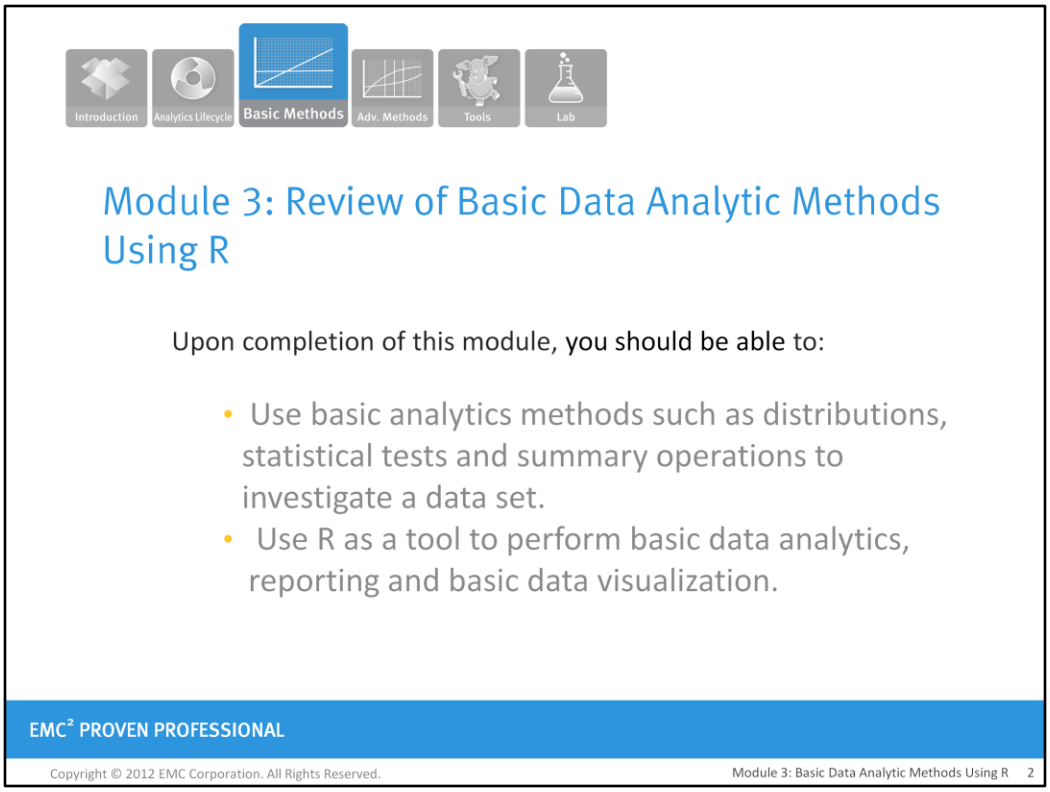
Basic Methods

Module 3 – Review of Basic Data Analytic Methods Using R

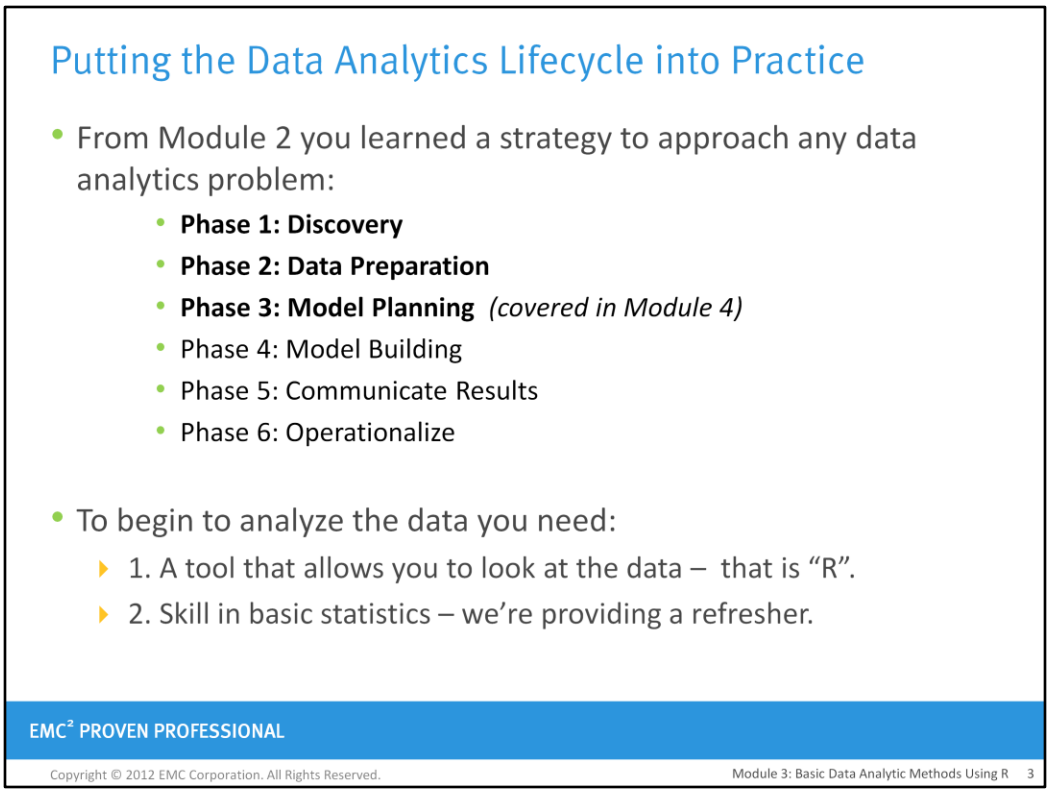
EMC² PROVEN PROFESSIONAL



These are the objectives for this module.

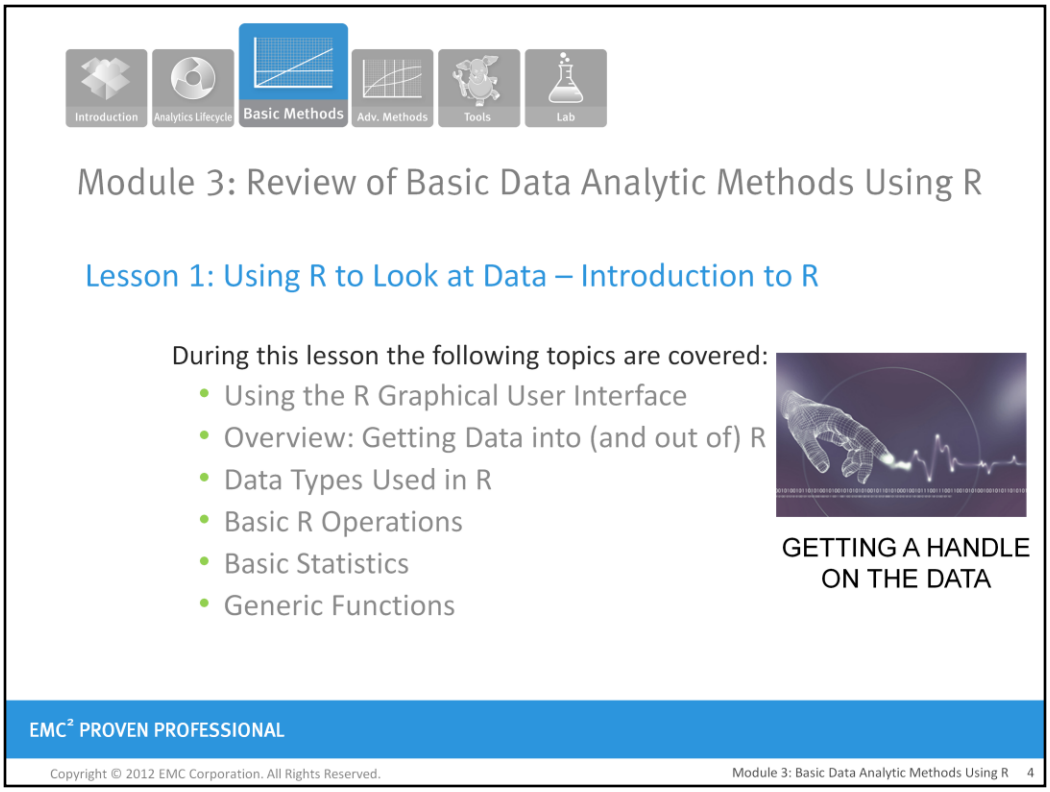
Specifically, after completing this module, you should be able to:

- Use the R package as a tool to perform basic data analytics, reporting, and apply basic data visualization techniques to your data.
- Apply basic analytics methods such as distributions, statistical tests and summary operations, and differentiate between results that are statistically sound vs. statistically significant.
- Identify a model for your data and define the null and alternative hypothesis.



Module 2 presented the data analytics lifecycle. The first three phases represent our initial exploration of our data and the results of that exploration.

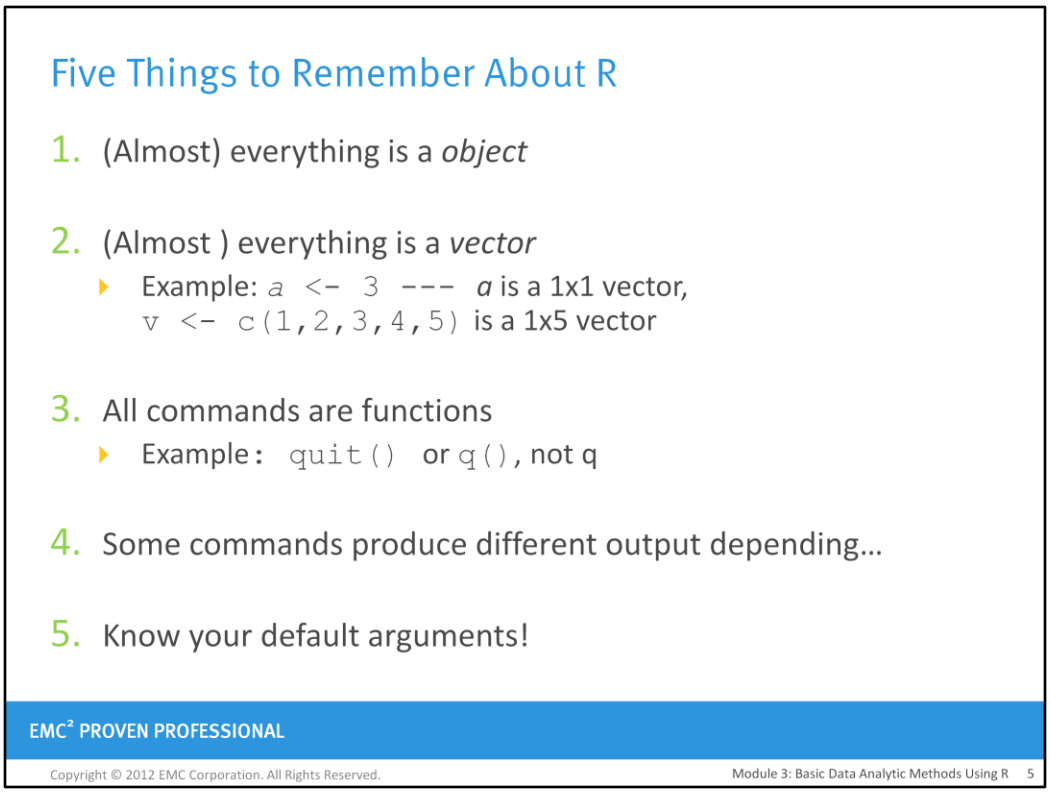
In order to begin to analyze the data, you need a way to "look" at the data and a tool to work with and present the data. What does "look" mean here? You need a way to "look" both in terms of basic statistical measure and in creating graphs and plots of that data in order to visualize relationships and patterns. Our tool of choice for this activity is the **statistical package, R**.



This lesson covers the topics listed above. The techniques you learn here will allow you to handle your data and get to know it: that is, **acquire, parse, and filter your data**.

We'll be using R to process the data and as well as to create basic summary statistics and datasets for analysis.

These processes will allow you to understand what you have, and apply these techniques to any data analytics project.



R is a big, complicated, messy, powerful, extensible framework for computing and graphing statistics. Written as a freeware version of the S language, it's widespread availability and use have resulted in several vendors supplying R interfaces to their products.

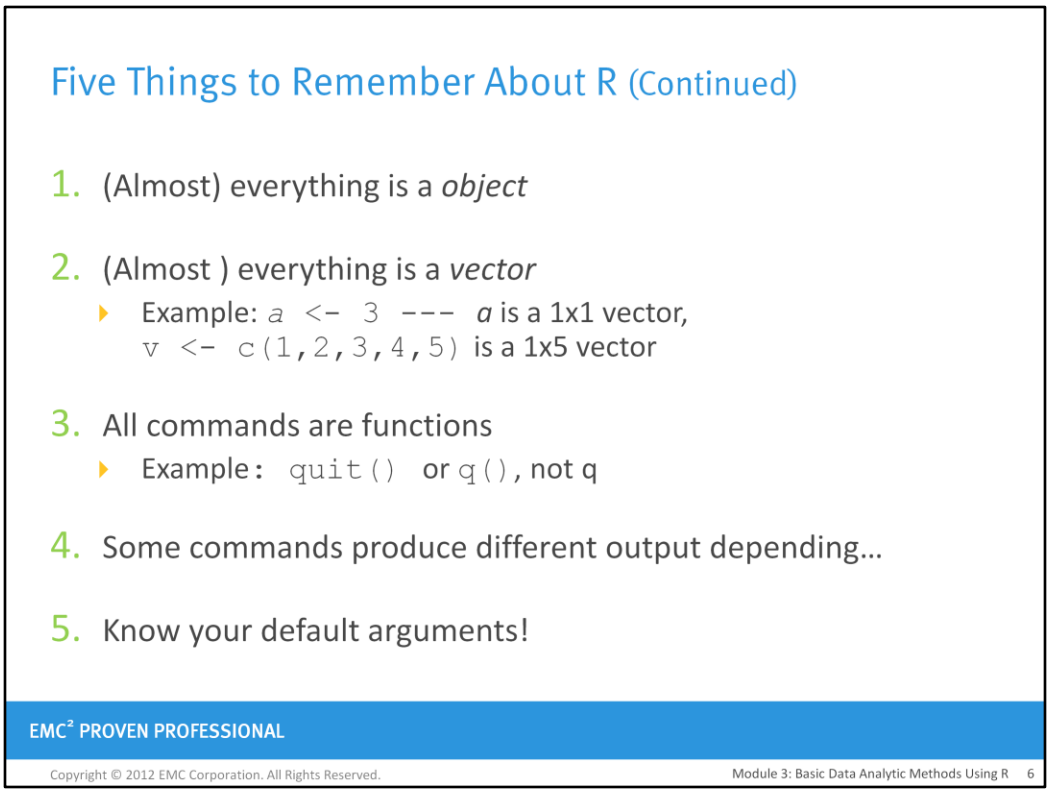
There are five things that you should remember about R. Doing so will help you in thinking about how to work with R, and, more importantly, when R proves stubborn and insists that it doesn't know what you're talking about.

First thing to remember is that underneath it all, **R is an object oriented language**. That means, for example, that the expression "**`x <- 3`**" is actually invoking a function of the x instance: e.g. **`x.assign(3).`**

Second, **almost everything in R is expressed as a vector or a group of vectors**. Although x <- 1 looks like a scalar variable, it's actually a 1-dimensional array (vector) with length 1. Similarly, **`v <- c(1,2,3,4,5)`** is a 1x5 vector (**`length(v)`** is 5).

As regards data structures, almost everything in R is defined as a vector: each element of a vector can be addressed by a numerical index (e.g. v[3] … subscripts in R are 1-based as in Python, not 0-based as in Perl or C). That means that scalar values (such as x) are actually a vector of length 1. The command (function) to create a vector is **`c()`**, and can contain all numbers, all character strings, or a mixture of the two.

Third, **all commands in R are actually functions**. Hence, you must type in either **`quit()`** or **`q()`** to exit R. q is a variable within a R workspace. Simply typing in q will provide you with a definition of that function (the same as **`str(q)`**).

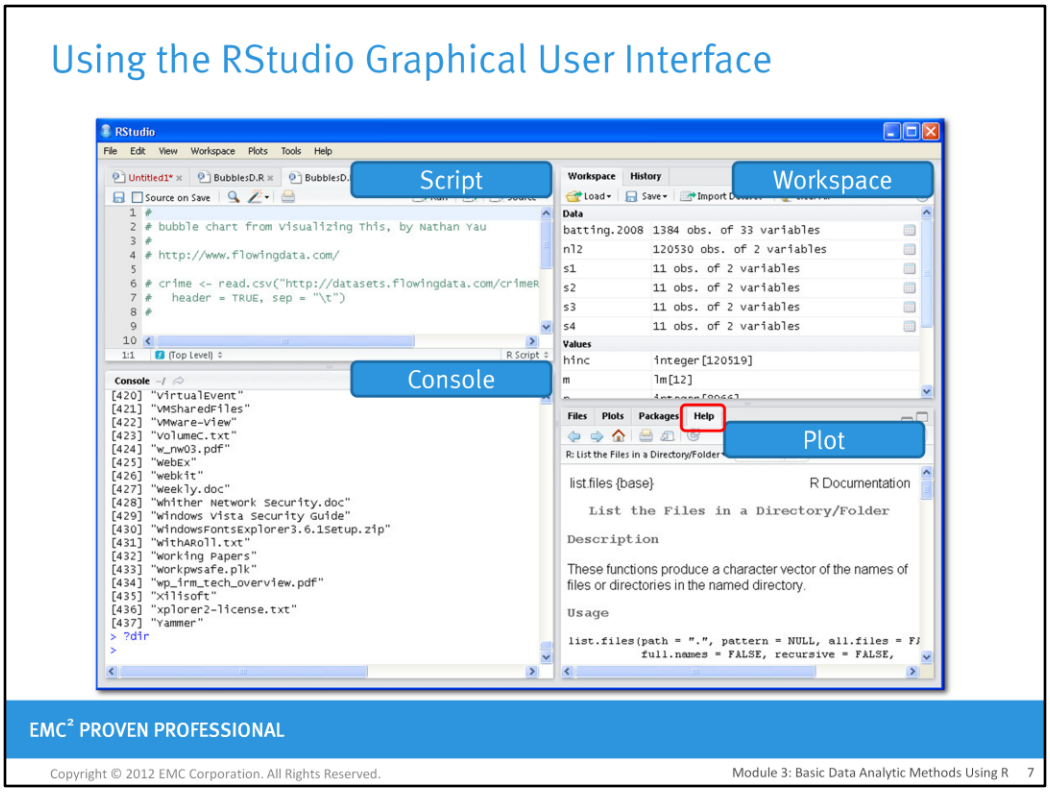


Fourth, since R is object oriented, and since we have said various operators are implemented as functions, it's no surprise that **there are multiple commands in R that are much like virtual functions in other OO languages**. Consider the *summary()* function. It's behavior will differ markedly depending on the class of the object passed as an argument. For instance, *summary(x)* will print basic summary statistics about each row if *x* is a data frame, but may generate a mosaic plot if *x* is a table. The *plot()* function works the same way. (We'll see an example of that in one of our labs).

Finally, **most commands in R have a large number of default arguments.** For example, the lm() function (univariate regression), looks like this:

lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)

For simple regression, the usual invocation is *lm(var1 ~ var2, data=<dataset>).* Other parameters take on a default value, or may not be needed based on the type of variables provided in the function call. Usually the simple invocation "just works" given the choice of default values. However, you may need to apply one of more of these parameters in order to get different visual results. The command *help(lm)* or *?lm* will provide more detail in the style of *nix manual pages – the documentation will describe the arguments, types, default values, etc., but it won't explain how or when to use this particular function.



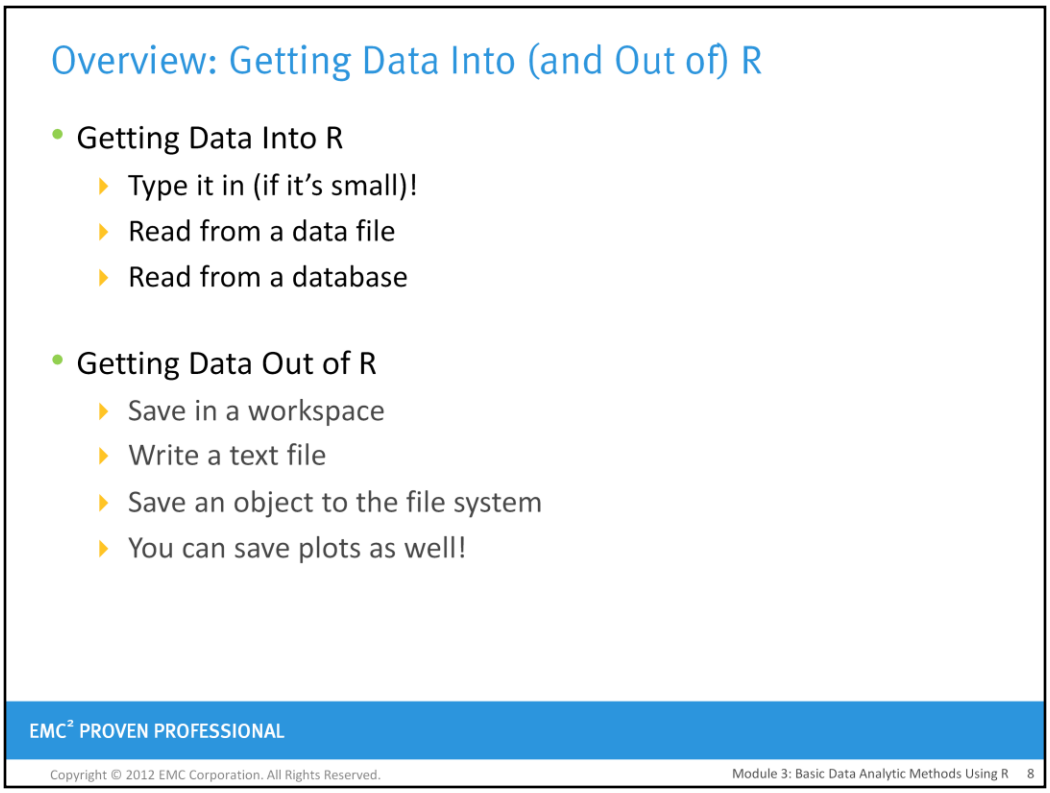
R comes in multiple flavors. The heart of the software is a command-line interface (CLI) that is very similar to the BASH shell in Linux or the interactive versions of scripting language like Ruby or Python.

The Window version of R supports multiple GUIs. The default GUI is invoked by simply invoking the R program either via the command line or via the Windows GUI. Within R, the *rcmdr* interface offers a more task-oriented view. The *Rattle* interface is another framework that is more task oriented: a user can load a dataset and automatically perform certain tests. Finally, ***RStudio*** **provides both a desktop and a Web browser interface. This is the UI that we will be using in this course.**

RStudio offers three panes that are fairly common to all R GUIs The upper left pane is for script editing. The lower left pane is the R console itself, where all commands are executed.

The lower right pane is the help screen, invoked by the `help(<topic>)` command, with which you will become very familiar, as well as tabs for file in the current directory, plots, and a tab that enables you to view which *packages* are available locally or can be downloaded from CRAN, the comprehensive R archive network. Finally, the upper right pane is unique to *RStudio*, and offers a table-oriented view of the variables stored in the current R workspace. Clicking on a variable or data structure in the workspace window will display the values of that object in the script window as a separate tab.

Note that many panes have multiple tabs that offer different views on the workspace: take a moment to familiarize yourself with their content during our first lab. Each pane can be grown or shrunk by clicking on the grow boxes in the upper right hand corner of each pane.



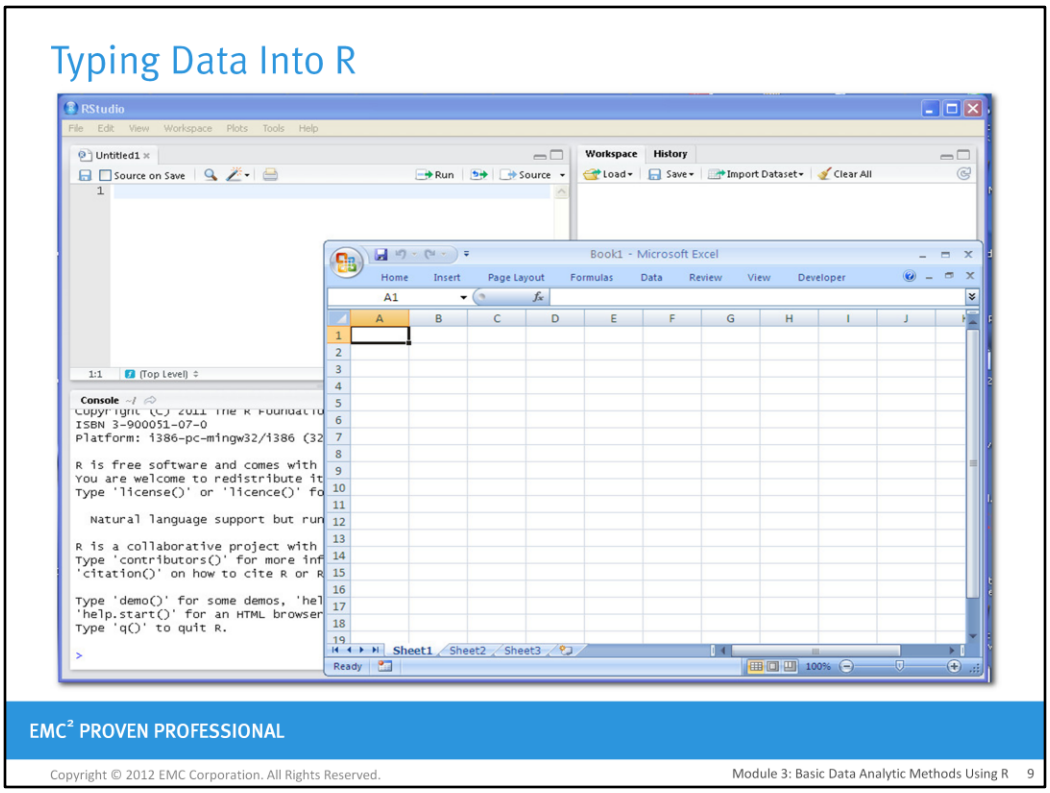
How do we input data into R? The first method, and sometimes the simplest, is: type the data in! This is a good method for small data sets.

You **can always read raw data from a data file using *read.table()*.** There are several help functions for reading delimited data as well as fixed length fields; the scan() function permits reading fields of variable length.

You **can also read data in from a database.** Both DBI (Java) and ODBC (Microsoft) interfaces are supported. Drivers are available for most popular databases.

Once in, there are **several ways to save data from an R workspace**. You can save the entire workspace and restore it in a later session. You can also write a R data object (usually a data frame) as a text file with field delimiters. Finally, you can save an R object or objects as a binary file, which can be loaded back into another session.

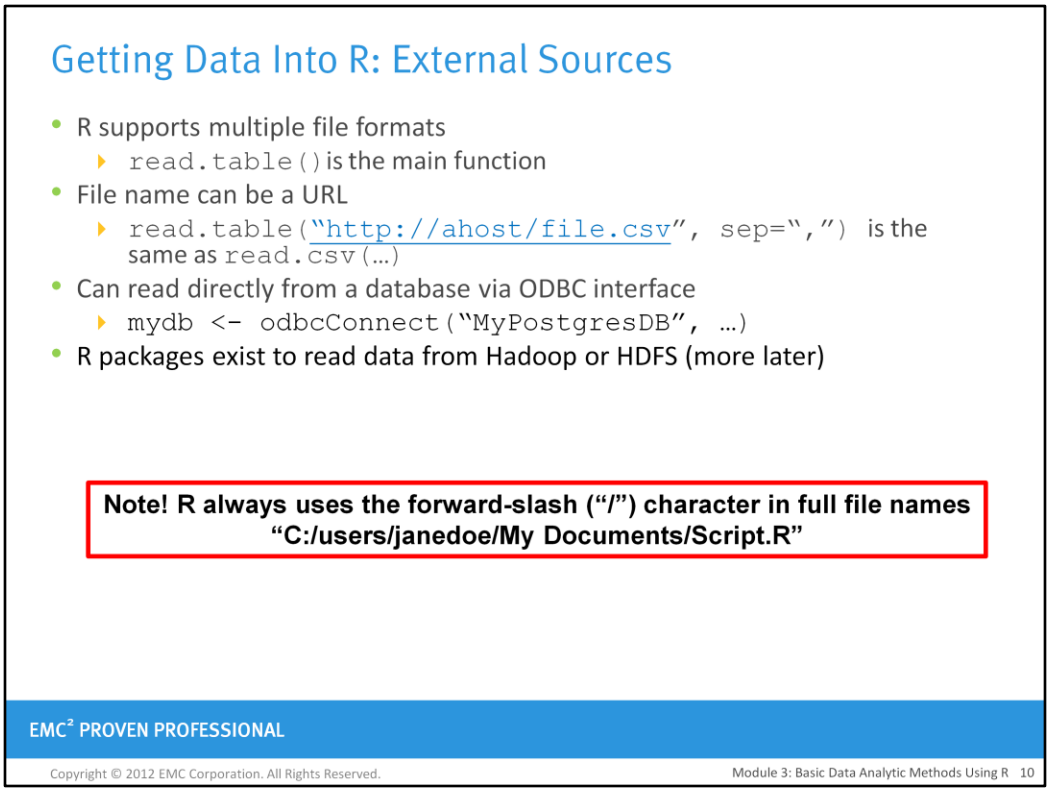
R also allows you to specify a particular output device, which is the standard way to save the results of a graph or a plot. RStudio allows you to save the graph as an image (.jpg, etc.) directly from the plot window.



Data can always be created by typing in values. For example, the vector assignment

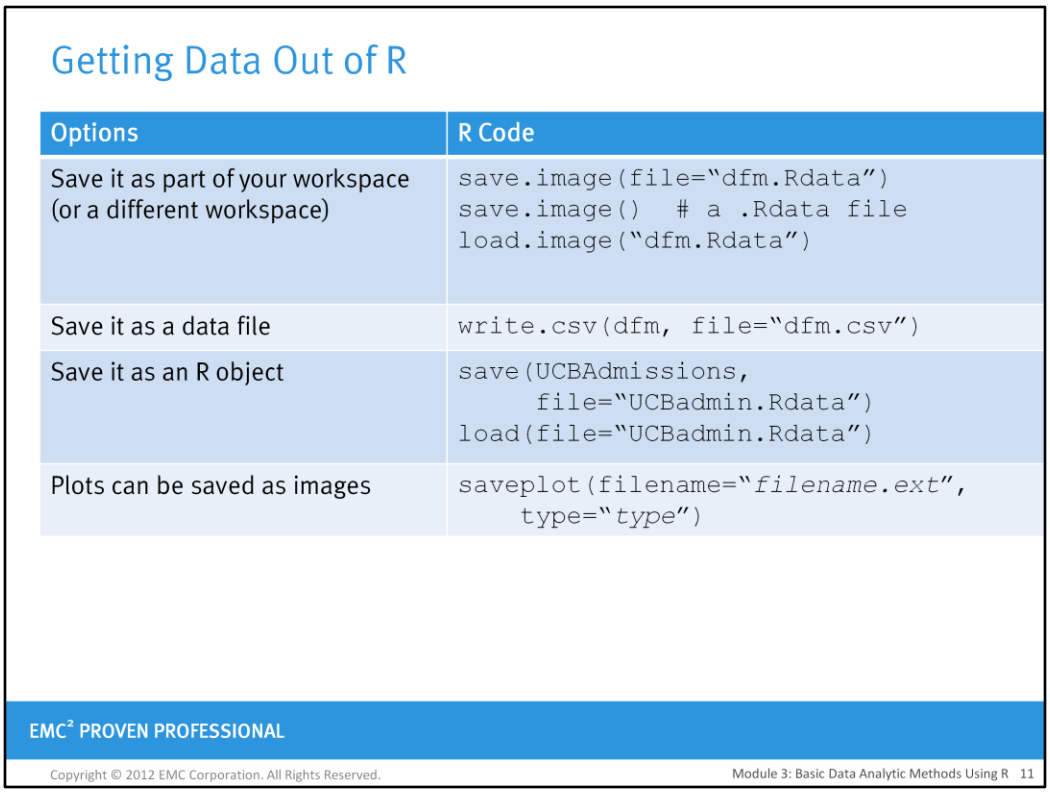
v <- c(1:10) creates a vector of 10 elements numbered 1 through 10. More complicated data structures **can be created by composing that data structure from a group of other data structures**. First create an empty data structure, and fill it in via the editor or cut and paste from external files. The R script editor allows tweaking of input, and is easier that editing keystrokes in the console window. Remember that we can transform from one object type to another, so we could read data in as a matrix and use the *as.data.frame()* function to create the data frame.

This graphic shows the use of Excel with RStudio. Unfortunately, RStudio does not yet provide the ability to edit matrices or data frames. The standard R-GUI interface allows you to create an empty data object, and then edit that object via the *edit*() or *fix*() functions. When creating a data frame you can create and name your variables as well (for example, LastName (character), etc.).



R has the ability to read in data in many different formats. The *read.table()* function is the most used, although there are multiple helper functions such as *read.csv()*, *read.delim()* and *read.fwf()* for reading fixed-length fields. Multiple import functions also exist, including reading in data from SPSS, SAS, Sysstat, and other statistical packages. The file name argument to `read.table()` can also be a URL: this is useful in reading a data file from the Internet. Consult the help subsystem (*help(read.table)* for more options).

R *always* uses a forward slash "/" as the separator character in full pathnames for files. A file in your documents directory in Windows would be written as "C:/users/janedoe/My Documents/Newscript.R". This makes script files somewhat more portable at the expense of some initial confusion on the part of Windows users.



R utilizes a *workspace* that consists of a collection of data objects, code libraries, and named data sets. Each workspace also support multiple *environments*, although we won't address this issue further (see the R reference manual for more details).

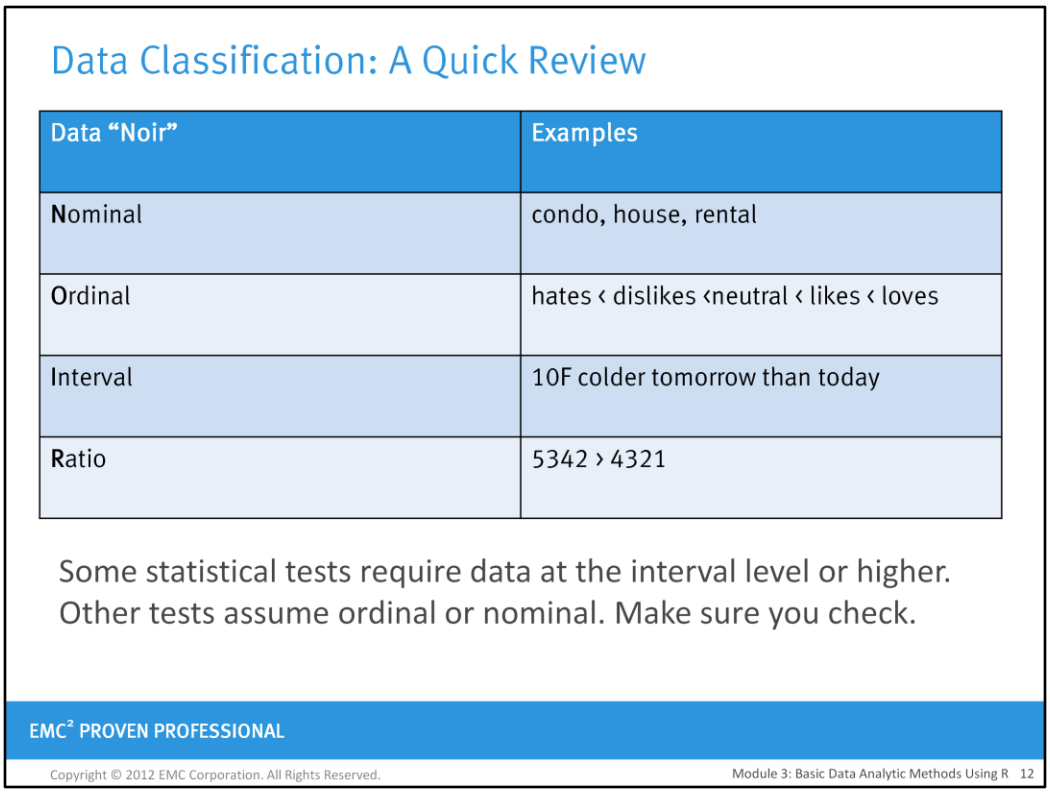
R libraries that are not automatically loaded can be loaded into the workspace via the `library(<dataset>)`; datasets can be loaded into R via the `load("<dataset>")` command.

Packages that are not part of the standard distribution can be obtained via `the install.package("<packagename>")` command (note the use of double quotes).

Data objects in R can be exported either as .csv file, or in native format (`save(<object name> …, file="<full file path name>")`) (usually with a .Rdata extensions) and then reloaded into the R workspace via a `load(file="full path name")`. This will repopulate workspace with that object or objects.

If you choose to save your R workspace, it can be reloaded automatically when R is re-started. Other workspaces can be loaded into R with `the load.image()` command.

Lastly, plots can be saved to a file using the saveplot() command. Most platforms will allow *.jpg* and *.png*, but check your local R documentation for your particular platform.

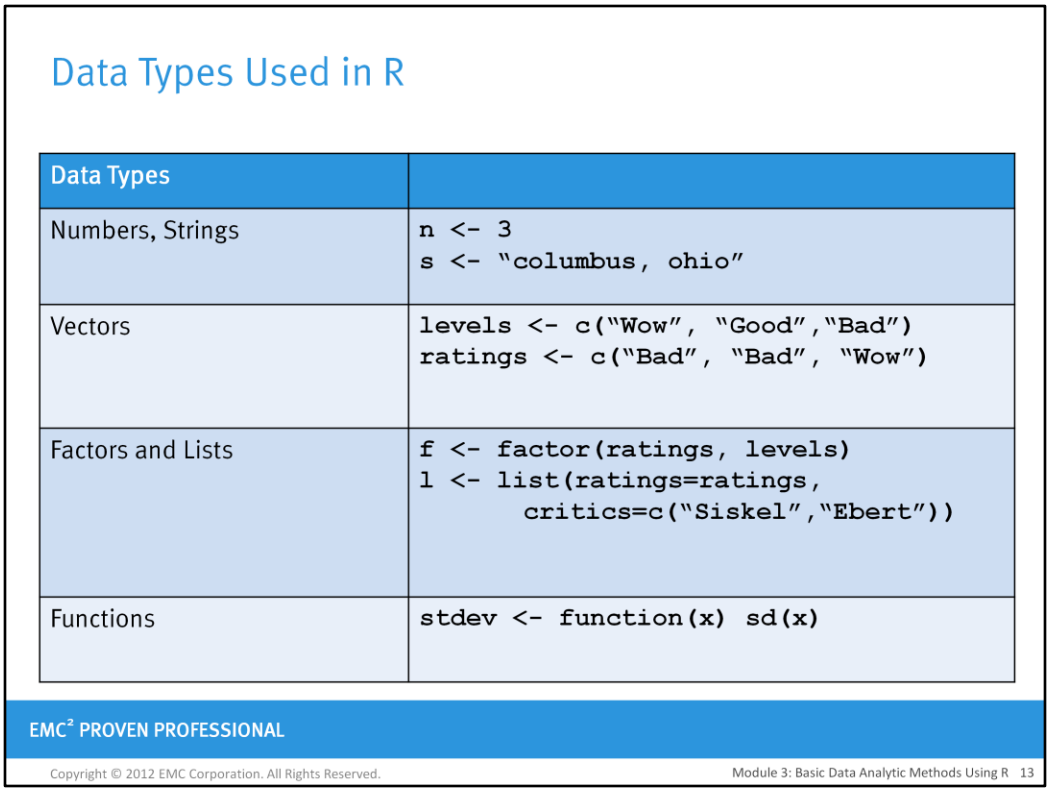


Recall our general classification of the measurement of data. Data can be either nominal, ordinal, interval or ratio level of measurement. Nominal is simply a label, there is no order implied. Ordinal data, on the other hand, does have an implied order. For example, I may assign the values of "Good", "Better" and "Best" such that best > better, and better > good. However, I don't know the measure of the distance between each value.

Interval data has a fixed value of distance between each element, but an arbitrary 0 point (temperature is an example of this level of measurement). We can distinguish one element from another, but we can't say that 30 degrees is ½ as cold as 60 degrees. Ratio data, on the other hand, does have a meaningful zero point (e.g. dollars spent on clothing: $0), and $20 is twice as much money as $10.

Some data can be converted to another form. By encoding an ordinal level of measurement, we can generate a single measure of whether someone approves of something or not. A mean value of 3.5 for a Likert scale (coded 1:5) could be interpreted to imply that generally people were positive about an event, but we don't know if everyone was mostly neutral or varied between love and hate. In this case, viewing a table of responses would be preferred. We can, however, recode binary values: for example, we can code "female" as 1 and "male" as 0 and determine gender balance.

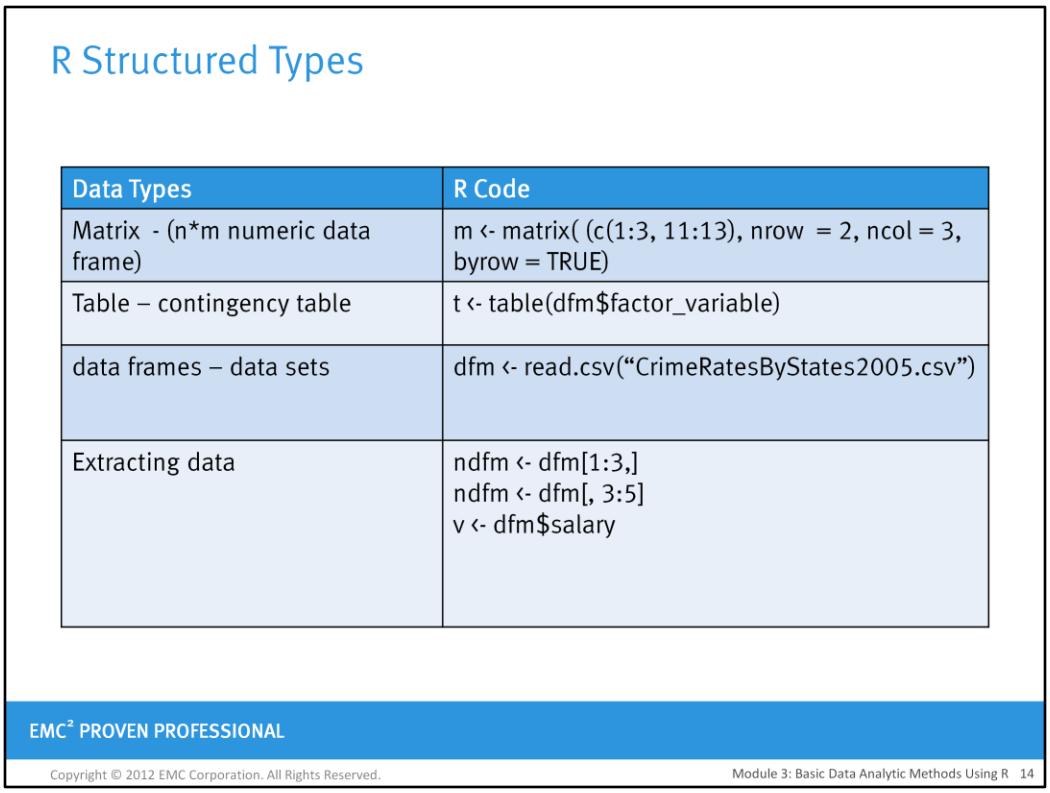
When choosing a statistical measurement, ensure that you have chosen one that is compatible for your data. Numbers will be calculated in some instances, but the result is misleading at best.



The workhorse data types of R are the vector and the data frame. Recall that (almost) everything in R is an object and a vector. Numbers and strings are 1 element vectors (that is length(n) == length(s) is true). Vectors can be numeric (c(1,2,3)) or character (c("WoW", "Good", "Bad")) or mixed (c(1, "two", 3)). **Mixed vectors are always considered to be character.**

Factors are categorical variables. If the available data doesn't include a particular label, it can be supplied as the 2nd argument to the `factor`() command. **Lists** are comprised of a set of named vectors. In the example above, we have defined two character vectors, levels and ratings. We create a factor, *f*, using ratings as our values and levels as the allowed levels, and then create a list structure using our ratings vector and a new vector for critics.

You can write your own functions in R. You can alias an existing R function as demonstrated in the example above: std(x) simply calls the R function sd() to compute the standard deviation of a vector, or your function can be arbitrarily complex. See help("function") in the on-line help for more details.



R structured types are the matrix, the table, and the data frame.

The matrix is what you think it is: an N by M array usually consisting of numeric values.

Tables are our old friend contingency tables, especially useful for observing nominal or ordinal data.

Finally, data frames are the real workhorse of R. These structures reflect most directly a dataset view of the world, where each row (record) contains several data fields. Usually rows are ordered by number (1..n) as opposed to tables, where rows are named entities ("High", "Medium", Low").

There are several ways to extract data from a structured type. You can select as subset of rows (*dfm[1:10,]*) or a subset of column (*dfm[,3:4]*). You can assign a column to a vector, and that vector will take on the resulting type (numeric, character, etc.) These "slices" can be transformed into other types by using the `as.<type>` function (e.g. `dfm <- as.data.frame(t)`).

Why does this matter? There are two reasons:

1. **knowing what the class of an R variable is (via *class(v)*) helps us understand where and when it can be used in a function, or it may need to be converted into a different representation (*foo <- (as.data.frame(t…))***
2. **Knowing the type of the underlying data helps us understand when data conversion is needed. Sometimes what appears to be numeric data is encoded as character strings ("12345" != 12345). Hence, in order to perform certain calculations, we may need to convert data (*as.numeric(t$age)*).**

Basic R Operations on Vectors

Function	R Code
Operations on Vectors	`v <- c(1:10); w <- c(15:24) ; nv <- v * pi ; nw <- w * v`
Vector transformations	radius <- sqrt(d$population)/ pi) t <- as.table(dfm$factor_variable) pct <- t/sum(t)* 100
Logical Vectors	v[v < 1000] ndf <- subset(dfm, d$population < 10000) nv <- v[c(1,2,3,5,8,13)]
Examining data structures	`dim(dfm); attributes(dfm) ;` `class(dfm); typeof(dfm)`

EMC² PROVEN PROFESSIONAL

Recall that a vector is a 1-dimensional array with a single data type (either character or numeric). We can **perform several different transforms** on a vector: multiplying each value by a scalar, creating a new vector by multiplying one vector by another, etc. We **also can transform the contents** of a vector by performing a transform on each element. If I have a vector called d$population, I can create a new vector as radius <- sqrt(d$population)/pi.

An example of this kind of manipulation is illustrated by creating a table using a factor from a larger dataset. This results in a table where each element of the factor has a count of the number of times it appears in that dataset. We can then create another vector containing percentages using the statement pct <- t/sum(t)*100, and create a second row in the tables via the t <- rbind(t,pct).

Logical vectors are created whenever an expression is used as an index. In the case above, a new vector is created with values of TRUE if the value of a particular element of v is < 10000. Any element of v that is marked as true is then added to the new vector. This is useful for creating subsets of larger data sets, as we shall see later on in this module. The subset() function provides another way to create a subset of values; the use of a specific range of indexes can be used as well (here we create a new vector consisting of values corresponding to the first six values of a Fibonacci sequence.)

Descriptive Statistics

Function	R Code
View the data	head(x); tail(x)
View a summary of the data	summary(x)
Compute basic statistics	sd(x); var(x); range(x); IQR(x)
Correlation	cor(x); cor(d$var1, d$var2)

EMC² PROVEN PROFESSIONAL

One of the first things to consider when receiving a dataset is to validate your assumption. Is the data clean? Does it make sense? I personally use `head(ds)` and `tail(ds)` to look at the 1st and last values.

The next command is `summary()` that provide the minimum, maximum, median, mean and the 1st and 3rd quartile values. (Compare this against the values returned from the `fivenum(ds)` function.)

Other functions include `sd` (standard deviation), `var` (variance), `range` (low value and high values), and `IQR` that displays the interquartile range (difference between 1st and 3rd quartiles). The `cor`() function computes the correlation between variables in the dataset, or, more specifically, the vectors provided as the values of x and y.

Generic Functions

- Also known as method overriding in OO-land
- Specific actions that differ based on the class of the object :

Code	Function
Plot the variable x	plot (x)
Histogram of x	hist (x)
Internal structure of x	str (x)

- Good for initial data exploration (more later)

EMC² PROVEN PROFESSIONAL

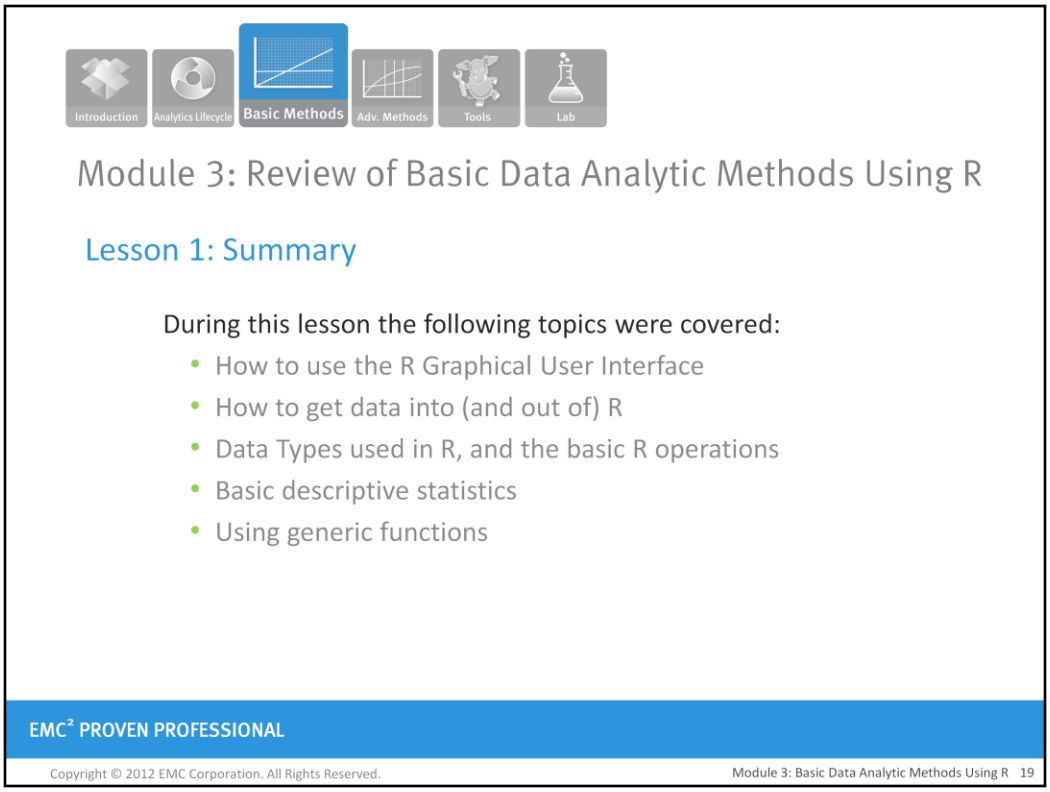
R makes use of a number of generic functions (we'll call them that because they explicitly take an object as their 1st argument, instead of the more OO notation of object.print()). In a strict OO language, these would be called virtual functions or methods and overridden by each class that wanted to make this capability available (consider the toString() function in Java). Such functions can have multiple parameters that affect their behavior. Review the help(plot) documentation as an example.

Check Your Knowledge

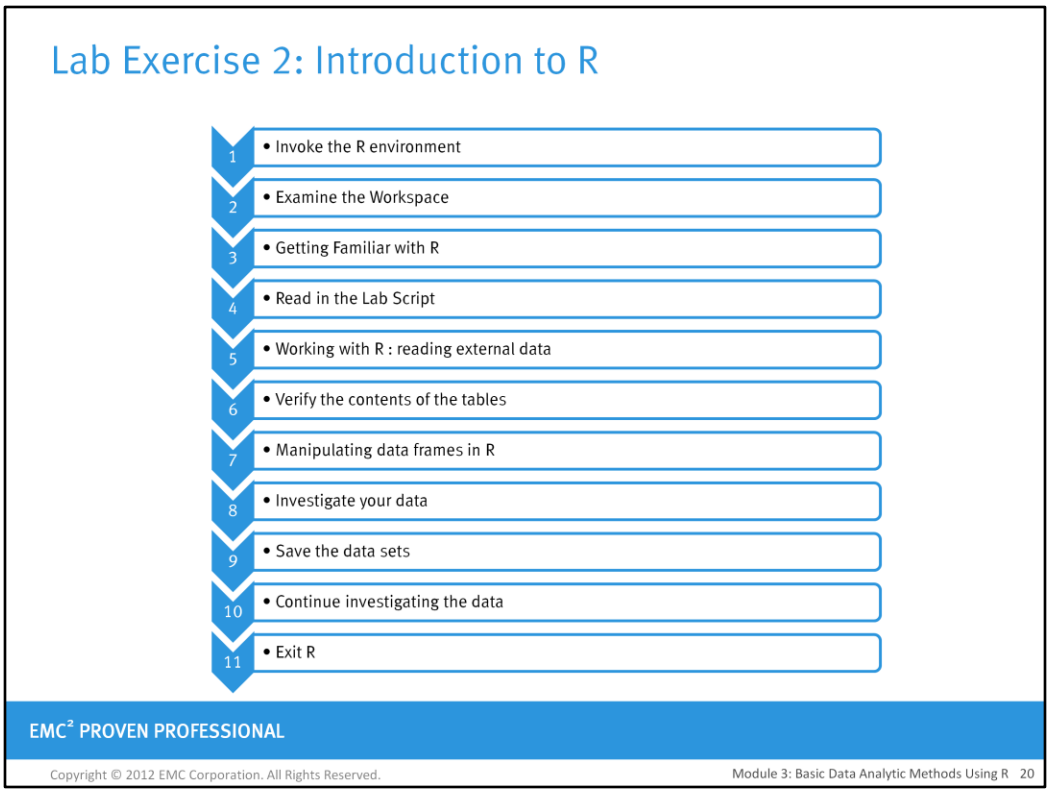
- Which data structures in R are the most used? Why?
- Consider the cbind() function and the rbind() function that bind a vector to a data frame as a new column or a new row. When might these functions be useful?

EMC² PROVEN PROFESSIONAL

Please take a moment to answer these questions.



This slide contains the key points covered in this lesson. Please take a moment to review them.



This slide captures the workflow from Lab exercise 2. Please take a moment to review each step before starting the lab.

Lab Exercise 2: Introduction to R

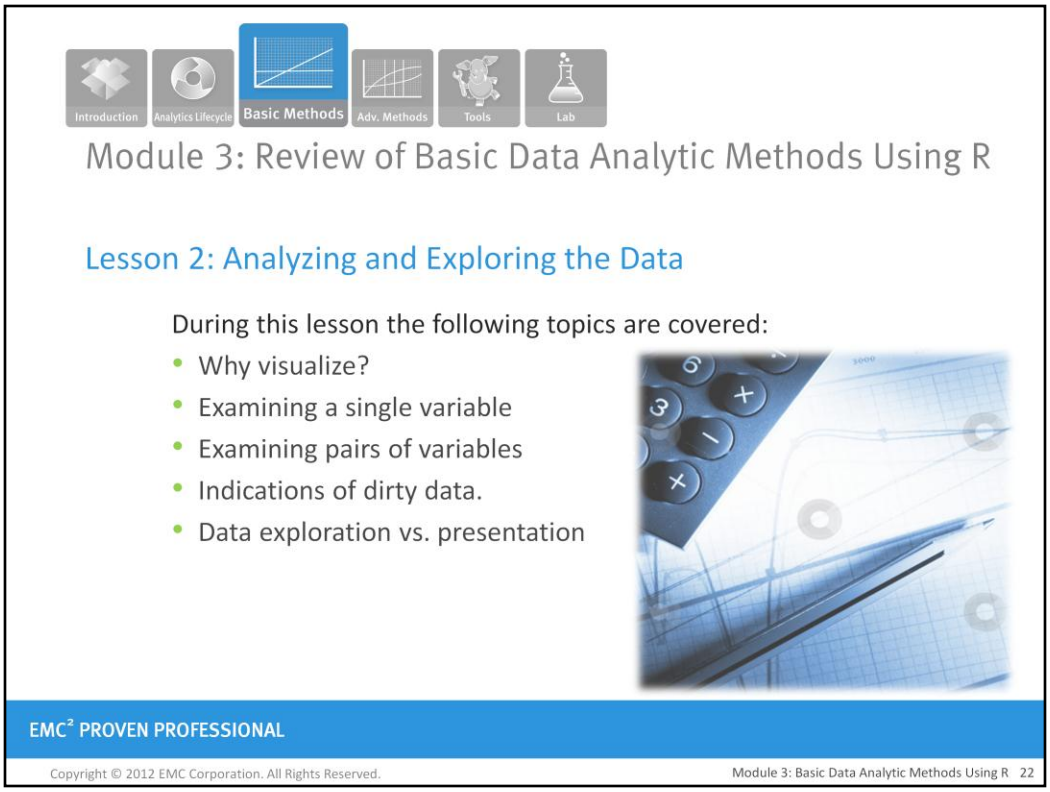
This lab is designed to investigate and practice working with R and using it to examine data.

- After completing the tasks in this lab you should able to:
  - Read data sets into R, save them, examine the contents

EMC² PROVEN PROFESSIONAL

In this lab, you are asked to read in some data into a data frame and display, summarize, view and experiment with some basic R capabilities.

Instructions for the lab are contained in your lab guide. This lab is structured as a tutorial; be sure to execute the commands in the order given and pay close attention to the output of each command.



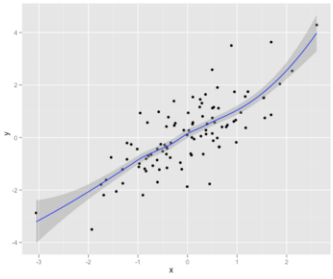
The topics for this lesson are listed.

Why Visualize?

Summary statistics give us some sense of the data:

- Mean vs. Median.
- Standard deviation.
- Quartiles, Min/Max.
- Correlations between variables.

```
summary(data)
       x                    y
 Min.   :-3.05439   Min.   :-3.50179
 1st Qu.:-0.61055   1st Qu.:-0.75968
 Median : 0.04666   Median : 0.07340
 Mean   :-0.01105   Mean   : 0.09383
 3rd Qu.: 0.56067   3rd Qu.: 0.88114
 Max.   : 2.60614   Max.   : 4.28693
```



Visualization gives us a more holistic sense

EMC² PROVEN PROFESSIONAL

In the previous lesson, we saw how to examine data in R, including how to generate the descriptive statistics: averages, data ranges, and quartiles (which are included in the summary() report).

We also saw how to compute correlations between pairs of variables of interest. These statistics do give us a sense of a data: an idea of its magnitude and range, and some obvious dirty data (missing values, values with obviously wrong magnitude or sign).

Visualization, however, gives us a succinct, more holistic view of the data that we may not be able to get from the numbers and summaries alone. It is an important facet of the initial data exploration. Visualization helps you assess data cleanliness, and also gives you an idea of potentially important relationships in the data before going on to build your models.

Anscombe's Quartet

4 data sets, characterized by the following. Are they the same, or are they different?

Property	Values
Mean of x in each case	9
Exact variance of x in each case	10
Exact mean of y in each case	7.5 (to 2 d.p)
Variance of Y in each case	3.75 (to 2 d.p)
Correlations between x and y in each case	0.816
Linear regression line in each case	Y = 3.00 + 0.500x (to 2 d.p and 3 d.p resp.)

i

X	y
10.00	8.04
8.00	6.95
13.00	7.58
9.00	8.81
11.00	8.33
14.00	9.96
6.00	7.24
4.00	4.26
12.00	10.84
7.00	4.82
5.00	5.68

ii

x	y
10.00	9.14
8.00	8.14
13.00	8.74
9.00	8.77
11.00	9.26
14.00	8.10
6.00	6.13
4.00	3.10
12.00	9.13
7.00	7.26
5.00	4.74

iii

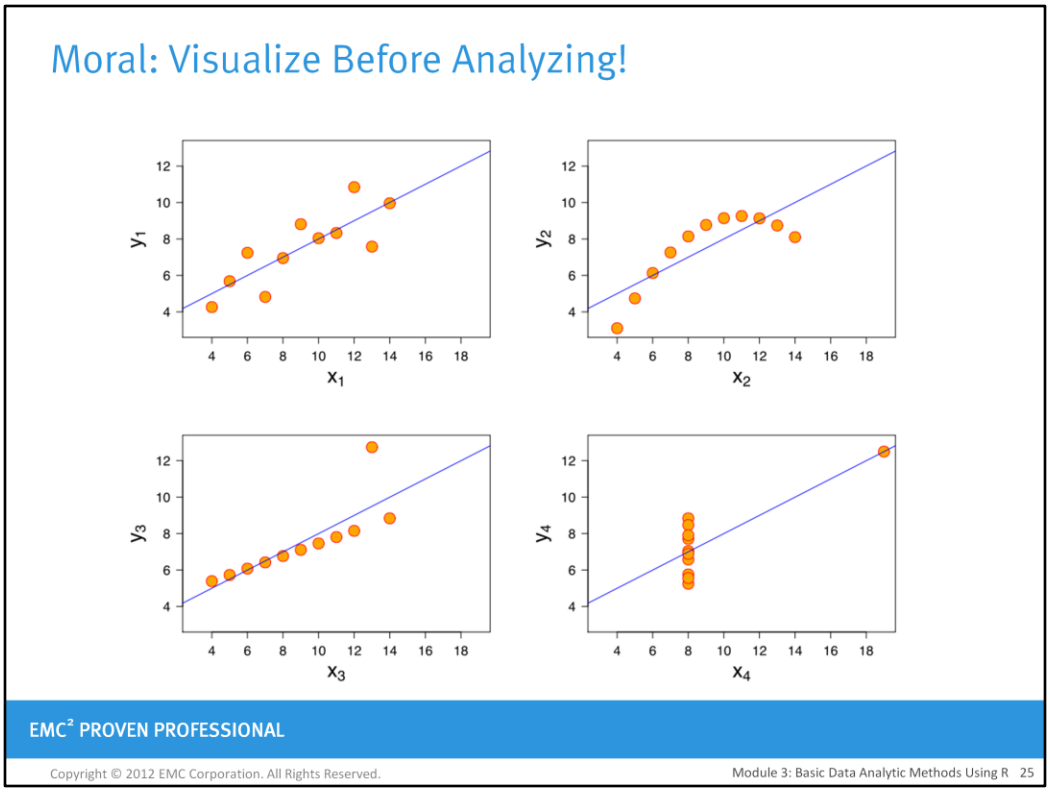
x	y
10.00	7.46
8.00	6.77
13.00	12.74
9.00	7.11
11.00	7.81
14.00	8.84
6.00	6.08
4.00	5.39
12.00	8.15
7.00	6.42
5.00	5.73

iv

x	y
8.00	6.58
8.00	5.76
8.00	7.71
8.00	8.84
8.00	8.47
8.00	7.04
8.00	5.25
19.00	12.50
8.00	5.56
8.00	7.91
8.00	6.89

EMC² PROVEN PROFESSIONAL

Anscombe's Quartet is a synthesized example by the statistician F. J. Anscombe. Look at the properties and values of these four data sets. Based on standard statistical measures of mean, variance, and correlation (our descriptive statistics), these data sets are identical. Or are they?



However, if we visualize each data set using a scatterplot and a regression line superimposed over each plot, the datasets appear quite different. Dataset 1 is the best candidate for a regression line, although there is a lot of variation. Dataset 2 is definitely non-linear. Dataset 3 is a close match, but over predicts at higher value of x and has an extreme outlier. And Dataset 4 isn't captured at all by a simple regression line.

Assuming we have datasets represented by data frames s1, s2, s3, and s4, we can generate these plots in R by using the following code:

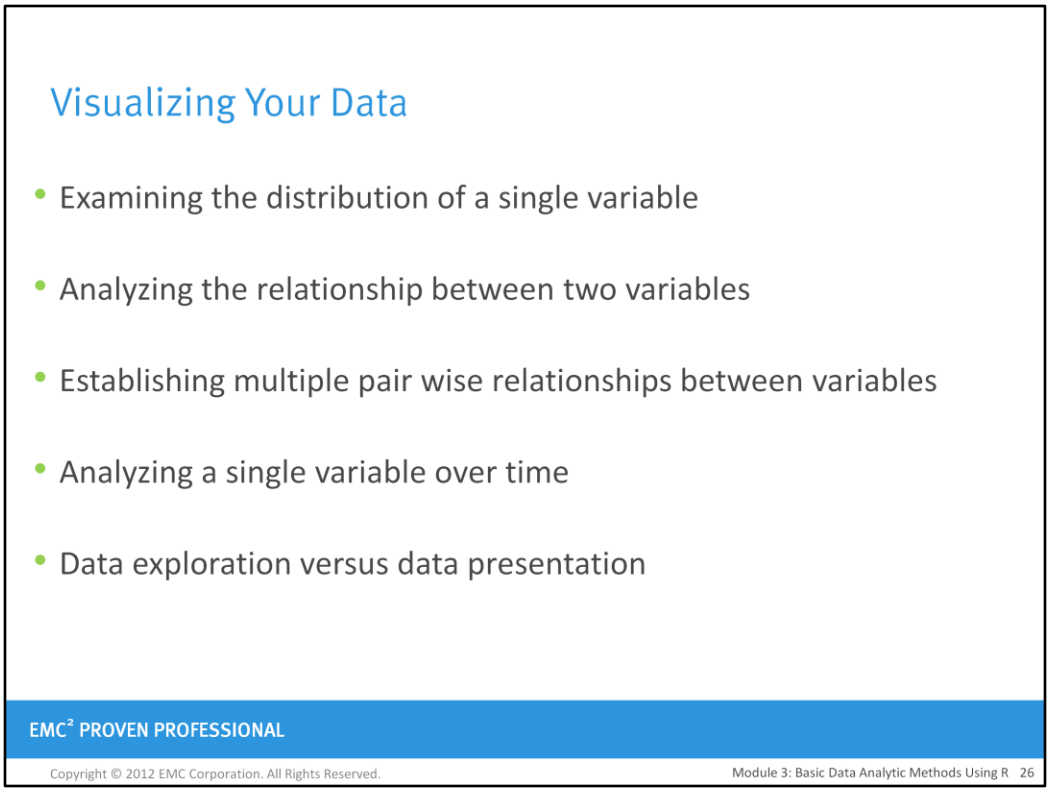
R-Code

```
plot(s1)
plot(lm(s1$y ~ s1$x))
```

…

(Yes, a loop is possible but requires more advanced data manipulation: for information, consult the R "eval" function if interested). We **also must take care to overwrite the preceding graph in each instance**.

Code to produce these graphs is included in the script *AnscombePlot.R*. Note that the dataset for these plots are included in the standard R distribution. Type *data()* for a list of dataset included in the base distribution. *data(name)* will make that dataset available in your workspace.



In a previous lesson, we've looked at how you can characterize your data by using traditional statistics. But we also showed how datasets could appear identical when using descriptive statistics, and yet look completely different when visualizing the data via a plot.

Using visual representations of data is the hallmark of exploratory data analysis: letting the data speak to us rather than necessarily imposing an interpretation on the data *a priori*. In the rest of this lesson, we are going to examine ways of displaying data so that we can better understand the underlying distributions of a single variable or the relationships between two or more variables.

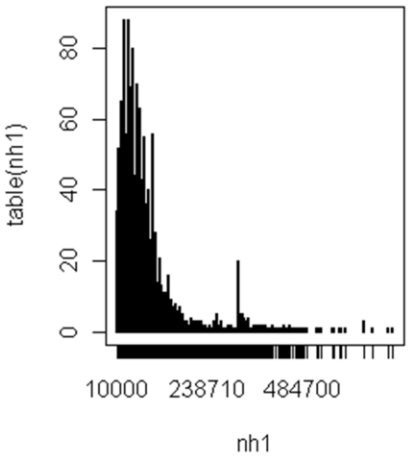
Although data visualization is a powerful tool, the results we obtain may not be suitable when it comes time for us to "tell a story" about the data. Our last slide will discuss what kind of presentations are most effective.

Examining the Distribution of a Single Variable

Graphing a single variable

- plot(sort(.)) – for low volume data
- hist(.) – a histogram
- plot(density(.)) – densityplot
  - A "continuous histogram"

- Example
  - Frequency table of household income



EMC² PROVEN PROFESSIONAL

R has multiple functions available to examine a single variable. Some of them are listed above. See the R documentation for each of these. Some other useful functions are `barplot()` and `dotplot()`.

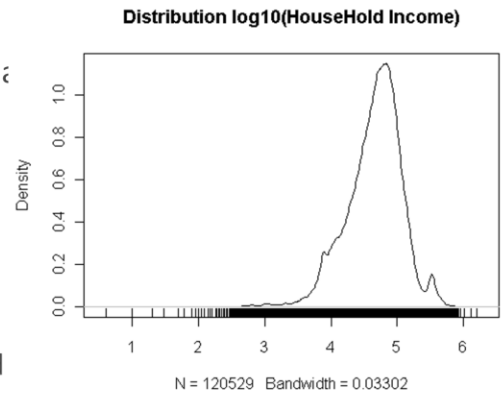
The example included is a frequency table of household income. We can certainly see a concentration of households in the leftmost portion of the graph.

Examining the Distribution of a Single Variable

Graphing a single variable

- plot(sort(.)) – for low volume da
- hist(.) – a histogram
- plot(density(.)) – densityplot
  - A "continuous histogram"

- Example
  - Frequency table of household income
    - rug() plot emphasizes distribution



EMC² PROVEN PROFESSIONAL

R has multiple functions available to examine a single variable. Some of them are listed above. See the R documentation for each of these. Some other useful functions are `barplot(), dotplot() and stem().`

The example included is a frequency table of log10 of household income. We can certainly see a concentration of households in the rightmost portion of the graph. The rug() function creates a 1-dimensional density plot as well: notice how it emphasizes the area under the curve.

What are we looking for?

A sense of the data range

- If it's very wide, or very skewed, try computing the log

Outliers, anomalies

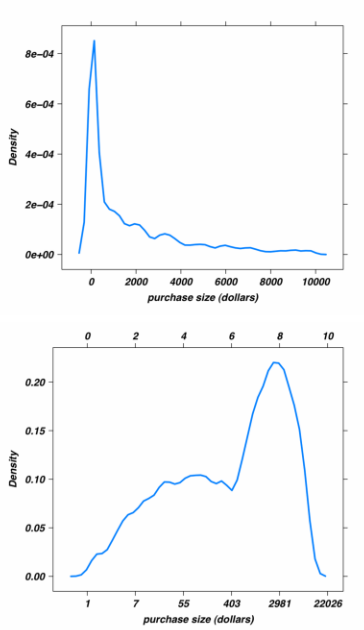
- Possibly evidence of dirty data

Shape of the Distribution

- Unimodal? Bimodal?
- Skewed to left or right?
- Approximately normal? Approximately lognormal?

Example - Distribution of purchase size ($)

- Range from 0 to > $10K, left skewed
- Typical of monetary data
- Plotting log of data gives better sense of distribution
- Two purchasing distributions
  - ~ $55
  - ~ $2900



EMC² PROVEN PROFESSIONAL

When viewing the variables during the data exploration phase, you are looking for a sense of the data range, and whether the values are strongly concentrated in a certain range. If the data is very skewed, **viewing the log of the data (if it's all positive) can help you detect structure that you might otherwise miss in a regularly scaled graph.**

This is your chance to **look for obvious signs of dirty data (outliers or unlikely looking values**). See if the data is unimodel or multimodal: that gives you an idea of how many distinct populations (with distinct behavior patterns) might be mixed into your overall population. Knowing if the data is approximately normal (or can be transformed to approximately normal – for example, by taking the log) is important, since many modeling techniquest assume that the data is approximately normal in distribution.

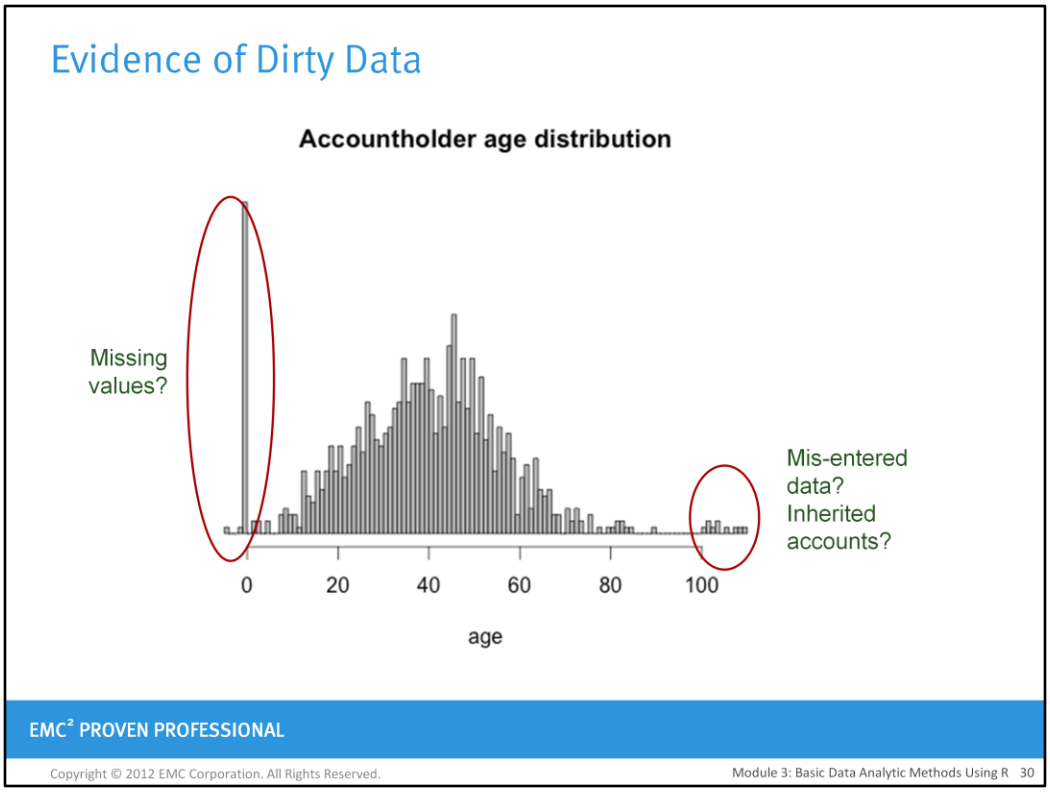
For our example, we can look at the densityplot of purchase sizes (in $ US) of customers at our online retail site. The range here is extremely wide – from around $1 US to over $10,000 US. Extreme ranges like this are typical of monetary data, like income, customer value, tax liabilities, bank account sizes, etc. (In fact, all of this kind of data is often assumed to be distributed lognormally – that is, its log is a normal distribution).

The data range makes it really hard for us to see much detail, so we take the log of it, and then density plot it. Now we can see that there are (at least) two distinct population in our customer base: One population that makes small to medium size purchases (median purchase size about $55 US) and one that makes larger purchases (median purchase size about $2900 US). Can you see those two populations in the top graph?

The plots shown were made using the lattice package. If the data is in the vector *purchase_size*, then the lattice plot is:

```
library(lattice)
densityplot(purchase_size)    # top plot
# bottom plot as log10 is actually
# easier to read, but this plot is in natural log
densityplot(log(purchase_size)
```

(the commands were actually more complicated than that, but these commands give the basic equivalent)



Here's an example of how dirty data might manifest itself in your visualizations. We are looking at the age distribution of account holders at our bank. Mean age is about 40, approximately normally distributed with a standard deviation of about 15 years or so, which makes sense.

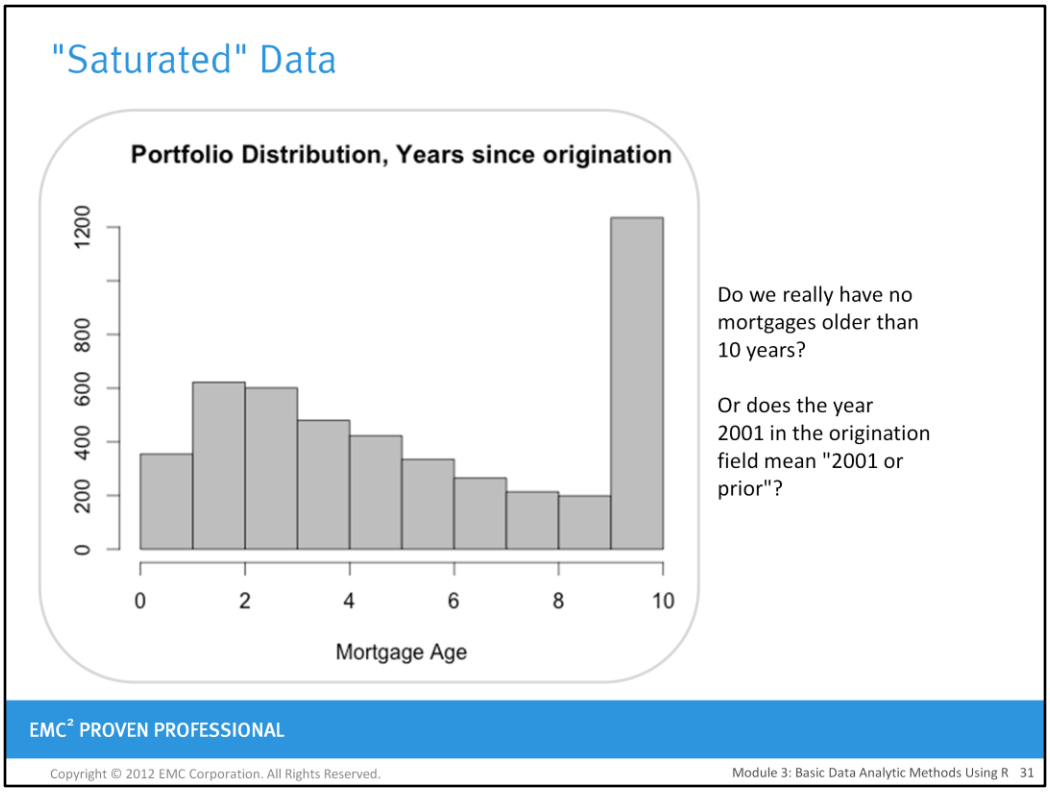
We see a few accounts with accountholder age < 10; unusual, but plausible. These could be custodial accounts, or college savings accounts set up by the parents of young children. We probably want to keep them for our analysis.

There is a huge spike of customers who are zero years old – evidence of missing data. We may need to eliminate these accounts from analysis (depending on how important we think age will be), or track down how to get the appropriate age data.

The customers with negative age are probably either missing data, or mis-entered data. The customers who are older than 100 are possibly also mis-entered data, or these are accounts that have been passed down to the heirs of the original accountholders (and not updated).We may want to exclude them as well, or at least threshold the age that we will consider in the analysis.

If this data is in a vector called *age*, then the plot is made by:

```
hist(age, breaks=100, main="Accountholder age distribution",
xlab="age", col="gray")
```



Here's another example of dirty (or at least, "incompletely documented" data). We are looking at the age of mortgages in our bank's home loan portfolio. The age is calculated by subtracting the origination date of the loan from "today" (2011).

The first thing we notice is that we don't seem to have loans older than 10 years old – and we also notice that we have a disproportionate number of ten year old loans, relative to the age distribution of the other loans.

One possible reason for this is that the date field for loan origination may have been "overloaded" so that "2001" is actually a beacon value that means "2001 or prior" rather than literally 2001. (This sometimes happens when data is ported from one system to another, or because someone, somewhere, decided that origination dates prior to 2001 are not relevant).

What would we do about this? If we are analyzing probability of default, it is probably safe to eliminate the data (or keep the assumption that the loans are 10 years old), since 10 year old mortgages default quite rarely (most defaults occur before about the 4th year). For different analyses, we may need to search for a source of valid origination dates (if that is possible).

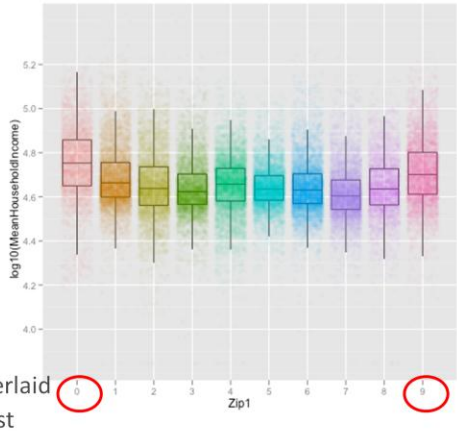
If the data is in the vector *mortgage*, the plot is made by:

```
hist(mortgage, breaks=10, main="Portfolio Distribution, Years
since origination", xlab="Mortgage Age", col="grey")
```

Analyzing the Relationship Between Two Variables

How?

- Two Continuous Variables (or two discrete variables)
  - Scatterplots
  - LOESS (fit smoothed line to the data)
  - Linear models: graph the correlation
  - Binplots, hexbin plots
    - More legible color-based plots for high volume data
- Continuous *vs.* Discrete Variable
  - Jitter, Box and whisker plots, Dotplot or barchart



Example:

- Household income by region (ZIP1)
- Scatterplot with jitter, with box-and-whisker overlaid
- New England (0) and West Coast (9) have highest mean household income

EMC² PROVEN PROFESSIONAL

Scatterplots are a good first visualization for the relationship between two variables, especially two continuous variables. Since you are looking for the relationship between the two variables, it can often be helpful to fit a smoothing curve through the data, for example loess or a linear regression. We'll see an example of that a little later on.

For very high volume data, scatterplots are problematic; with too much data on the page, the details can get lost. Sometime the *jitter()* function can create enough (uniform) variation to see the associations more clearly. *Hexbin* plots are a good alternative: you can think of hexbin plots as two dimensional histograms that use color or grayscale to encode bin heights.

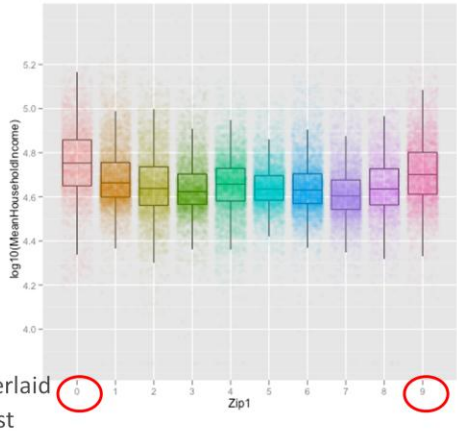
There are other alternatives for plotting continuous vs. discrete variables. Dotplots and barcharts plot the continuous value as a function of the discrete value when the relationship is one-to-one. Box and whisker plots show the distribution of the continuous variable for each value of the discrete variable.

The example here is of logged household incomes as a function of region (first digit of the zip). (Logged in this case means data that uses the logarithm of the value instead of the value itself.) In this example, we have also plotted the scatterplot beneath the box-and-whisker, with some jittering so each line of points widens into a strip. The "box" of the box and whisker shows the range that contains the central 50% of the data; the line inside the box is the location of the median. The "whiskers" give you an idea of the entire range of the data. Usually, box and whiskers also show "outliers" that lie beyond the whiskers, but they are turned off in this graph. This graphs shows how household income varies by region. The highest median incomes are in New England (region 0) and on the West Coast (region 9). New England is slightly higher, but the boxes for the two regions overlap enough that the difference between the two regions probably is not significant. The lowest household incomes tend to be in region 7 (TX, OK, Ark, LA).

Analyzing the Relationship … (Continued)

How?

- Two Continuous Variables (or two discrete variables)
  - Scatterplots
  - LOESS (fit smoothed line to the data)
  - Linear models: graph the correlation
  - Binplots, hexbin plots
    - More legible color-based plots for high volume data
- Continuous *vs.* Discrete Variable
  - Jitter, Box and whisker plots, Dotplot or barchart



Example:

- Household income by region (ZIP1)
- Scatterplot with jitter, with box-and-whisker overlaid
- New England (0) and West Coast (9) have highest mean household income

EMC² PROVEN PROFESSIONAL

If the data is a column called *MeanHouseholdIncome*, in a data frame called *data*, then the base graphics equivalent of the box and whisker plot for this data is

```
boxplot(log10(MeanHouseholdIncome) ~ Zip1, data=data,
xlab='Zip1', ylab='log10(income))
```

Assume there is a data frame called data, with columns MeanHouseholdIncome and Zip1, the basic graphics code for a box and whisker:

```
boxplot(log10(MeanHouseholdIncome) ~ Zip1, data=data,
xlab='Zip1', ylab='log10(income)`)
```

EXTRA:

The graph shown on the slide is in ggplot (which is fairly complicated). The commands are

```
library(ggplot2)
```

the outlier.size=0 prevents the boxplot from plotting the outlier

```
ggplot(data, aes(x=Zip1, y=log10(MeanHouseholdIncome))) +
geom_boxplot(outlier.size=0, alpha=0.1) +  points
```

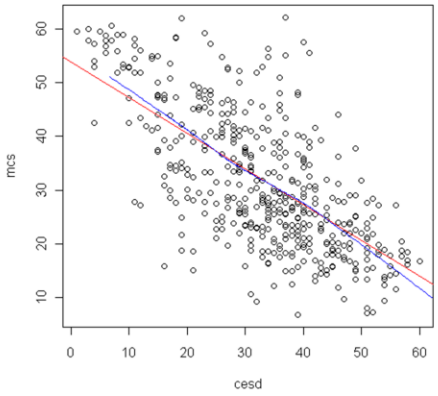
plot the jittered scatterplot, color-code the points

```
geom_point(aes(colour=Zip1), alpha=0.02,
position="jitter").
```

You can read more about ggplot2 at <http://had.co.nz/ggplot2/ >

Two Variables: What are we looking for?

- Is there a relationship between the two variables?
  - Linear? Quadratic?
  - Exponential?
    - Try semi-log or log-log plots
  - Is it a cloud?
    - Round? Concentrated? Multiple Clusters?
- How?
  - Scatterplots
- Example
  - Red line: linear fit
  - Blue line: LOESS
  - Fairly linear relationship, but with wide variance



EMC² PROVEN PROFESSIONAL

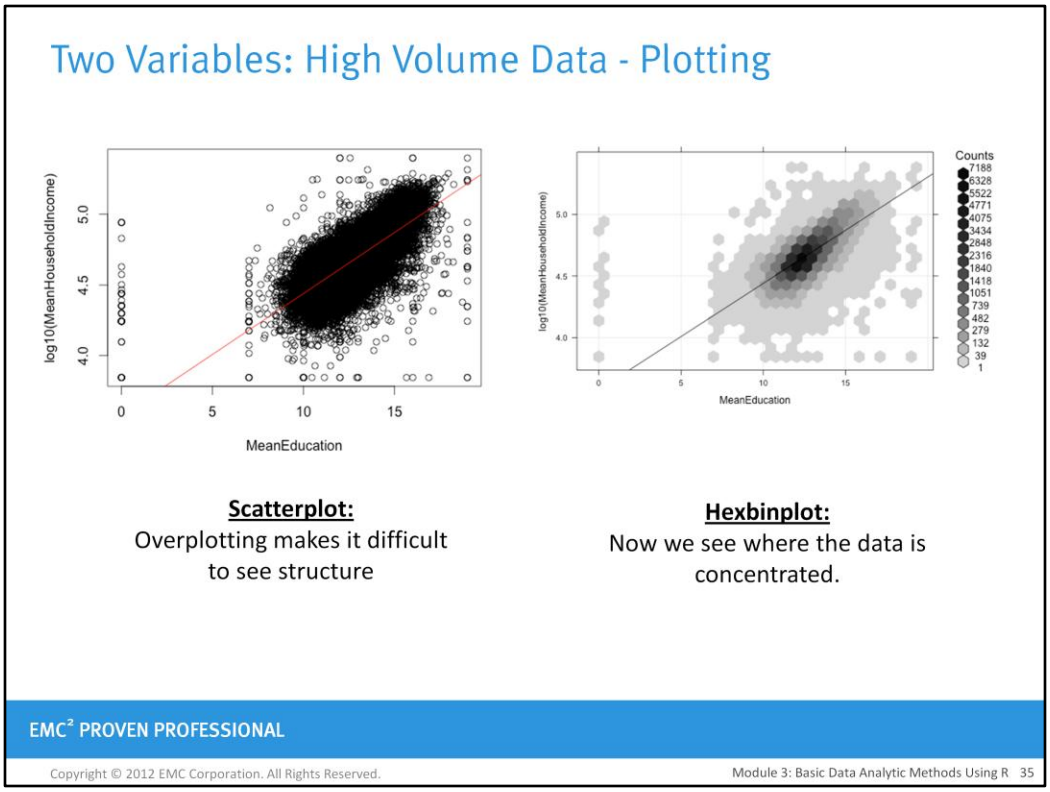
We are looking for a relationship between the two variables. If the functional relationship between the variables is somewhat pronounced, the data lies roughly along a curve: a straight line, a parabola, or an exponential curve. If y is related exponentially to x, then the plot of (x, log(y)) will be approximately linear. If the data is more like a cloud, the relationship is weaker.

In the example here, the relationship seems approximately linear; we've plotted the regression line in red. There are times when a standard regression line just doesn't capture the relationship. In this case, the loess() function in R (also lowess()) will fit a non-linear line to the data. Here we've drawn the loess curve in blue.

R-Code

Assume a dataset named *ds* with variables *cesd* and *mcs.* The R code to generate the above plot is as follows.

```
with(ds,
{
    plot(mcs ~ cesd)
   abline(lm(mcs ~ cesd), lcol="red")
   lines(lowess(mcs ~ cesd), lcol="blue")
}  )
```



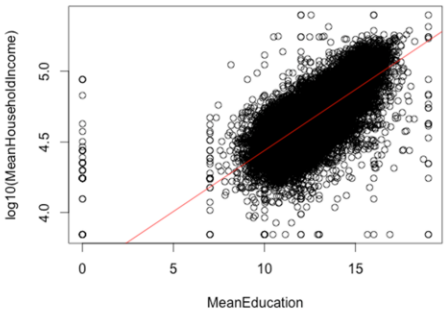
When we have too much data, the structure becomes difficult to see in a scatterplot. Here, we are plotting logged household income against years of education. The "blob" that we get on the scatterplot on the left suggests a somewhat linear relationship (this suggests, but the way, that an extra year of education multiplies your expected income by 10^M, where M is the slope of the regression line). However, we can't really see the structure of how the data is distributed.

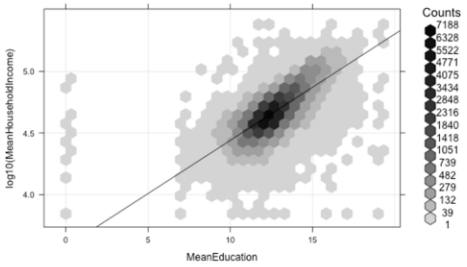
On the right we have plotted the same data using a hexbinplot. Hexbinplots are a bit like 2-d histograms, where shading tells us how populated the bin is. Now we can see that the data is more densely clustered in a streak that runs through the center of the data cloud, roughly along the regression line. The biggest concentration is around 12 years of education, extending about to about 15 years.

Notice also the outlier data at MeanEducation = 0. Missing data perhaps?

<Continued>

Two Variables: High Volume Data – Plotting (Continued)





Scatterplot:
Overplotting makes it difficult to see structure

Hexbinplot:
Now we see where the data is concentrated.

EMC² PROVEN PROFESSIONAL

The scatter plot on the left is plotted by:

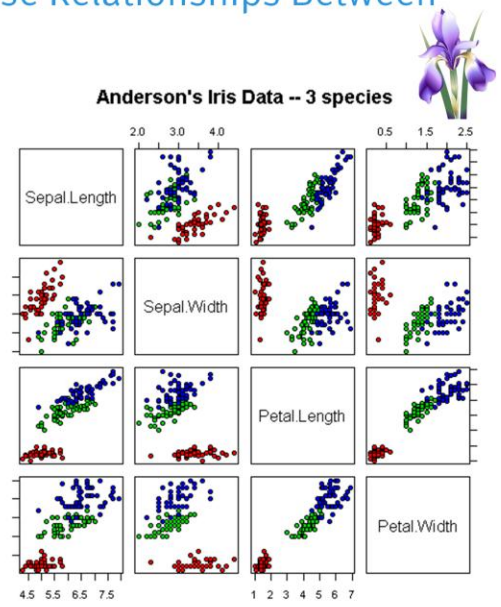
```
plot(log10(MeanHouseholdIncome) ~ MeanEducation, data=zcta)
abline(lm(log10(MeanHouseholdIncome) ~ MeanEducation,
data=zcta), col='red')
```

The hexbinplot:

```
library(hexbin)
#
# "g" adds the grid, "r" the regression line
# sqrt transform on the count gives more dynamic range to the shading
#
hexbinplot(log10(MeanHouseholdIncome) ~ MeanEducation,
  data=zcta, trans = sqrt, inv = function(x) x^2,
  type=c("g", "r"))
```

Establishing Multiple Pairwise Relationships Between Variables

- Why?
  - Examine many two-way relationships quickly
- How?
  - pairs(ds) can generate a plot of each pairs of variables
- Example
  - Iris Characteristics
    - Strong linear relationship between petal length and width
    - Petal dimensions discriminate species more strongly than sepal dimensions



EMC² PROVEN PROFESSIONAL

There are times when it's useful to see multiple values of a dataset in context in order to visually represent data relationships so as to magnify differences or to show patterns hidden within the data that summary statistics don't reveal. In the graphic represented above, the variable sepal length, sepal width, petal length and petal width are compared with three species of irises (the key is not listed in the graphic). Colors are used to represent the different species, allowing us to compare differences across species for a particular combination of variables.

Consider the values encoded in the second square from the top right, where sepal length is compared with petal length. Values for petal length are encoded across the bottom; values for sepal length are encoded on the right hand side of the graphic. We can observe that the green and blue species are well matched, although the blue species has longer petals in the main. The petal length for the red species, however, remain markedly the same, and vary only in the lower half of sepal length values. As an exercise, imagine fitting a regression line to each of these individual graphs. What would you make of the relationship between sepal length and sepal width?

The R code for generating the plot is:

```
pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
pch = 21, bg = c("red", "green3",
"blue")[unclass(iris$Species)] )
```

and uses the iris dataset included with the R standard distribution. Here colors include the species, as well as proving the spirit of APL is alive and well.

Analyzing a Single Variable over Time

What?

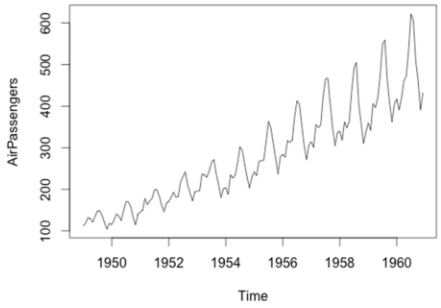
- Looking for …
  - Data range
  - Trends
  - Seasonality

How?

- Use time series plot

Example

- International air travel (1949-1960)
- Upward trend: growth appears superlinear
- Seasonality
  - Peak air travel around Nov. with smaller peaks near Mar. and June



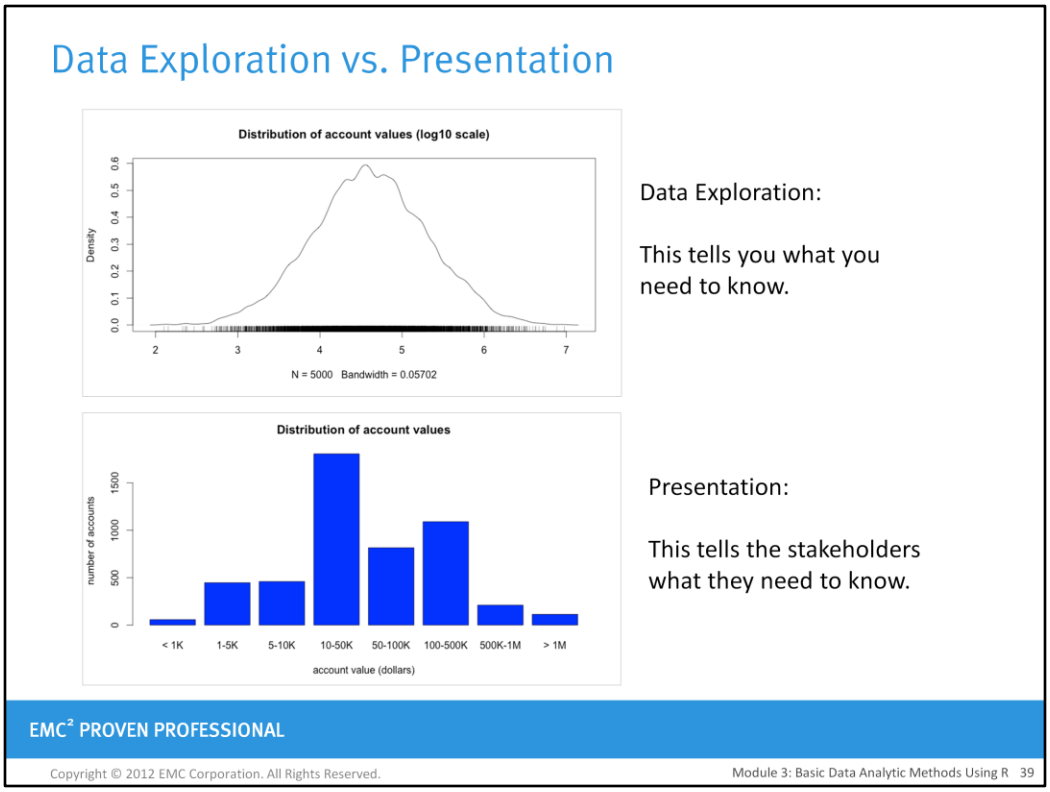
EMC² PROVEN PROFESSIONAL

Visualizing a variable over time is the same as visualizing any pair of variables, but in this case we are looking for some specific patterns.

Data range, of course, tells us how much our y variable has increased or decreased over the period of time we are considering. We want to get a feeling for the growth rate, and whether or not we see and changes in that growth rate. We are also looking for *seasonality*: a regular pattern in the fluctuations over a fixed period of time. We can think of those patterns as marking "seasons".

In the air travel data example that we show, we can see that air travel peaks regularly around Nov/Dec (the holiday season), with a smaller peak around the middle of the year (summer travel) and an even smaller one near the beginning of the year (spring break?).

We can also see that the number of air passengers increased steadily from 1949 to 1960, and that the growth appears to be faster than linear, at least during peak travel season.



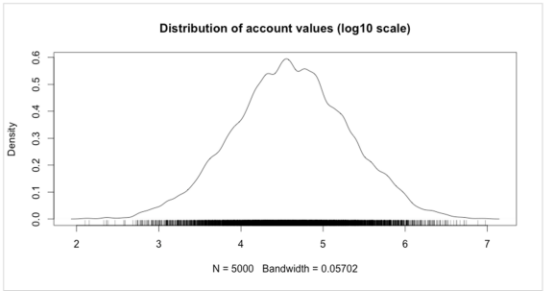
Finally, we want to touch on the difference between using visualization for data exploration, and for presenting results to stakeholders. The plots and tips that we've discussed try to make the details of the data as clear as possible for the data scientist to see structure and relationships. These technical graphs don't always effectively convey the information that needs to be conveyed to non-technical stakeholders. For them, we want crisp graphics that focus on the message we want to convey.

We will touch more on this topic in Module 6, but for right now we'll share a small example. The top graph shows the density plot of logged account values for our bank. This graph gives us, as data scientists, information that can be relevant to downstream analysis. The account values are distributed approximately lognormally, in the range from 100 to 10M dollars. The median account value is in the area of $30,000 (10^4.5), with the bulk of the accounts between $1000 US and $1M US dollars.

It would be hard to explain this graph to stakeholders. For one thing, densityplots are fairly technical, and for another, it is awkward to explain why you are logging the data before showing it. You can convey essentially the same information by partitioning the data into "log-like" bins, and presenting the histogram of those bins, as we do in the bottom plot. Here, we can see that the bulk of the accounts are in the 1000-1M range, with the peak concentration in the 10-50K range, extending out to about 500K. This gives the stakeholders a better sense of the customer base than the top graphic would.

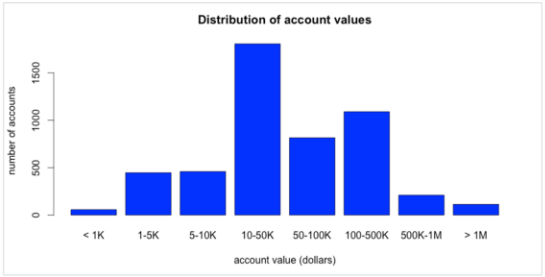
[**Note** – the reason that the lower graph isn't symmetric like the upper graph is because the bins are only "log-like". They aren't truly log10 scaled. Log10 scaled bins would be closer to: 1-3K, 3K-10K, 10K-30K..... As an exercise, we could try splitting the bins that way, and we would see that the resulting bar chart would be symmetric. The bins we chose, however, might seem more "natural" to the stakeholders.]

Data Exploration vs. Presentation (Continued)



Data Exploration:

This tells you what you need to know.



Presentation:

This tells the stakeholders what they need to know.

EMC² PROVEN PROFESSIONAL

[generating the data used in the graph]

```
income = rlnorm(5000, meanlog=log(40000), sdlog=log(5))
```

Plot for the top graphic:

```
plot(density(log10(income), adjust=0.5), main="Distribution of
account values (log10 scale)")
rug(log10(income))
```

Plot for the bottom graphic:

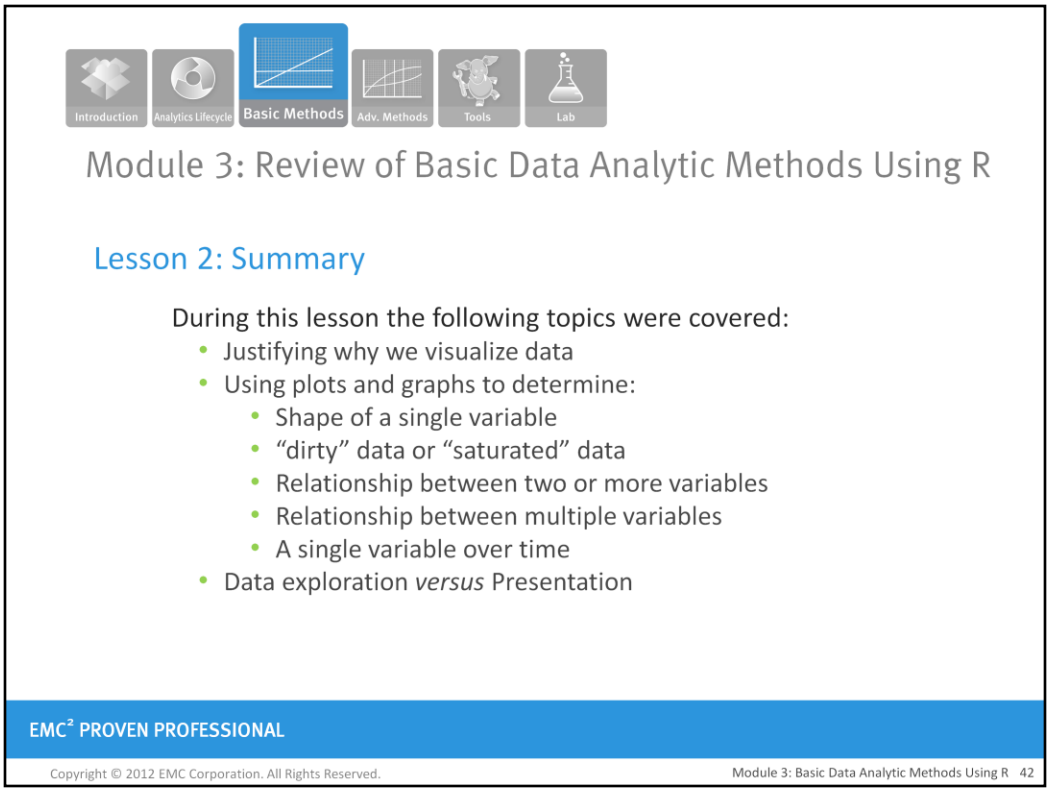
```
# create "log-like bins"
breaks = c(0, 1000, 5000, 10000, 50000, 100000, 5e5, 1e6, 2e7)
# bin and label the data
bins = cut(income, breaks, include.lowest=T,
           labels = c("< 1K", "1-5K", "5-10K", "10-50K", "50-100K",
"100-500K", "500K-1M", "> 1M"))
# plot the bins.
plot(bins, main = "Distribution of account values", xlab = "account
value (dollars)", ylab = "number of accounts", col="blue")
```

Check Your Knowledge

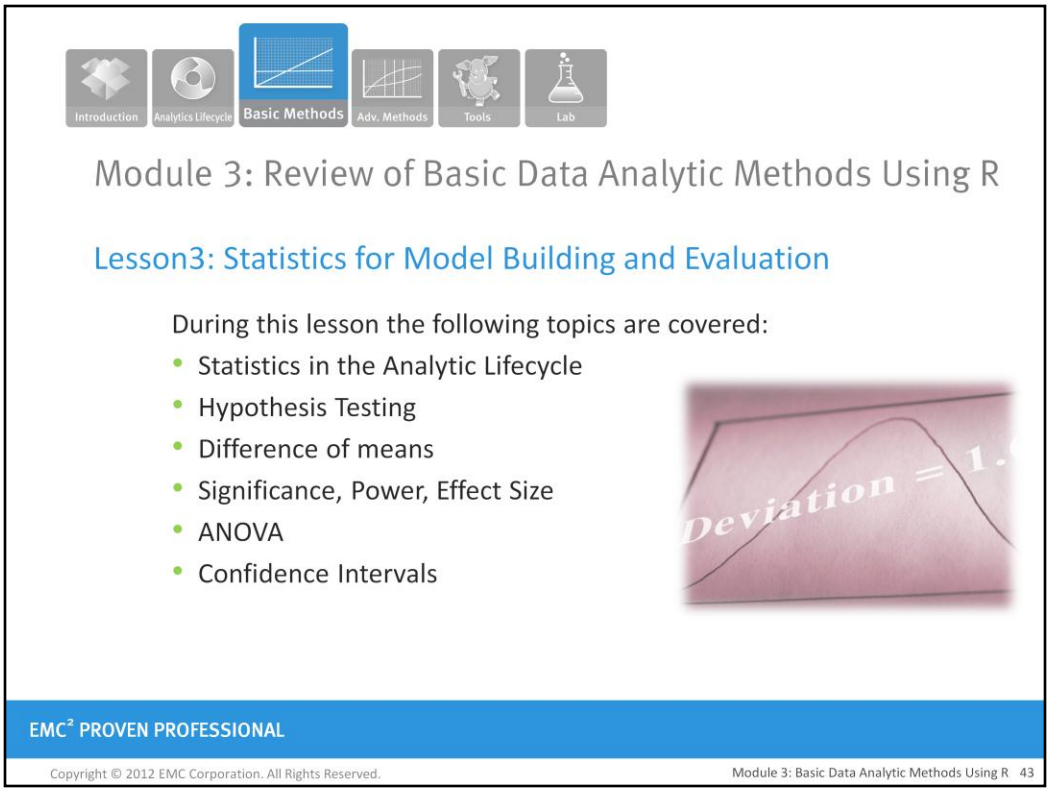
- Do you think the regression line sufficiently captures the relationship between the two variables? What might you do differently?
- In the Iris slide example, how would you characterize the relationship between sepal width and sepal length?
- Did you notice the use of color in the Iris slide? Was it effective? Why or why not?

EMC² PROVEN PROFESSIONAL

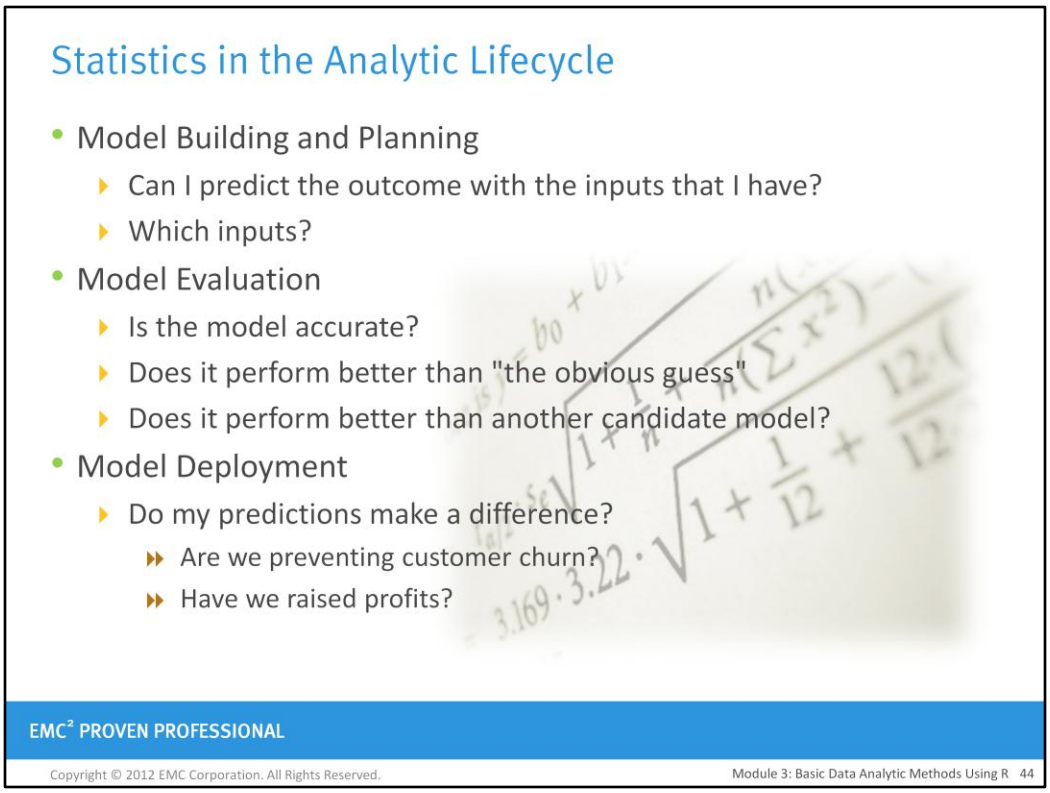
Please take a moment to answer these questions.



This slide captures the key topics from this lesson.



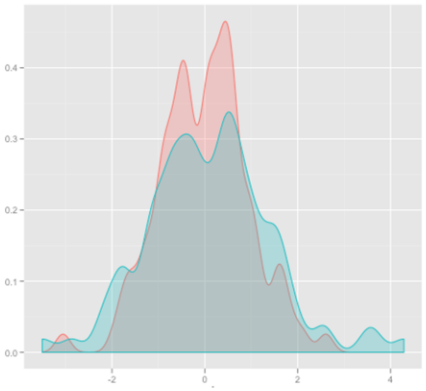
In this lesson, we'll be concentrating on model building and evaluation, using the topics described.



As Data Scientists. we use statistical techniques not only within our modeling algorithms but also during the early model building stages, when we evaluate our final models, and when we assess how our models improve the situation when deployed in the field. In this section we'll discuss techniques that help us answer questions such as those listed above? Visualization will help with the first question, at least as a first pass.

Evaluating a Model: Hypothesis Testing

- Fundamental question: "Is there a difference?"
  - Specifically: "Would I see this value if there is no difference?"
- The baseline scenario: "There is no difference."
  - Statisticians call this the Null Hypothesis
  - "There is a difference." – The Alternative Hypothesis



EMC² PROVEN PROFESSIONAL

The questions that we've just listed, and others questions that we would ask while planning or evaluating a model, are **comparisons of "before the model" and "after the model"**. Specifically – **does the model make a difference**? Does the model explain anything? Would I see the values that I've observed if the model make no difference?

The baseline assumption is that there is no difference between before and after – the model doesn't explain anything. Statisticians call this "the null hypothesis". Of course, we want to reject the null hypothesis in favor of what is called "the alternative hypothesis" – that the model does make a difference: that is; that the values we observe (1) do not equal what we would see if there is no difference, or (2) are greater than, or (3) are less than what we would see if there is no difference.

Null and Alternative Hypotheses: Examples

Null Hypothesis	Alternative Hypothesis
The best estimate of the outcome is the average observed value: • The mean is the "Null Model"	The model predicts better than the null model: • The average prediction error from the model is smaller than that of the null model
This variable does not affect the outcome: • The coefficient value is zero	The variable does affect outcome: • Coefficient value is non-zero
The model predictions do not improve revenue: • Revenue is the same with or without intervention	Interventions based on model predictions improve revenue: • A/B Testing, ANOVA

EMC² PROVEN PROFESSIONAL

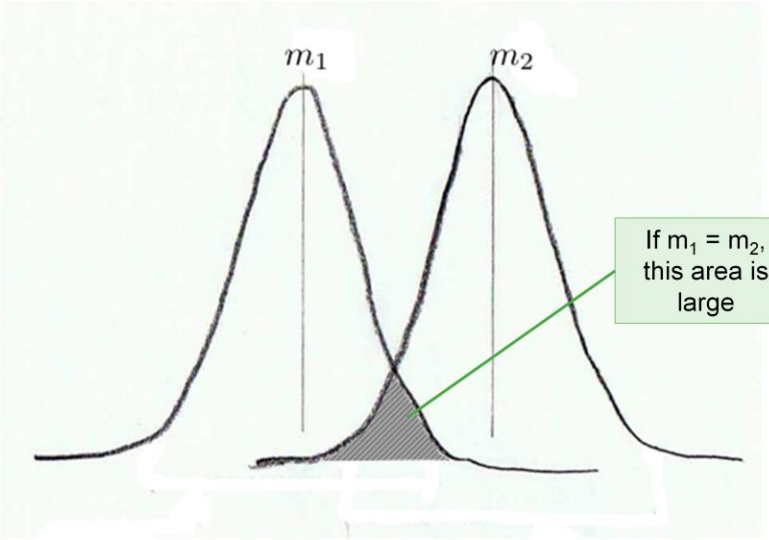
Here are some examples of null and alternative hypotheses that we would be answering during the analytic lifecycle.

1. Once we have fit a model – does it predict better than always predicting the mean value of the training data? If we call the mean value of the training data "the null model", then the null hypothesis is that the average squared prediction error from the model is the same as the average squared prediction error from the null model. The alternative is that the model's squared prediction error is less than that of the null model. A variation of that is to determine whether your "new" model predicts better than some "old" model. In that case, your null model is the "old" model, and the null and alternative hypotheses are the same as describe above.
2. When we are evaluating a model, we sometimes want to know whether or not a given input is actually contributing to the prediction. If we are doing a regression, for example, this is the same as asking if the regression coefficient for a variable is zero. The null hypothesis is that the coefficient is zero; the alternative is that the coefficient is non-zero.
3. Once we have settled on and deployed a model, we are now making decisions based on its predictions. For example, the model may help us make decisions that are supposed to improve revenue. We can test if the model is improving revenue by doing what are referred to as "A/B tests". Suppose the model tells us whether or not to make a customer a special offer. Over the next few days, every customer who comes to us is randomly put into the "A" group, or the "B" group. Customers in the A group get special offers (or not) depending on the output of the model. Customers in the B group get special offers (or not) depending on the output of the model. Customers in the B group get special offers "the old way" – either they don't get them at all, or they get them by whatever algorithm we used before.

If the model and the intervention are successful, then group A should generate higher revenue than group B. If group A does not generate higher revenue than group B (if we accept the null hypothesis that A and B generate the same revenue), then we have to determine if the problem is whether the model makes incorrect predictions, or whether our intervention is ineffective.

If we are testing more than one intervention at the same time (A, B, and C), then we can do an ANOVA analysis to see if there is a difference in revenue between the groups. We will talk about ANOVA in a bit.

Intuition: Difference of Means

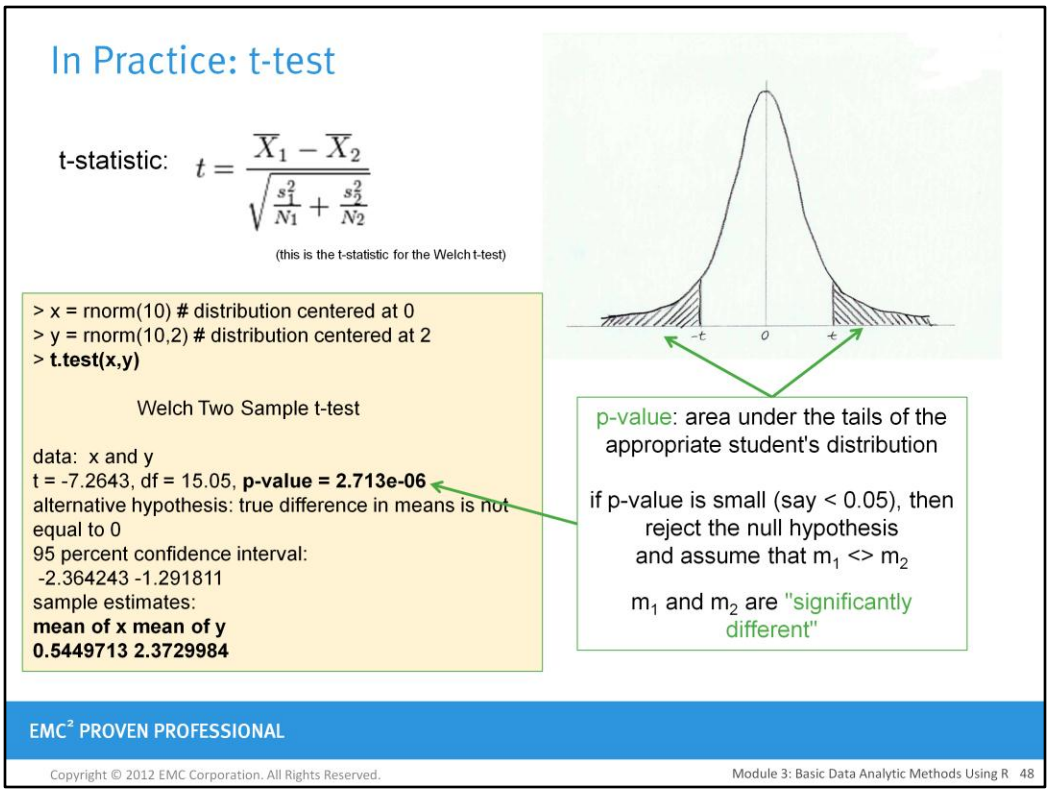


EMC² PROVEN PROFESSIONAL

For examples 1 and 3 on the previous slide, we can think of verifying the null hypothesis as verifying whether the mean values of two different groups is the same. If they are not the same, then the alternative hypothesis is true: the introduction of new behavior did have an effect.

Suppose both group1 and group2 are normally distributed, with the same variance *sigma*. We have *n1* samples from group1 and *n2* samples from group2. It happens to be true that the empirical estimate of the population means *m1* and *m2* are also normally distributed with variances *sigma/n1* and *sigma/n2* – in other words, the more samples we have, the better our estimate of the mean.

If the means are really the same, then the distributions of *m1* and *m2* will overlap substantially.



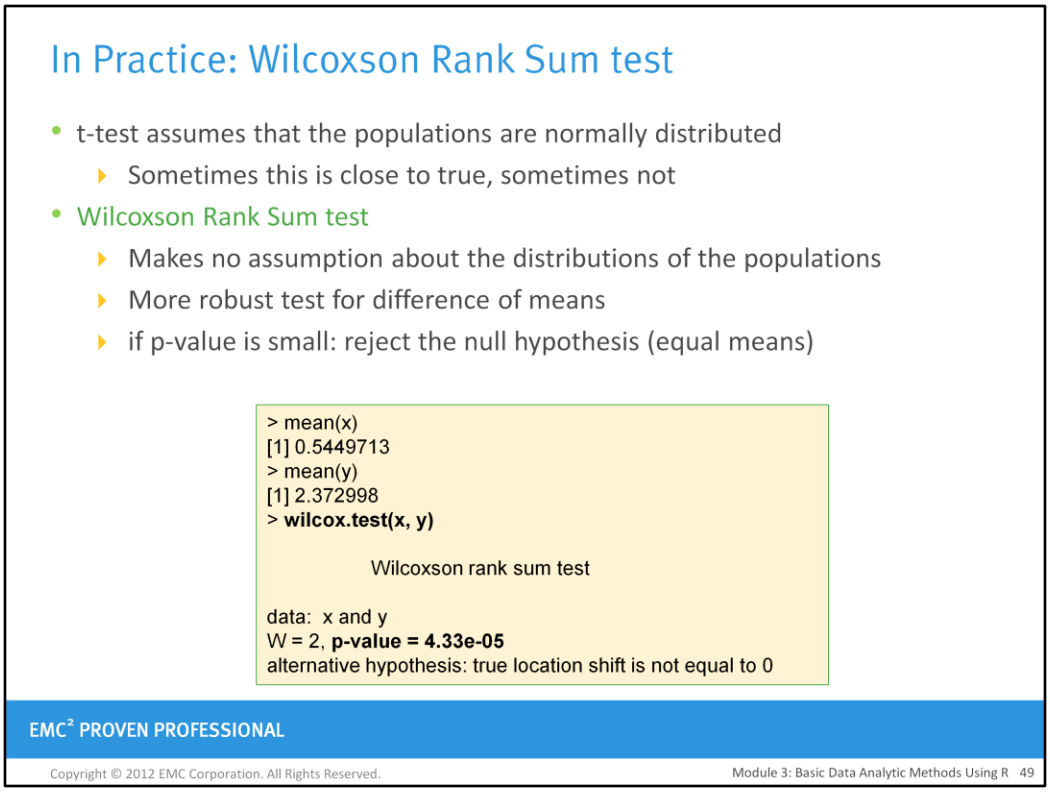
In practice, we don't calculate the area directly. Instead we calculate the t-statistic, which is the difference in the observed means, divided by a quantity that is a function of the observed standard deviations, and the number of observations. If the null hypothesis is true (m1 = m2) then *t* should be "about zero". Specifically, *t* is distributed in a bell shaped curve around 0 called the *Student's distribution* – the specific Student's distribution is a function of the number of observations. For a very large number of observations, the Student's distribution converges to the normal distribution.

How do we tell if the t-statistic that we observed is "about zero"? We calculate the probability of observing a *t* of that magnitude or larger under the null hypothesis – this probability is the area under the tails of the appropriate student distribution.

If the alternative hypothesis is that m1 <> m2, then we look at the area under both tails. If the alternative hypothesis is that m1 > m2 (or m2 > m1), then we look at the area under one tail.

This area is called the "p-value". If p is small, then the probability of seeing our observed *t* under the null hypothesis is small, and we can go ahead and accept the alternative hypothesis.

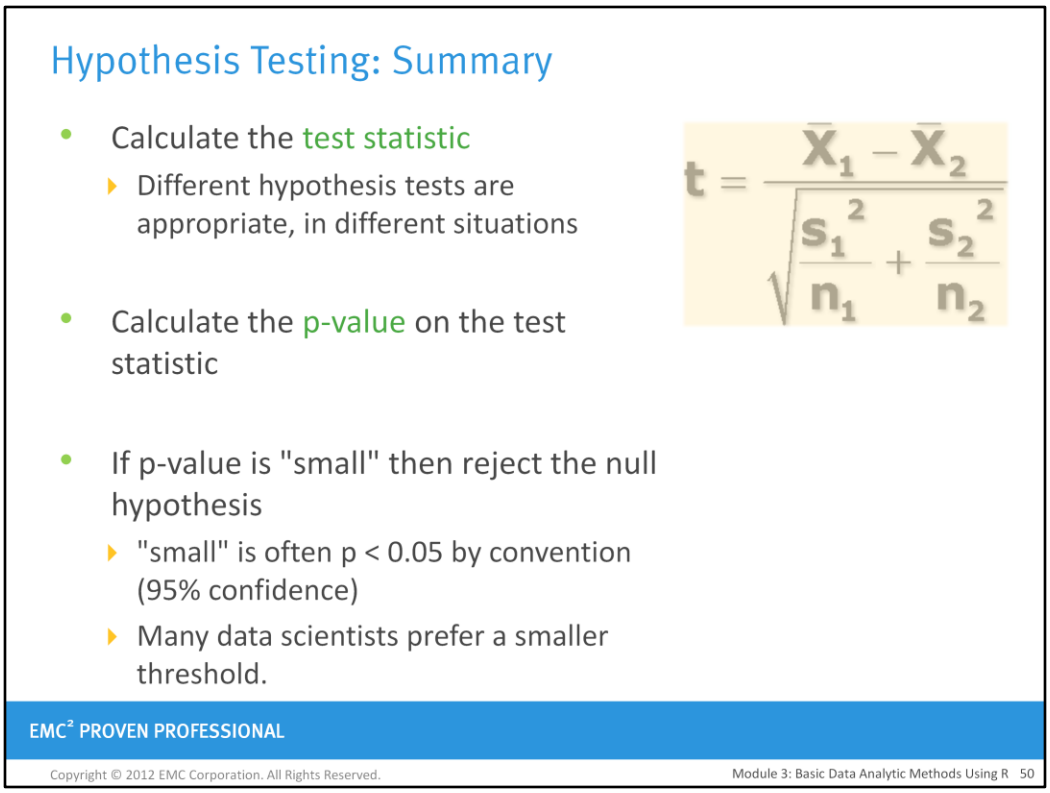
[**Note** – Welch's t-test does not assume equal variance, and is a more robust variation of Student's t-test]



A t-test represents a parametric test. Student's t-test assumes that both populations are normally distributed with the same variance. Welch's t-test (the `t.test()` function in R is Welch's t-test by default) does not assume equal variance, but it does assume normality. Sometimes, this is approximately true (true enough to use a t-test), and sometimes, it isn't.

If we can't make the normality assumption, then we should use a nonparametric test. The Wilcoxson Rank Sum test will test for difference of means without making the normality assumption. Without getting into the details, Wilcoxson's test uses the fact that if two populations are centered in the same place, then if we merge the observations from each population, sort them, and rank them, then the observations of each population should "mix together". Specifically, if we sum the resulting ranks for each population, the sum should be "about the same".

Since Wilcoxson's test doesn't assume anything about the population distribution, it is strictly weaker than t-test when it is applied to normally distributed data. Here, we show the results of wilcox.test() on the same (normally distributed) data from the previous slide. wilcox.test() does reject the null hypothesis, but the p-value is an order of magnitude larger than it is with the t-test. So if you know that you can assume the data is near normally distributed, then you should use the t-test.



Every hypothesis test calculates a **test statistic** that is assumed to be distributed a certain way if the null hypothesis is true.

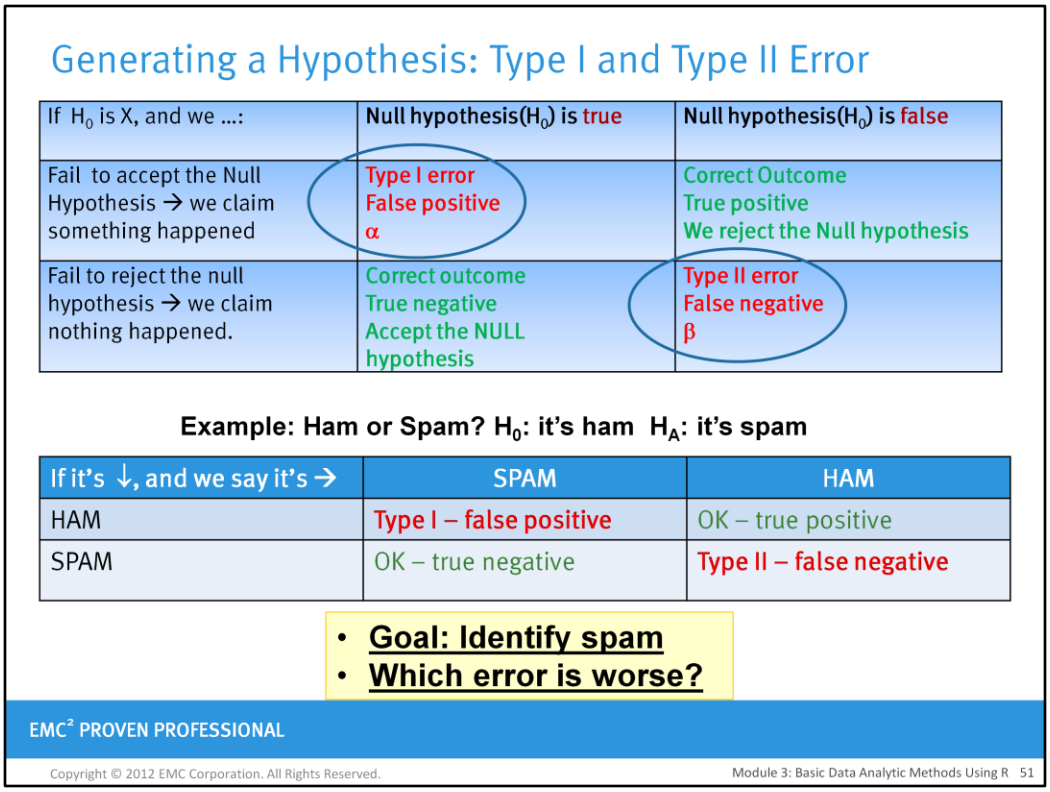
- Usually around 0 for difference, or around 1 for ratios
- Different hypothesis tests are appropriate in different situations: check the assumptions of the test, and whether they are valid (enough) for your situation.

The **p-value** is the probability of observing a value of the test statistic like the value that you saw if the null hypothesis is true. The p-value depends on how the test statistic is assumed to be distributed.

If p-value is "small" then reject the null hypothesis

- "small" is often $p < 0.05$ by convention (95% confidence)
- Many data scientists prefer a smaller threshold, often 0.01, or 0.001

Of course, most statistical packages have functions that will do steps 1 and 2 automatically, for you. Sometimes, you have to find the appropriate distribution and do it by hand.



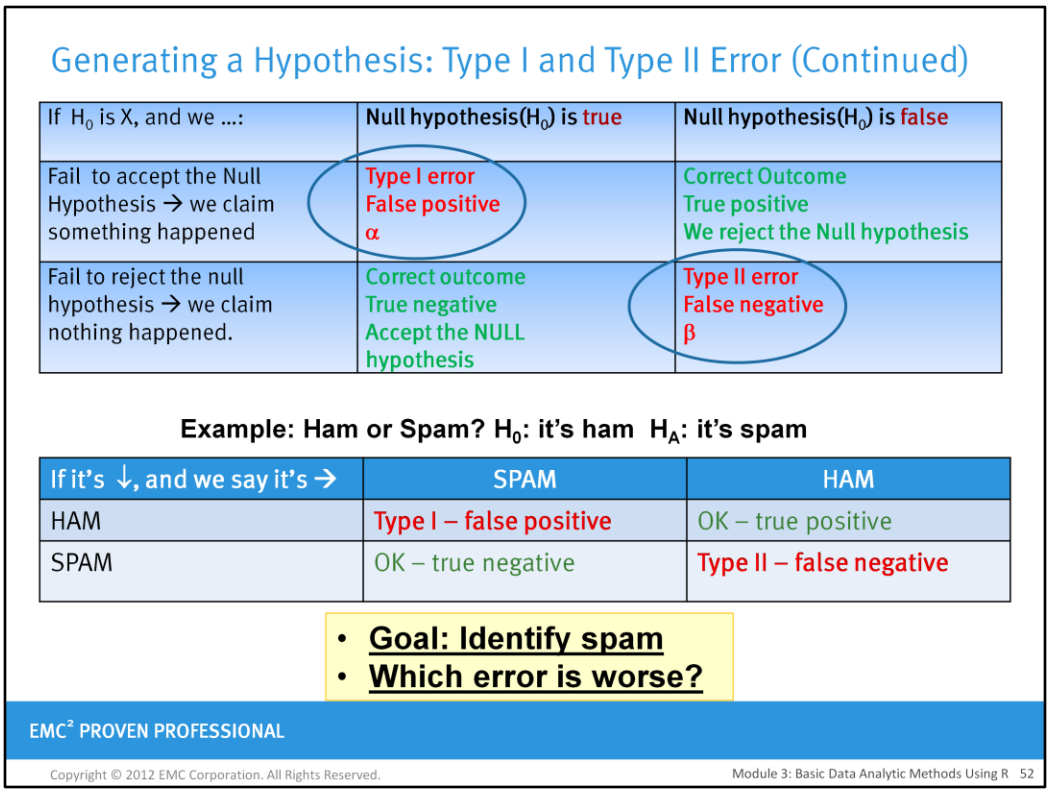
So, we have developed our null hypothesis and its alternate. Once we collect the data and begin our analysis, what kind of errors might we make? There are two kinds: type I errors and (oddly enough) type II errors, based on whether we fail to accept the null hypothesis or fail to reject the null hypothesis.

Type I error is the failure to accept the null hypothesis. This is a "False positive" -- finding significance where none exists. Type II error is the failure to reject the null hypothesis, thereby create a "false negative". This means that we have failed to find significance when it does exist.

Let's use the example of SPAM filtering (spam refers to "unsolicited commercial email".) Here our H_0 is that the email is legitimate (also know as "ham" [that is; not spam]); our alternate hypothesis is that it's not legitimate (it's "spam"). A false positive means that we treat legitimate email as spam; a false negative implies that we treat spam messages as legitimate.

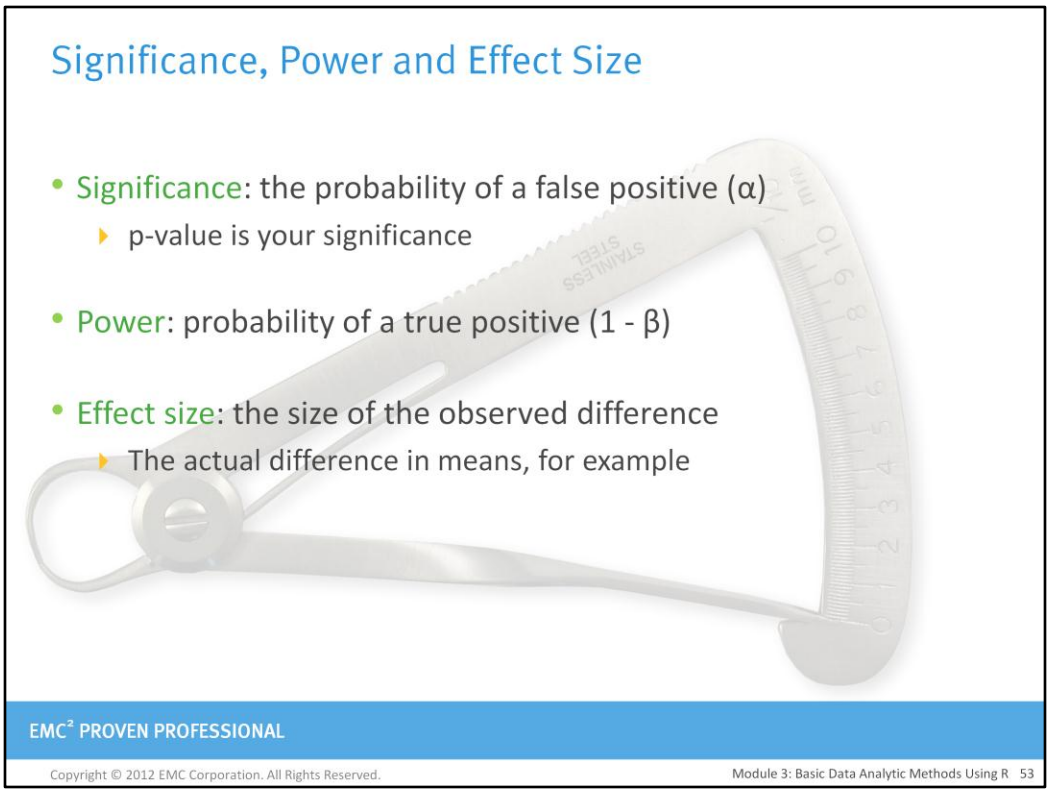
We could frame the following question: using this Email filter, how often will we identify a valid email message (ham) as spam? We consider this to be a more serious error than labeling a spam email as valid , since spam messages can be filtered from the user's mailbox, whereas a message incorrectly labeled as spam may contain information critical to the recipient.

<Continued>



Let's take another example from computer security. Suppose we have authentication system that validates user's identity, and we want to determine how well it's working. Our null hypothesis (H_0) is that the user is indeed an authorized user of the system (this is usually the case). Our alternative hypothesis (H_A) is that the user is an imposter. A false positive would mean that we declare the authorized user to be an imposter (Type I error), while a false negative would indicate that an imposter is recognized as an authorized user. In this case, we want to make sure that we don't commit type II errors, since this would represent a bad outcome for our authentication system (allowing an intruder inside as an authorized user has more negative consequences than holding up an authorized user who is incorrectly indentified as an intruder.)

Regardless of which way we choose, how do we calculate the probability of committing a type I error (false positive)?



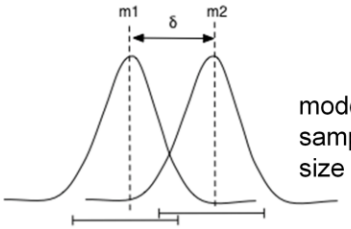
The *significance* of a result is the probability of a false positive – rejecting the null hypothesis when it should be accepted. This is exactly the p-value of the result.

The threshold of p-values that you will accept depends on how much you are willing to tolerate a false positive. So a p-value threshold of 0.05 means that you are willing to have a false positive 5% of the time.

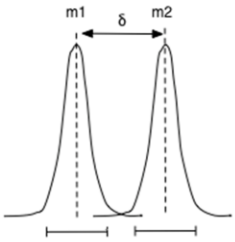
The *power* of a result is the probability of a true positive – correctly accepting the alternative hypothesis. The desired power is usually used to decide how big a sample to use.

Effect size is the actual magnitude of the result: the actual difference between the means, for example.

Always Keep Effect Size in Mind!



moderate sample size



larger sample size

Both power and significance increase with larger sample sizes.

So you can observe an effect size that is *statistically* significant, but *practically* insignificant!

EMC² PROVEN PROFESSIONAL

For a fixed effect size (*delta* in the above diagrams), both power and significance increase with larger sample sizes. This is because, for a difference in means (assuming Gaussian distributions), the estimate of the mean gets tighter as the sample size increases.

So even if the difference between the means stays the same, the Gaussian distributions around each mean overlap less, and the t-statistic gets larger, which pushes it further out on the tail of the Student's distribution.

Since there is no limit on how tight the Gaussian distribution can get, you can make any effect size appear statistically significant, even if, for all practical purposes, the difference is "insignificant" (in English terms). So always take into consideration whether or not the effect size you observe truly means "a difference" in your domain.

Hypothesis Testing: ANOVA

ANOVA is a generalization of the difference of means

- One-way ANOVA
  - k populations ("treatment groups")
  - n_i samples each – total N subjects
  - Null hypothesis: ALL the population means are equal

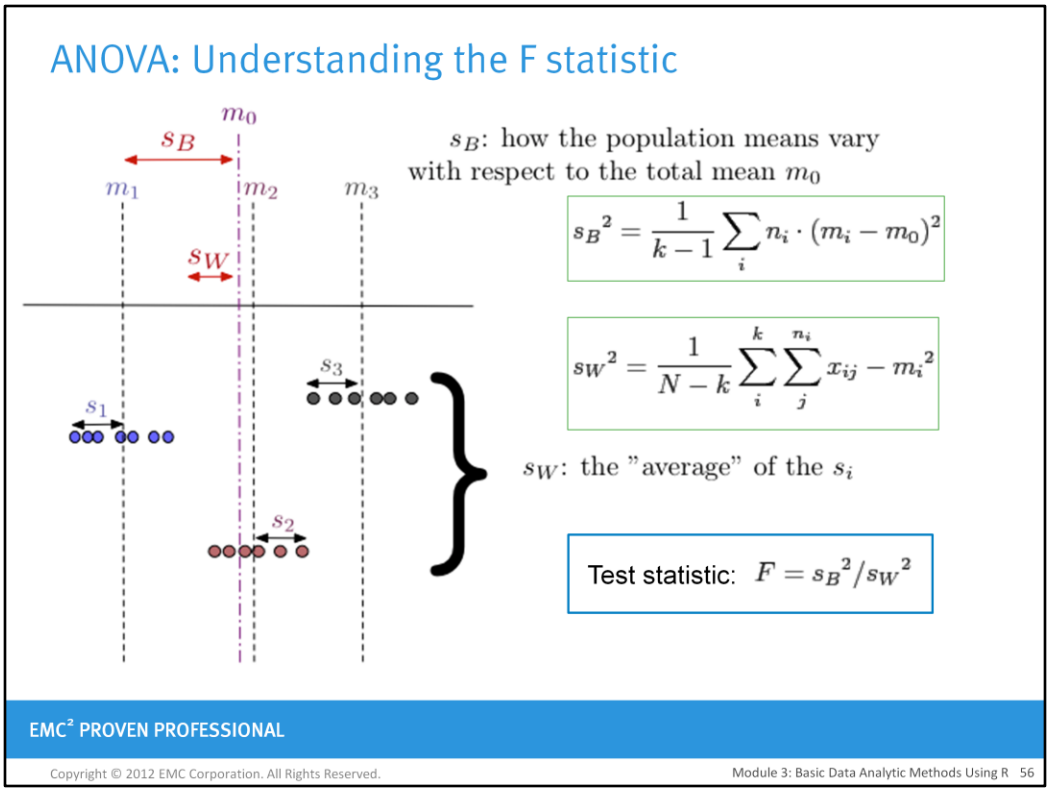
Population	n_i: # offers made	m_i: avg purchase size
Offer 1	100	$55
Offer 2	102	$40
No intervention	99	$25

EMC² PROVEN PROFESSIONAL

ANOVA (Analysis of Variance) is a generalization of the difference of means. Here we have multiple populations, and we want to see if any of the population means are different from the others. That means that the null hypothesis is that ALL the population means are equal.

An example: suppose everyone who visits our retail website either gets one of two promotional offers, or no promotion at all. We want to see if making the promotional offers makes a difference. (The null hypothesis is that neither promotion makes a difference. If we want to check if offer 1 is better than offer 2, that's a different question).

We can do multi-way ANOVA (MANOVA) as well. For instance if we want to analyze offers and day of week simultaneously, that would be a two-way ANOVA. Multi-way AVNOVA is usually done by doing a linear regression on the outcome, using each of the (categorical) treatments as an input variable. Here, we will only talk about 1-way ANOVA.



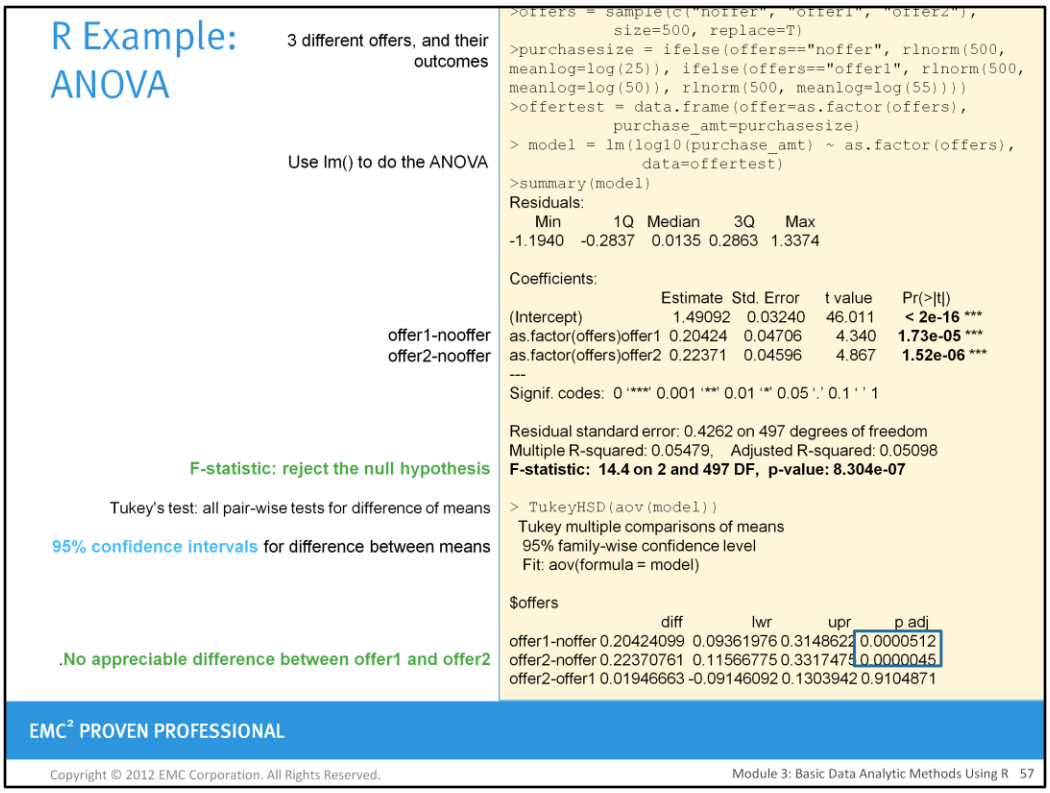
The first thing to calculate is the test statistic. Here we sketch the intuition behind the test statistic for ANOVA. Essentially, we want to test whether or not the clusters formed by each population are more tightly grouped than the spread across all of the populations.

The *between-groups mean sum of squares,* s_b^2, is an estimate of the *between-groups variance*. It is a measure of how the population means vary with respect to the grand mean – the "spread across all of the populations".

The *within-group mean sum of squares,* s_w^2 , is an estimate of the *within-group variance*: It is a measure of the "average population variance" – the average "spread" of each cluster.

If the null hypothesis is true, then s_b^2 should be about equal to s_w^2 – that is, the populations are about as wide as they are far apart – they overlap. Their ratio, the test statistic *F*, will then be distributed as the *F* distribution with *k*-1, *n*-1 degrees of freedom, which is right skewed and has its mode near 1. In the equations above, *k* is the number of populations, n_i is the number of samples in the it^h population, and *N* is the total number of samples.

If we observe that F < 1, then the populations clusters are wider than the between group spread, so we can just accept the null hypothesis (no differences). Otherwise, we only need to consider the area under the right tail of the F distribution.



Here is an example of how to do one-way ANOVA in R. We have a data frame with the outcomes under the three different offer scenarios you saw previously. We can use the linear regression function **`lm()`** to do the ANOVA calculations for us.

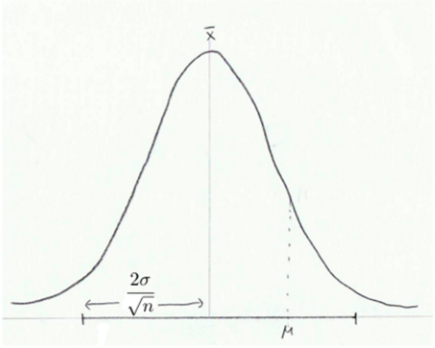
The F-statistic on the linear regression model tells us that we can reject the null hypothesis – at least one of the populations is different from the others. Since we used **`lm()`** to do the ANOVA, we have additional information: The intercept of the model is the mean outcome for nooffer. The coefficients for offer1 and offer2 are the difference of means of offer1 and offer2 respectively, from nooffer. The **lm()** function does a Wald test on each of the coefficients for the null hypothesis that the coefficient value is really zero. We can see from the p-values that the null hypothesis was rejected for both coefficients, with highly significant p-values. So, we can assume that both offer1 and offer2 are significantly different from nooffer.

However – we don't know whether or not offer1 is different from offer2. That requires additional tests. Tukey's test does all pair-wise tests for difference of means. We can see the 95% confidence interval for the difference of each pair of means, and the p-value for the test on the difference. A p-value of 0.9104871 for offer1 and offer2 suggests that we really can't tell the difference between them.

A small p-value (p = 0.049) demonstrates statistical vs. practical significance – with more data, the difference gets more statistically significant, but the effect size is still fairly small. Is the effect practically significant?

More references (2-way anova, etc): *Practical Regression and ANOVA using R*, Julian Faraway (you can get a .pdf file of an old edition of the book online from <http://cran.r-project.org/>)

Confidence Intervals



Example:
- Gaussian data N(μ, σ)
- x is the estimate of μ
  - based on n samples

μ falls in the interval

$$x \pm 2\sigma/\sqrt{n}$$

with approx. 95% probability ("95% confidence")

If x is your estimate of some unknown value μ,
the P% confidence interval
is the interval around x that μ will fall in, with
probability P.

EMC² PROVEN PROFESSIONAL

The *confidence interval* for an estimate *x* of an unknown value *mu* is the interval that should contain the true value *mu*, to a desired probability. For example, if you are estimating the mean value (mu) of a Gaussian distribution with std. dev sigma, and your estimate after *n* samples is X, then *mu* falls within +/- 2* sigma/sqrt(n) with about 95% probability.

Of course, you probably don't know sigma, but you do know the empirical standard deviation of your n samples, *s*. So you would estimate the 95% confidence interval as x +/- 2**s*.

In practice, most people estimate the 95% confidence interval as the mean plus/minus twice the standard deviation. This is really only true if the data is normally distributed, but it is a helpful rule of thumb.

Example

The defect rate of a disk drive manufacturing process is within 0.9% - 1.7%, with 98% confidence. We inspect a sample of 1000 drives from one of our plants.

- We observe 13 defects in our sample.
  - Should we inspect the plant for problems?
- What if we observe 25 defects in the sample?

EMC² PROVEN PROFESSIONAL

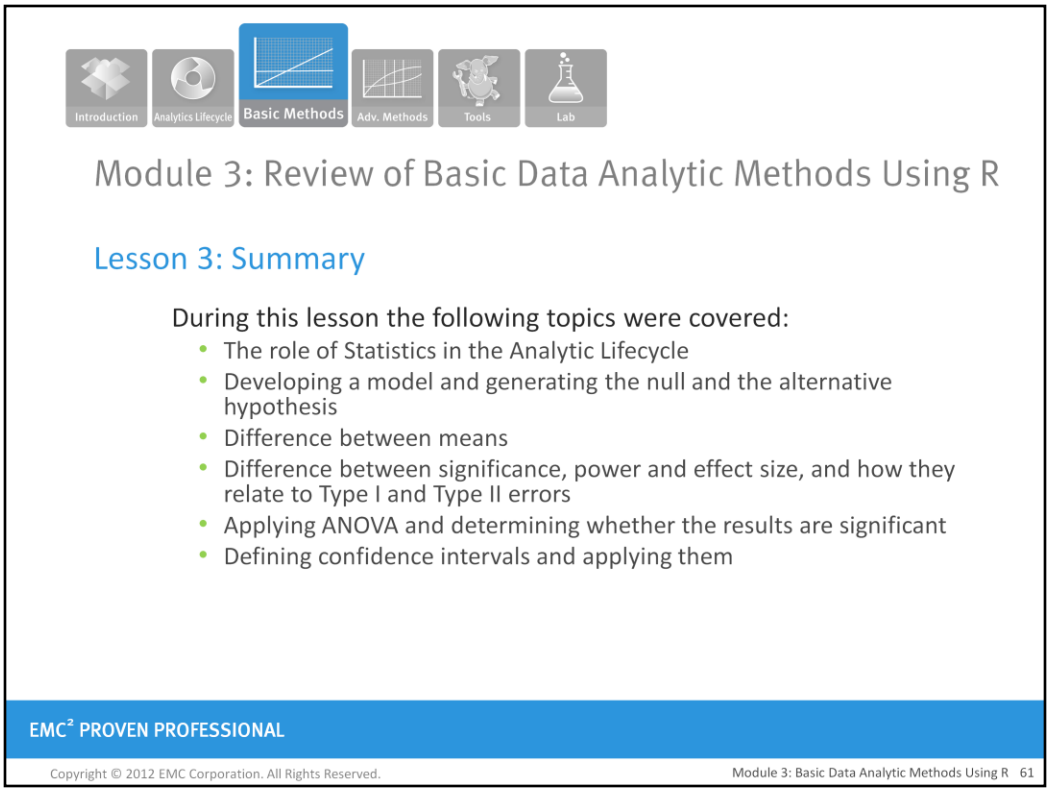
Suppose we know that a properly functioning disk drive manufacturing process will produce between 9 and 17 defective disk drives per 1000 disk drives manufactured, 98% of the time. On one of our regularly scheduled inspections of a plant, we inspect 1000 randomly selected drives. If we find 13 defective drives, we can't reject the assumption that the plant is functioning properly, because 13 defects is "in bounds" for our process.

What if we find 25 defects? We know that this would happen less than 2% of the time in a properly functioning plant, so we should accept the alternate hypothesis that the plant is **not** functioning properly, and inspect it for problems.

Check Your Knowledge

- Refer back to the Anova example on an earlier slide. What do you think? Does the difference between *offer1* and *offer2* make a practical difference? Should we go ahead and implement one of them?
- If yes, and the costs were US $25 for each offer1 and US $10 for *offer2*, would you still make the same decision?
- In our manufacturing plant example, assuming you would check the plant for problems in the manufacturing process, how might you justify this decision financially?

EMC² PROVEN PROFESSIONAL



These are the key points covered in this lesson.

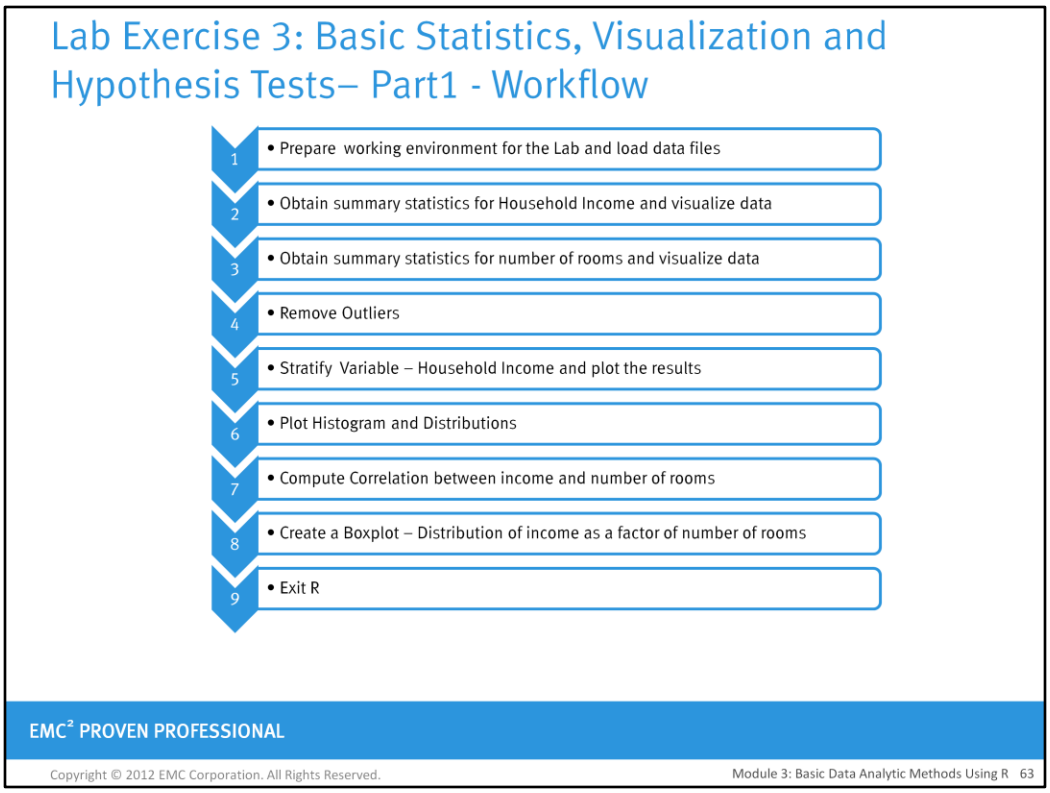
Lab Exercise 3: Basic Statistics, Visualization and Hypothesis Tests

This lab is designed to investigate and practice using R to perform basic statistics and visualization on data and to perform hypothesis testing.

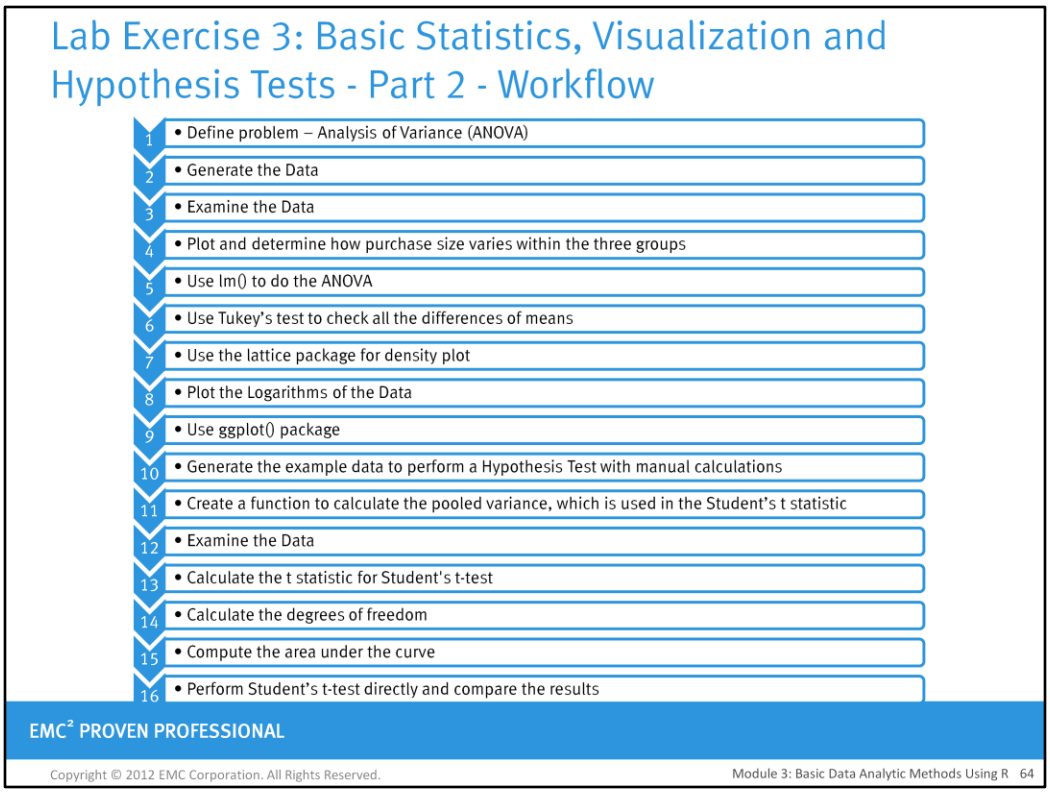
- After completing the tasks in this lab you should able to:
  - Perform basic data analysis
  - Visualize data with R
  - Create and test a hypothesis

EMC² PROVEN PROFESSIONAL

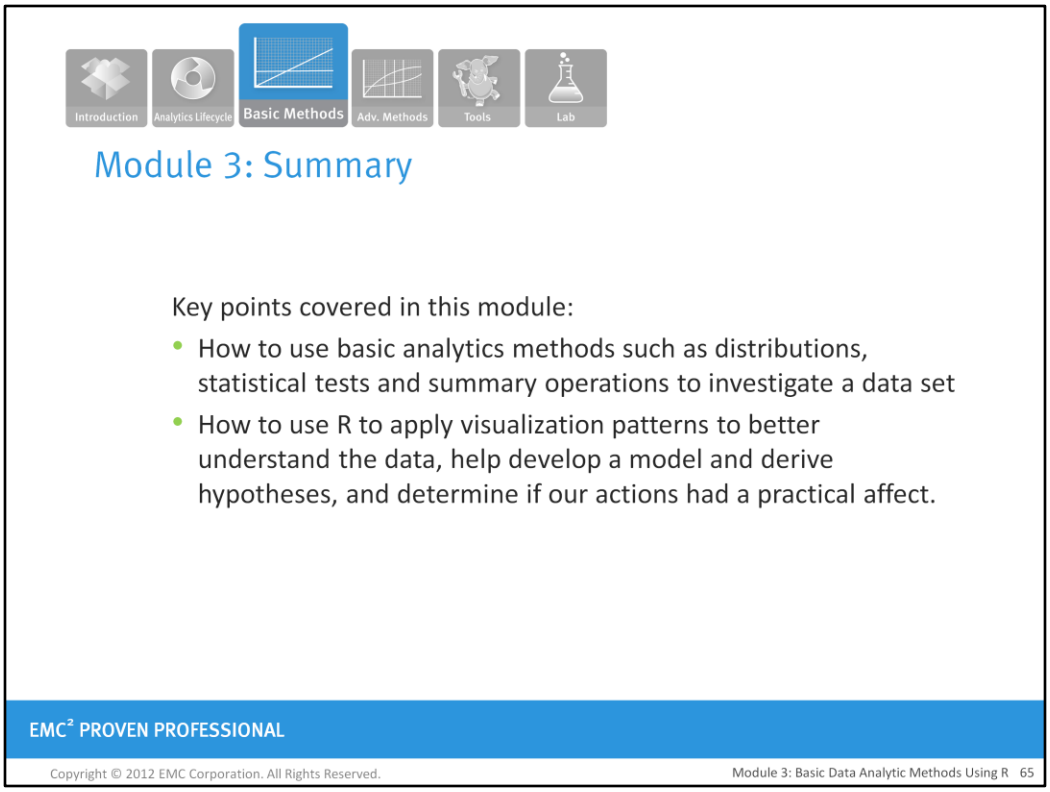
In this lab, you are asked to read in some data into a data frame and display, summarize, view analyze that data. First thing is to load that dataset into R, and then go from there. Instructions for the lab are contained in your lab guide.



This slide captures the workflow from part1 of Lab Exercise 3. Please take a moment to review each step before starting the lab.



This slide captures the workflow from part2 of Lab Exercise 3. Please take a moment to review each step before starting the lab.



These are the main points covered in the module. Please take a moment to review them.

In addition, pause and consider what you learned from the lab exercises in this module.

This slide intentionally left blank.

EMC² PROVEN PROFESSIONAL