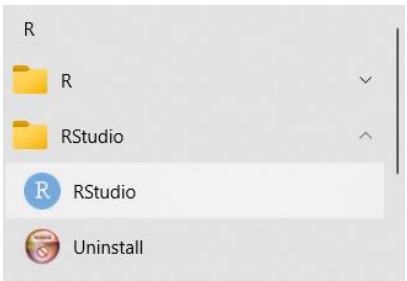
Week 8 Laboratory Activity

Prior to starting this week's tutorial, you would need to have installed RStudio. You can download the Open Source License RStudio Desktop Version from the RStudio Download site. It is available for most common operating systems, including Windows, MacOS and Linux. Do spend some time to familiarize yourself with the interface. Download the RStudio Desktop.

In this week's tutorial we will investigate R as a language for performing exploratory data analysis in Data Science. For this tutorial we will be using RStudio. To launch the application type "RStudio" in the application search box on the start menu (in Windows).



Activity: Exploratory Analysis of Big Data in R

In the first part of the tutorial, we will generate some data and familiarise ourselves with R commands. In the second part we will load and analyse a "large" data file.

Part A: Generating and Graphing Data

In the first part of the tutorial, we will have a play around with R, getting used to its syntax and generating some random data.

First things first, let's generate a sequence of integer values and store them in a variable:

```
x <- 1:10
x
```

Traditionally in R we use "<-" for value assignment although "=" can also be used. R allows us to do perform operations directly on vectors, with the operation being performed on each element of the vector, e.g.:

```
y <- x^2 + 10
y
```

Even division of vectors works in an element wise fashion:

```
(y-10)/x
```

R was designed for doing statistical analysis so it contains many inbuilt functions for visualising data. We can plot the vectors above by simply calling the plot function:

```
plot(x,y)
```

Note that R already labels the *x* and *y* axes using the names of the variables (in this case `x` and `y`). You can add a title and rename the labels as follows:

```
plot(x, y, main="A Simple Plot", xlab="X-value", ylab="Y-value")
```

Practice 1: To convert temperatures in degrees Celsius to Fahrenheit, multiply by 1.8 (or 9/5) and add 32. Generate a sequence of integers with value 0 to 100, as the degree in Celsius. Then write an equation to get the Fahrenheit, and plot a graph for these values.

To make the data a little more interesting, we can concatenate variables:

```
x <- c(1,4)
```

Here we have used the "`c`" command to concatenate the sequences. Note that the `c()` function is required in R whenever we need to define a vector containing arbitrary values, e.g. if we want `x` to be a vector containing the values 1 and 4, we need to type:

```
x <- c(1,4)
```

and cannot just type: `x <- (1,4)`

R contains many inbuilt commands for generating data. Let's generate 1000 uniformly distributed random values in the range 50 to 100 using the `runif()` command:

```
z <- runif(1000,50,100)
z
```

Practice 2: Using the `runif()` command to generate two sequences of random variables over different ranges:

I. 500 random values in the range 10 to 20
II. 500 random values in the range 20 to 50

Change `z` to be the concatenation of these two sequences.

We can easily calculate the average value, as well as the maximum, minimum, and standard deviation of these values by simply typing:

```
mean(z)
max(z)
min(z)
sd(z)
```

Now let's generate some more interesting data by sampling values from a Normal distribution:

```
x <- rnorm(1000)
y <- 3*x + 10 + rnorm(1000)
```

Let's combine this new data with the uniform random values in z to make a table (called a "data frame" in R) as follows:

```
myData <- data.frame(x,y,z)
head(myData)
```

Rather than printing out the whole data frame by typing "`myData`", we have used the `head()` command to print out just the first few lines of the table. We can inspect the data by getting R to plot it:

```
plot(myData)
```

Since there are 3 variables in `myData`, there are 6 different ways to plot the data (i.e. assign variables to the axes x and y) so R plots them all! This functionality can be very helpful when we are looking to understand what relationships exist in our data, e.g. correlations between variables as shown in plots containing `x` and `y`, or clusters in plots containing `z`.

Let's have a closer look at the relationship between the first two variables `x` and `y`, by plotting just them:

```
plot(y~x,data=myData)
```

Note that we have used a special syntax here "`y~x`", which is called a formula in R and in this case simply means that we are going to plot y against x. (In general, the formula means that we are going to try to predict y based on x.)

Practice 3: Interpret the output, e.g. relation between x and y.

We'll now look at fitting a simple linear model (linear regression) to the data. We can do this using the function lm and specifying the same formula we used to plot the variables.

```
fit <- lm(y~x,data=myData)
```

```
summary(fit)
```

Here the summary function has been called in order to print out details of the linear fit including the slope and intercept terms. We can now add this line-of-best-fit to the plot:

```
abline(fit,col='red')
```

Practice 4: Interpret the output from the aspect of coefficients, e.g. slope, intercept, and Multiple R-squared. Your tutor will discuss this finding in class. Observe also statistics like 1st Quartile, Median, 3rd Quartile, etc. We can discuss this in your tutorial session.

Part B: Analysing a Large Data File

Note that this part of the tutorial is a modified and extended version of that given in Chapter 4 of "The Art of Data Science: A Guide for Anyone Who Works with Data" by Roger D. Peng and Elizabeth Matsui. The book provides an excellent introduction to Exploratory Data Analysis with R.

Now that we have seen a little R, we will use it to work on a large datafile. The file is called "`hourly_44201_201406.csv.gz`" and contains 12 MB of compressed data in gzipped CSV format. (Uncompressed, the file is 335 MB) The data consists of hourly ozone level readings across the US. The CSV file is available on Moodle.

Before loading the file into R, we will need to change RStudio's working directory to be the appropriate location. Make sure that the working directory (folder) is the same location as the location of your file. (For those who are relatively new to computing, please do not hesitate to ask on Moodle Forums or in-class if you are lost here, it is quite common. For those who can assist, please do).

To do that use the `setwd()` command with the appropriate directory, for example (assuming that you downloaded the file to your "`Downloads`" folder):

```
setwd("C:\\Users\\<AuthcateID>\\Downloads")
```

Replace the `<AuthcateID>` with your computer login name (change `<AuthcateID>` to be your login). Note the double backward slash (for Windows), else can use the forward slash.

We can now read in the file. Since it is compressed, we will need to let R know that it needs to decompress the file while reading it in. To do this we use the `gzfile()` command. To read the file in as as dataframe use the `read.table()` command:

```
ozone <- read.table(gzfile("hourly_44201_2014-06.csv.gz"),header=TRUE,sep=",")
```

Now we can have a look at the data frame. To see only the first few lines of the table use the "`head`" command:

```
head(ozone)
```

Practice 5a: How to display the last few records (the command is the same as in your previous Python exercises?

Practice 5b: How many records in this file? (For you to find out online)

We can see the details of the table by typing the "`str`" command.

```
str(ozone)
```

In R, a table is called a "data frame", a row is an "***observation***" and a column is a "***variable***". So the output tells us that the frame contains 1600483 observations over 24 variables. R has detected the type of each variable (column) as either integer, numeric (for decimal values), logical (true/false) or "factor" (categorical).

We can easily select a subset of the columns to display (in this case 6, 7 and 10) using the concatenation operator discussed above. Note the need for the comma before the `c()` function,

Since we are selecting columns not rows.

```
head(ozone[,c(6:7, 10)])
```

We can select a particular variable (column) using the `$` notation. In this case we'll display the first 10 rows only. (Note that ***unlike Python, R counts from 1 not from 0***)

```
ozone$Latitude[1:10]
```

In order to show all of the distinct values present in a particular column, use the "`unique`" function:

```
unique(ozone$State.Name)
```

Practice 6: How many states in the data? Does that make sense?

You can also compute summary statistics for a column using the summary function as seen in the previous part of this tutorial.

Practice 7: Display the summary statistics for the column Sample Measurement.

Here we see the median value, which is the midpoint value for the data, such that half the values are smaller (or equal) and half are larger (or equal). We are also given the (1st and 3rd) quartiles, which are the values with 25% of the data on one side and 75% on the other, and the maximum and minimum values.

We can generate as many percentage levels as we wish using the quantile method. For example we can generate quantile values at 10% increments by using the seq function to generate a sequence of values 0,0.1,0.2,...,1:

```
quantile(ozone$Sample.Measurement, seq(0, 1, 0.1))
```

We'll now look at plotting the data. Let's consider first a basic boxplot of ozone levels:

```
boxplot(ozone$Sample.Measurement, ylab="Ozone level (ppm)")
```

The boxplot visualises the information that we were given by the summary function above, and is a standard way to try to visualise the distribution of data for a particular variable.

We are not limited to generating boxplots for single variables, however. We can use a formula of the form "`A ~ B`", to tell R to generate different boxplots over variable `A` for every distinct value of variable `B`. In this case we'd like to see the ozone level distributions broken down by state:

```
boxplot(ozone$Sample.Measurement~ozone$State.Name,ylab="Ozone level (ppm)")
```

We can't read the x-axis, so we can clean up the presentation with an additional command call to fix the formatting (supplementary material for the `par()` command, https://www.datamentor.io/r-programming/subplot/):

```
par(las = 2, mar = c(10, 4, 2, 2), cex.axis = 0.8)

boxplot(Sample.Measurement ~ State.Name,ozone,range=0,ylab="Ozone level (ppm)")
```

Now lets do some simple spatial analysis on the data. We'll see if the western states of the US have higher or lower ozone levels compared to the eastern states. To do this we will first define a new factor variable (column in our table) indicating whether each state is in the east or west of the country:

```
ozone$region <- factor(ifelse(ozone$Longitude > -100, "west", "east"))
str(ozone)
```

Practice 8: Based on this statement, name three west states, and three east states.

Note the new factor (i.e. column) has been added to the ozone data frame (ie. table) with the name "`region`". To visualise what we're doing, we'll create a visualisation using the maps package. If the maps package has not been installed on your machine type:

```
install.packages("maps")
```

And then follow the instructions to install the package. Note how easy it is to extend your R install with new packages!

Once you have installed the maps package run the following code:

```
library(maps)
map("state")
abline(v = -100, lwd = 3)
text(-120, 30, "West")
text(-75, 30, "East")
```

Next let's calculate summary statistics for each of the two regions: East and West. In order to do that we'll import a library called "`dplyr`" and use a function called `summarise()` after first grouping the data by region (using the `group_by()` command):

```
install.packages("dplyr")
library(dplyr)
```

```
grp <- group_by(ozone,region)
summarise(grp,mean=mean(Sample.Measurement),median=median(Sample.Measurement))
```

When computing the averages above, two non-states (namely Puerto Rico and Mexico) were included in the calculations. Mexico spans the east-west divide and has different pollution levels to the US, so perhaps it should be excluded from the analysis. We can exclude data using the filter command:

```
library(dplyr)
fltr <- filter(ozone, State.Name != "Country of Mexico")
grp <- group_by(fltr, region)
summarise(grp, mean=mean(Sample.Measurement),median=median(Sample.Measurement))
```

Practice 9a: Is ozone pollution higher in the East or West? Does that make sense?

Practice 9b: Write statements to remove the islands (Puerto Rico and Hawaii) and perhaps also Alaska. How does the mean and median of sample measurement changes?

As we have learned, the boxplot is a good way to visualise the distribution of data. You also can use the summary function to check the distribution of data.

Practice 10: Plot a boxplot to show the distribution of the resulting data, i.e. sample measurement for each region.

We see that there do appear to be some differences in Ozone pollution levels between the east and west of the US. Finally, we can also generate some other visualisations in using the maps package, such as colouring code the map by their level of ozone pollution relative to the best and worst states:

```
ozone_bystate <-
summarise(group_by(ozone,State.Name),mean=mean(Sample.Measurement))
min <- min(ozone_bystate$mean)
max <- max(ozone_bystate$mean)
relative_values <- (ozone_bystate$mean - min)/(max - min)
map_statenames <- map('state',plot=FALSE,namesonly=TRUE)
names_mapping <-
match(gsub('(:.*)','',map_statenames),tolower(ozone_bystate$State.Name))
map('state',fill=TRUE,col=grey(relative_values)[names_mapping])
```

The second part of the code looks a little complicated but all it is doing is mapping the names used in our data to refer to the states to those used on the map to refer to them.

Part C: Investigate other Datasets

R comes packaged with a number of datasets that you can investigate. To see the datasets available, type the data command:

```
data()
```

Press the up/down keys to scroll type "q" to exit. To have a look at a particular dataset, load it with the data command, e.g.:

```
data(iris)
iris
```

Practice 11a: What are the different species of iris?

Practice 11b: Which species has the min sepal length of iris, which has the max?

Practice 12: Load whichever data you think looks interesting and do some initial graphing and analysis of it.

Part D: Investigating other Packages

Researchers have developed packages for extending R with all sorts of analysis functionality. For instance the `ggplot2` library provides more sophisticated plotting:

```
install.packages("ggplot2")
library(ggplot2)
ggplot(data = airquality, mapping = aes(x = Ozone, y = Temp)) +
geom_point() + geom_smooth(method = lm)
```

Including more sophisticated regression techniques like lowess regression.

```
ggplot(data = airquality, mapping = aes(x = Ozone, y = Temp)) +
geom_point() + geom_smooth(method = loess)
```

Practice 13: Have a look at this list of useful packages to see the sorts of functionality that is available.