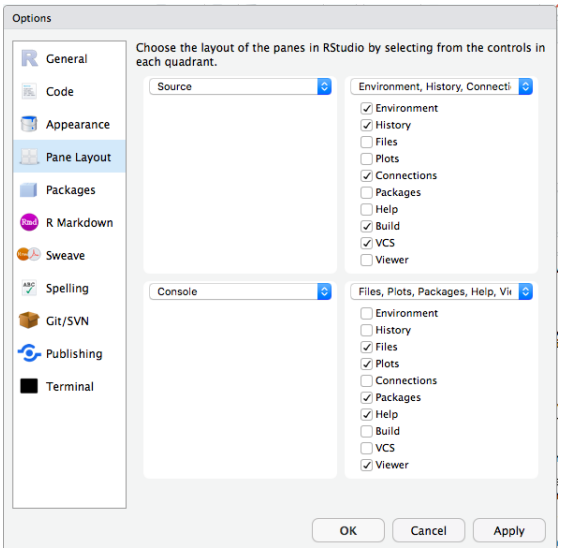
HUDM 5133 - Introduction to Data Analysis and Graphics in R
01 - Introduction

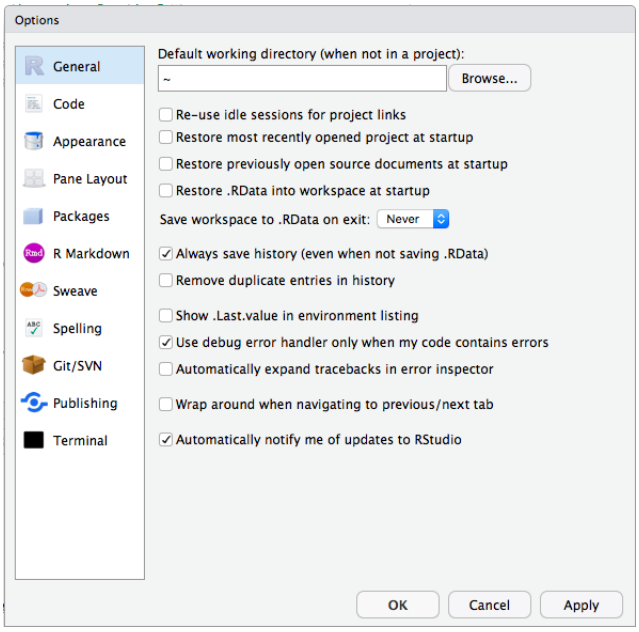
0.1 Downloading R and RStudio

Visit the Comprehensive R Archive Network (CRAN) website at `https://cran.r-project.org/` to download R for your computer's operating system. If you have an older version of `R` on your machine, now is the time to download the most recent version from CRAN. This is especially important because some of the packages we will use are not compatible with older versions of `R`.

Although you don't need to use RStudio to work with R, it makes R easier to work with. RStudio is a separate download; go to `https://www.rstudio.com/` and download and install the right version for your operating system. You will likely want to put an RStudio shortcut icon in your menu bar or start menu. Open RStudio and notice the pane design for integrated viewing of different processes. Go to the "RStudio" drop-down menu and select "Preferences". Then select "Panes". You will see a screen that looks like the figure below. My preference is to put the source pane in the top left, the console pane in the bottom left, the environment/history pane in the top right, and the plots/help pane in the bottom right.



Next, go back to the "Preferences" menu and select the "General" tab. I prefer to uncheck all boxes except the three shown in the figure below. Furthermore, I recommend setting the drop down menu so that RStudio never saves your workspace on exit. The rationale for setting this to never save is that if you don't, R will save whatever data is in your workspace (i.e., objects visible using `ls()`) in a history file so that the next time you open RStudio, it will all be available in your working environment. While, in principal, this is a nice idea, in practice, you end up storing way more data and objects than are necessary and the clutter causes RStudio to open slowly and get glitchy.



Instead of storing important information in your workspace, I will encourage you to begin to think of your source file, which is a text file that contains your saved lines of code, as the best place to store your important information in R. Most operations you will ask R to do will take only a fraction of second to run, so storing your code and then running the code each time you need it is a good habit to get into. To that end, you will work on writing efficient code that is understandable, so that the next time you read it you can see clearly what you were trying to do when you wrote it. Adding comments to code are a big part of making it understandable.

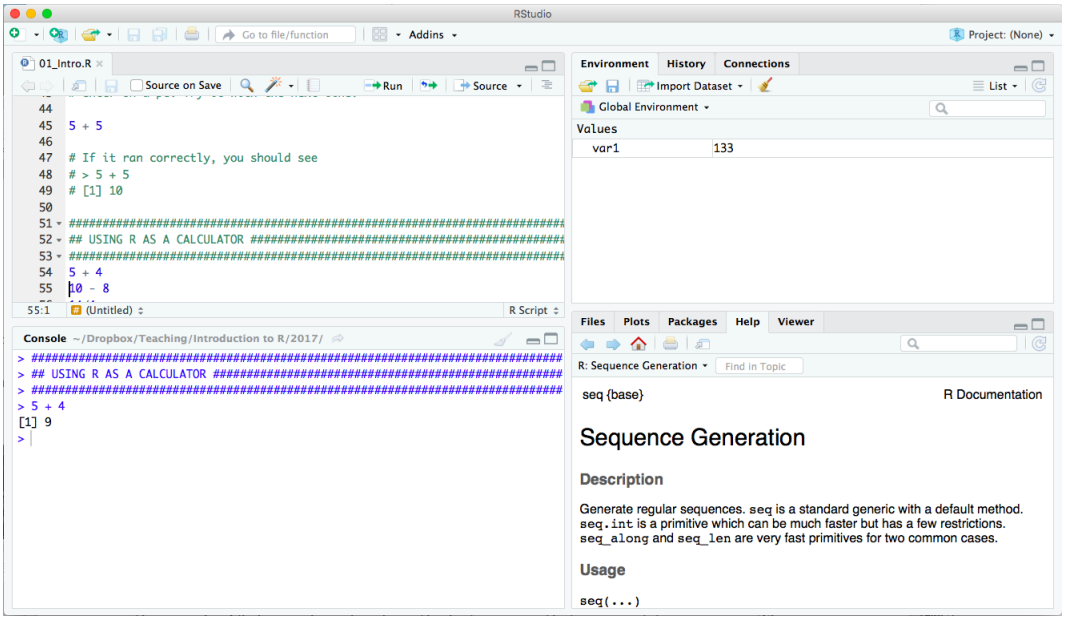
0.2 The Four RStudio Panes

The **console pane** is where you may interact directly with the R command line. If you type code in the console and press enter (or return), your code will run, and, if called for, R will produce output, also in the console. Let's try it. Do some basic math in the console, like `5 + 5`, and press enter. You should see the answer `10` printed as output. Note that you must use the asterisk "`*`" for multiplication and the forward slash "`/`" for division.

The history of code run in your console will be recorded in the **history pane**. Go to the top right pane and click on the history tab. You should see all the code that you just ran in the console. Click on a line of code in the history pane that you want to run again. Then, with your cursor on the line (you don't have to highlight the whole line) find and click the "To Console" button. You should notice that the line now appears in your console.

In general, as I mentioned above, you will work in the **source pane** rather than the console because the source panel allows you to save your code as text and adds intelligent color coding and tabbing to help make code easier to read and debug. Go to the "File" menu and select "New File" and "R Script". A new text file should open up in your source window pane. Save the file. You may send code from your history to your source text file by clicking on "To Source". Try that as well. You should now see the line in your source panel. To run a line of code in your source file, put your cursor anywhere on that line and, on a Mac OS, push command and return at the same time. On Windows OS, push control and enter at the

same time. The line will run and the cursor will move to the next line. This is a convenient way to move through a document. If you wish to run a specific part of a line, or more than one line at a time, simply select the code you wish to run and then push command and return or control and enter. There is also a "Run" button at the top of the RStudio source pane in case you prefer to point and click to run.



0.3 The CRAN Website and the R Community

In addition to downloading R, you may also visit the Comprehensive R Archive Network (CRAN) to access various help manuals at `https://cran.r-project.org/manuals.html`. If you have a question about something R-related, a Google search is typically a great first step toward finding answers to R questions. Check out the R help pages at Stack Overflow here `https://stackoverflow.com/`. Chances are, if you have an R related question, someone else has already asked about it on one of the stack exchange sites. The Quick R website is another resource. It is available at `http://www.statmethods.net/`.

1 Working with Code in the Source Pane

Download the R file for today called "01_Intro.R". Right click on the file and either get properties or get info and select open with. Find the option to change the default so that all .R files open with RStudio by default. Then, double-click on the "01_Intro.R" and verify that it opens in RStudio. If not, try again. Eventually, open the file in RStudio and you should see it as a tab in the source pane.

1.1 Using R as a calculator

- The hashtag "#" is the comment character in R. Anything on a line following a hashtag will be ignored.

- R will do arithmetic operations using the usual symbols for addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (`^`).
- Other math functions include `sqrt` for square root, `exp` for the exponential function, `log` for log base e, trigonometric functions using `sin`, `cos`, and `tan`.
- R will follow order of operations. For example, running `5 + 3 * 4` will return `17`, not `32`.
- Parenthesis may be used as well. For example, running `(5 + 3) * 4` will return `32`.
- Scientific notation can be specified with the letter `e`, which is interpreted as "times ten to the power of" when written in a numerical expression. For example, `2e2` is 200.

1.2 Assigning Values

- Before we get into assigning values, look at your environment tab in the environment/history pane. It should be empty at this point, meaning that no variables have been assigned, and nothing is stored in your R workspace. To check this with code, you may enter `ls()`, which will list all the names of objects stored in your environment.
- The help page for the `assign` function notes that the first two arguments passed to the function are called `x` and `value`, where `x` needs to be a character string representing the name of the variable you want to create, and `value` is the value you want the variable to have. Let's call the variable `var1`, and let's assign it a value of 5133.
- The value of `var1` may be overwritten. Suppose we want to update `var1` to be its old value less 5000. See the source file for code to do just that.
- There is a shortcut for the `assign` function that involves using the less than symbol and the hyphen to construct an assignment arrow, `<-`.
- To assign the variable name `x` to have the value `5`, we could write `assign(x = "x", value = 5)`. Or, with the shorthand, we could simply write `x <- 5`.

  Now is a good time to note that R is case sensitive. So, even though a variable named `x` is defined, R will not recognize `X` because of the case difference. Similarly, if you try to assign a variable by writing `Assign` instead of `assign`, R will throw an error because of the capitalization.

2 Installing and Using Packages

Packages are collections of functions, data, and compiled code bundled up into a well-defined format that makes them easily importable into R.

- Packages may be installed via code or via the RStudio help pane. Let's install the `car` package via the RStudio help pane.

- The `car` package name is an acronym for 'Companion to Applied Regression'. The package contains, among other things, a number of useful functions for diagnostics for linear regression.
- Go to the 'Packages' tab in the help pane and click the 'Install' button.
- By selecting to install from 'Repository (CRAN)' you instruct R to download the package from the CRAN website. You would choose the other option if you already had a compressed file on your computer that contained the package you wanted to install to R.
- In the space for 'Packages', put the name `car`.
- Check the box to 'Install dependencies' to enable packages that `car` depends on to also be downloaded automatically.
- Then click 'Install'.

If you get an error that package "`car`" is not available for your version of `R`, update to the latest version by going to the CRAN website and downloading. After downloading the latest version of `R`, restart RStudio and try to install package `car` again. It should work now. The other way to install a package is to do it with code. You can go to the console and type `install.packages("car")`, for example, to install the package manually. In any case, when you install the package for the first time, all the pacakges that `car` depends on will also need to be downloaded. Thus, you will get the following message:

```
also installing the dependencies 'backports', 'digest', 'glue', 'zeallot',
'ellipsis', 'magrittr', 'vctrs', 'R6', 'clipr', 'BH', 'rematch',
'prettyunits', 'assertthat', 'utf8', 'forcats', 'hms', 'readr', 'cellranger',
'progress', 'zip', 'cli', 'crayon', 'fansi', 'pillar', 'pkgconfig', 'rlang',
'SparseM', 'MatrixModels', 'sp', 'haven', 'curl', 'data.table', 'readxl',
'openxlsx', 'tibble', 'minqa', 'nloptr', 'Rcpp', 'RcppEigen', 'carData',
'abind', 'pbkrtest', 'quantreg', 'maptools', 'rio', 'lme4'
```

Subsequently, you will receive status reports on the downloads of each of those packages. Once they are all done downloading, you will get a message that "The downloaded binary packages are in ..." Now the package is downloaded, but it is not yet loaded in your workspace. To load the `car` package, type `library(car)`. You will not have to install the package again, but you will need to load it using the library function any time you want to use it.

3 OLS Regression in R

Download and open the data file called "ecls.Rdata" from Canvas. Verify that it now shows up in your environment pane on Rstudio. The object `ecls` is a data frame. That means that is a matrix with named data columns that may be of different types. We may apply some functions to learn more about the contents of the data frame.

- `dim()` gives the dimensions of the data frame (rows first, then columns).
- `head()` shows the first six rows of the data frame.
- `names()` prints the names of the columns, in order.
- `str()` shows the structure of the data frame.

The `lm()` function is the main workhorse used to fit linear models. The first argument for lm is the formula. Formulas in R have the following form:

$$\text{name.y} \sim \text{name.x1} + \text{name.x2} + \cdots + \text{name.xp}$$

where the variable name supplied on the left hand side of the tilde, $\sim$, is the name of the dependent variable (also called the outcome variable), and the names separated by addition symbols on the right hand side of the tidle are the names of the independent variables (also called predictors, regressors, covariates, and explanatory variables).

The next important argument you will need to supply to the `lm()` function is the name of the data frame where the variable names may be found. There are a number of other arguments that may be used with this function, but for our purposes in this course, these two are the big ones. Thus, if the data frame you are working with is called `ecls` (and it should be), and you want R to estimate least squares regression coefficients for the simple regression of 5th grade math score on kindergarten math score, your function call will look like this:

```
lm(formula = MATH5 ~ MATHK + SES, data = ecls)
```

or

```
lm(formula = MATH5 ~ MATHK + SES,
   data = ecls)
```

Although you don't have to put each argument on a new line, I find that it helps with organization and readability, so I recommend that you do it. Running the code above would cause output to be printed but would not save any output. To save the results from the function call it needs to be assigned. Let's assign it to the name `lm1`.

```
lm1 <- lm(formula = MATH5 ~ MATHK + SES,
          data = ecls)
```

Now we can use other functions to extract the output. The `summary` function will summarize the output from any `lm()` function call.

```
summary(lm1)
```

Task 1 *How many cases and variables are there in the* `ecls` *data frame?*

Task 2 *What are the names of the variables in the* `ecls` *data frame?*

Task 3 *Run a simple linear regression of fifth grade math score predicted by kindergarten math score and assign it to the name* `lm1`*, as above. Describe and interpret the estimates for the intercept and slope. Also report and interpret the value of the multiple squared correlation,* R^2*.*

Task 4 *Run a multiple linear regression of fifth grade math score predicted by kindergarten math score, kindergarten SES, and gender, and assign it to the name* `lm2`*. Describe and interpret the estimates for the intercept and slopes. Also report and interpret the value of the multiple squared correlation,* R^2*.*

4 Going Further with R Regression Output

There is a lot of information stored in the output from running the `lm()` function. Here we will discuss some of the ways to access and use that information using the regression saved as `lm1` above. First, take a look at the output again that get from running the `summary()` function.

```
> summary(lm1)

Call:
lm(formula = MATH5 ~ MATHK, data = ecls)

Residuals:
    Min      1Q  Median      3Q     Max
-63.174 -10.625   1.621  12.139  55.744

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 75.45441    0.69990  107.81   <2e-16 ***
MATHK        1.54790    0.02009   77.05   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 17.39 on 7360 degrees of freedom
Multiple R-squared:  0.4465,Adjusted R-squared:  0.4464
F-statistic:  5936 on 1 and 7360 DF,  p-value: < 2.2e-16
```

Notice the estimate for the residual standard error, s, is 17.39. Recall the formula for residual standard error:

$$s = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}.$$

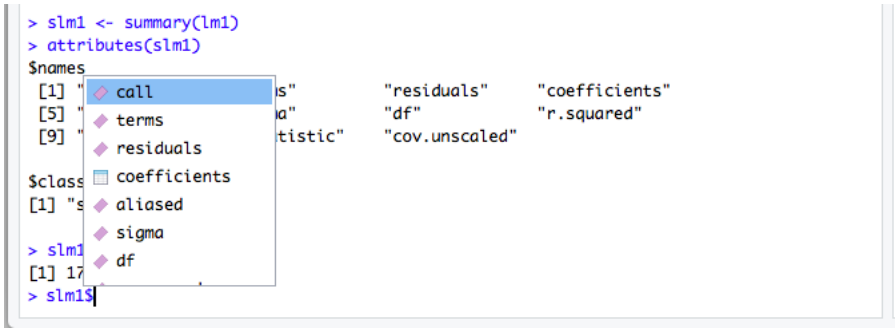
Furthermore, notice the degrees of freedom for the residual standard error are 7360. This is because the total sample size is 7362 and there are two parameters estimated in the model

`lm1`. Finally, note that the multiple R-squared is reported as 0.4465. The adjusted R-squared makes a correction for upward bias in R-squared that is important for small samples. With a large sample like this, the difference is inconsequential.

As a first step toward working with the output from `lm1`, we will save the output from running the `summary()` function on `lm1`.

```
slm1 <- summary(lm1)
```

The object `slm1` now holds a lot of information, which can be accessed by using the dollar sign, `$`. One way to see directly what named objects are stored in `sm1` is to use the `attributes` function. However, a nice feature of Rstudio is that it offers drop down lists of the attributes stored in objects, so no function is necessary.



- `call` gives the original function call:

  ```
  lm(formula = MATH5 ~ MATHK, data = ecls)
  ```

- `terms` gives the variable names.

- `residuals` is a vector of all 7362 residuals, in the same order as the data.

- `coefficients` is a matrix of coefficients, their standard errors, t scores for testing the null hypothesis that the coefficient is identical to zero, and p values for those tests.

  ```
               Estimate Std. Error   t value Pr(>|t|)
  (Intercept) 75.454408 0.69990280 107.80698        0
  MATHK        1.547902 0.02009004  77.04821        0
  ```

  Note that this particular matrix of coefficients has two rows and four columns. The first row corresponds with the intercept and the second row with the slope on MATHK. Thus, it is possible to extract specific information from the matrix by subsetting it. For example, if I want to save the standard error of the intercept, which is in the first row, second column, I could access and save it (calling it SE1) as follows:

  ```
  SE1 <- slm1$coefficients[1,2]
  ```

- `aliased` lets you know which variables, if any, were perfectly multicollinear with a linear combination of other variables in the model.

- `sigma` is the estimate of the residual standard error, s.
- `df` is the number of degrees of freedom for the residual SE.
- `r.squared` is the squared multiple correlation.
- `adj.r.squared` is the adjusted version.
- `fstatistics` is the F statistic associated with testing the full model against an intercept only model.
- `cov.unscaled` is the estimated variance/covariance matrix of model coefficients (i.e., intercept and slopes).

Also note that detailed descriptions of these roles may be found by accessing the help on the `summary.lm()` function via `help(summary.lm)`.

Task 5 *Save the summary from `lm2` as `slm2` and explore the output using both the `attributes()` function and the `$`.*