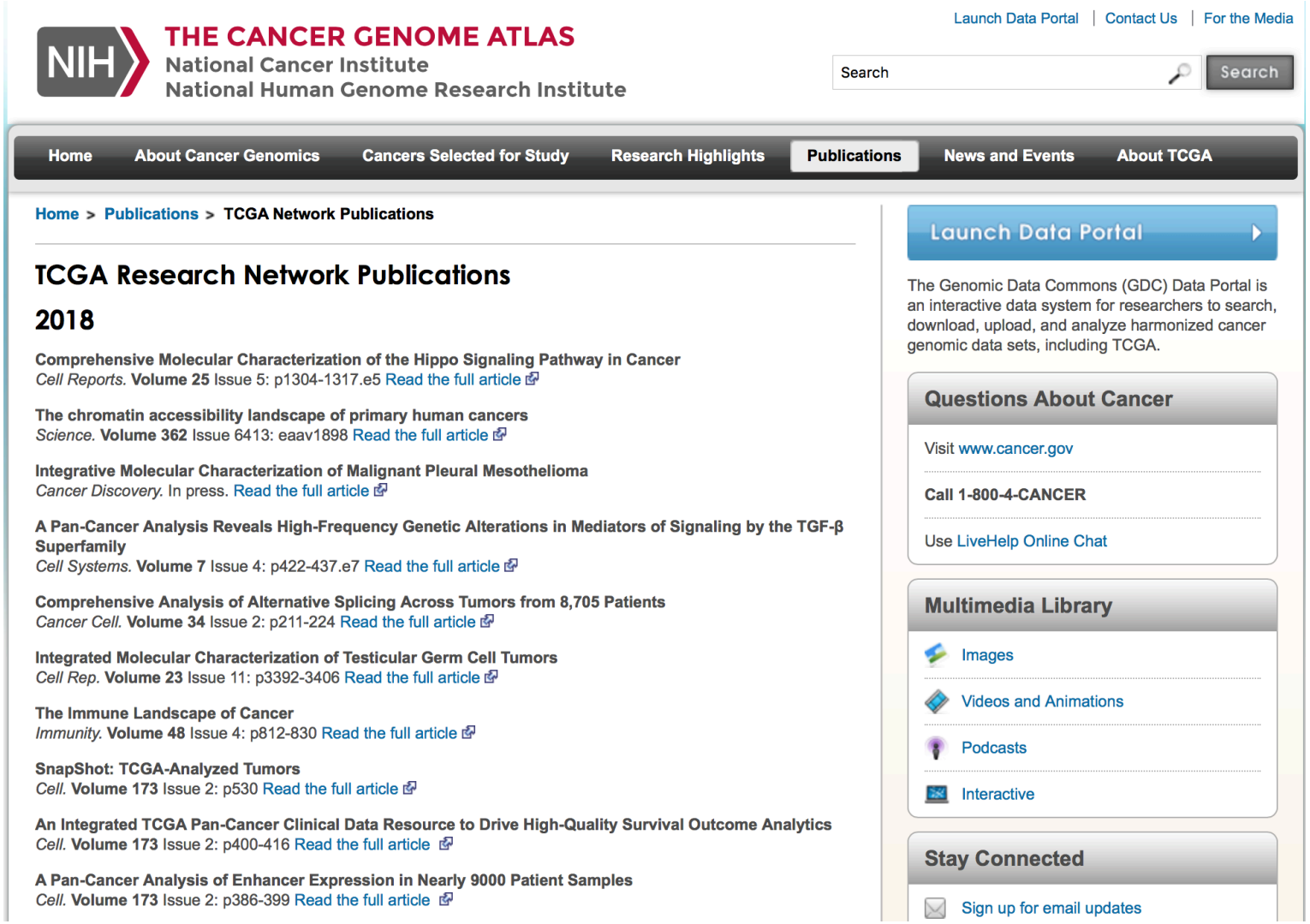
PROJECTS	PRIMARY SITES	CASES
43	69	33,096

FILES	GENES	MUTATIONS
358,092	22,872	3,142,246





To do



THE CANCER GENOME ATLAS
National Cancer Institute
National Human Genome Research Institute

Launch Data Portal | Contact Us | For the Media

Search Search

Home | About Cancer Genomics | Cancers Selected for Study | Research Highlights | Publications | News and Events | About TCGA

Home > Publications > TCGA Network Publications

TCGA Research Network Publications

2018

Comprehensive Molecular Characterization of the Hippo Signaling Pathway in Cancer
Cell Reports. **Volume 25** Issue 5: p1304-1317.e5 Read the full article

The chromatin accessibility landscape of primary human cancers
Science. **Volume 362** Issue 6413: eaav1898 Read the full article

Integrative Molecular Characterization of Malignant Pleural Mesothelioma
Cancer Discovery. In press. Read the full article

A Pan-Cancer Analysis Reveals High-Frequency Genetic Alterations in Mediators of Signaling by the TGF-β Superfamily
Cell Systems. **Volume 7** Issue 4: p422-437.e7 Read the full article

Comprehensive Analysis of Alternative Splicing Across Tumors from 8,705 Patients
Cancer Cell. **Volume 34** Issue 2: p211-224 Read the full article

Integrated Molecular Characterization of Testicular Germ Cell Tumors
Cell Rep. **Volume 23** Issue 11: p3392-3406 Read the full article

The Immune Landscape of Cancer
Immunity. **Volume 48** Issue 4: p812-830 Read the full article

SnapShot: TCGA-Analyzed Tumors
Cell. **Volume 173** Issue 2: p530 Read the full article

An Integrated TCGA Pan-Cancer Clinical Data Resource to Drive High-Quality Survival Outcome Analytics
Cell. **Volume 173** Issue 2: p400-416 Read the full article

A Pan-Cancer Analysis of Enhancer Expression in Nearly 9000 Patient Samples
Cell. **Volume 173** Issue 2: p386-399 Read the full article

Launch Data Portal

The Genomic Data Commons (GDC) Data Portal is an interactive data system for researchers to search, download, upload, and analyze harmonized cancer genomic data sets, including TCGA.

Questions About Cancer

Visit www.cancer.gov

Call 1-800-4-CANCER

Use LiveHelp Online Chat

Multimedia Library

Images
Videos and Animations
Podcasts
Interactive

Stay Connected

Sign up for email updates

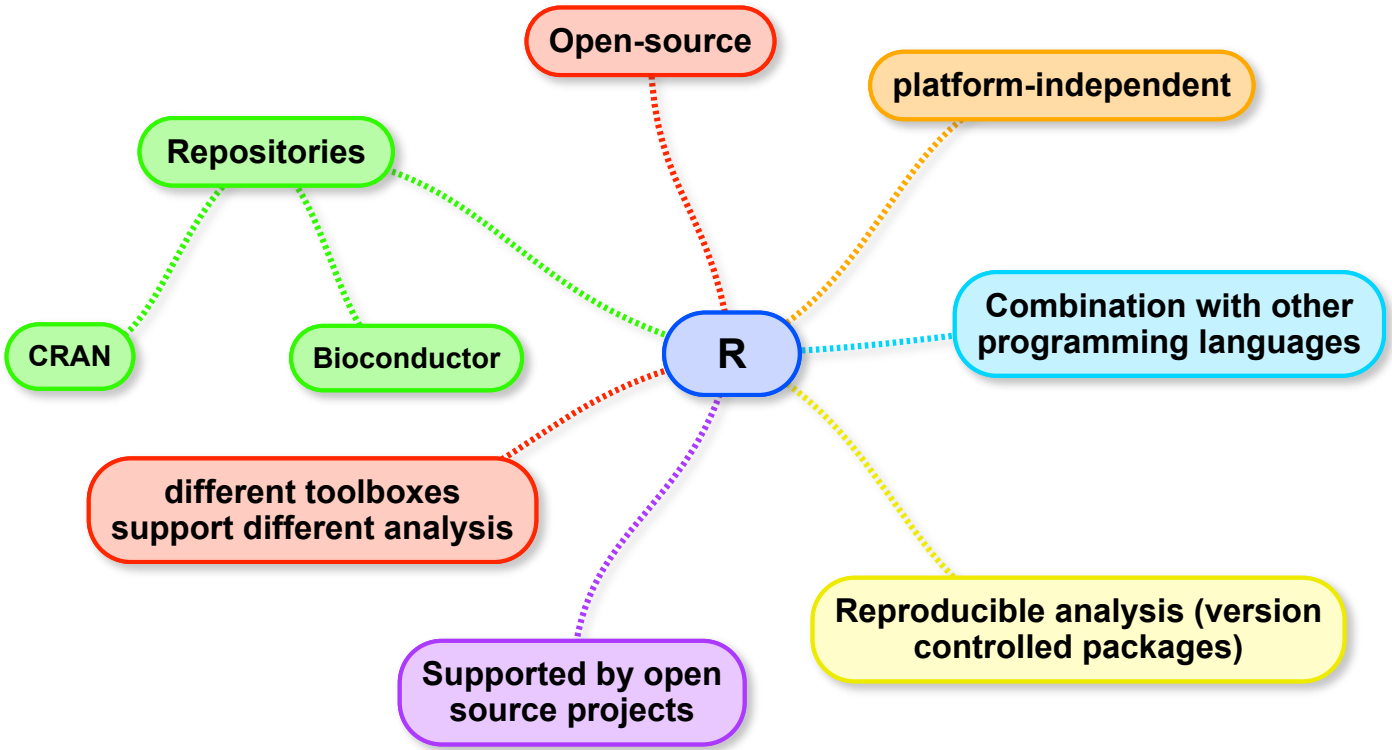
Lecture 1: Introduction to R

Introduction to R

- R is an environment in which you can perform statistical analysis and produce graphics.
- R can be used as a toolbox for standard statistical techniques.
- For advanced users, the main appeal of R is as a programming environment suited to data analysis.
- R is one of the most scripting programming languages especially in the area data science.
- R provides a command line interface as well as several graphical user interfaces (GUI) – e.g. Rstudio (you will use it in the tutorials)
- More information is available on the R project homepage:

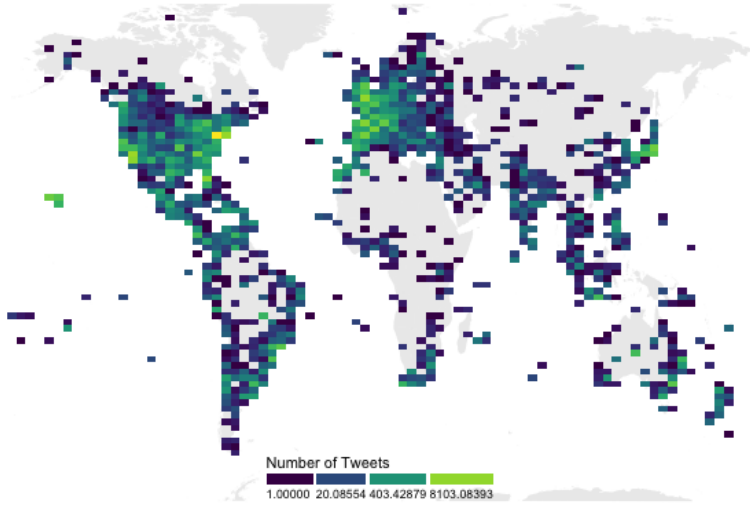
http://www.r-project.org

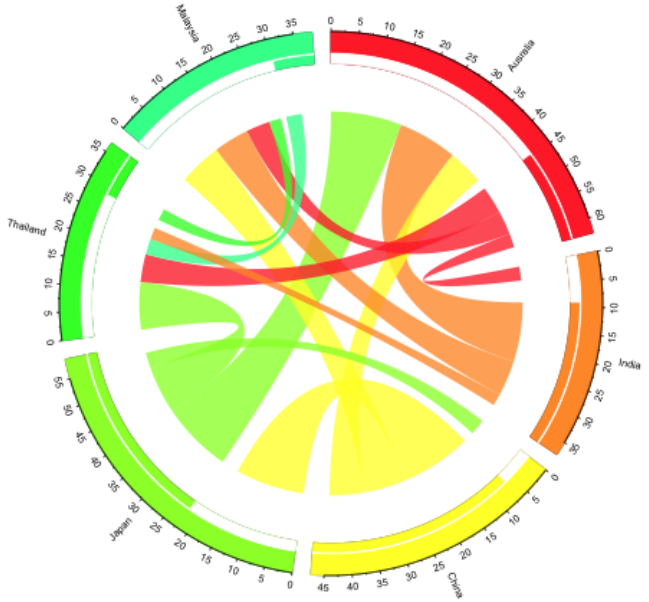
Introduction to R



Introduction to R

- Popular also among biologists
- Easy to learn
- There is a lot of introduction material , including tutorials, comprehensive documentation etc.
- Enriched programming language features (not only scripting)
- Production of high quality graphics for publication (e.g. www.r-graph-gallery.com)





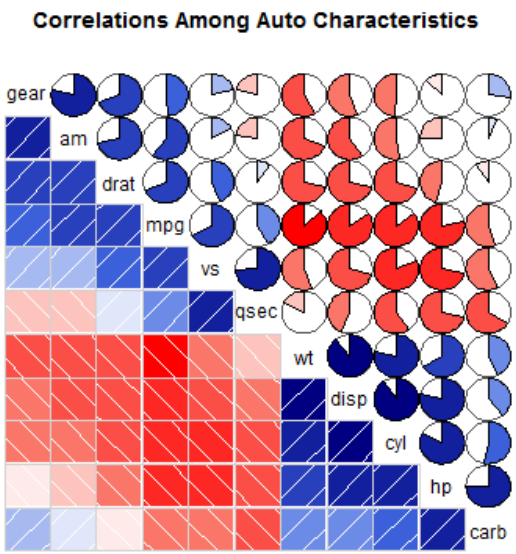
Why R? I

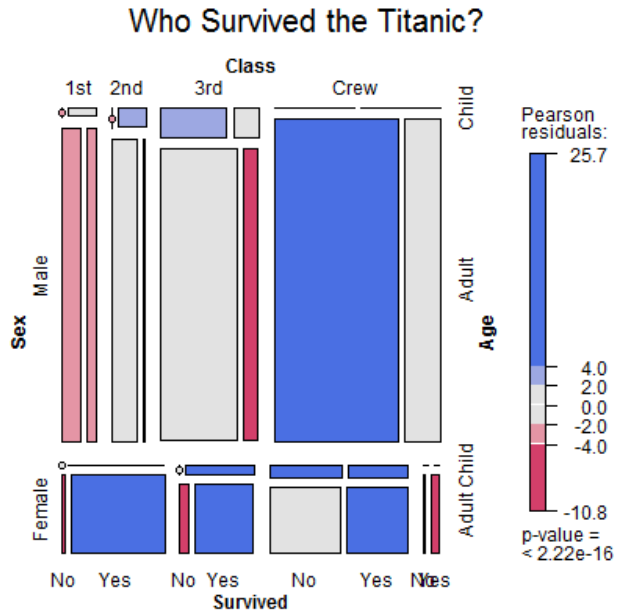
- R is free and available on any platform (Mac, PC, Linux, etc.) and also, via RStudio, in a web browser.
- R is powerful – you won't outgrow it.
- R helps you think about data in ways that are useful for statistical analysis.
- R promotes reproducible research.
  - R Commands provide an exact record of how an analysis was done and can be edited, rerun, commented, shared, etc.
- R is up-to-date.
  - Many new analysis methods appear first in R.
- There are many packages available that automate particular tasks. The CRAN (Comprehensive R Archive Network) repository contains thousands of *packages* that provide a wide range of additional capabilities (including many with biological applications.)

Why R? II

- The Bioconductor repository (`http://www.bioconductor.org/`) contains hundreds of additional packages that focus on biological and bioinformatics applications.
- R can be combined with other tools. R can be used within programming languages (like Python) or in scripting environments to automate data analysis pipelines.
- R is popular – including among biologists.
- R provides a gentle introduction to general informatics.
  - Although R can be used "one command at a time", R is a full featured programming language.
- Although R is very comprehensive, a small palette of useful things can be learned fairly easily.
- R produces publication quality graphics.

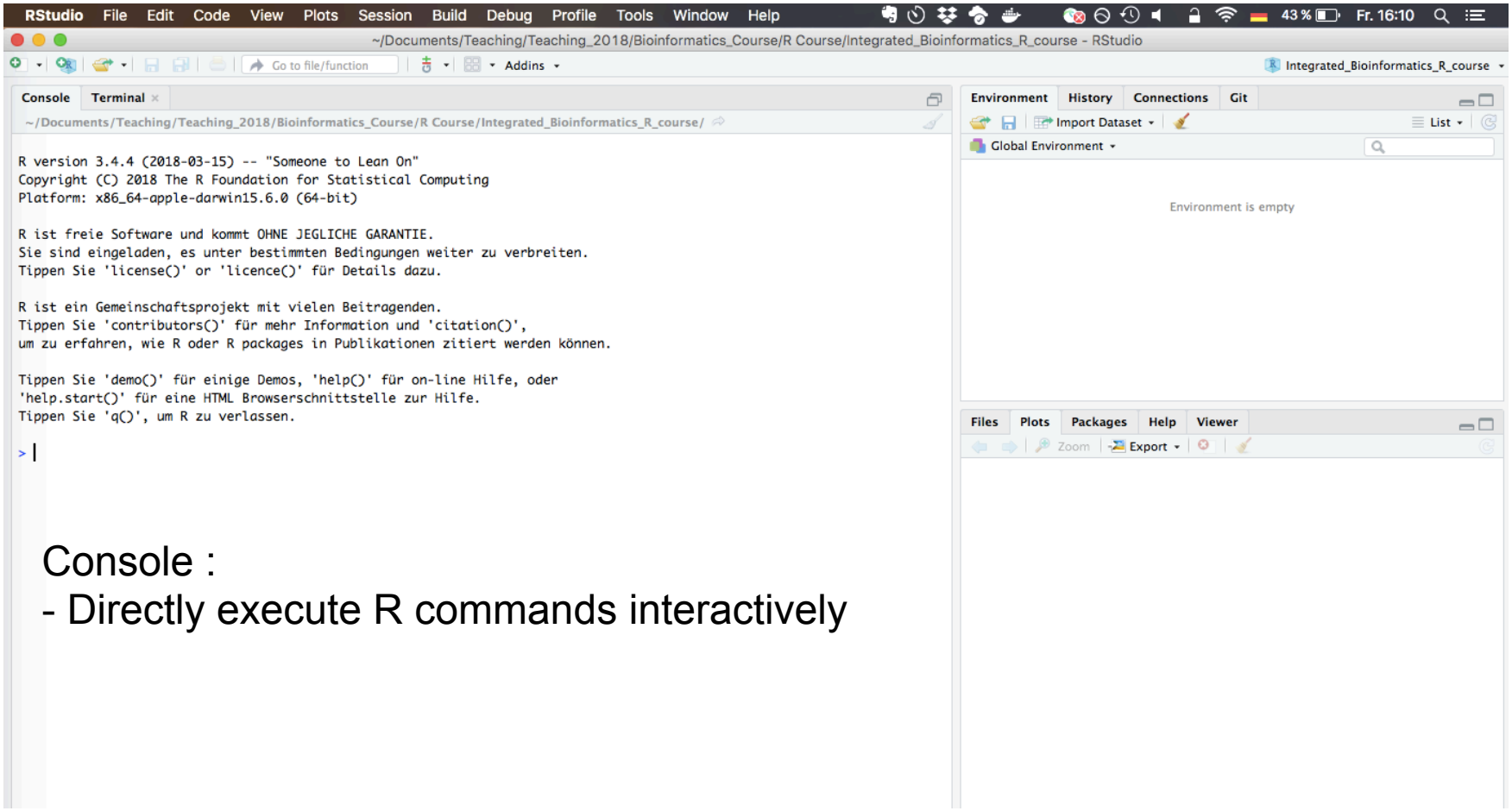
Why R? III





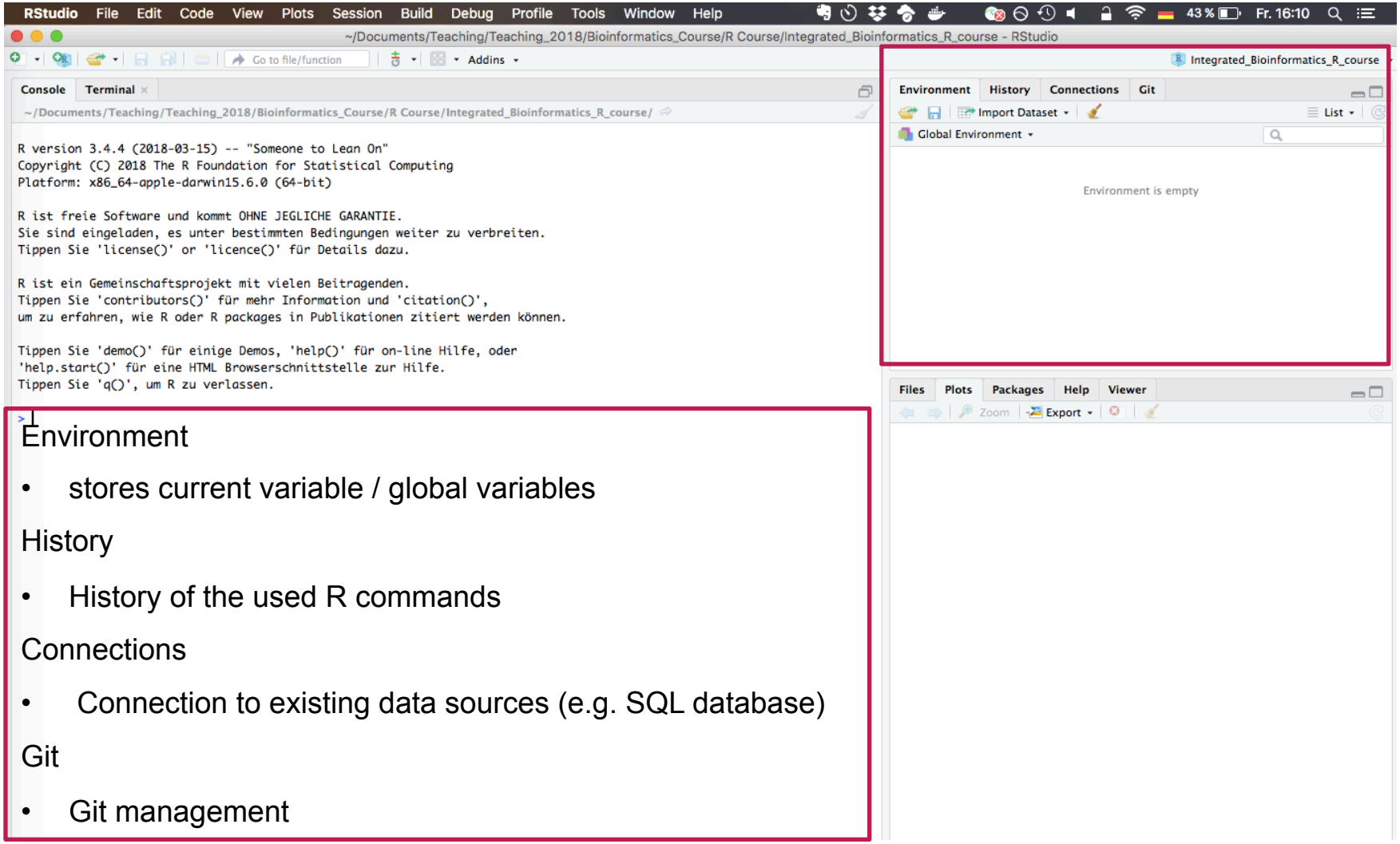
Source: *http://www.statmethods.net/*

RStudio



Console :
- Directly execute R commands interactively

RStudio



Environment

- stores current variable / global variables

History

- History of the used R commands

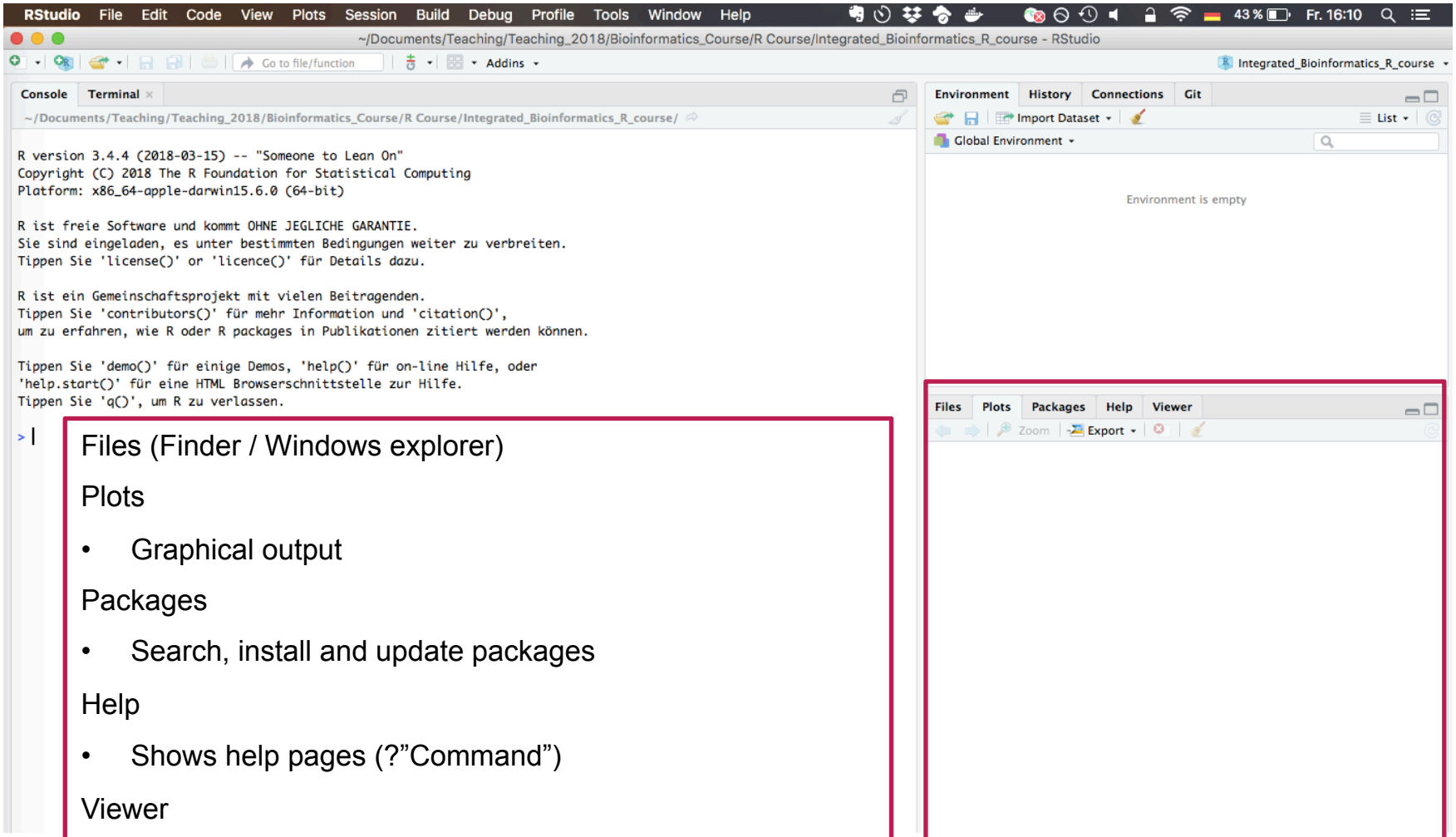
Connections

- Connection to existing data sources (e.g. SQL database)

Git

- Git management

RStudio



Files (Finder / Windows explorer)

Plots

- Graphical output

Packages

- Search, install and update packages

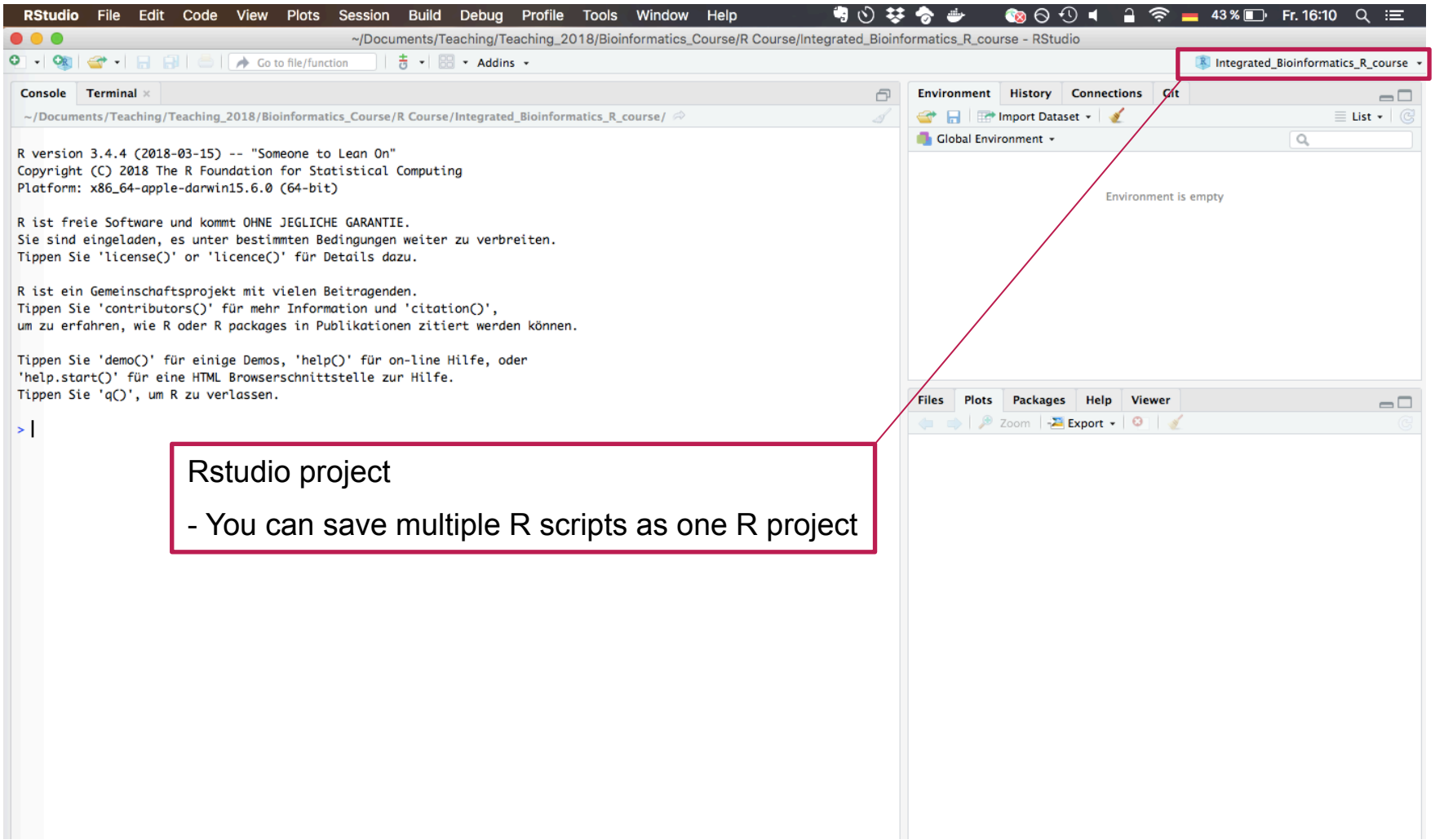
Help

- Shows help pages (?"Command")

Viewer

- Shows your data: View("yourdata")

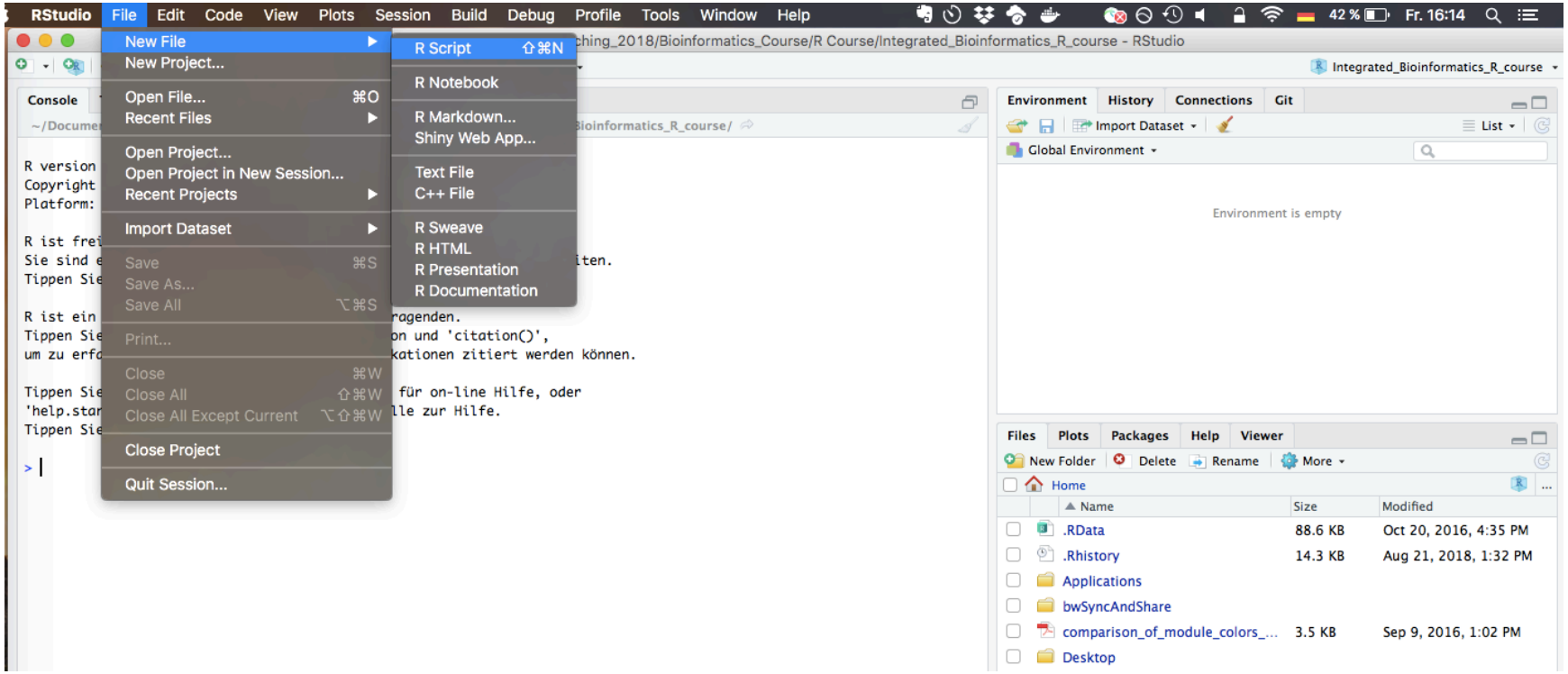
RStudio



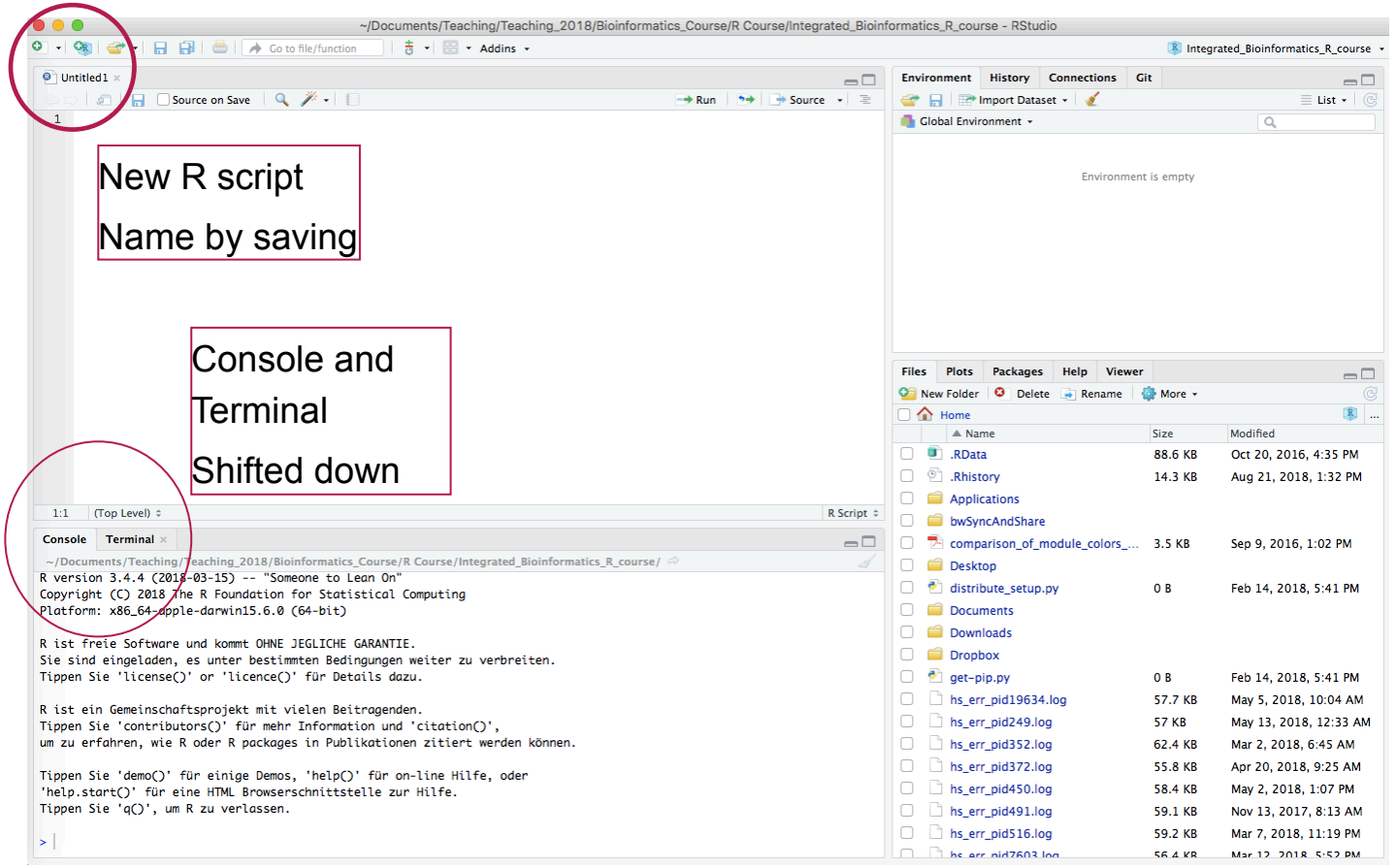
Rstudio project

- You can save multiple R scripts as one R project

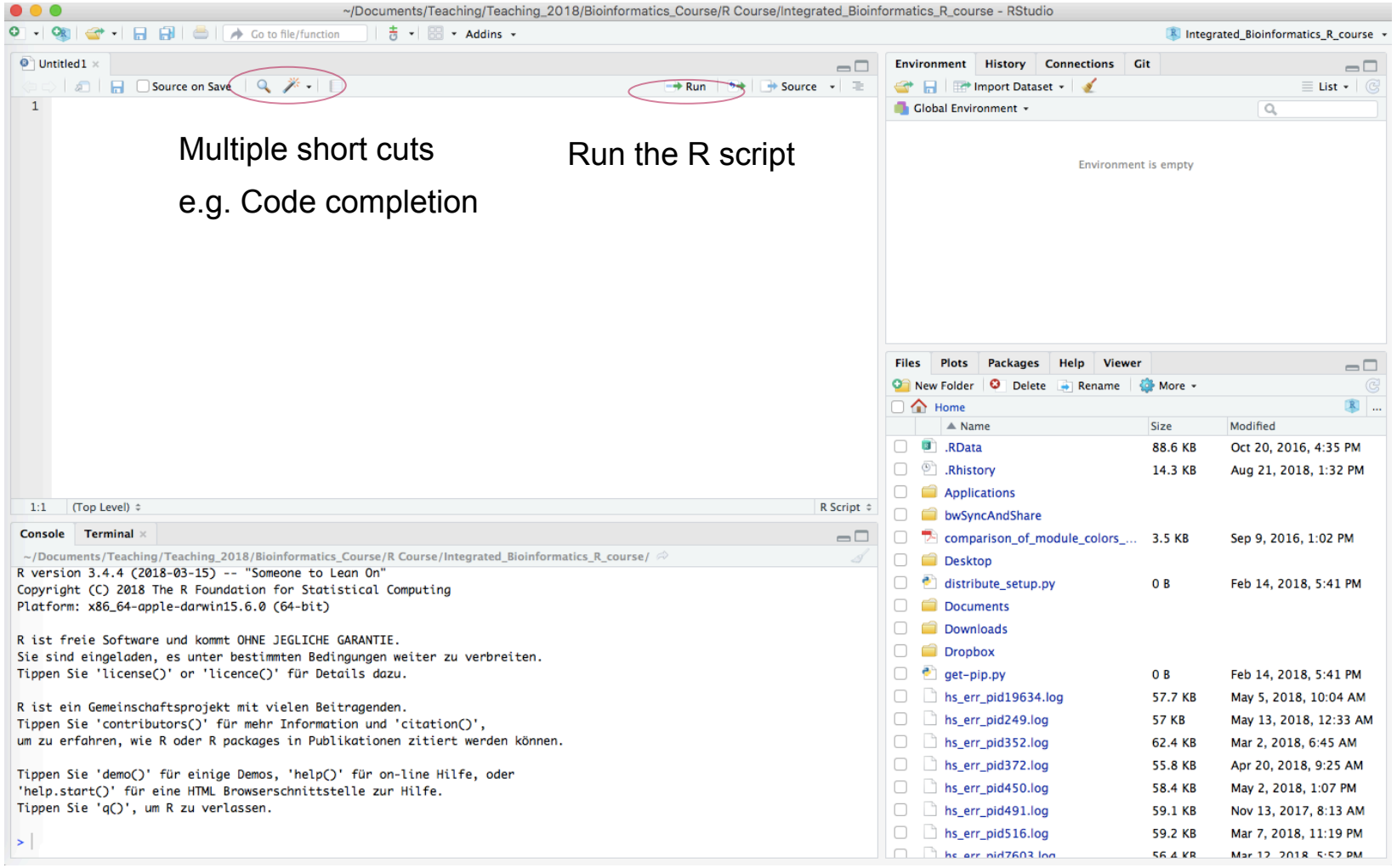
Making a new R Script



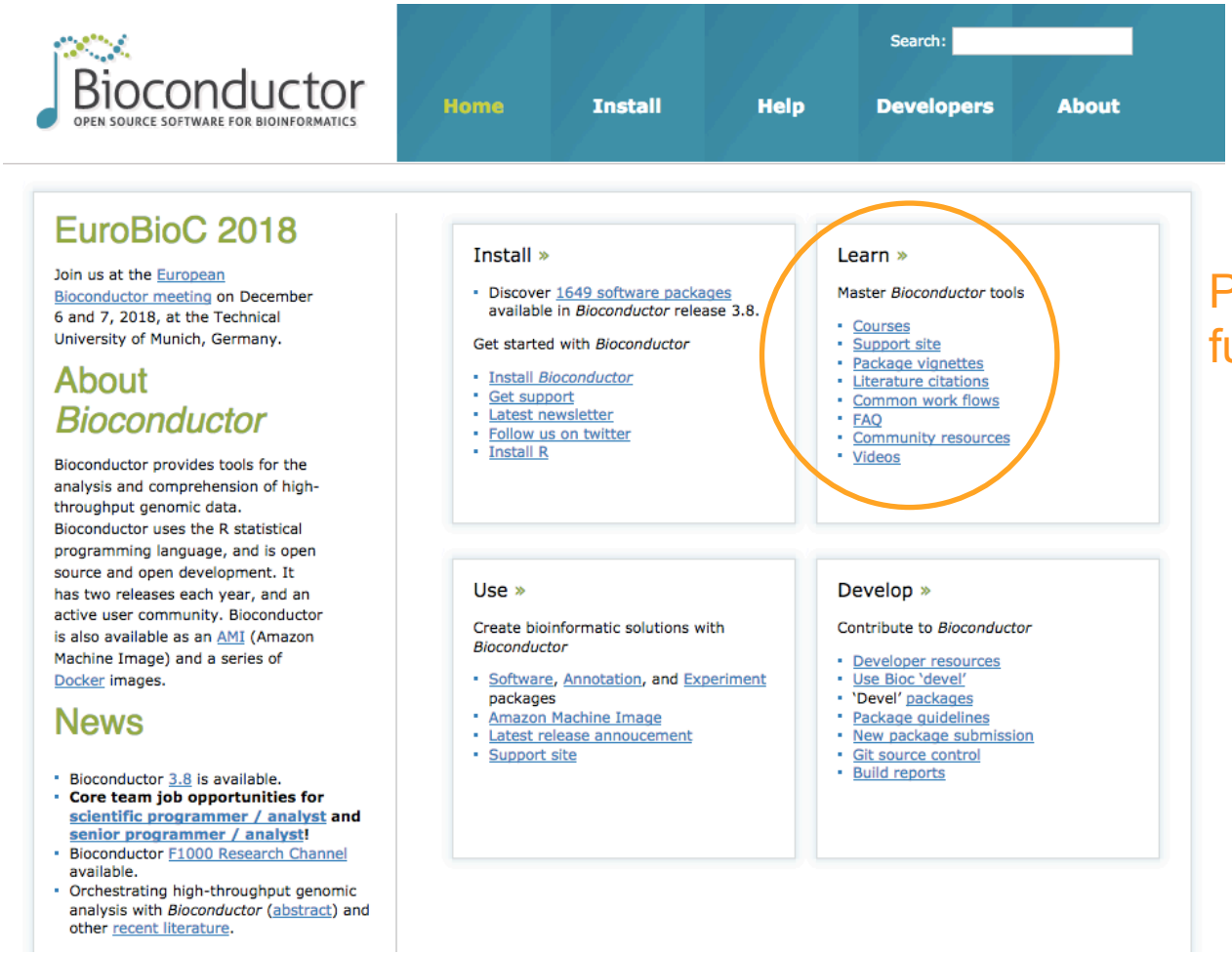
RStudio with R Script



RStudio – Run R Script



Bioconductor – Collection of R packages for bioinformatics



Provide Tutorial / Links to further learning material

Installing bioconductor

Type : sessionInfo()
In RStudio Console

- If R version <3.5.0

```
source("https://bioconductor.org/biocLite.R")
```

Install specific packages, e.g., "GenomicFeatures" and "AnnotationDbi", with

```
BiocInstaller::biocLite(c("GenomicFeatures", "AnnotationDbi"))
```

- R version 3.5.0 or higher

```
if (!requireNamespace("BiocManager"))
    install.packages("BiocManager")
BiocManager::install()
```

Update your R version to the newest one :
https://www.r-project.org/

```
BiocManager::install(c("GenomicFeatures", "AnnotationDbi"))
```

Version control – also with packages

Update Installed *Bioconductor* Packages

Bioconductor packages, especially those in the development branch, are updated fr
identify packages requiring update within your version of *Bioconductor*, start
enter

```
BiocManager::install()
```

Use the argument ask=FALSE to update
?install for additional details.

Upgrading inst

Some one version of *Bioconductor*. To use the latest version of
R, enter

```
Namespace("BiocManager"))
tall.packages("BiocManager")
ocManager::install()
```

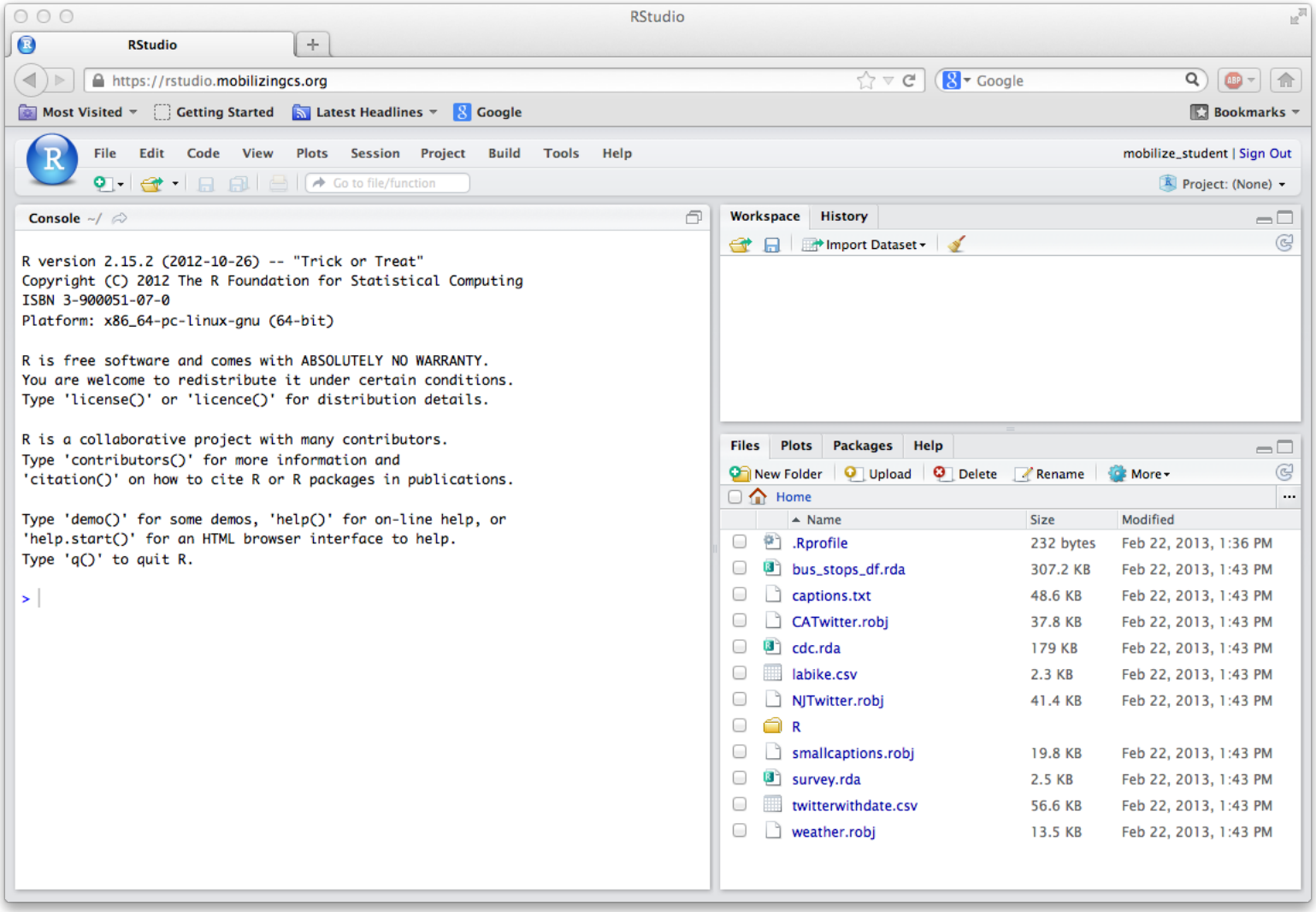
Remember that more recent versions of *Bioconductor* may be available if your version of R is out-of-date.

For more details on *Bioconductor* approaches to versioning, see the newsletter and version numbering developer reference.

This may be later important if you want to reproduce research with specific versions of R and of the corresponding packages

YOUR TURN
START WITH THE HANDOUT

RStudio I


RStudio
RStudio
https://rstudio.mobilizingcs.org
Google
Most Visited
Getting Started
Latest Headlines
Google
Bookmarks
File Edit Code View Plots Session Project Build Tools Help
mobilize_student | Sign Out
Go to file/function
Project: (None)
Console ~/
R version 2.15.2 (2012-10-26) -- "Trick or Treat"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-linux-gnu (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.
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.
>
Workspace History
Import Dataset
Files Plots Packages Help
New Folder Upload Delete Rename More
Home
Name Size Modified
.Rprofile 232 bytes Feb 22, 2013, 1:36 PM
bus_stops_df.rda 307.2 KB Feb 22, 2013, 1:43 PM
captions.txt 48.6 KB Feb 22, 2013, 1:43 PM
CATwitter.robj 37.8 KB Feb 22, 2013, 1:43 PM
cdc.rda 179 KB Feb 22, 2013, 1:43 PM
labike.csv 2.3 KB Feb 22, 2013, 1:43 PM
NJTwitter.robj 41.4 KB Feb 22, 2013, 1:43 PM
R
smallcaptions.robj 19.8 KB Feb 22, 2013, 1:43 PM
survey.rda 2.5 KB Feb 22, 2013, 1:43 PM
twitterwithdate.csv 56.6 KB Feb 22, 2013, 1:43 PM
weather.robj 13.5 KB Feb 22, 2013, 1:43 PM

RStudio II

- RStudio is organized into four panes, some with multiple tabs. Which tabs are in which panels is configurable. Some of the important tabs include:
  - Console: This is where you can execute `R` commands interactively
  - Workspace: A listing of the objects available in your `R` session
  - History: A record of past commands (can be saved, reloaded, etc.)
  - Files: A file manager for locating, loading, moving, renaming, files.
  - Plots: Where plots show up
  - Packages: Install and load packages here.
  - Help: Where documentation files appear when you ask for them
  - Open Files: Open files have a tab labeled with the file name.

A Fancy Calculator

```
2+3+1023 #addition
```

```
## [1] 1028
```

```
1-1 #substraction
```

```
## [1] 0
```

```
43 * 67 * 2 #multiplication
```

```
## [1] 5762
```

```
33/699 #division
```

```
## [1] 0.0472103
```

```
5%%3   #modulo (remainer)
```

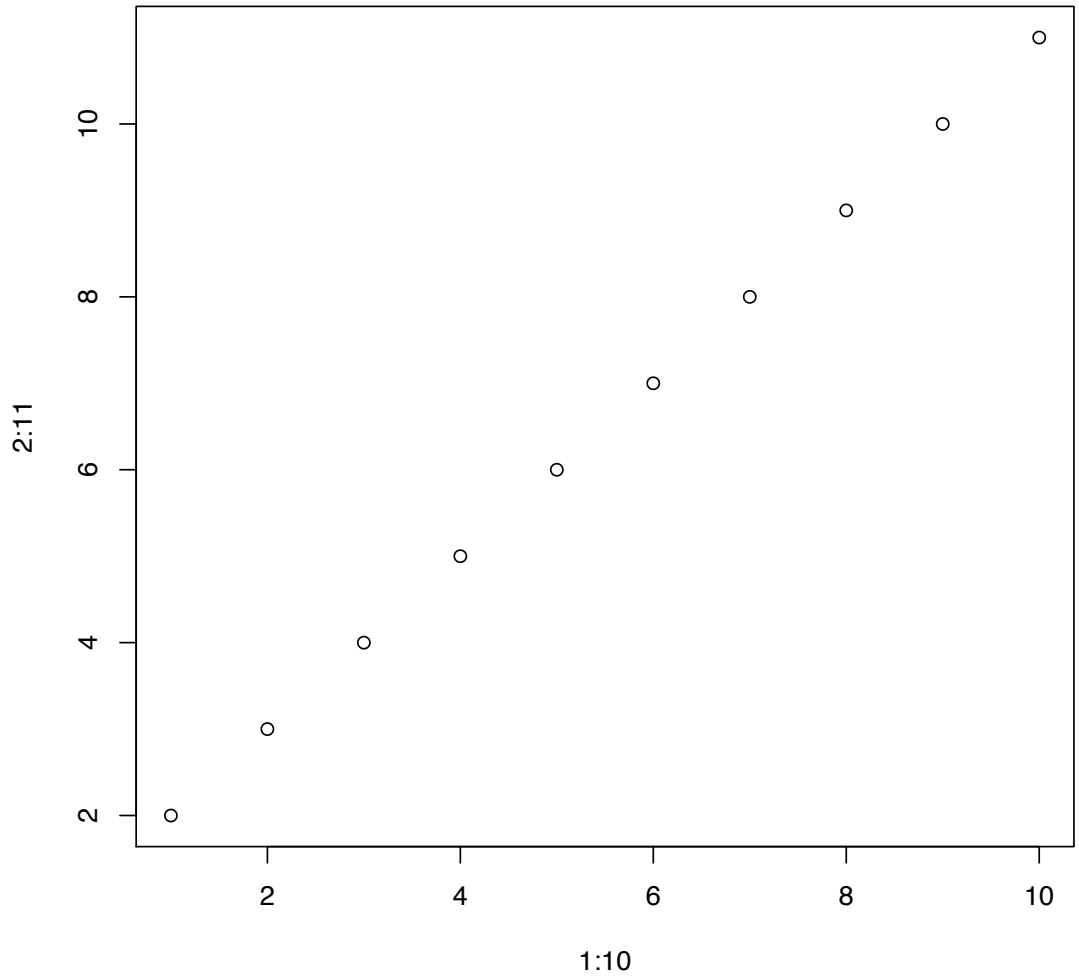
```
## [1] 2
```

```
3^4 #power
```

```
## [1] 81
```

A Powerful Tool for Graphics

```
plot(1:10, 2:11)
```



Functions I

Definition (Function)

A **function** is a set of instructions that perform a specific task.

- Functions in `R` work like they do in mathematics: they specify a transformation from one or more inputs (*arguments* or *parameters*) to one or more outputs (or *return* values).
- A function is *called* by writing its names, followed by parentheses, with any arguments going inside the parentheses.

Functions II

```
log(17.44) #natural log
## [1] 2.858766
exp(2.33)  #exponentiation
## [1] 10.27794
sin(2*pi)  #sine
## [1] -2.449294e-16
sqrt(9) # square root
## [1] 3
sqrt(9, 1) # too many arguments


## Error in sqrt(9, 1):  2 arguments passed to 'sqrt' which requires 1

sum(3,5) # sum
## [1] 8
```

- The `print` function prints its argument and returns it *invisibly*.

```
print("This is R!")

## [1] "This is R!"
```

Getting Around in R I

- You can operate R entirely from the command line, entering text in interactive mode.
- There are a few basic commands that will help you navigate your workspace.
  - This command will show you the path to the current folder:

```
getwd()
```

```
## [1] "/media/taherl/data/Dropbox/Work/Teaching/RostockMB/IntroR"
```

  - If you type getwd without the parentheses you will see the actual definition of the *function* getwd in R:

```
getwd
```

```
## function ()
## .Internal(getwd())
## <bytecode: 0x2f82860>
## <environment: namespace:base>
```

  - R is case-sensitive:

```
Getwd
```

```
## Error in eval(expr, envir, enclos):  object 'Getwd' not found
```

Getting Around in R II

- You can change the working directory by typing

```
setwd("pathname")
```

- Within R, lots of Unix conventions are used, so paths are specified with a single forward-slash separator, even on Windows systems. For example,

```
setwd("C:/Documents and Settings/username/My Documents")
```

would be used to point to the My Documents directory.

- You can list the objects in your workspace with `ls()`, and remove them with `rm(objectname)`.

Getting Help

- Ask!
- `help.start()` and the HTML help button in the Windows GUI.
- `help` and `?`: `help("objectname")` or `?help`.
- `help.search()`, `apropos()`
- `browseVignettes("package")`
- rseek.org
- use tab-completion in RStudio, this will also display help-snippets

Data Types

- There are the following elementary types ("modes"):
  - `numeric`: real numbers;

```
is.numeric(5)
```

```
## [1] TRUE
```

  - `character`: chain of characters, text;

```
is.numeric("November 2016")
```

```
## [1] FALSE
```

  - `logical`: `TRUE` or `FALSE`;

```
is.logical(is.numeric("November 2016"))
```

```
## [1] TRUE
```

  - `factor`: characters or numbers, describing certain categories;
  - special values: `NA` (missing value), `NULL` ("empty object"), `Inf`, `-Inf` (infinity), `NaN` (not a number).
- You can check the data type using `mode`:

```
mode(2 == 2)
```

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

```
mode("these are characters")
```

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

Comparison Operators

- Comparison operators return the logical values `TRUE` or `FALSE`:
  - equal (==)
  - not equal (!=)
  - greater than (>)/less than (<)
  - greater than or equal (>=)/less than or equal (<=)

```
1 == 1
```

```
## [1] TRUE
```

```
2 != 3
```

```
## [1] TRUE
```

```
2 > 3
```

```
## [1] FALSE
```

```
4 <= 3
```

```
## [1] FALSE
```

Logical Operators

- AND: `&`, returns `TRUE` if both comparisons return `TRUE`.

```
2 > 1 & 6 > 5
```

```
## [1] TRUE
```

- OR: `|`, returns `TRUE` if at least one comparison returns `TRUE`.

```
2 == 2 | 2 != 3
```

```
## [1] TRUE
```

- NOT: `!`, returns the negation (opposite) of a logical vector.

```
!(2 > 1)
```

```
## [1] FALSE
```

Variables I

Definition (Variable)

A **variable** is a symbolic name to which a value may be associated. The associated value may be changed.

- Variable names
  - cannot start with a digit;
  - are case-sensitive;
  - some are already used by `R`, e.g., `c`, `q`, `t`, `C`, `D`, `F`, `I`, `T`, and should be avoided.

Variables II

- Giving a new value to a variable is called **assignment**:

```
weight<-15
weight

## [1] 15

weight<-10*weight+2
weight

## [1] 152

weight="fifteen"
weight

## [1] "fifteen"
```

Vectors I

Definition (Vector)

A **vector** is an ordered group of elements of the same type with a single dimension.

- The elementary data structures in R are vectors.
- The c(...) construct can be used to create vectors:

```
weight <- c(60, 72, 57, 90, 95, 72)
weight

## [1] 60 72 57 90 95 72
```

- To access vector elements we use an *index* inside a single square bracket ("[]"):

Vectors II

```
weight
```

```
## [1] 60 72 57 90 95 72
```

```
weight[1]
```

```
## [1] 60
```

```
weight[c(2,4,6)]
```

```
## [1] 72 90 72
```

```
weight[10]
```

```
## [1] NA
```

```
weight[c(-1,-2,-3)]
```

```
## [1] 90 95 72
```

Vectors III

- To generate a vector of regularly spaced numbers, use

```
seq(0, 1, length=11)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
1:10
```

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

```
weight[1:3]
```

```
## [1] 60 72 57
```

- Alternatively, you can use the `rep` function:

```
weight <- rep(1:13, 2)
weight
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13  1  2  3  4  5  6  7  8  9 10
## [24] 11 12 13
```

```
weight <- rep(1:13, each=2)
weight
```

```
##  [1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10 11 11 12
## [24] 12 13 13
```

Vectors IV

- Common arithmetic operations (including `+`, `-`, `*`, `/`, `^`) and mathematical functions (e.g. `sin`, `cos`, `log`) work *element-wise* on vectors, and produce another vector:

```
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
height^2

## [1] 3.0625 3.2400 2.7225 3.6100 3.0276 3.6481

height+0.5

## [1] 2.25 2.30 2.15 2.40 2.24 2.41
```

- When two vectors are not of equal length, the shorter one is *recycled*:

Vectors V

```
weight
```

```
##  [1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10 11 11 12
## [24] 12 13 13
```

```
height
```

```
## [1] 1.75 1.80 1.65 1.90 1.74 1.91
```

```
bmi <- weight/height^2
```

```
## Warning in weight/height^2:  longer object length is not a multiple of shorter object length
```

```
bmi
```

```
##  [1] 0.3265306 0.3086420 0.7346189 0.5540166 0.9908839 0.8223459 1.3061224
##  [8] 1.2345679 1.8365473 1.3850416 1.9817677 1.6446918 2.2857143 2.1604938
## [15] 2.9384757 2.2160665 2.9726516 2.4670376 3.2653061 3.0864198 4.0404040
## [22] 3.0470914 3.9635355 3.2893835 4.2448980 4.0123457
```

- A few particularly useful functions are
  - `length`, which returns the length of a vector (i.e. the number of elements it contains);

```
length(height)
```

```
## [1] 6
```

Vectors VI

- `sum`, which calculates the sum of the elements of a vector;

```
sum(height)
```

```
## [1] 10.75
```

- `unique`, which returns a vector with duplicate elements removed;

```
unique(c(5, 1, 2, 1, 3, 1, 4, 5))
```

```
## [1] 5 1 2 3 4
```

- `which`, which returns the locations where a logical vector is `TRUE`. This can be useful for switching from logical indexing to integer indexing:

```
which(height > 1.7)
```

```
## [1] 1 2 4 5 6
```

- simple statistics:
  - `mean`

```
mean(height)
```

```
## [1] 1.791667
```

  - `var`

```
var(height)
```

```
## [1] 0.01005667
```

Vectors VII

- `summary`

```
summary(height)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.650   1.742   1.775   1.792   1.875   1.910
```

- `min, max`

```
min(height)
```

```
## [1] 1.65
```

```
max(height)
```

```
## [1] 1.91
```

- `which.min` and `which.max` are more efficient shortcuts for `which(min(x))` and `which(max(x))`, respectively:

```
height
```

```
## [1] 1.75 1.80 1.65 1.90 1.74 1.91
```

```
which.min(height)
```

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

```
which.max(height)
```

```
## [1] 6
```

- `sqrt`, `sin`, ...

Converting between Data Types

- There are some functions to convert vectors between types:

```
v = c(1, 0, 1)
as.character(v)

## [1] "1" "0" "1"

as.logical(v)

## [1]  TRUE FALSE  TRUE

v = c("1", "0", "100")
as.numeric(v)

## [1]   1   0 100
```

Sorting I

- There are some useful functions to order and sort vectors:
  - `sort`: sort in increasing order;

```
height
```

```
## [1] 1.75 1.80 1.65 1.90 1.74 1.91
```

```
sort(height)
```

```
## [1] 1.65 1.74 1.75 1.80 1.90 1.91
```

  - `order`: orders the indexes of a vector so that the elements of the vector are sorted;

```
order(height)
```

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

Sourcing a Script

- You can tell `R` to execute several statements one after the other without waiting for additional instructions using the `source()` function.
- To prepare your *script* to be sourced, you first have to write the entire script in a text editor. In RStudio, the editor window is in the top-left corner of the screen.
- Sourced scripts behave differently from interactive code in printing results. In interactive mode, a result is printed even without a `print()` function. But when you source a script, output is printed only if you have an explicit `print` function.

Exercise

- The nth triangular number is given by $\frac{n(n+1)}{2}$. Create a sequence of the first 20 triangular numbers.

Matrices I

Definition (Matrix)

A **matrix** is a multidimensional collection of data entries of the same type. Matrices have two dimensions. It has rownames and colnames.

- A matrix with m rows and n columns is said to be of order $m \times n$; (m, n) is the **dimension** of the matrix.
- In `R`, matrices can be created in several ways:
  - with the function `matrix`:

```
A <- matrix(nrow=2, ncol=3, data=c(1,2,3,4,5,6))
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
dim(A)
```

```
## [1] 2 3
```

Matrices II

- with the functions `cbind` and `rbind` add columns and rows to an existing matrix, or to another vector:

```
A <- rbind(c(1,3,5), c(2,4,8))
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    8
```

- You can set the names of columns and rows as follows:

```
rownames(A) <- c("x","y")
colnames(A) <- c("a1","b1","b2")
A
```

```
##   a1 b1 b2
## x  1  3  5
## y  2  4  8
```

- By default, `R` fills a matrix column-wise. This can be overruled using the parameter `byrow`:

```
A <- matrix(nrow=4, ncol=3, byrow=TRUE, data=1:12)
```

Matrices III

- To transpose a matrix use the `t()` function:

```
t(A)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

- Matrix multiplication is done using %*%:

```
A <- matrix(nrow=2,ncol=3,data=6:1)
A
```

```
##      [,1] [,2] [,3]
## [1,]    6    4    2
## [2,]    5    3    1
```

```
B <- matrix(nrow=3,ncol=4,data=1:12)
B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
A%*%B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   20   56   92  128
## [2,]   14   41   68   95
```

Lists I

Definition (List)

A list is a special type of vector. Each element can be a different type.

- Lists can be created using `list()`.
- Specific elements of a list can be accessed by writing the name of the list and then the `$` character, followed by the name of the element (if any), or the double square bracket `"[[]]"`.

Lists II

```
example1 <- list(foo="a character", bar=1)
example1

## $foo
## [1] "a character"
##
## $bar
## [1] 1

example1$foo

## [1] "a character"

is.numeric(example1$bar)

## [1] TRUE

example1[["foo"]]

## [1] "a character"

example1[[1]]

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

Lists III

- Lists can also be created by coercing other objects using `as.list()`.

```
example2 <- as.list(c(1, "a character", TRUE, 1 + (0+4i)));
example2

## [[1]]
## [1] "1"
##
## [[2]]
## [1] "a character"
##
## [[3]]
## [1] "TRUE"
##
## [[4]]
## [1] "1+4i"

example2[[1]]

## [1] "1"
```

Data Frames I

Definition (Data Frame)

A **data frame** is essentially a matrix where the columns can have different data types.

```
age=c(2, 3, 5)
gender = c("female", "male", "female")
wt=c(TRUE, FALSE, TRUE)
df=data.frame(age, gender, wt)
df
```

```
##   age gender    wt
## 1   2 female  TRUE
## 2   3   male FALSE
## 3   5 female  TRUE
```

- Columns are named and can be accessed by their name.
- If rows are named, they can be accessed by their names.

Data Frames II

```
str(df)
```

```
## 'data.frame': 3 obs. of  3 variables:
##  $ age   : num  2 3 5
##  $ gender: Factor w/ 2 levels "female","male": 1 2 1
##  $ wt    : logi  TRUE FALSE TRUE
```

```
colnames(df)
```

```
## [1] "age"    "gender" "wt"
```

```
# equivalent
names(df)
```

```
## [1] "age"    "gender" "wt"
```

```
rownames(df)
```

```
## [1] "1" "2" "3"
```

Data Import I

- `read.table` is the Swiss army knife of file importing in `R`, and can handle any kind of delimited file.

```
importdata <- read.table("myfile.txt")
```

- `R` assumes that the file is a tab- or space-delimited file that has variable names in the header row and a first line with a length one shorter than subsequent lines.
- We can adjust for any unique characteristics of our data. For example, to tell `R` to read a data file with a header row and semi-colon-separated data:

```
importdata<-read.table("myfile.txt", header=TRUE, sep=";", row.names="id", na.strings="..", stringsAsFac
```

- There are also functions `read.csv`, `read.delim`, `read.csv2`.

Data Import II

- When creating a `data.frame` strings are automatically coded as factors.

```
patients <- read.csv("http://www-huber.embl.de/users/klaus/BasicR/Patients.csv")
head(patients)

##   PatientId Height Weight Gender
## 1        P1   1.65     75      f
## 2        P2   1.90     NA      m
## 3        P3   1.60     50      f

patients$Gender

## [1] f m f
## Levels: f m
```

  - In order to turn it off, set `options(stringsAsFactors=FALSE)` globally or to specify this explicitly in the creation of the data frame as in:

```
patients <- read.csv("http://www-huber.embl.de/users/klaus/BasicR/Patients.csv",
stringsAsFactors=FALSE)
### not a factor
patients$Gender

## [1] "f" "m" "f"
```

Conditional Execution I

Definition (Conditional statement)

Conditional statements are statements that are only performed if certain conditions are met.

```
if ( test_expression ) {
statement1
}
else {
statement2
}
```

- `test_expression` can be a logical or numeric vector, but only the first element is taken into consideration.
  - In the case of numeric vector, zero is taken as `FALSE`, rest as **TRUE**.
  - If `test_expression` is `TRUE`, the statement is executed.
  - If `test_expression` is `FALSE`, the statement is **not** executed.

Loops I

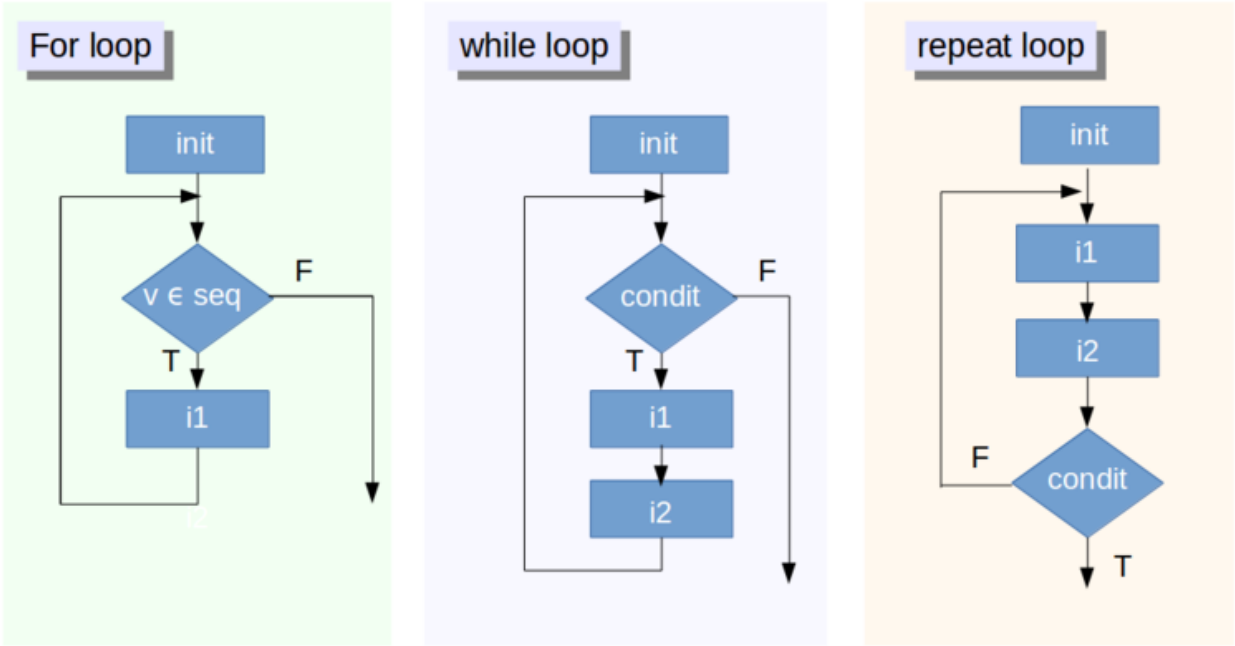
Definition (Loop)

A **loop** is a sequence of instructions that are continually repeated until a certain condition is reached.

- "Looping", "cycling", "iterating" or just replicating instructions is an old practice that originated well before the invention of computers.
  - It is nothing more than automating a multi-step process by organizing sequences of actions or "batch" processes and by grouping the parts that need to be repeated.
- All modern programming languages provide special constructs that allow for the repetition of instructions or blocks of instructions.
- There are two types of loops:
  (1) loops that execute instructions for a prescribed number of times, as controlled by a counter or an index, incremented at each iteration cycle; and

Loops II

(2) loops are based on the onset and verification of a logical condition. The condition is tested at the start or the end of the loop construct.



for Loops I

- A `for` loop iterates over a vector:

```
for (val in sequence){
        statement
}
```

- `sequence` is a vector and `val` takes on each of its value during the loop.
- The instructions to be repeated are contained within curly braces.
  - In each iteration, "statement" is evaluated.

```
a <- c()
for(i in 1:10) {
        a[i] <- i;
}
a
```

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

for Loops II

- for loops can also be nested:

```
# Create a 10x7 matrix
mymatrix <- matrix(nrow=10, ncol=7)
# For each row and each column, assign values based on the position of the rows and columns
for (i in 1:dim(mymatrix)[1]){
        for (j in 1:dim(mymatrix)[2]){
                mymatrix[i,j]=i*j
        }
}
# Just show the upper left 3x7 chunk
mymatrix[1:3, ]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    1    2    3    4    5    6    7
## [2,]    2    4    6    8   10   12   14
## [3,]    3    6    9   12   15   18   21
```

while Loops I

- The `while` loop is made of an initialization block followed by a logical condition.
  - This condition is an expression that evaluates to a logical value, `TRUE` or `FALSE`.
    - If the result is `FALSE`, the loop is not executed. The program will then execute the first instruction it finds `after` the loop block.
    - If the result is `TRUE`, the instruction or block of instructions are executed. The iterations stop once the condition evaluates to `FALSE`.
  - An instruction or block of instruction may update the control variable, altering the result of the condition at the start of the loop.

```
while (test_expression){
        statement
}
```

- `test_expression` is evaluated and the loop is entered if the result is `TRUE`.
- `statement` is executed and the flow returns to evaluate the `test_expression` again.

while Loops II

- This is repeated each time until `test_expression` evaluates to `FALSE`, in which case, the loop exits.

```
cnt <- 2
while (cnt < 7) {
        print("Hello, this is a while loop!")
        cnt <- cnt + 1
}

## [1] "Hello, this is a while loop!"
## [1] "Hello, this is a while loop!"
## [1] "Hello, this is a while loop!"
## [1] "Hello, this is a while loop!"
## [1] "Hello, this is a while loop!"
```

Exercise

Without using `max` or `min`, create an `R` script that returns the maximum value out of the elements of a numeric vector `x` of any length.

Plotting in base R I

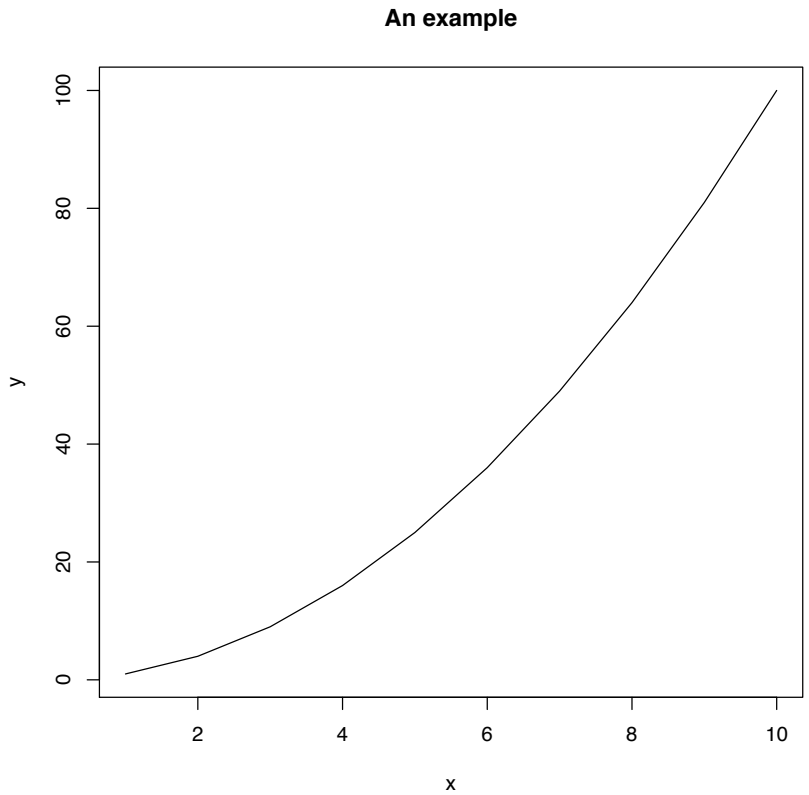
- The most basic plotting function is `plot`:

```
plot(x,y,type="l",main="my header", ...)
```

  - `x`: x-axis data;
  - `y`: y-axis data (may be missing);
  - `type=l,p,h,b`: lines, points, horizontal lines, etc;
  - `main`: plot heading; and
  - additional graphical parameters: see `?par` for more information.

Plotting in base R II

```
x <- 1:10
y <- x^2
plot(x, y, type="l", main="An example")
```



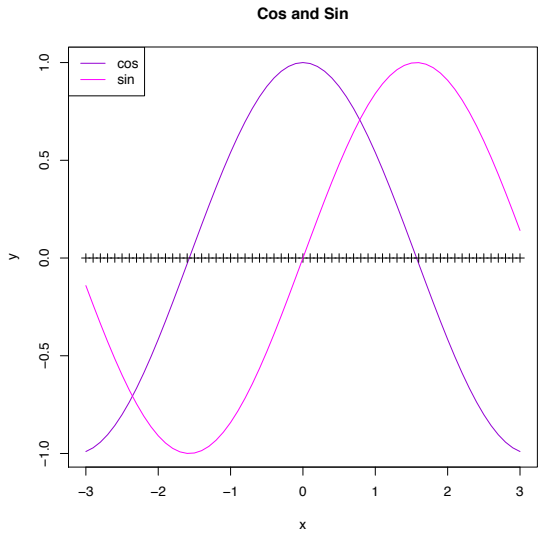
Plotting in base R III

- Additional lines, points and so on can be added with `lines` and `points`.
- The function `pdf` will open a pdf document as a "graphical device".
  - Subsequently, everything will be plotted to the same pdf file.
- `dev.off()` closes the graphical device. The pdf file becomes viewable.
- `dev.new()` opens a new graphical device.
  - This allows you to create a new plot without overwriting the current active one.
- With the graphical option `par( mfrow=c(<no.rows>, <no.columns>) )` you can produce an array of plots.

Plotting in base R IV

```
pdf(file="plot-example.pdf", width=12, height=6)
```

```
x <- seq(-3,3, by = 0.1);
y <- cos(x);
z <- sin(x)
plot(x,y, type="l", col="darkviolet", main="Cos and Sin")
points(x, rep(0, length(x)), pch=3)
lines(x,z, type="l", col="magenta")
legend("topleft", c("cos","sin"), col=c("darkviolet", "magenta"), lty=1)
```




```
dev.off()
```

Exercise

The file `weight_chart.txt` contains data for a growth chart for a typical baby over the first 9 months of its life.

- Read in the data using the `read.table` function.
- Use the `plot` function to draw this as a point an line graph with the following changes:
- the point character (`pch`) should be a filled square;
- the plot point size (`cex`) should be 1.5x normal size;
- the thickness of the line (`lwd`) should be 2 pixels;
- the x-axis title (`x.lab`) should be "Age [months]";
- the y-axis title (`y.lab`) should be "Weight [kg]";
- add suitable title (`main`) to the top of the plot.

Saving and Loading your Workspace

- `save(objectname)` saves a specific object from your current workspace.
- `save.image` saves the entire workspace. A complete workspace is usually saved with the extension `.RData`:

```
save.image("mydata.RData")
load("mydata.RData")
```

Packages I

- In order to use a package, you must first install it:

```
install.packages("DESeq2", dependencies=TRUE)
update.packages()
```

- R will ask us which mirror we want to use.
- The `dependencies=TRUE` option will check for other required packages and install those too.
- `update.packages()` indicates how to maintain the packages in your system. Empty parentheses will update all installed packages.
- R will automatically locate packages that have been officially accepted into CRAN.
  - For local packages, you have to specify an explicit path to where R can find the package.
- `library()` shows all the packages that have been installed on your system.

Packages II

  - Used with the name of a package as an argument, `library` will load the specific package into your current environment.
- `search()` shows which packages have been actively loaded in your current environment.

Packages III

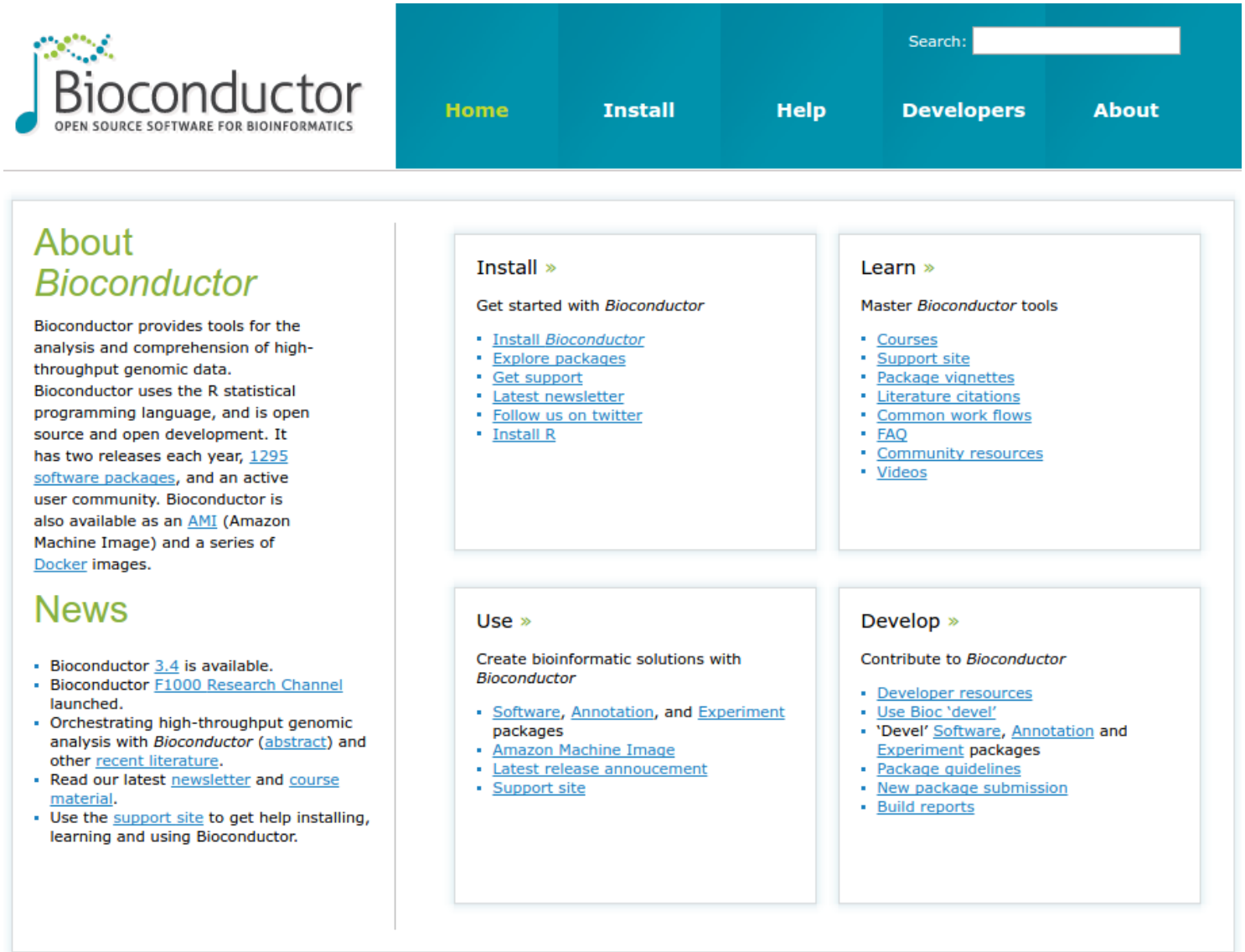
```
library()
```

```
## Warning in library():  library '/usr/local/lib/R/site-library' contains no packages
```

```
library("DESeq2")
```

```
## Loading required package:  S4Vectors
## Loading required package:  methods
## Loading required package:  stats4
## Loading required package:  BiocGenerics
## Loading required package:  parallel
##
## Attaching package:  'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##    clusterExport, clusterMap, parApply, parCapply, parLapply,
##    parLapplyLB, parRapply, parSapply, parSapplyLB
## The following objects are masked from 'package:stats':
##
##    IQR, mad, xtabs
## The following objects are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##    as.data.frame, as.vector, cbind, colnames, do.call,
##    duplicated, eval, evalq, get, grep, grepl, intersect,
##    is.unsorted, lapply, lengths, mapply, match, mget, order,
##    paste, pmax, pmax.int, pmin, pmin.int, rank, rbind, rownames,
##    sapply, setdiff, sort, table, tapply, union, unique, unlist,
##    unsplit
## Loading required package:  IRanges
## Loading required package:  GenomicRanges
## Loading required package:  GenomeInfoDb
```

The Bioconductor Packages



What is Bioconductor Good for?

- Microarrays
  - expression, copy number, SNPs, methylation, ...
- Sequencing
  - RNA-seq, ChIP-seq, called variants, ...
    - Especially *after* assembly / alignment
- Annotation
  - genes, pathways, gene models (exons, transcripts, etc.), ...
- Flow cytometry, proteomics, image analysis, high-throughput screens, ...

Installing Bioconductor

- To automatically download and install Bioconductor and its core packages, use the following script, which can be sourced directly:

```
source("http://www.bioconductor.org/biocLite.R")
biocLite()
```

- To install a specific package, e.g., "DESeq2", use:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("DESeq2")
```