

Lab 1 – Introduction to R and RStudio

This lab contains some numbered questions. You are required to submit:

1. One pdf document (Lab 1.pdf) that contains:
 - your written answers to the question
 - the commands you used to answer the question, when asked
2. One R script (Lab 1.R) that contains all the code used to produce your answers. The script must run without errors.

Submit your Word file in the Lab 1 folder in Learning Hub by 11:59pm TOMORROW.

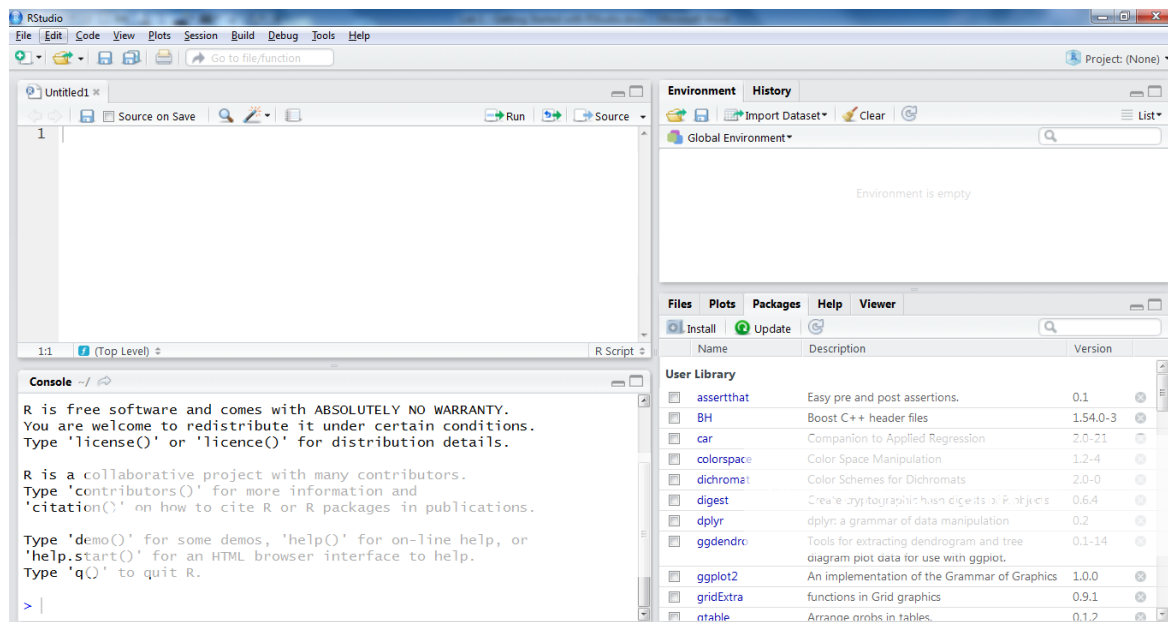
R and RStudio: Overview

R is a high-level computer language for statistical computing and data visualization. RStudio is an integrated development environment (IDE) used to write and execute R code.

Both R and RStudio are freely available. At BCIT, R/RStudio is installed on lab computers. To install on your personal computer:

- download R here <https://www.r-project.org/>
- download RStudio here <https://posit.co/download/rstudio-desktop/>

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



- The **Source** pane is where we write and save R code.
- The **Console** pane is for interactive coding with R. Output is displayed here.
- The **Environment** pane displays all active variables and data.
- The **Plot** pane shows graphs/charts and contains tabs for *packages*, *files*, and *help*.

Installing the mosaic package

By default, the basic R installation contains many useful commands/functions. An R *package* is a repository of commands that can be installed and loaded to augment the basic installation.

For the purposes of this course, we will be using the **mosaic** package extensively. The **mosaic** package is developed and maintained by a group of American educators (<https://www.mosaic-web.org/>).

Installing mosaic

In the miscellaneous window, select the *packages* tab. If **mosaic** does not appear in the User Library, click the install button and type “mosaic” in the Packages field. Install it. You can also install **mosaic** in the Console using the command:

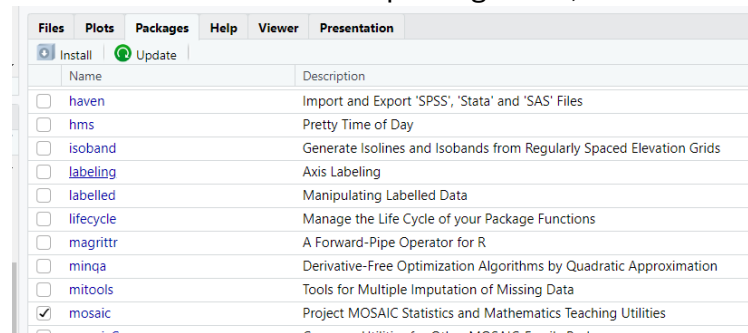
```
> install.packages("mosaic")
```

R might tell you that another package used by **mosaic** was not installed. If this happens, install that package separately.

Loading mosaic

The **mosaic** package must be loaded *each time* you open RStudio. There are two ways:

- click the checkbox next to it in the packages tab, or

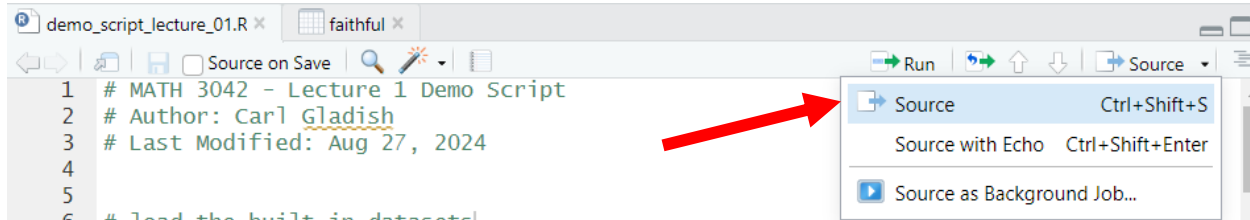


- run the command

```
library(mosaic)
```

Using the Console versus Source

R is a fully functional programming language. Typically, you write scripts in the Source pane and then run the script from there using:



For the moment, we will just execute commands by directly typing them into the Console.

Try Some Commands

Type the following mathematical expressions into the console. R will behave like a calculator. (The `>` is the prompt. 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

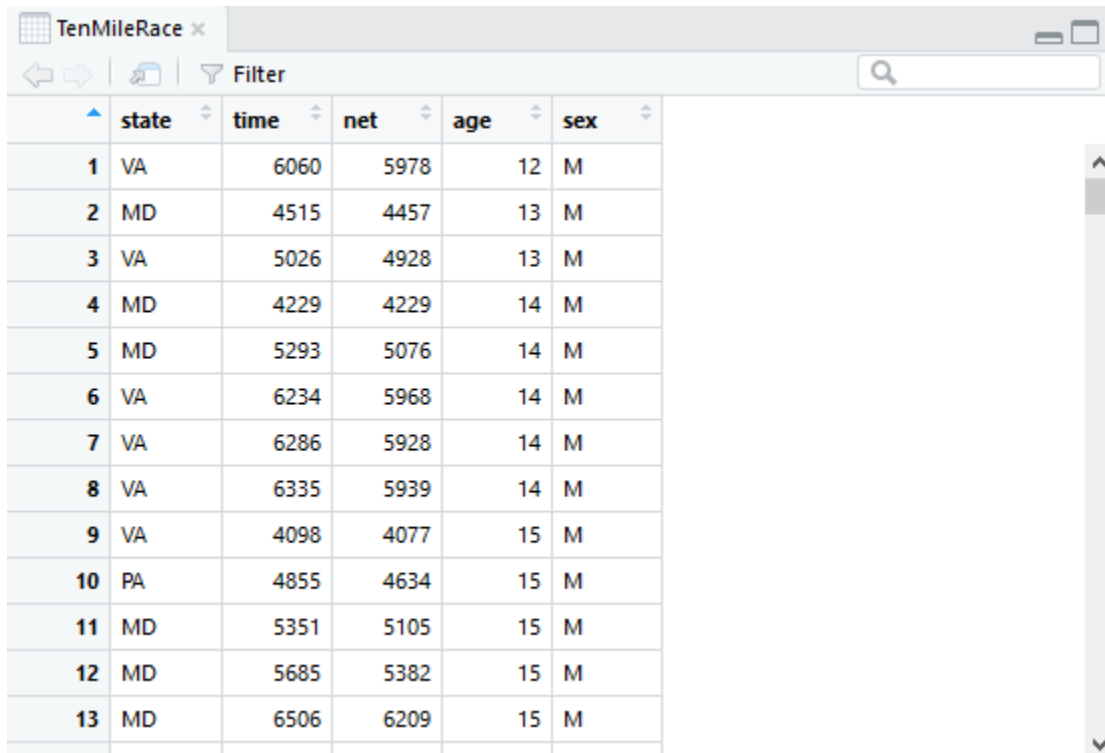
Data comes in many forms: text files, Excel files, SQL databases, and so on. In this course, for simplicity we will often use data sets that are either built-in to base R or available in a package. You can see a list of those datasets by clicking on “datasets” in the Packages tab (scroll down to “System Library”).

Other data sets 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 the packages tab. We are going to use the data in the dataset **TenMileRace** within the **mosaicData** package.

Investigate this dataset by running the following in the console (Note, R is case sensitive):

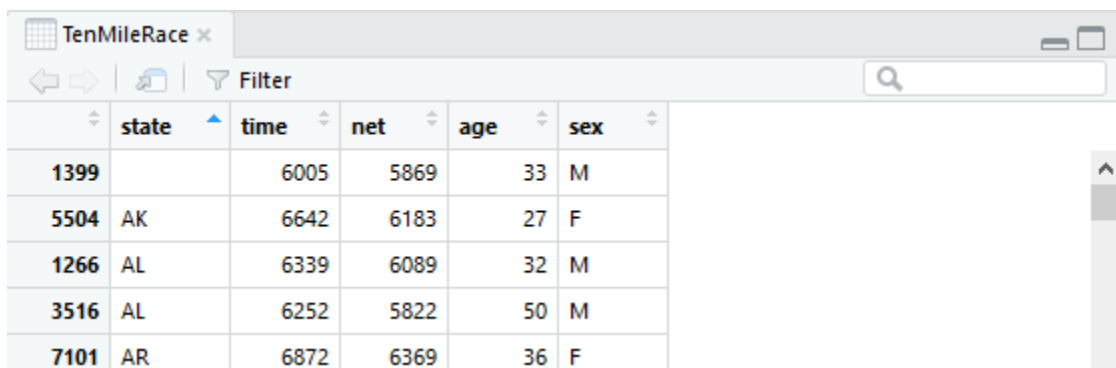
```
> help(TenMileRace)
> View(TenMileRace)
```

The **TenMileRace** data will appear in the top left of the screen under the View tab. The grid of data is called a *data frame* in R. Each column is a *variable*, and each row is an *individual*.



	state	time	net	age	sex
1	VA	6060	5978	12	M
2	MD	4515	4457	13	M
3	VA	5026	4928	13	M
4	MD	4229	4229	14	M
5	MD	5293	5076	14	M
6	VA	6234	5968	14	M
7	VA	6286	5928	14	M
8	VA	6335	5939	14	M
9	VA	4098	4077	15	M
10	PA	4855	4634	15	M
11	MD	5351	5105	15	M
12	MD	5685	5382	15	M
13	MD	6506	6209	15	M

You can sort the data frame by any of its variables by clicking on the sorting arrows. For instance, click the down arrow under **state** to sort the rows by runners' state of residence.



	state	time	net	age	sex
1399		6005	5869	33	M
5504	AK	6642	6183	27	F
1266	AL	6339	6089	32	M
3516	AL	6252	5822	50	M
7101	AR	6872	6369	36	F

Filtering Data – Carry Out These Steps

Step 1

Suppose you want a list of the runners' times to display in a table or graph. The command

```
> TenMileRace$net
```

outputs the variable **net**, which is the net time from the starting line to finish line for each runner. Store this output in a variable called **race.times** using

```
> race.times <- TenMileRace$net
```

The variable **race.times** now appears in the Environment window to the right.

Note: R does *not* use the dot operator (.) in the same way as C or Java or python. In R, the dot is often used in variable names (“period separated” names).

Step 2

Suppose we are interested in the “fast” runners – the ones who finished the race in less than 4000 seconds. Execute the command

```
> race.times < 4000
```

The output is a huge list of boolean (TRUE and FALSE) values. To just see the first few values, use the function **head**:

```
> head(race.times)
[1] 5978 4457 4928 4229 5076 5968

> head(race.times < 4000)
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

The first boolean value is FALSE, since 5978 is not smaller than 4000.

Before we continue, evaluate the following commands and try to understand each output.

```
> c(1, 1, 1, 2, 0)
> sum(c(TRUE, FALSE, TRUE))
```

To determine the number of fast runners, evaluate the command (output shown):

```
> sum( race.times < 4000 )
[1] 346
```

This works because, in the R language, TRUE = 1 and FALSE = 0.

To get a list of the fast runners (< 4000 sec), we will extract the correct rows with **subset**:

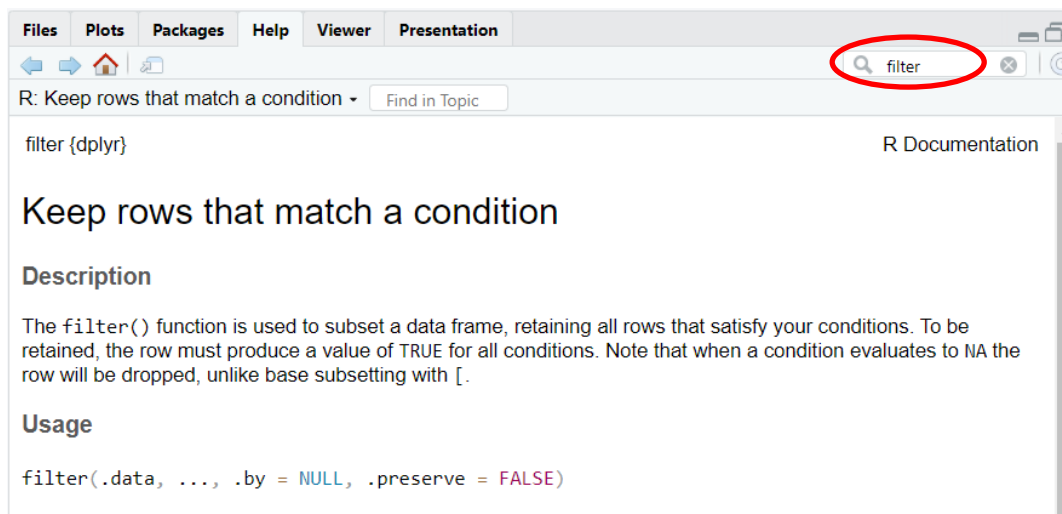
```
> fast.runners <- subset(TenMileRace, net < 4000 )
```

To count the number of fast runners, use the function *nrow*

```
> nrow( fast.runners )  
[1] 346
```

Another way to filter the data frame uses the package **dplyr**. Install and load **dplyr** the same way you installed **mosaic**. We will use the command “filter” in this package.

(Read about the command in the Help tab by searching “filter”.)



For instance, to pull out the runners from Virginia (“VA”), use:

```
> VA.runners <- filter( TenMileRace, state == "VA")
```

Hints/Observations

- Note that the USA runners are those who have a two-letter code in the **state** variable (e.g., “VA” or “MD”). Non-USA runners are those with the country named spelled out (e.g., “Kenya”), which always has more than two letters.
- In **TenMileRace** the **state** variable is stored as *levels* (don’t worry about the technical meaning at this moment). To convert to the character data type, use

```
> as.character( TenMileRace$state )
```

Lab 1 Questions

1. Record a command that creates a variable **slow.runners** containing the data frame made up of runners whose **net** time was more than 8000 seconds.
2. Record a command that determines how many runners are in **slow.runners**.
3. Record commands to create the following filtered datasets:
 - a. all female runners
 - b. all non-American runners (*Hint: nchar*)

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
```

4. Record the commands to create a table showing the number of runners from each **state** (i.e., country) besides USA. (Your table will still contain USA state labels, but with counts of 0.)
5. Do the previous question again but use the built-in function *droplevels()* to avoid displaying **state** values with a count of 0.
6. Record commands that print the **state** from which the *greatest* number of runners come (USA or non-USA).

Manipulating Data Numerically

We can use mathematical commands to find, for instance, the *mean* (i.e., average) race time. Try the following commands:

```
> mean(race.times)
> min(race.times)
> max(race.times)
```

We can also obtain these values by filtering on one of the other variables. For instance, if we want to compare the mean race times by sex, execute:

```
> mean( data=TenMileRace, net~sex )
```

This gives the mean net race times by sex. The expression “net ~ sex” is something unusual in R called a *model formula*. Here it means net values are *grouped* by sex values.

7. Give the R command(s) to find the number of male runners who finished the race in less time than the fastest female runner. Your command(s) must not include any literal numbers (i.e., do not hard code any numbers)!

Another useful function is **rank**, which gives the rank (1st, 2nd, 3rd, etc.) of a vector of numerical values, from lowest to highest. (Ties are assigned fractional ranks.)

Applying this to our **fast.runners** data, we get the following:

```
> head(rank( fast.runners$net ))  
[1] 77.0 274.5 56.0 84.0 282.5 257.0
```

The 77.0 here means that the first runner in the list placed 77th (had the 77th lowest time).

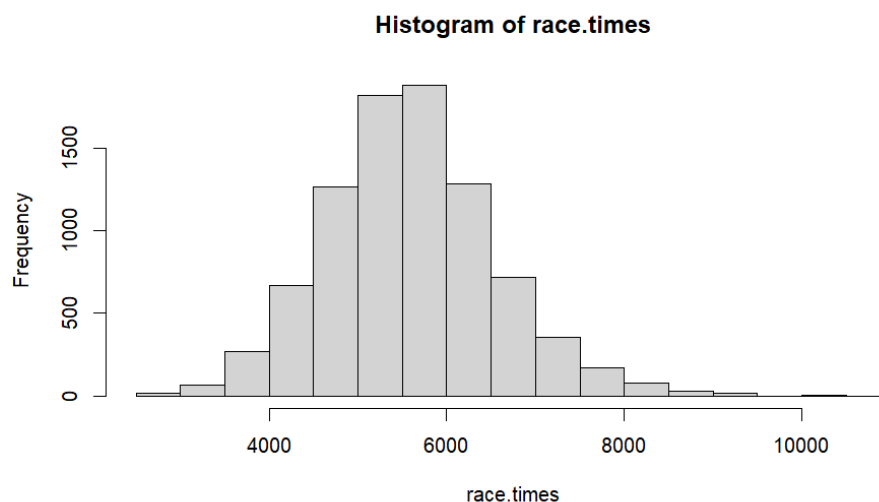
8. Record R commands to find the **age**, **sex**, and **state** of the first runner who finished the race who ranked better (i.e., lower) according to the variable **net** than according to **time**.

Histograms

To get a visual overview of how race times are distributed, we can use the **hist** command to generate a histogram of race times:

```
> hist(race.times)
```

The histogram will be shown in the lower-right Plot pane:



Based on the histogram, we observe that times around 5000 to 6000 seconds are quite common, while very few runners had times below 4000 or above 8000.

9. Let's say that "typical" runners are all those runners remaining once we *exclude* the fastest 5% and the slowest 5%. Record the commands to create a new data frame containing the "typical" runners. Apply **head()** to show just the first few rows. (Show the output in your answer.) *Do not hard code any literal numbers, except 0.05 and 0.95.*
10. Create a histogram of the **net** time for typical runners and include your histogram in your answers. (You can save your histogram by selecting Export -> save as image.)

