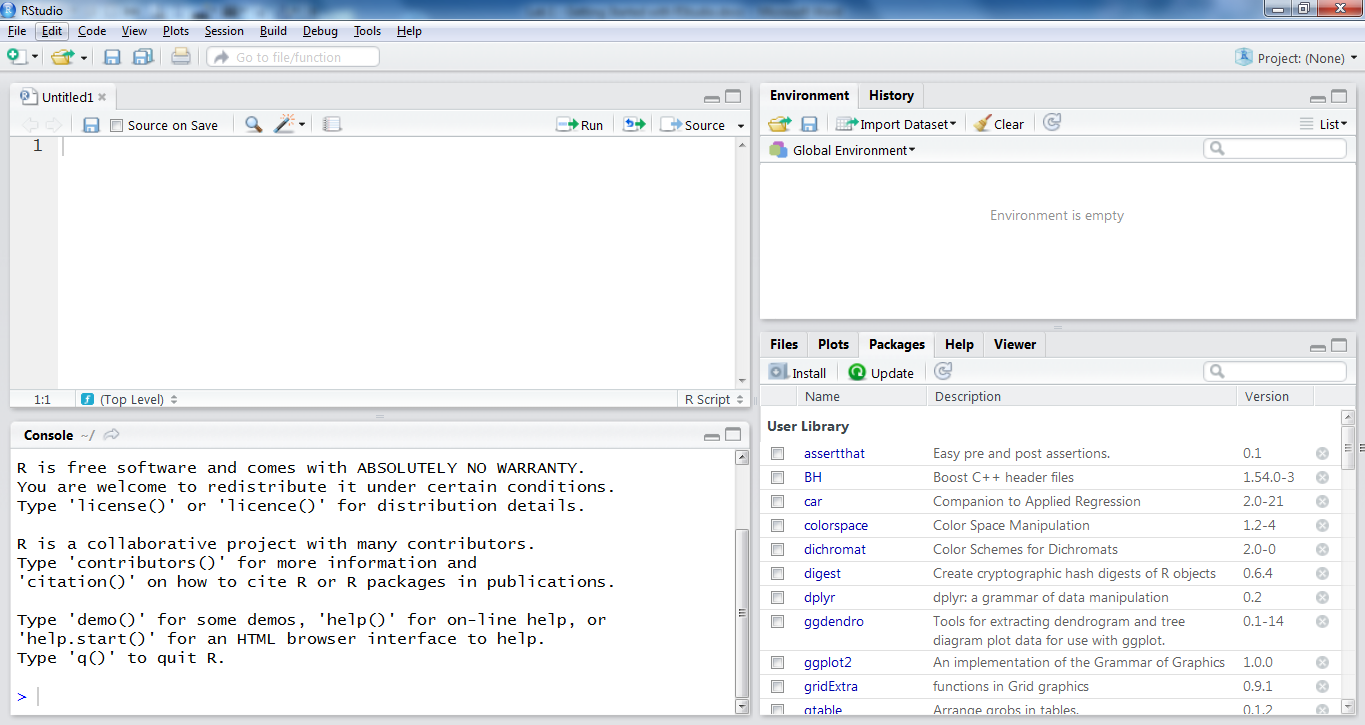
# **Lab 1 – Introduction to R: Getting started with R and R Studio**

**Note: This lab contains some numbered questions. For all of them, you are required to submit (in a Word document) the commands you used to answer them. In some cases, you may also be required to submit program output and/or graphs. Submit your file in the Lab 1 assignment folder in Learning Hub in the lab section of this course before your next lab.**

R, along with its graphical interface RStudio, is free statistical software that is available on the lab computers through Citrix. You can download R on your home computer through this link: <https://cran.rstudio.com/>. When you have finished, you can download R Studio here: <https://www.rstudio.com/products/rstudio/download/>.

# R Studio: an overview

Open RStudio. The RStudio interface shows 4 main windows, as shown below:



Miscellaneous

Environment

Console

Script

The **script window** (upper left) is the window into which we will be typing code to run.

The **console window** (lower left) is the window that displays all the code that has been run, and any printed output from running that code.

The **environment window** (upper right) displays all active elements in the working environment (any data you have loaded, and any output you have assigned to a variable name).

The fourth window (lower right) serves several functions, so it is labelled as the **miscellaneous window**. There are several tabs in this window. The *files* tab shows the files in your “working directory”. The *plots* tab displays the most recent plot you have created (if any, and you can use the back and forward buttons to cycle through plots you created previously). The *packages* tab shows packages which are installed, and can be used to install new packages. The *help* tab displays “help” documents. The *viewer* tab can be used to “view local web content”, and we will not be using it.

A **package** in R is a repository of a group of commands that may be related in some way. These commands are not available in the basic R installation, but can be added on by installing packages. There are currently over 5000 packages available.

For the purposes of this course, we will be using the “mosaic” package extensively. This package must be installed once, but must be loaded each time you open RStudio.

# Installing the “mosaic” package:

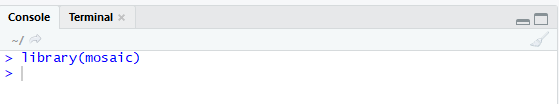
* In the **miscellaneous** window, select the *packages* tab.
* Click the checkbox next to *mosaic*. Note that a checkbox will automatically appear next to *mosaicdata* as well.

The “mosaic” package is now installed (which means it is available to run), but it is not actually running (so any commands that rely on the mosaic package won’t work). The package needs to be loaded **each time** you open RStudio. There are two ways to do this:

1. Type the command:

library(mosaic)

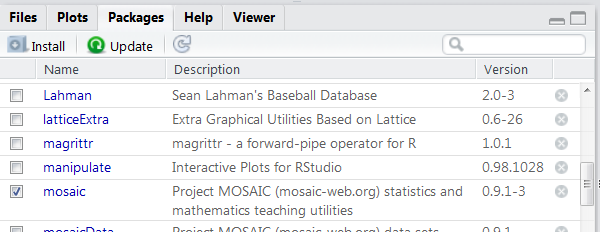
in the **console window**.



This can be handy if you may not finish running the code in one session and need to come back to it; you can just re-run this line of code when you come back to it.

**OR**

1. Click the check-box next to “mosaic” on the *packages* tab of the **miscellaneous window**.



This may be easier for you than remembering the command.

**TROUBLESHOOTING TIPS:**

* sometimes the **mosaic** package does not appear in the list. If this is the case, Type the command:

install.packages(“mosaic”)

in the **console window**.

* R might return an error, telling you that a package used by **mosaic** was not installed. If this happens, install that package separately.

R is a fully functional programming language, and you can write and run scripts in R Studio. We will not be writing scripts quite yet, so we will not be needing the **Scripts** window. In this lab, we will just be using simple commands, which we can type directly in the console.

You can try typing some simple mathematical expressions into the console, and R will behave like a calculator. (The > is the prompt, and the output appears below the command.)

> 73\*15-237

[1] 858

> 34^2+43/13

[1] 1159.308

> sqrt(17)

[1] 4.123106

# Loading Built-In Data

To use any commands, you will need some data available. Some datasets are loaded automatically. You can see a list of those datasets by clicking on “datasets” in the **Packages** tab. Others are available through the “mosaicData” package, which is loaded automatically with the *mosaic* package. You can see a list of these by clicking on “mosaicData”. In both cases, clicking on the package name will give you more information about it.

We are going to use the data in the dataset **TenMileRace**. Take a couple of minutes to read about this dataset either by clicking on its name or typing

> help(TenMileRace)

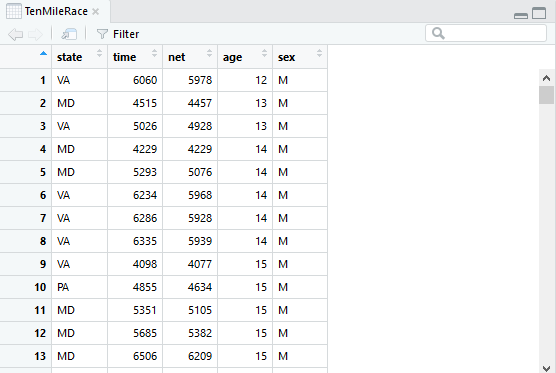
in the console.

Now view this dataset with the command

> View(TenMileRace)

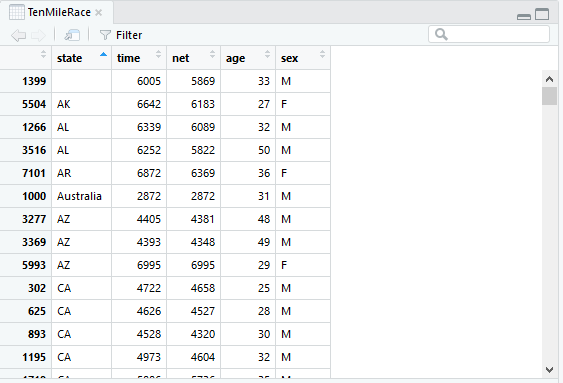
Note that R is case sensitive and will return an error if you don’t copy that command exactly.

The TenMileRace data should now appear in the top left of the screen under the **View** tab.



Note: sometimes RStudio, for reasons known only to itself, fails to load the dataset. If this happens, just close the **TenMileRace** tab and try again. Failing that, exit RStudio and open it again, and try to load the dataset again. (You people know your way around computers, so I probably don’t have to tell you this.)

You can sort the data by any of the fields by clicking on the little arrows. For instance, if you click on the down arrow under **state**, you can sort your data by the runners’ states of residence.



# Filtering Data

You can manipulate this dataset in various ways. For instance, suppose you want a list of the runners’ times, which you can then display in a table or graph (we will learn how to create tables and graphs soon). The command

> TenMileRace$net

outputs a list of times. (See the description of the dataset to see why the **net** field is more useful than the **time** field.) You can then store this in a variable called **racetimes** with either of the following commands:

> racetimes=TenMileRace$net

> racetimes<-TenMileRace$net

Note that these variables now appear in the Environment window to the right.

We may be interested the “fast” runners – say, the ones who finished the race in less than 4000 seconds. Try this command:

> racetimes<4000

You’ll get a huge list of TRUE and FALSE entries. What your command did was assign a Boolean value to the statement “racetimes<4000”for each entry in your list of race times. For instance, your first listed time was 5978, which is greater than 4000, so the statement “racetimes<4000” was false for that case.

The list of 8636 TRUE and FALSE statements probably isn’t very useful. You’re probably more interested in one of the following:

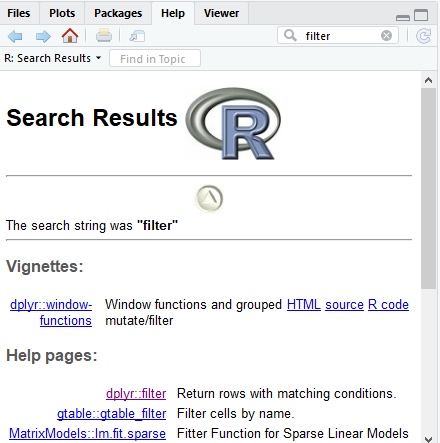
* The number of fast runners
* A list of fast runners

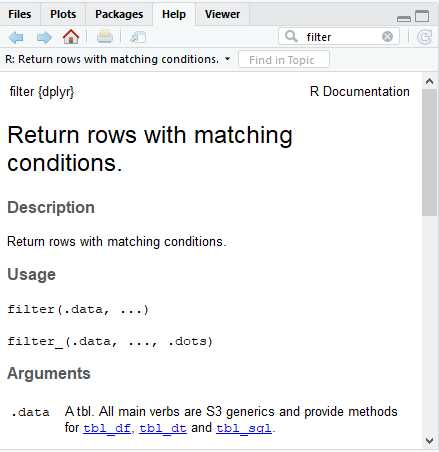
The first of those is easiest, and we can use the list you just generated. R, like other languages, assigns a value of 1 to TRUE, and a value of 0 to FALSE, so we can just add up the TRUE/FALSE list.

> sum(racetimes<4000)

[1] 346

So, there were 346 “fast” runners.

To get a list of fast runners, we need to filter our entire dataset. We do this with the **filter** command, which is part of the **dplyr** package. Install and load that package in the same way you installed and loaded the **mosaic package**, and then read over the help file. ****

****

1. Give the command that creates a dataset called **fastrunners** consisting of runners who completed the race in under 4000 seconds. Hint: your command should be of the form  
   > fastrunners=filter(…, …)

You can now view your **fastrunners** dataset just like you’d view a built-in data set:

> View(fastrunners)

How many rows does your new dataset have? You can get the number from the display, or store it in a variable using the command nrow(fastrunners). Does your number seem right?

1. Now give the commands you use to create the following filtered datasets:
2. The female runners
3. The foreign runners. (For this, we use the fact that the American runners are listed by their two-letter state abbreviations, while the foreign runners are listed with their longer country names. So you can create this set by extracting the entries whose “state” names have three or more characters. Hint: you’ll need to install the **stringi**package and use the **stri\_length** function.)

We can also get tables of data. For instance, the following table gives us a simple breakdown of runners by sex:

> table(TenMileRace$sex)

F M

4325 4311

1. Now give the commands to create a breakdown of the runners by state. (The display here will be a bit unwieldy, because there are so many states.)

# Manipulating data numerically

We can use mathematical commands to find, for instance, average race times. (“Average” is a vague term in statistics, so we use the more precise “mean”.) Since we already have a list of race times, we can use that. Try out the following commands:

> mean(racetimes)

> min(racetimes)

> max(racetimes)

> range(racetimes)

We can also obtain these values by filtering on one of the other variables. For instance, suppose we want to compare the mean race times by sex. For this, we need to use our original **TenMileRace** dataset. Try this command:

> mean(data=TenMileRace, net~sex)

This gives the mean net race times by sex. Do these numbers seem believable?

1. Give the R command or commands that find the number of male runners who finished the race in less time than the fastest female runner. Provide the output of your command(s) as well. Your command(s) should not require have any numbers as inputs – all arguments should be variables.

Another useful command is **rank**, which gives a list of the runners’ ranks. That is, the fastest runner has rank 1, the second fastest has rank 2, and so on. (Ties have fractional ranks.) Applying this to our **fastrunners** data, we get the following:

> rank(fastrunners$net)

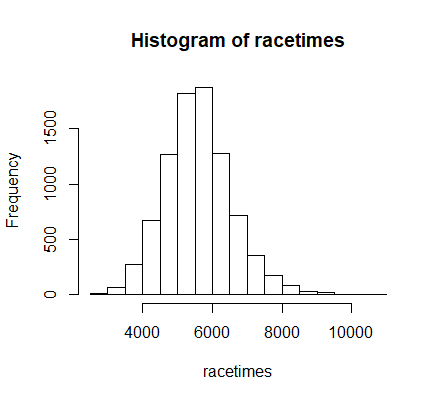
[1] 77.0 274.5 56.0 84.0 282.5 257.0 304.0 322.0 3.0

So, the runner who appears first in the list of fast runners placed 77th.

We can also get a feel for how the race times are distributed. The **hist** command produces a histogram of race times.

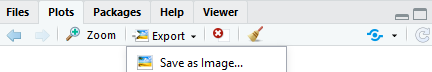
> hist(racetimes)

A histogram will appear to the right:



Soon we will learn how to customize histograms, but for now we can get a decent idea of what the **racetimes** data looks like. The medium times are fairly common, while very few runners were very fast or very slow. We might consider the very fast and very slow runners to be outliers. Previously, we used R to determine that the racers’ times ranged from 2814 seconds to 10536 seconds. Suppose we want to know what the range of times “typical” runners was. One way we may define “typical” is by excluding the fastest 5% as well as the slowest 5% of runners.

1. Create a series of commands that gives the range of “typical” runners. Include this output. As before, your command(s) should not require have any numbers (except 0.05 and 1) as inputs – all arguments should be variables. Create a histogram of these typical runners, and include your histogram in your file. (You can save your histogram by selecting Export -> save as image.)



**Submit a Word file containing all of your answers to the Lab 1 assignment folder in Learning Hub before your next lab.**