OLCF and NICS/RDAV Tutorial: Introduction to R

`http://olcf.ornl.gov and http://rdav.nics.tennessee.edu/`
Prepared by Amy Szczepaski, Pragnesh Patel, and George Ostrouchov.

1. You can download R from `www.r-project.org`. Installation will depend on your OS. Launch R on your computer.

   Once R is running, give the command:

   ```
   library(ggplot2)
   ```

   If you don't have ggplot2 installed, then you can install it with:

   ```
   install.packages("ggplot2")
   ```

   or by using the appropriate menu item from the GUI (will vary by operating system). Once you have installed ggplot2, you can then load it with:

   ```
   library(ggplot2)
   ```

   One of our exercises is based on the **diamonds** dataset that is included with the ggplot2 package. Next week's lesson will be an in-depth look at ggplot2.

   R has thousands of packages available, most of which were contributed by the user community. Much like with LaTeX, some of them are more useful than others.

2. Let's do some elementary calculations with R

   ```
   2 + 2
   a <- 2 + 3
   a
   A
   b <- c(1, 2, 3, 4, 5)
   b
   b <- 1:10
   b
   1:100
   sum(b)
   mean(b)
   sd(b)
   summary(b)
   ls()
   b[1] <- 100
   b
   b[5] = 50
   b
   ```

3. Getting help in R

   ```
   ?mean
   ?clustering
   ??clustering
   ```

4. Continuing with basics:

   ```
   example(mean)
   ls()
   ls
   str(a)
   str(b)
   b[11] <- 15
   b
   ```

```
mean(b)
b[13] <- 20
b
mean(b)
```

5. What do we do now to find the mean of `b`? What does the help file tell us?

6. Now some matrix and array operations:

   ```
   x <- matrix(rpois(6, 3), ncol=2)
   x
   t(x)
   x %*% t(x)
   z <- array(1:24, dim=c(2,3,4))
   z
   z[, , 3]
   x %o% x
   ```

7. We can also type in small programs at the command line. We usually don't want to do this, but we can.

   ```
   for (i in 1:10) print(i)

   We can do multi-line programs, too.

   a <- 0
   for(i in 1:10)
   {
   a <- i + 2
   print(a)
   }
   ```

   Later, we'll see how to run our R programs from a script.

8. The function `plot` can be used to make basic graphs.

   ```
   plot(b)
   ```

   We won't do much today with graphics; next time we'll take an in-depth look at ggplot2.

9. How do you think that we **remove** the objects `a` and `b`? And the objects that the `?mean` left for us?

10. Normally we want to use data that already exists. Since a lot of the users of R keep their data in spreadsheets, one of the easiest formats to import into R is a spreadsheet. I have taken the **diamonds** dataset that is distributed as part of the ggplot2 package and have saved it to a spreadsheet on my hard drive. I will use this to demonstrate loading a spreadsheet.

    ```
    diamonds <- read.csv("/Users/george/data/diamonds.csv")
    ```

    The argument to `read.csv()` is the path to the file. You can also work with this exact same file, as it is distributed with the ggplot2 package. If you have loaded the ggplot2 package with the `library()` command, then you can load the dataset with the command:

    ```
    data(diamonds)
    ```

    The function `read.table()` can load data that is delimited with something other than commas. There are readers in R for many different types of data (e.g. Excel, NetCDF, HDF 5). More and more readers are being contributed by the community. The data we are working with today will be represented in R as a **data frame**, which is an extremely common format within R.

11. Let's try to understand this data.

    ```
    ls()
    head(diamonds)
    ```

```
tail(diamonds)
str(diamonds)
summary(diamonds)
```

If we were to just type the name of the dataset, `diamonds`, like we did with the earlier examples of **a** and **b**, then R would start to print out the entire contents of **diamonds**—with over 50,000 observations! Fortuantely, most R installations have a default setting of the maximum number of rows that they will print to the screen.

12. To refer to the **carats** variable in the **diamonds** dataset, we use the notation `diamonds$carats`. In general, we refer to variables in the form `data$variable` (substituting in the appropriate names). There are times when we will explicitly tell R which dataset we are looking at; in those cases, we can refer to the variables just by their names, such as **carats**. The default is to always tell R both the name of the dataset *and* the name of the variable.

13. We can fit a **linear model** using the `lm()` function. We will model the price of the diamond as a function of carats.

```
mymodel <- lm(diamonds$price ~ diamonds$carat)
mymodel
summary(mymodel)
anova(mymodel)
```

14. We need the object **mymodel** to hold the output of the `lm()` function. If we had just given the command `lm(diamonds$price ~ diamonds$carat)`, then R would have done the calculation but not had anywhere to save the results. R has built-in capabilities for almost any statistical function that you can think of—as well as for many that you have never heard of.

15. The `plot()` function can plot just about anything.

```
plot(diamonds$cut)
plot(diamonds$carat)
plot(anova(mymodel))
plot(mymodel)
methods(plot)
```

Next time we'll do more with graphics.

16. You can also define your own functions with R

```
plustwo <- function(x) x+2
plustwo(4)
```

17. This is useful in combination with the `lapply()` function. This applies a function to every member of a vector, list, or similar object. It returns an list with the results. We will take one of the **for** loops from before and replace it with the `lapply()` function.

```
y <- lapply(1:10, plustwo)
y
str(y)
```

18. Depending on the version of R that you are running, it may have a built-in editor. You can open the editor and type in a short program such as:

```
a <- 0
for(i in 1:10)
{
     a <- a + i
     cat("Right now a = ", a, ".\n")
}
print("We're done now!")
```

On a Mac, you run the code by selecting it and doing command-return. Here's an example of another short program

```
plustwo <- function(x)
{
     x+2
}

plusthree <- function(x)
{
     a<-0
     a<-x+1
     a<-a+2
     return(a)
}

for(i in 1:10)
{
     cat(plustwo(i), plusthree(i), "\n")
}
```

The exact usage of the built-in editor will vary depending which version of R you have. Code from the editor will run as if you had typed it in directly at the command line.

Debugging is done by

```
debug(plusthree)
plusthree
```

Q quits the debugger.

19. Let's leave the interactive interface now. We have some options about what we might want to save:
    - Do we want to save our command history?
    - Do we want to save the program in the editor?
    - Do we want to save the objects in memory?
20. On Unix-like systems, installing R also installs utilities `Rscript` and `R CMD BATCH` that can be accessed from the command line. These can also be used in shell scripts. Here is an example script that I have saved as a file on my hard drive in a file that I have named **myscript.R**.

    ```
    mydata <- 1:10
    mymean <- mean(mydata)
    print(mymean)
    ```
21. I will run it two different ways: Once with `Rscript myscript.R` and once with `R CMD BATCH myscript.R`. Both of these can take options that will change their bahavior.
22. How do we save the plots that result from running R from the command line? Suppose we had the script below.

    ```
    mydata <- 1:10
    mymean <- mean(mydata)
    print(mymean)
    plot(mydata)
    ```

    R creates a file called **Rplots.pdf** for the image. To control what is saved and how, R has functions for graphics output. One called `pdf()` that will redirect all graphical output to a PDF file; its arguments allow you to set a filename, the size of the image (in inches) and other attributes. There is a similar function called `png()`; in this case the image size is set in pixels. (R handles BMP, JPEG, and TIFF files in a similar way.) We could edit our script as follows to specify more about our saved image. The function `dev.off()` tells R that we're done building the graph and it is ready to save.

```
mydata <- 1:10
mymean <- mean(mydata)
print(mymean)
png(filename="awesomegraph.png", width=700, units="px", pointsize=24)
plot(1)
dev.off()
```