# 01. Introduction to Statistical Programming and R

# What is Statistical Programming?

- There are three elements: statistics, algorithms and carrying out statistical analysis on a computer.
- In a typical study we begin by loading the data into the computer, and making an exploratory analysis of the data.
- We look at the raw numbers, and visualize them by plotting graphs that reveal patterns in the data. We may have a model we wish to fit to the data, and a hypothesis we wish to test.
- Statistics tells us what objects we need to calculate in order to make the fit and carry out the test.

# What is Statistical Programming?

- An algorithm tells us how to calculate those objects. It is a sequence of operations that carry out a given task. An algorithm is a mathematical object.
- Computer programming should carry out the calculations dictated by the algorithm. Two correct implementations of the same algorithm can differ.
- Statistical programming involves doing computations to aid in statistical analysis. Statistical programming is also closely related to other forms of numerical programming. It involves optimization and approximation of mathematical functions.

# Introduction to R

- A free software programming language and software environment for statistical computing and graphics
- Initially developed by R. Ihaka and R. Gentleman (1996)
- A GNU project that is freely available under the GNU General Public License
- Written primarily in C, Fortran, and R
- Pre-compiled binary versions are provided for many popular operating systems (Windows, Mac, and Linux)
- Contains tens of thousands of packages to analyze many different types of data.
- The official R home page is

`http://www.r-project.org`

# Introduction to R

- R source code is freely available.
- Anyone can re-use, modify and distribute the code
- R is a software most widely used for statistical analysis and data science .
- R can perform many different statical analysis
  - Statistical tests
  - Linear/non-linear modelling
  - Multivariate analysis
  - Time series analysis
  - Machine learning and deep learning
  - ...

`https://cran.r-project.org/web/views/`

# Introduction to R

- R GUI (graphical user interface) environment
- R is not a menu-driven program, but codes are to be typed in for running commands.
- In general, R consists of 3 windows such as
  - R console
  - R editor
  - R graphics
- Several R programming IDE (integrated development environments):
  - Rstudio (`https://rstudio.com/products/rstudio/`)
  - R Tools for visual studio
  - Tinn-R
  - ESS
  - Rattle

# R on Linux Operating System

- Operating systems for R
  - Microsoft Windows
  - MacOS
  - Linux: Red hat enterprise linux (RHEL), Fedora, CentOS, Ubuntu, SUSE, ...
- Windows vs. linux
  - R isn't inherently faster on linux, compared to Windows.
  - For high-performance computing and big data analysis, linux is recommended to use multiple cores.
  - Some parallel processing packages work better on linux.
- Rtools for building R packages on Windows

  `https://cran.r-project.org/bin/windows/Rtools/`

  - Windows users should install Rtools to build R packages with C/C++/Fortran codes from source files.
  - Linux users do not need this.

# Installing R and Contributed R Packages

- You will want to install the base system. To do so visit the R website and follow the installation directions.
- In addition to the base system there are user contributed add-on packages.
- Packages are a collection of functions, examples and documentation that usually focus on a specific task.
- The base system contains some packages. To install an additional package, say "ISLR", type

  ```
  > install.packages("ISLR")
  ```

  You will be asked to select the mirror site nearest to you, after that everything is automatic.
- Before using the contents of the package we need to load it,

  ```
  > library(ISLR)
  ```
- See the R website for a complete list of contributed packages.

# R Console

- The R Console is where computations are performed
- An expression is inputed into the console and the expression is evaluated. Depending on the expression, the system may respond by outputting results to the console or creating a graph in a new window. Then another expression is inputed and evaluated.
- An R session is this interaction between you and the system
- To get the last expression entered use the up arrow
- Press Esc to stop evaluating the current expression
- To exit the R console

```
> quit()
> q()
> q(save="no")
```

# Calculator

- Enter a math expression and the result is outputted to the console.
- Binary Operators

  `+` `-` `*` `/` `^` `%%`
- Math functions

  `abs()` `sqrt()` `log()` `exp()` `log10()` `factorial()`
- Trigonometric functions

  `sin()` `cos()` `tan()` `asin()` `acos()` `atan()`
- Rounding

  `round()` `ceiling()` `floor()`
- Math quantities

  `Inf` `-Inf` `NaN` `pi` `exp(1)`

# Examples

```
> 5 %% 4
[1] 1
> log(2)
[1] 0.6931472
> cos(pi)
[1] -1
> 6*9+3/5
[1] 54.6
> 3^4
[1] 81
> 5^(-5.6+3)
[1] 0.01522923
> exp(-4*4/2)/sqrt(2*pi)
[1] 0.0001338302
```

```
> ceiling(3.2)
[1] 4
> floor(3.2)
[1] 3
> round(exp(1),4)
[1] 2.7183
> 0/0
[1] NaN
> 1/Inf
[1] 0
> -Inf/Inf
[1] NaN
> sin(cos(tan(0)))
[1] 0.841471
```

# Exercise

- Compute the following values

1. $\frac{e^2}{6!}\sin(5)$
2. $\frac{e^{\tan(\pi/5)}\log_{10}5}{e+\tan(e)+e^{-1}}$
3. $\frac{2^k e^{-2}}{k!}$, where $k = 0, 1, 2, 3, 4$
4. $\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, where $x = -3, -2, -1, 0$

# Assigning Objects

- What we normally do is create objects and then perform functions on those objects.
- Assign an object a name "x" using either

```
x <- object
x = object
```

- Suppose I want to find the mean of a set of numbers. I first assign the vector of numbers a clever name x and then call the function mean().

```
> x = c(0,5,7,9,1,2,8)
> x
[1] 0 5 7 9 1 2 8
> mean(x)
[1] 4.571429
> X
Error:  object 'X' not found
```

- R is **CASE SENSITIVE**.

# Function 'sort'

- Now suppose I want to sort a vector `y` so that the numbers are descending. By default `R` will sort ascending so I need to change the formal argument decreasing to `TRUE` (the default value for decreasing is `FALSE`).

```
> y <- c(4,2,0,9,5,3,10)
> y
[1] 4 2 0 9 5 3 10
> sort(y)
[1] 0 2 3 4 5 9 10
> sort(y, decreasing=TRUE)
[1] 10 9 5 4 3 2 0
> sort(y, decreasing=T)
[1] 10 9 5 4 3 2 0
```

# Script Editor

- The script editor is used for writing programs in R.
- To start a new script, click File > New Script
- When you save a script file in R, it is saved with .R extension.
- You don't need semi-colons at the end of each line. However, if you enter more than one expression on the same line you will need semi-colons.
- You can wrap a long expression onto several lines, but you need to be careful that R does not treat your unfinished code as a complete expression.
- To comment a line of code, use a #.
- There is no block commenting in R.

# Commenting Code

- Comments are notes that help users understand portions of your code. They are also useful reminders to yourself of what was done and why it was done. Including meaningful comments in code is a major part of writing good programs, this semester we will practice commenting our programs.

```
> # Calculate the mean of x
> x = c(0,5,7,9,1,2,8)
> mean(x)
[1] 4.571429
> # Two expressions on the same line requires a ;
> mean(x); sum(x)
[1] 4.571429
[1] 32
```

# Documentation

- R function documentation contains:
  - A general description of the function
  - A list and description of the function's formal arguments and default settings
  - Details about the function
  - A list and description of values returned by the function
  - References
  - An executable example
- Sometimes the documentation is very complete and other times it is vague. It is more of a quick reference and not intended for instruction.
- Better educational resources are the R manuals, contributed manuals, and FAQs available on the R website.

# Documentation

- To look at the documentation for a specific function from a loaded package,

```
> ?function name
> help("function name")
```

- To search the documentation of all installed packages for key words,

```
> ??"key words"
> help.search("key words")
```

- Suppose we need help with the summation function. If we know the name of the function, `sum`, but have a question about the syntax,

```
> ?sum
```

- On the other hand if we don't know what function to use,

```
> ??sum
```

# Working Directory

- When we load/save datasets, load source files or save graphs we will need to specify the file path. To avoid typing the path every time we can specify a working directory.
- Create the destination directory, do only once
  ```
  > dir.create("C:/StatProg")
  ```
- To set the working directory click File > Change dir... or type
  ```
  > setwd(file path)
  ```
- To know a current path
  ```
  > getwd()
  ```
- Example of change of a working directory
  ```
  > getwd()
  > setwd("C:/StatProg")
  > getwd()
  ```

# Workspace

- The workspace is where all the objects you create during a session are located.
- When you close R you will be prompted to "Save workspace image?" If you say yes, the next time you open R from the same working directory the workspace will be restored. That is all of the objects you created during your last session will still be available. Also type

```
> save.image()
> save.image(file="C:/StatProg/data.Rdata")
```

- It is usually wiser to write a script and then run the entire script with each new session. This way you know exactly what objects you have created
- If a particular object is very important then save it to a file.

# Managing the Workspace

- To list what variables you have created in the current session,
  `> ls()`
- To see what libraries and dataframes are loaded,
  `> search()`
- To see what libraries have been installed,
  `> library()`
- To remove an object,
  `> rm(object name)`
- To remove all objects,
  `> rm(list=ls())`