# Basics - not only the basics of using R, but also the basics of using

# code as a tool to get something done... that means thinking about

# what you want to do and figuring out how to make it happen!

################################################

## LESSON 1 - Document within your code ##

################################################

# You can comment above and to the right of lines of code or you

# can comment our an entire line of code

# Age of myself and two of my brothers

**age <- c(37, 49, 50)**

#age <- c(35, 47, 48) # line commented out because old data

**mean\_age = mean(age)**  # added stored variable: Bob (01/19/2016)

###########################################

## LESSON 2 - Dynamic Code ##

###########################################

# A good way to learn to code is to try to re-write simple functions

**sum(age)/3**  # static, so will not 'update' with change

**sum(age)/length(age)** # dynamic change

# Let's see why writing dynamic code is better...add a 4th element

**age <- c(36, 48, 49, 51)**

**mean\_age = mean(age)**

# Re-run same two lines

**sum(age)/3** # incorrect

**sum(age)/length(age)** # correct b/c line of code is dynamic

# Now let's look at data as characters

**name <- c("Bob", "Arthur", "Tracy", "Rod")**

# Use the function 'rbind' to bind together two vectors

**dat1 <- rbind(age,name)**

# Let's be more conventional (remember, variables as columns) by

# Transposing our vectors... but how to transpose in R?

#############################################

## LESSON 3 - Build Your Resources ##

#############################################

# Knowing how to transpose is extremely important because

# some functions act on rows while others act on columns

# So.....how do we find out how to perform a transpose?

# 1) type 'transpose' into the Help search box, or

# 2) same thing as typing ‘??transpose’ into the R console

# 3) simply google -> transpose in r

# We need the R function, t(), which transposes vectors & matrices

**dat1 = t(dat1)**

######################################################

## LESSON 4 - Be Mindful of Behind the Scenes Actions ##

######################################################

# Here we will see variable class can change when combining objects

**mean(dat1[,1])** # This will produce an error

**class(dat1[,1])**  # We see that age, changed from numeric to character

# We could force it to be numeric

**mean(as.numeric(dat1[,1]))**

# But this doesn't seem practical...let's ask the all-knowing Google!

# Google -> combine numeric and character in r

**dat2 = list(age,name)** # using the list() function is suggested

# Verify by investigating dat2 in the workspace

####################################################

## LESSON 5 - Break Your Problem into Smaller Pieces ##

####################################################

# Now, how do we call from this list and find the mean age?...

# Take this one step at a time

# 1) How do you call something from a list? -> Use double brackets

**dat2[[1]]**

# 2) Now combine with the mean() function

**mean(dat2[[1]])**

# OK, now let's look at creating a matrix of values & how to index

# Create a matrix using the matrix() function

#M <- matrix(c(2,4,6,8,11,13,15,17, nrow=4) # What is wrong with this line of code?

# Does simply adding a ')' in the console fix it? -> view result of M

**M <- matrix(c(2,4,6,8,11,13,15,17), nrow=4)** # Note that view of M doesn't update -> re-load

# How to tell what something looks like?

# 1) Use Environment window for class/size info, and for display options

# 2) Highlight and run variable to display in console

# 3) Use the print() function

**print(M)**

# Many of the errors I make are due to indexing mistakes -> difficult to debug sometimes

# Indexing a full row or column of a matrix

**M[3,]** # third row of matrix m

**M[,1]**  # first column of matrix m

#################################################

## LESSON 6 - Smart with Naming Variables ##

#################################################

# Being smart about naming variables is useful -> simple, but not too simple

# Which do you think is better?

**mr2 = M[2,]** # or

**second.row.of.matrix.m = M[2,]**

# Why? mean(mr2) vs mean(second.row.of.matrix.m)

# What if the vector, matrix, array, etc. is too large to display? -> index a subset

# For example:

**M2 = matrix(rnorm(10000, 19, 2), nrow=100)**

**M2[1:4,1:10]** # Show how re-sizing Console window influences display

# or you could simply view uing the display option from the Environ window

# Let's revisit the mean() function to find column means of matrix m

**mean(M)** # Is the result what we expect? -> what is the result?

# Let's say I want the mean of each column...what do we do?

**mu.colM = colMeans(M)**

# And what if we want row means?

**mu.rowM = rowMeans(M)**

# Or...recall what we've learned so far -> transpose and using a f\_n w/in a f\_n

**mu.rowM2 = colMeans(t(M))**

## Loading data into R

## First, download the sales dataset from the following url

# <https://github.com/Ying-Ju/Miami-DataFest-Workshops>

# Download as zip file & extract sales.csv from the zipped folder

# We can use the 'Import Dataset' tool

# Or we could use the read.csv() function

# BUT!!!, there are options you should understand

# most importantly...how to properly state the file directory

**dat <- read.csv("C:/Users/Bob/Dropbox/Miami/DataFest/Datasets/sales.csv", header=T)**

# Explore your dataset and check that everything is OK first

# Note that the dataset already has header names (default is header = T)

# Create a histogram of sales, the response

# Recall what we've learned about going step by step . . .

# How do we call out 'sales' from 'dat'?

# What class is 'dat'? -> useful b/c it dictates what syntax to use

# Google -> access variable in data.frame in r

# You should find that a ‘$’ sign is used to access variables within a data frame

# 1st, check if you're doing what you think you're doing -> recall viewing subset using indices

**dat$sales[1:5]**

# Looks good, so how do we create a histogram? -> hist() f\_n

**hist(dat$sales)**

# or

**hist(dat[[1]])**

# Can you see the difference in the plot titles?

################################################

## Lesson 7 - Performing Linear Regressions ##

################################################

# Simple Linear Regression

**reg1 <- lm(dat$sales ~ dat$GPA)**

**summary(reg1)**

# Is this a good model to predict sales? Why or why not?

# Multiple Linear Regression

**reg2 <- lm(sales ~ GPA + experience + certification + days\_late, data=dat)**

**summary(reg2)**

# Now you try...

# Fit a reduced model using what we've learned from the full model

**reg3 <- lm(sales ~ experience + certification + days\_late, data=dat)**

**summary(reg3)**

# Can you interpret the results of the final fitted regression model?

# Next let's look at some diagnostic plots

**plot(reg3)**

# Do we see any possible issues?

**hist(dat$experience)**

**hist(dat$certification)**

# Let's looks at observation #15

# How can we predict with our model?

**y15 = 2.47 + (2.21\*50) + (1.52\*13) - (0\*-2.43) # PEMDAS followed but () help to organize**

# or

# What about the lm() f\_n? -> Where to look to find info on how to do this? -> type lm() in Help search

**reg3$fitted.values**

################################################

## Lesson 8 - Clean Coding Comes in Time ##

################################################

# Things can become very messing when you first begin to code. IT IS OK!!! -> Just improve as you go!!

**reg3.bs = reg3$coefficients** # This is simple

a**s.numeric(reg3.bs)%\*%c(1, as.numeric(dat[15,3:5]))** # This is not so simple, but still works

##################################################

## Lesson 9 - Downloading & Installing Packages ##

##################################################

# Next we'll download and install the R package ggplot2 to create cleaner histograms

# Most packages are downloading from the CRAN, an online archive of R routines

**library(ggplot2)**

**ggplot(dat, aes(experience)) + geom\_histogram()**

**ggplot(dat, aes(experience)) + geom\_histogram(binwidth = 6)**

# What can we assume from the histogram? -> How does this help us "get to know" the company?

# More importantly, we should think of questions we can ask those representing the company!!

###################################################

## Lesson 10 - Run Entire Routine & Check for Errors ##

###################################################

# Be sure you are ready to do this!! Save Often!!

# Clear Console using Ctrl+L

# Clear Plots

# Clear workspace in Environment window

# Ctrl+A to select all lines of code

# Ctrl+Enter to execute

# Scroll through Console -> Do you see any errors?

# Why did I clear the workspace, plots, and console?

# Important to run line by line, then section by section, etc.

# Save as you go, maybe create versions of your routines along the way, and stay organized!!!

# Above all else, HAVE FUN AND LEARN FROM OTHERS!!!!!!

# Thank you for your interest in coding!

#####################################################################################

# Next Workshop will focus on loading/managing medium sized datasets, model fitting techniques, creating functions, and other useful tips when coding.

# Hope to see you there!