**R and System Code Notes**

**General**

* No line end syntax (no ; at end of line)
* Assignment uses the “<-” operator: x <- 5, not x = 5
* R is 1-based: indices start at 1, not 0
* Can iterate through loops on elements of a list, not just numbers.
  + Ex. for( i in unique(df$iso)){ <body of loop> }
* Data frames are by far the most useful structure. They are essentially an R object version of an Excel spreadsheet, complete with column names. (They are, in reality, sets of coupled vectors.)
* Comments, comments, comments! Document the code!

**Operators**

* \*,+,-,/,==,!=,<,<=,>,>=,!,||,&&: Standard operators
* \*\*: Exponentiation. (x \*\* 2 == x^2)
* $: Takes a specific column from a dataset. (df$iso)
* “:”: List operator. (1:5 == c(1,2,3,4,5))
* [ ]: Extraction operator. [Row,Col], [Row,], [Col], [,Col], [Boolean,], [Boolean,Col]
  + Applicable to data frames and lists
  + df[1] == df[,1] == Column 1 of data frame df
  + df[1:5] == df[,1:5] == Columns 1-5 of data frame df
  + df[1,] == Row 1 of data frame df
  + df[1:5,] == Rows 1-5 of data frame df
  + df[“iso”] == df[,“iso”] == Column with name “iso” in data frame df
  + df[1,1] == Entry at Row 1, Column 1 of df
    - This will default to a single piece of data (number or string). To keep it as a single entry in a data frame, include the argument “drop = FALSE”: ( df[1,1,drop = FALSE] )
  + df[-1,] == df without its first row
  + df[df$iso == target, names(df)] == df[df$iso == target, ] == All columns, only those rows in which the “iso” column entry is the target.
    - The exception to this syntax is the is.na() function.
    - df[ is.na( df ) ] == All NA elements of data frame df, in list format.
    - df[ is.na( df ) ] <- 0 : Sets all NA elements within data frame df to 0.
  + df[[“iso”]] == list of iso column’s contents- no longer a data frame
  + If “list” is a list of data frames, list[[1]] retrieves the first data frame on its own- list[1] still treats it as an entry in the list, and will not allow access to the contents of the data frame directly.
* %in%: Boolean operator for containment of target. (x %in% y == is x in y?)
* %>%: Pipe operator. Passes the results of the line preceding it to the first argument of the function called in the next line.

**Commonly Useful Functions** *(use help() or ?? for details and syntax)*

*Note: does not include CEDS system functions*

* arrange( df, column names ): sort a data frame by column
* as.numeric() / as.character(): convert parameters into the specified data type
* aggregate(): aggregates data by specific id columns
* c(): concatenates the input strings into a vector, string and vectors, or vectors and vectors.
* cast(): reshapes melted data (melt()) back into wide format
* cbind(): combines two data frames by column
* data.frame(): Creates a data frame from its inputs
* duplicated(): Gives a boolean vector indicating which rows in a data frame are repeated, by specific columns
* filter(): Can be used similarly to extraction with a boolean condition
* getwd(): shows the current working directory path
* head(): displays first few rows of a data frame
* help(<function>): opens R documentation on the function passed as a parameter
* ifelse(x, y, n): returns a value of the same shape as x, with elements pulled from either y or n depending on whether the corresponding element of x is true or false
* is.na(): Boolean test to see whether the parameter is NA
* length(): Gives the number of entries in the parameter- for a data frame, # of columns
* match(): returns a vector of positions matching the elements of its first argument to its second
* melt(): Converts wide format data frames to long format- multiple id variables, but only one observation per column. Column names become entries in the “variable” column.
* merge(): Combine two data frames by shared columns.
* names(): When used on a data frame, lists the column names. (same as colnames() )
* nrow(): Gives the number of rows in a data frame
* order(): sorting. ( df[with( df, order( iso, sector, fuel ) ), ] ) [Also see arrange() ]
* paste0(): concatenates multiple strings
* rbind(): combines two data frames by row
* read\_excel(): reads an xlsx file [Requires readxl package]
* read.table() / read.csv(): imports files from a given filepath or URL
* select(): subsets a data frame by columns; can remove columns easily
* setwd(<path>): sets the current working directory
* subset(): takes subset of given data that meets Boolean statement criteria
* tail(): displays final few rows of a data frame
* unique(): gives a list containing only one instance of each unique entry in the parameter
* write.table / write.csv(): Exports and saves files made from R data objects

**Quirks**

* Lots and lots of packages with helpful functions- some are well documented, others are not.
* R syntax declares column names that start with numbers as invalid, and by default whenever a data table is read in, it will add an "X" to the front of any column names that would otherwise start with numbers.
  + Ex. “1960” will become “X1960”
  + There are ways to get it to drop the "X", but are not worth pursuing. If we dropped all of the "X"s, here are some of the effects:
    - Year columns could no longer be referenced using the "$" operator (e.g., df$X2010 would need to be written as df[["2010"]])
    - When a data frame with years as columns is melted, the resulting year column would be stored as a character variable, and would still need to be re-set for many of the operations that we would be performing. (e.g., df$year <- as.numeric(df$year)
  + Instead, recommend removal immediately before output of final product.
* Faster to apply functions to list and other data structure than to iterate across them.
* Partial matching: df$x will retrieve a column from df named “xsdfghj”, or anything starting with x.
* R will often interpret a list of strings (including a column of strings) as being factors instead (especially when using the read.table functions). This changes their behavior and can cause unexpected errors.
  + Spot factors by printing out variable values to the console. If you see “Levels:” and a unique list of contents beneath the variable’s actual value, it’s a factor.
  + Convert factors back to strings with as.character(). Alternately, if using read.table() or read.csv(), pass them the parameter “StringsAsFactors = FALSE”.
* In R, any number divided by 0 becomes infinity (Inf).
* Like in C, 1 (and all other nonzero numbers) are equivalent to TRUE and 0 is equivalent to FALSE. Be careful when writing functions that can take either booleans or numbers as arguments.
* Function locality applies in R just as much as many other languages.
  + Unless you are modifying a section or component of an existing variable that was declared externally, any variables declared within a function will cease to exist once the function returns.

**Bucket O’ Tricks**

* When calling functions, you can pass it parameters out of order by naming them in the call.
  + readData( “MED\_OUT”, “A.energy\_data” )
  + readData( file\_name = “A.energy\_data”, domain = “MED\_OUT” )
  + readData( file\_name = “A.energy\_data”, “MED\_OUT” )
* Create new columns by saying that they exist:
  + If data frame df has columns “one” and “two”:
  + df$three <- other\_df$data
  + df$three <- 5
* Identical:
  + df <- subset( activity\_db, activity\_db$iso == target )
  + df[ df$iso == target, names( df ) ]
  + df[ df$iso == target, ]
* Can alter specific sections of data frames
  + results[ results$iso == i, data\_cols ] <- data
* Can pull (and alter) by multiple different arguments
  + results[ results$iso == i & results$activity == act, data\_cols ] <- data
  + results[ results$iso == i | results$activity == act, data\_cols ] <- data
  + Note: Combined Boolean arguments within an extraction as seen above use only single instances of & (AND) and | (OR) instead of the double &&, || seen in regular Boolean expressions.
* Can pull from and alter entire dataframe
  + data\_new[data\_new == "start-year",] <- min( model\_base\_years )
* Identical:
  + df <- cbind( iso = df$iso, df[ data\_start:( len - 1 ) ] )
  + df <- cbind( df[ "iso" ], df[ c( "othercol1", "othercol2", etc. ) ])
* Can use re-ordering of a list of column names to shift the actual columns in a data frame
  + df <- df[ c( "sector","iso","etc." ) ]
* Can easily remove specific element(s) from a list through extraction:
  + all\_fuels <- all\_fuels[ all\_fuels != "process"]
    - Note: this does not work for data frames. Attempting it returns a list, as one cannot remove specific entries in a data frame without compromising its structure.
* arrange() does the same thing as with(order()), with less typing.
* If an object is considered a data frame, but prints a message along the lines of “source: local data frame [986 x 58]” and does not show all columns, it is likely to cause errors in standard data frame operations.
  + To coerce it back into normal data frame form, just cast it using as.data.frame().
* Ctrl-r on highlighted code within RGui or RStudio to run immediately.
* Ctrl-l in console to clear the console.
* Ctrl-d to duplicate the line of code where the cursor rests.
* Notepad++:
  + Automatically colors code appropriately and allows multiple files open simultaneously in tabs, but requires code to be copypasted into the console to run.
  + Ctrl-F to open the very useful find-and-replace feature
    - Has options to find, replace, replace all, and replace all in all open files: very useful for changing names of files and variables across large scripts or multiple scripts.
    - It is preferable to use find-and-replace for global renaming so that no errors are made and no instances are missed.
    - (There are likely to be similar features in most code editors.)