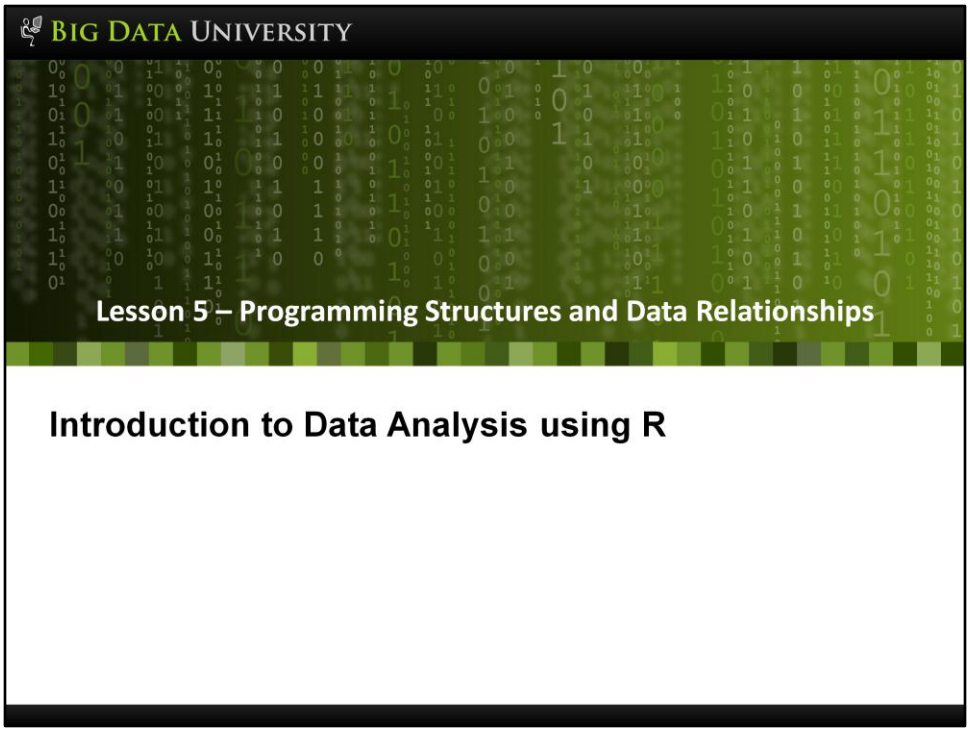
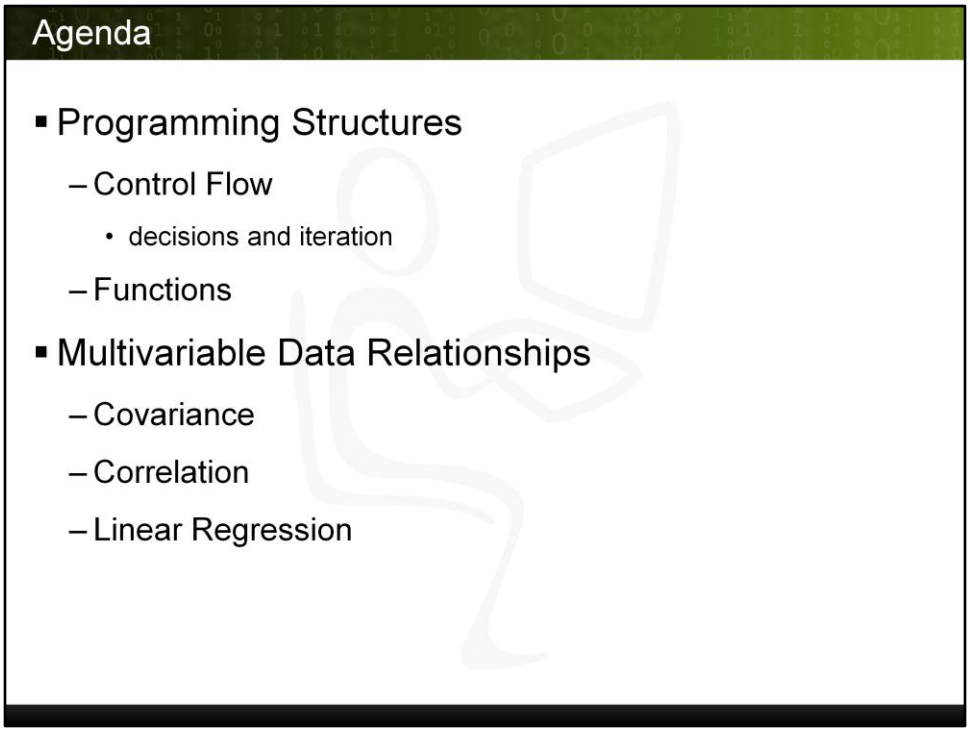


Welcome to this lesson on R programming structures and analysis of data relationships.

The programming structures that we will examine include: control flow statements and user defined functions.

In previous lessons we either performed summary or descriptive statistics across single variables, but in this lesson we will examine the interrelationships between variables. Interrelationship analysis comes in different forms and in this lesson we will examine covariance, correlation, and linear regression techniques.

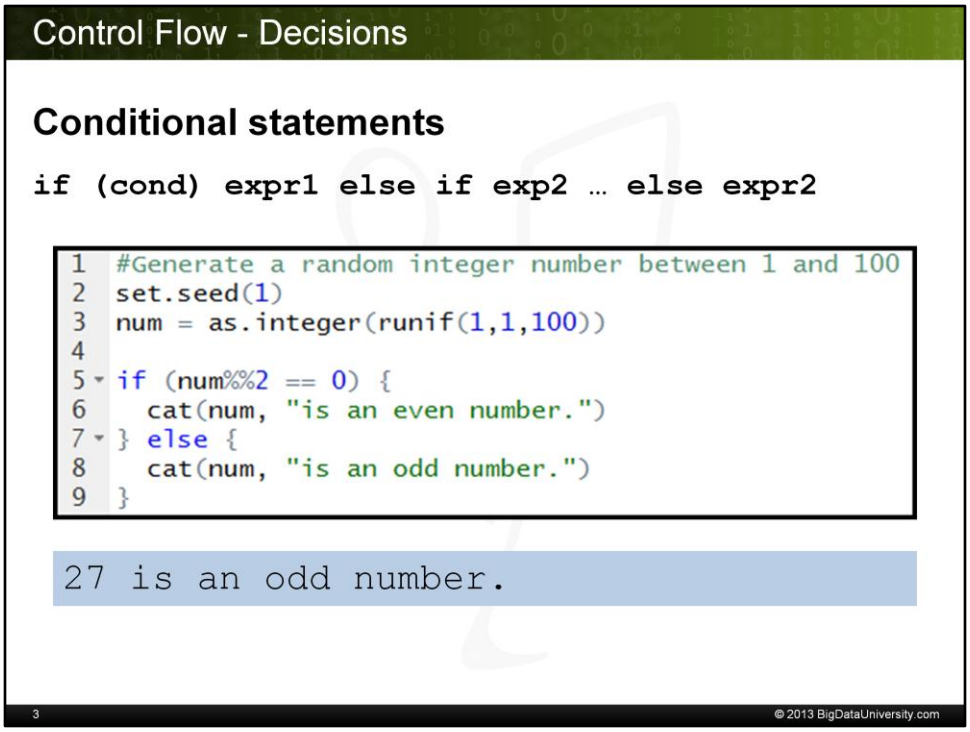


You have probably discovered that as we write more complex R scripts there are scenarios that require decision making and R provides many flexible options to control to flow of execution using conditional and iteration expressions.

We also mentioned that R is a functional and an object-oriented programming language. In this lesson we will examine how we define and use functions during our analysis.

We will learn how to create different types of R functions.

We have learned how to examine data trends over a single variable using a histogram or over 2 variables using scatterplots. In the final part of this lesson we will learn how to analyze the relationship between 2 variables through learning about covariance, correlation, and linear regression.



Let's examine decision making in R.

In this example we generate 1 random number between 1 and 100 using the runif() or random uniform distribution function. This function returns a floating point value and in this scenario we would like to simply deal with integer values so we use as.integer() function to discard the decimal portion of our number.

To enable reproducible results we have decided in line 2 to set a static seed value to R's pseudo-random number generator prior to the request for a number.

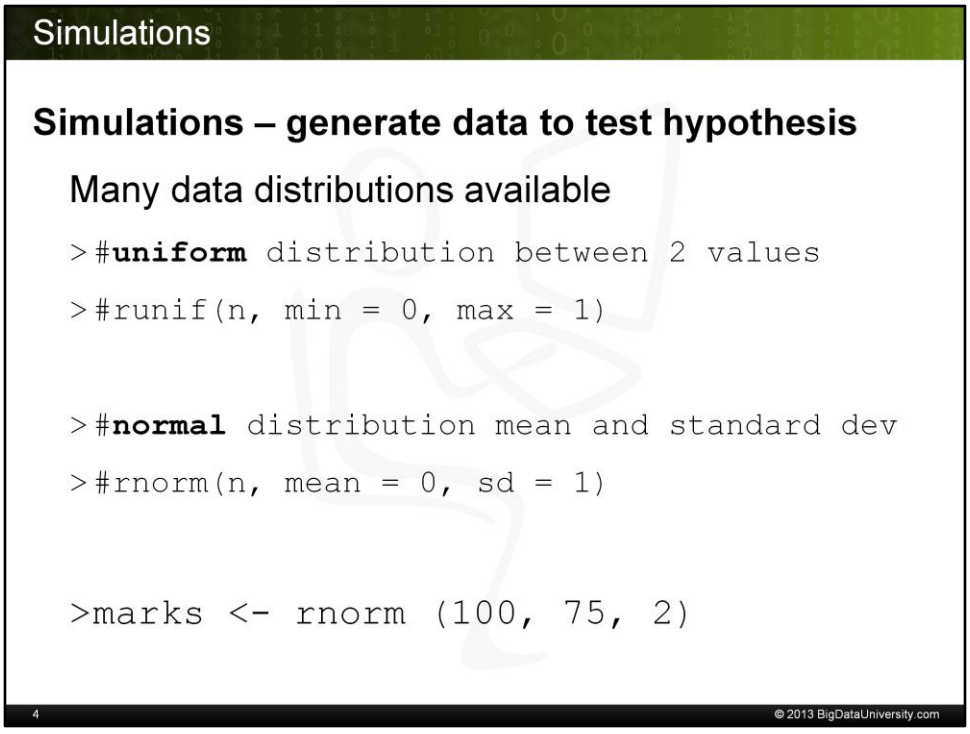
If you would like randomness when this script is executed you would obviously use a different seed value each time.

The purpose of the script is to print a statement that the random number is either an odd number of an even number.

We state the conditional expression instead parenthesis following the if statement. Here we are checking if the result of applying the modulus 2 operator to the number is zero (0) or not. If there is no remainder then we know that the number is indeed even and R will execute the code defined within the curly {} brackets.

If the first conditional expression is evaluated FALSE then the next conditional expression will be tested or in this case the final else clause will be executed.

Note the location of the brackets and indentation shown in this example. This example is consistent with various R coding style guidelines.

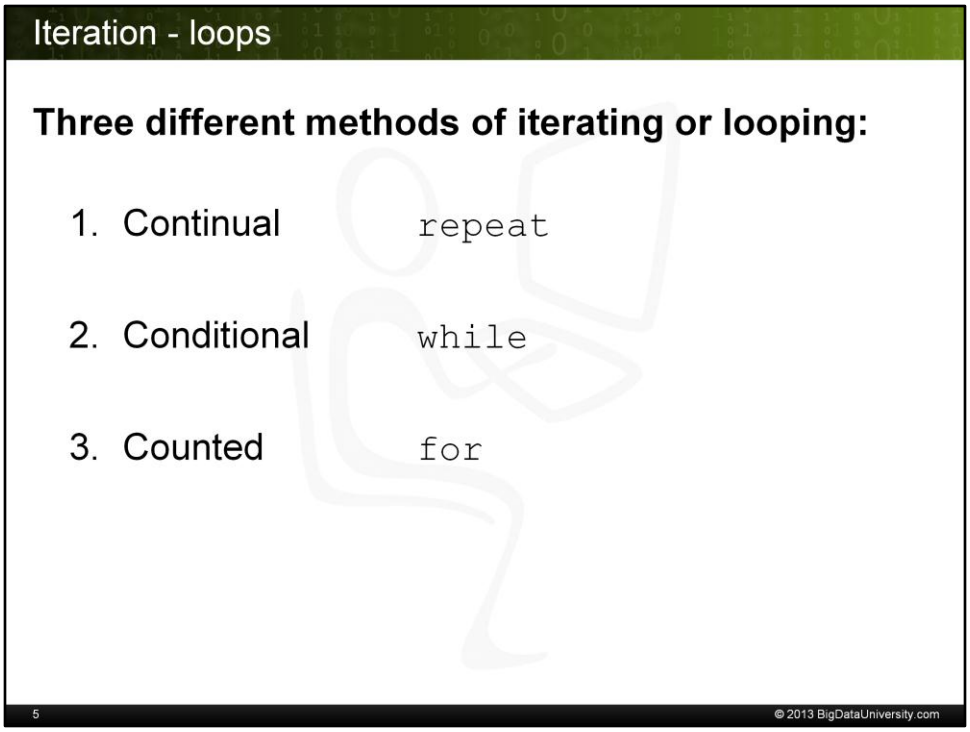


Throughout this lesson we will be using simulated data.

R has many methods of creating data that can be used for analysis. Two of the most common types of distributions are uniform distributions and normal distributions.

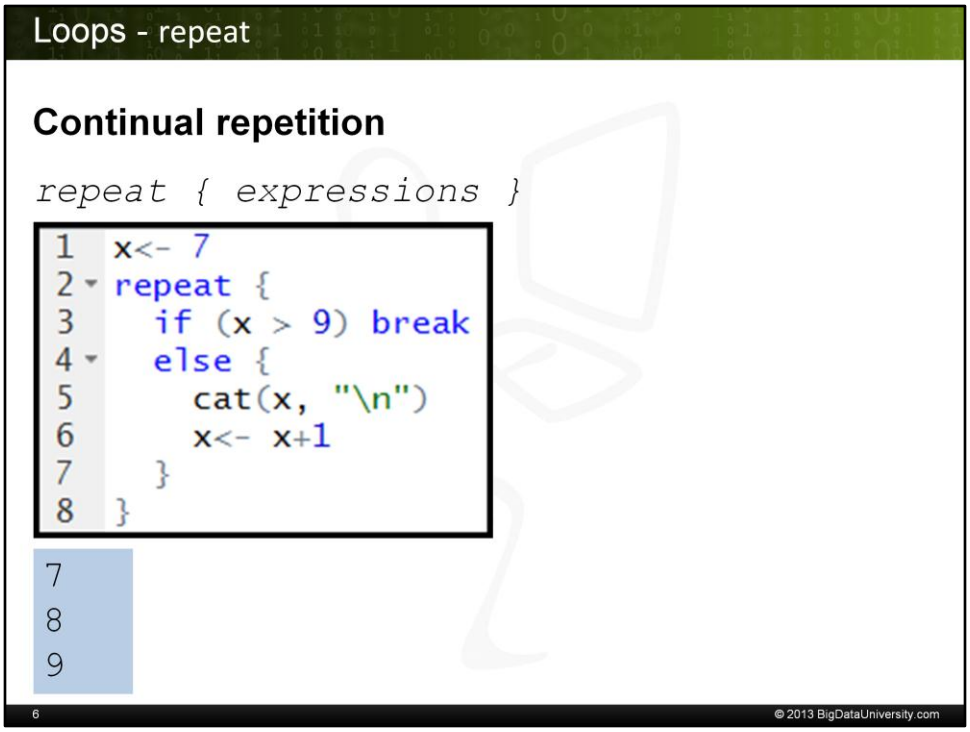
The functions runif and rnorm can be used to generate a vector of values with the corresponding distribution properties.

With normal distributions you provide a mean and a standard deviation. For example, if you wanted to create a large data set of 100 possible test scores with a normal distribution around a mean mark of 75 and a standard deviation or measure of distribution of 2 you could use the example as shown. To visually verify your data distribution you could use the plot or hist functions.



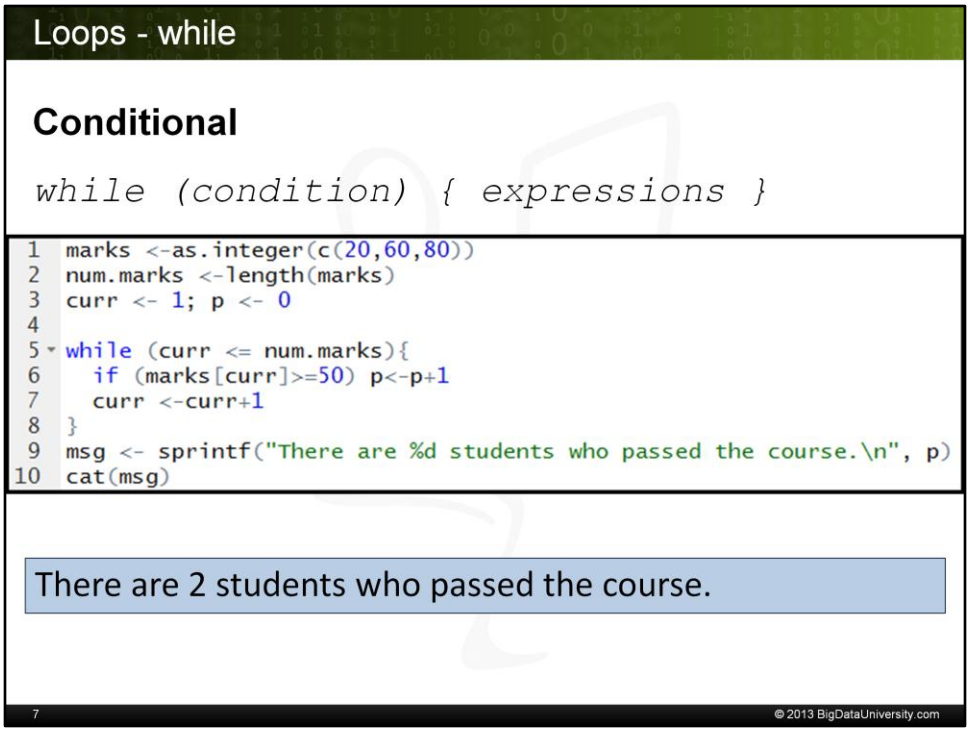
As with most functional programming languages R provides program structures to control iteration or looping behaviour.

We will examine each of these iterations options.



The repeat statement can be used to define an block of statements that will continue to iterate indefinitely.

The break statement is used to exit the loop. In this example we simply check if the value stored in the variable x is greater than 9 we exit the loop otherwise we will print out the current value of x and increment the value by one and the loop repeats. When the value stored in the variable x is greater than 9 we use the break statement to leave the loop and continue in our script.



A conditional looping structure defines the initial or precondition within parenthesis.

In this scenario our entry condition to the looping code involves checking if the value stored in the variable curr is less than or equal to the number of marks in our integer vector called marks.

Recall that in R code blocks are usually defined using the curly or brace brackets {}. It is possible to avoid using brace brackets if the expression is a single line and another option is to combine multiple expressions on a single line using the semi-colon (:). In this example we are actually doing both of these techniques. This may seem confusing so style guidelines should be considered if you have multiple developers working on a single project.

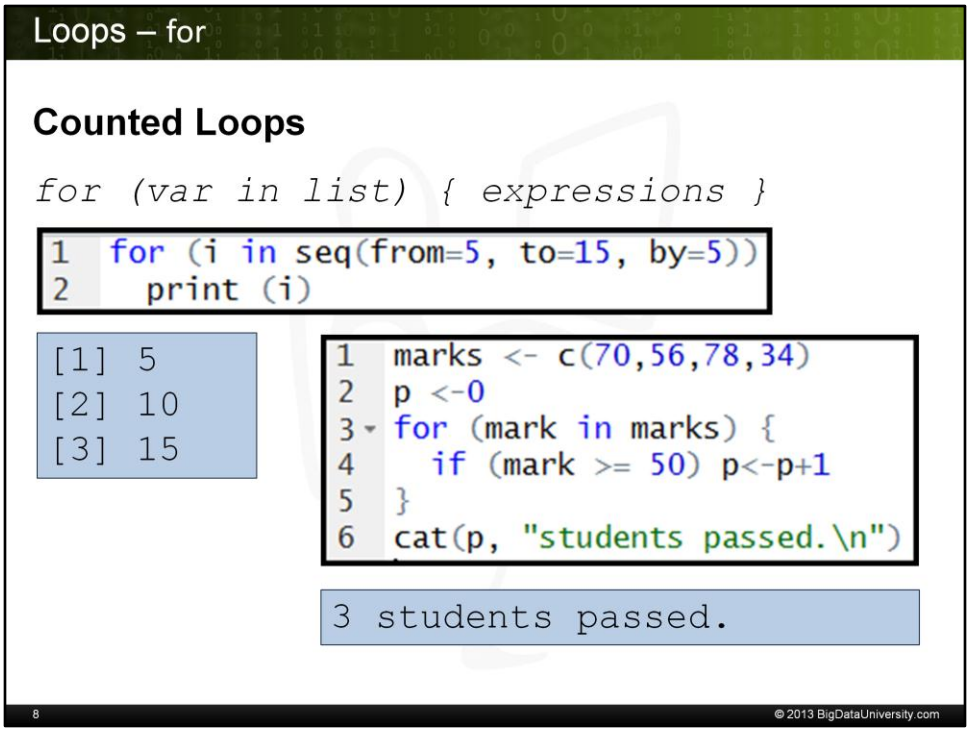
On line 3, we initialized the value of the variables p and curr on a single line.

On line 6, we use an if statement to check if a value in the marks vector is greater than or equal to 50, if it is we increment the value in the p variable by one.

On line 7 the if statement above is considered a complete expression and therefore the code on line 7 will execute for each iteration.

Once the looping condition is no longer true the msg will be created using the

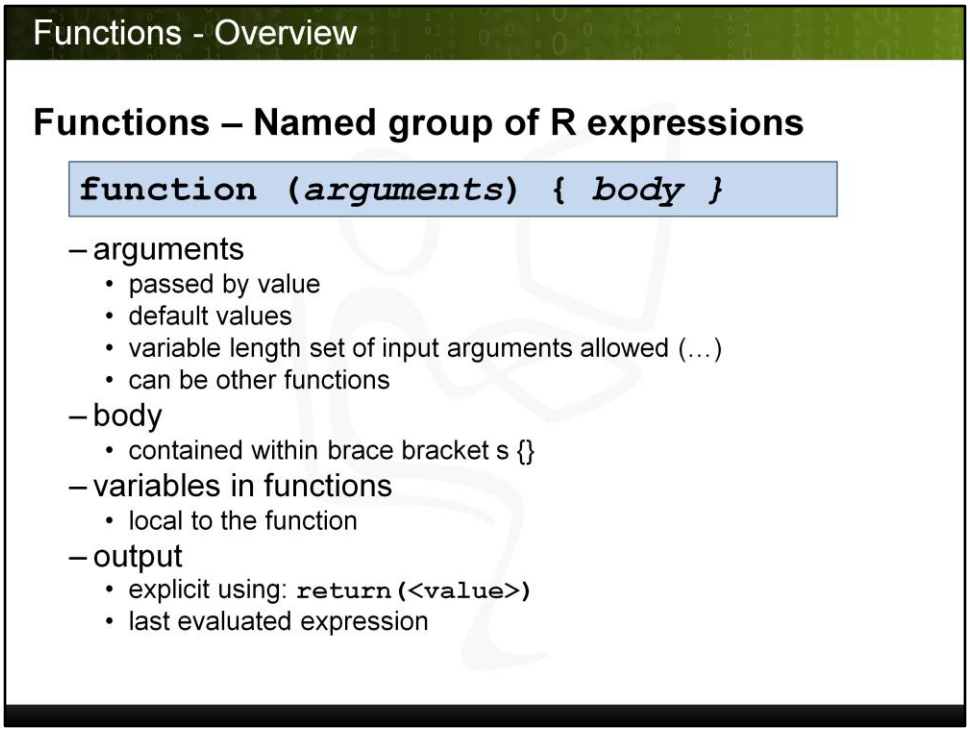
sprintf() function and then sent to the standard output display using the cat() function.



When the iteration scenario has a well defined number of iterations a for statement can be used in R.

The first example uses a sequence function to create a temporary vector of values. The value of I will begin with a value of 5 and then the value will be incremented by 5 until it reaches 15 for the final iteration.

The second example iterates over an existing vector of integers and since the index value of the marks vector is not required we can use a simplified version of the for statement. As we iterate through the values, the counter p will be incremented when the value in 50 or more. This style of for loop can be used to iterate over any vector data structure and it will always examine each element of the vector from the first position to the final position.



We have been using many built-in R functions throughout the course already.

After you start using R more, you will reach the point when you will want to create reusable sections of statements.

Functions are simply a named group of R expressions that are considered an R object of type function().

Input values or function arguments are passed to functions by value. Each argument is either matched with the function definition by position or by name.

We have used built-in functions that are considered generic, such as the plot() function. Generic functions are designed to accept a variety of different arguments. R functions can accept a variable length set of input arguments and they can even accept references to other functions.

When functions are called the actual arguments are assigned to local variables within the function. The statements in the function body are evaluated based on the input data. Control will be returned to the invoking function when the return() statement is reached or when the final expression in the function body is reached.

If there is no explicit data returned to the calling function, the output of the last expression will be returned.

Functions - Example

Function – `numPassed( course )`

```
1  numPassed <- function (course) {
2    # Determines the number of successful students.
3    # Args:
4    #   course: data frame or list with an attribute marks
5    # Returns:
6    #   The number of students with a mark of 50 or higher.
7    pass <- (course$Marks >= 50)
8    num.pass <- sum(pass)   # counts the TRUE values
9    return(as.integer(num.pass))
10 }
11
12 df1 <-data.frame("Marks" = c(70,34,67,78))
13 cat("Number of students passed =", numPassed(df1))
```

```
>#Display the function definition

>numPassed
```

Let's take a look at an example of a simple R function.

We want find that we are often writing code to determine the number of students who passed a course so we decide to define a function to simply our code.

The function is defined using the function() expression. The name of this function has been specified as numPassed and the function takes a single argument.

The comments on line 3 through 6 help explain the usage of the function includes the input and output datatypes expected.

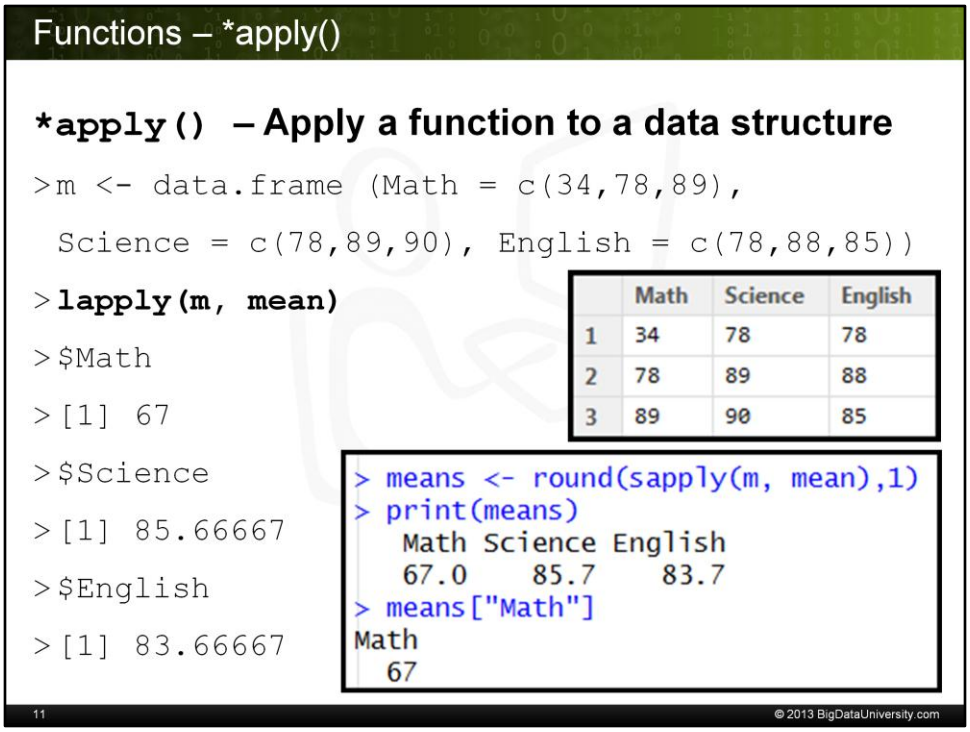
The body or content of the function is contained within the brace brackets ({}). We decided to use the vector enabled sum() function with a conditional expression to calculate the number of students with marks of 50 or higher. The pass object on line 8 is actually a vector of boolean values based on the criteria defined in line 7.

If you want to examine the source code of a function you can type the name of the function, without specifying any arguments, in the R Console.

This function will accepts a single dataframe or a list and it will perform a conditional expression tests for each element of the data structure. If the student has obtained a

mark of 50 or more a logical value of TRUE is generated. These logical values are collected in a local vector called pass. We then use the sum() function to count the number of TRUE values and return the data to the calling function as an integer.

Finally, the function is being called within the concatenate or cat() function.



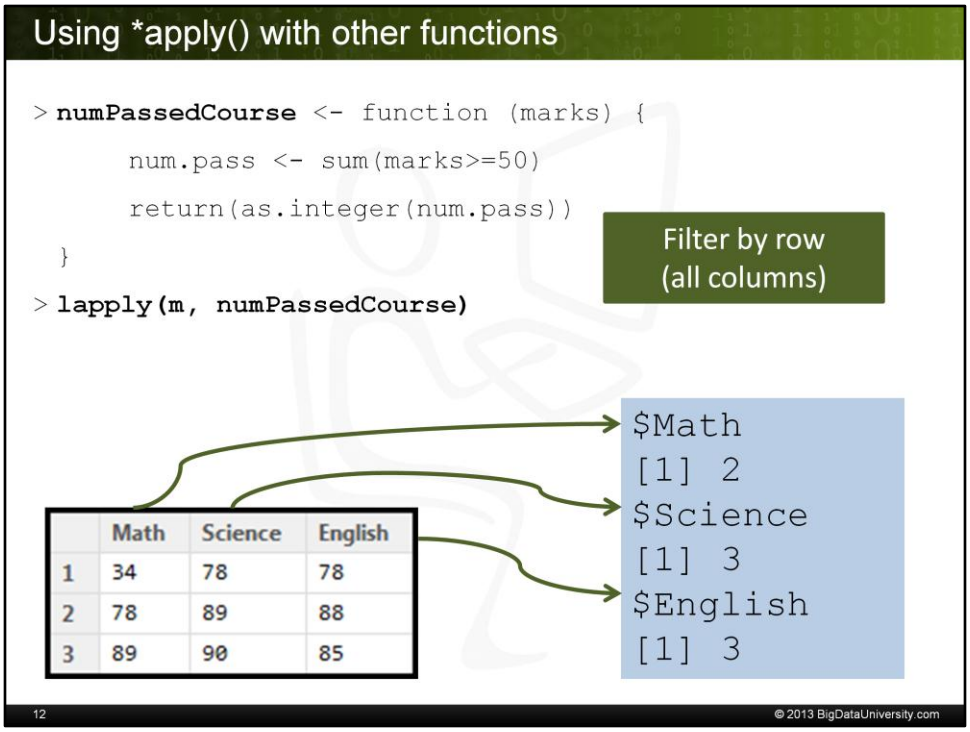
One of the most used features of R is the *apply() family of functions.

Here we are using the lapply() or List Apply function. The function that we are invoking is the mean or average function within the base package.

The mean function will be executed across each column of the data.frame called m.

The lapply() function always returns a list as its ouput. In our scenario, the list with the mean or average by subject is provided.

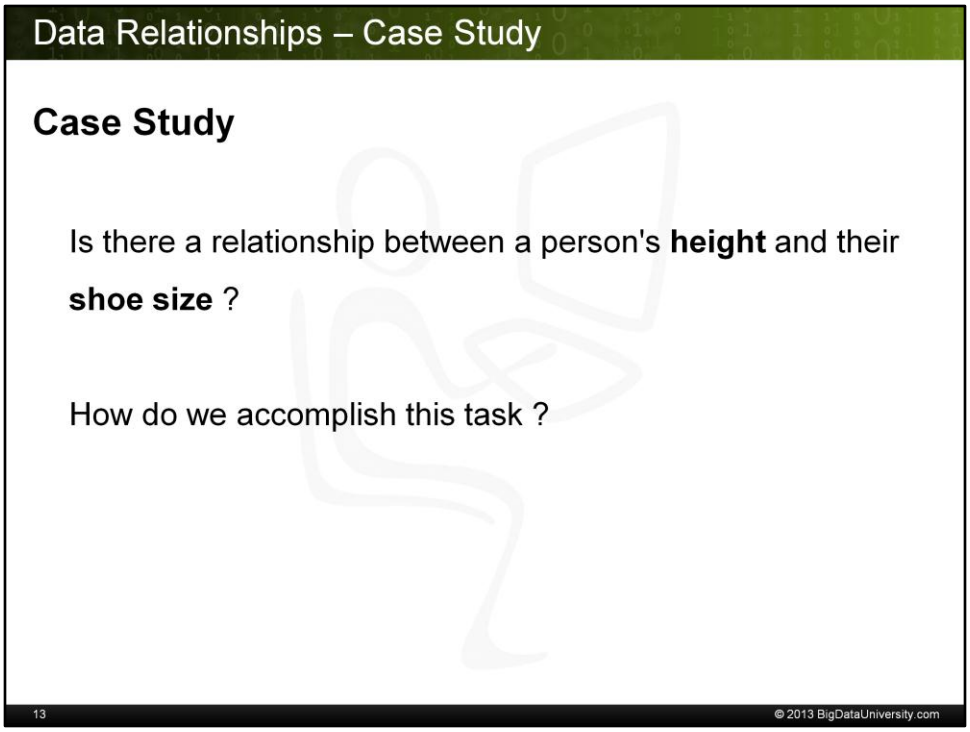
In the box of the right side we see how the sapply() or simplified apply function can be used. sapply() basically performs the same task as lapply(), but it coerces the output into a vector or a matrix. Notice how sapply generated a vector of double values where each element has a name associated with it. The final expression demonstrates how to retrieve the element named "Math".



Here we are using the same approach of using lapply() to invoke a function across a dataset.

In the previous example we used the built-in function mean and in this example we are using the function numPassedCourse to find the number of students who passed the course.

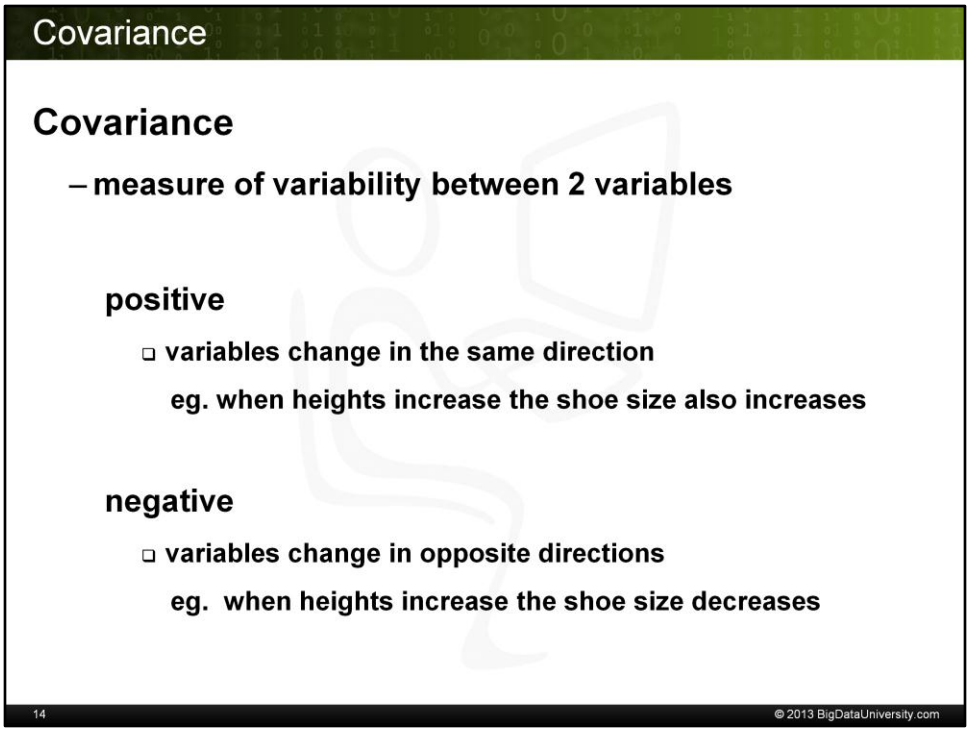
These examples of using the apply() function are quite simple, but you can probably see how a single line of R code can actually be performing a complex operation including iteration.



Now that we have learned about flow of control and functions, let's examine data relationships across multiple variables.

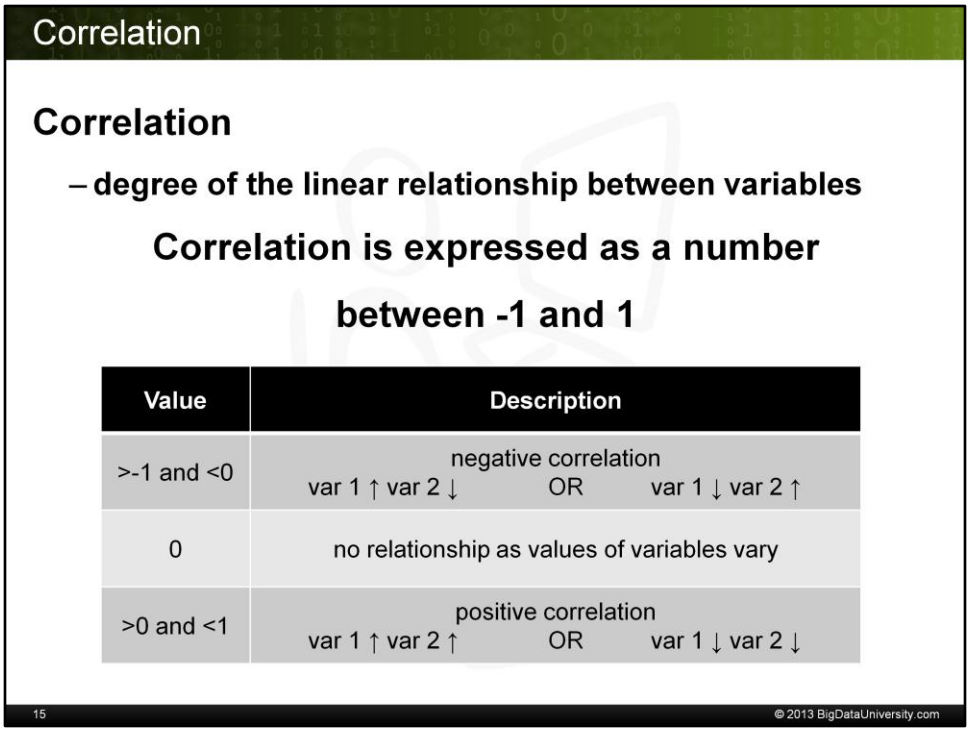
In our case study we want to determine if there is a relationship between the height of a person and their shoe size.

We will use R to simulate data sets for our analysis and we will use quantitative measures and visualizations to perform our data analysis.



There are a few different quantitative measurements that can be used to measure the variables of multiple variables.

A positive covariance would indicate that there is a positive linear relationship between the two variables, and a negative covariance would indicate the opposite.



Correlation is a similar statistical measurement of the degree of linear relationship between variables, but it has the added benefit that it has a well defined range of values. The coefficient range is from -1 to 1.

For example, a correlation coefficient of 0.8 would indicate that there is a strong linear relationship between the variables. Therefore, if variable 1 increases there is a similar observed increase in variable 2. If the correlation coefficient is close to zero (0) there is minimal relationship between the changes in values of the variables. Negative coefficients would indicate that the direction of change in value of the 2 variables are opposite one another.

R Analysis – Generate data and coefficients

```
1  #set the randomization seed for consistency
2  set.seed(1)
3  #uniform distribution of heights
4  height <- runif(100, 150, 230)
5
6  #uniform distribution of shoe sizes
7  shoe.size <- runif(100,6,14)
8
9  #determine covariance
10 cov1<-cov(as.shoe.size,height)
11
12 #determine correlation
13 cor1<-cor(shoe.size,height)
```

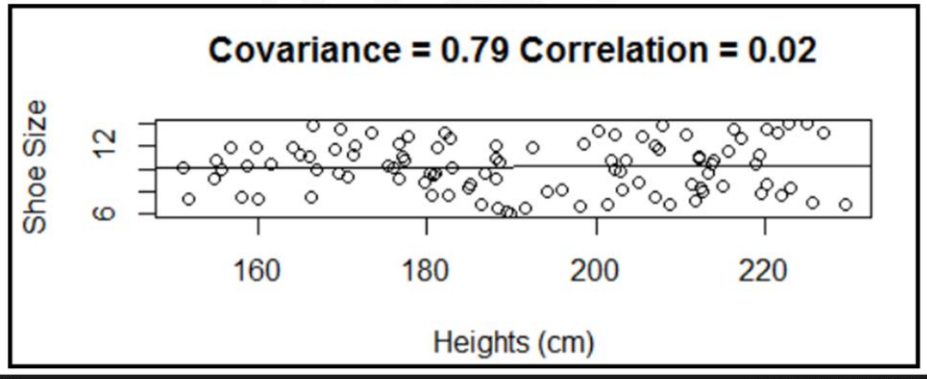

Here we are creating 2 uniform distributions of heights and shoe size for our analysis.

We use the covariance or cov() function to compute the covariance measurement and we use the correlation or cor() function to compute the correlation.

These values are stored in variables as they will be used in our final visualization.

R Analysis

```
15 #create a plot of heights vs. shoe sizes
16 msg <- sprintf("Covariance = %.2f Correlation = %.2f", cov1,cor1 )
17 plot (height,shoe.size, main=msg,
18       xlab = "Heights (cm)",
19       ylab = "Shoe Size")
20
21 #create a linear regression model and plot
22 lm1 <- lm (shoe.size~height)
23 abline(lm1)
```



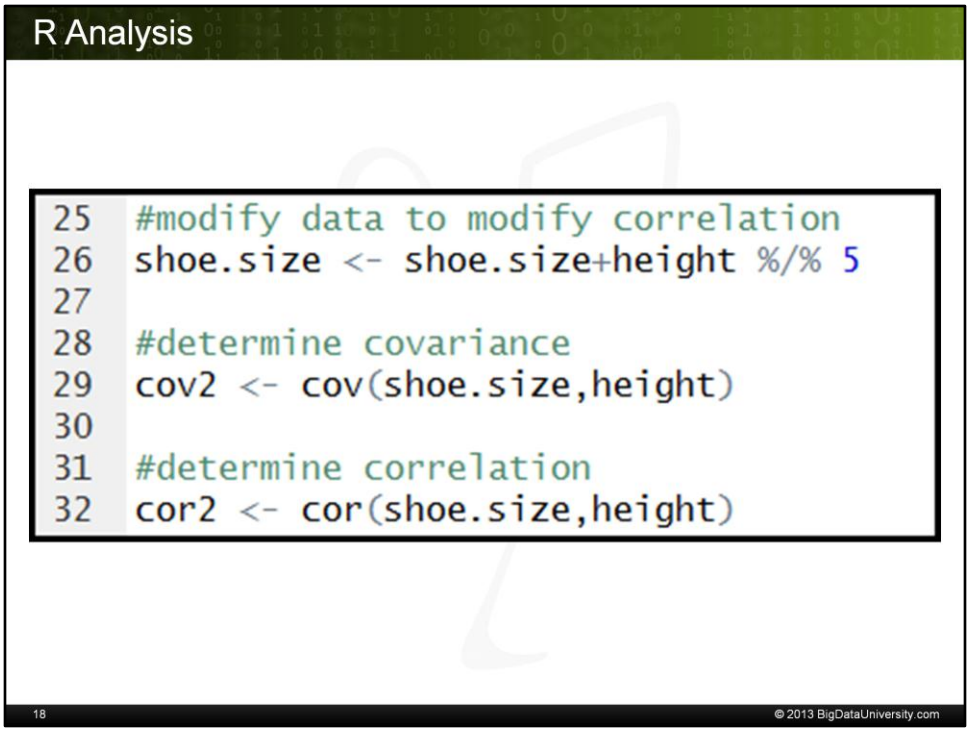
17 © 2013 BigDataUniversity.com

We are using the base graphics package plot() function to create a scatterplot of the generated data. Visually it seems like there is no relationship between these two measurements. The values of correlation and covariance support our visual understanding as their values are close to (0) zero.

On line 22 a linear regression model is created and stored in a variable called lm1.

The linear model function or lm() can be used to perform an analysis of multiple data sets and determine an linear approximation of the relationship between the variables. Notice that the lm() function is provided a formula. In R a formula consists of a tilde (~) character where in this case the variable on the left is considered the response and the variable on the right of the tilde is the term under consideration.

The object lm1 contains quite a large amount of detailed information. In this scenario we use the object to add a line to our graph to visualize the relationship.



We use a math expression on line 26 to adjust the shoe size for our next analysis.

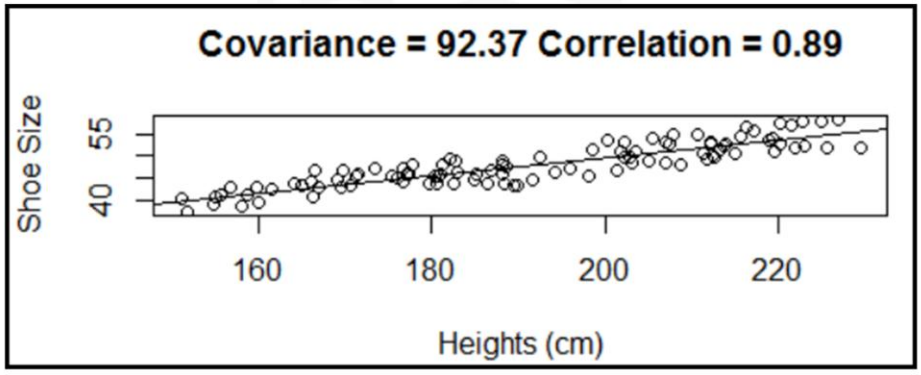
As an aside, the expression uses integer division to introduce a relationship between the variables.

Again we determine the covariance and correlation coefficients of the revised variables.

R Analysis

```
#create a new plot of heights vs. shoe sizes
msg <- sprintf("Covariance = %.2f Correlation = %.2f", cov2,cor2 )
plot (height,shoe.size, main=msg,
      xlab = "Heights (cm)",
      ylab = "Shoe Size")

#create a linear regression model and plot
lm2 <- lm (shoe.size~height)
abline(lm2)
```

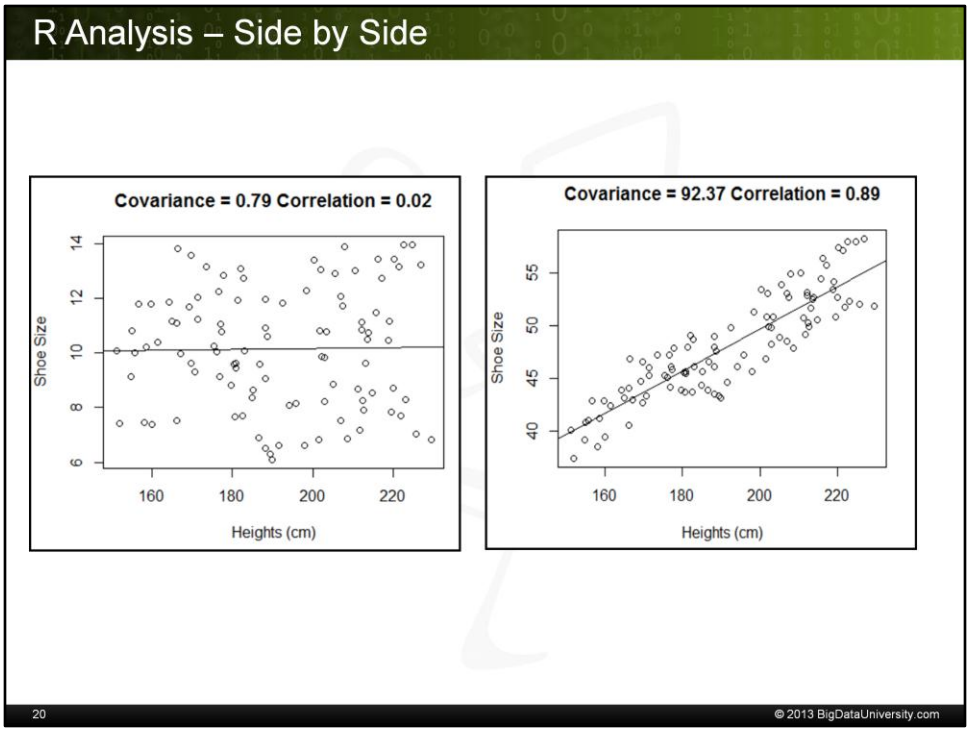


19 © 2013 BigDataUniversity.com

In our initial simulation of randomly generated uniform shoe sizes and heights there was very little correlation.

Here we see visually that there is a strong linear correlation and the value of the correlation coefficient agrees with our visual analysis.

We have also performed a revised linear regression model and plotted the result on the graph.



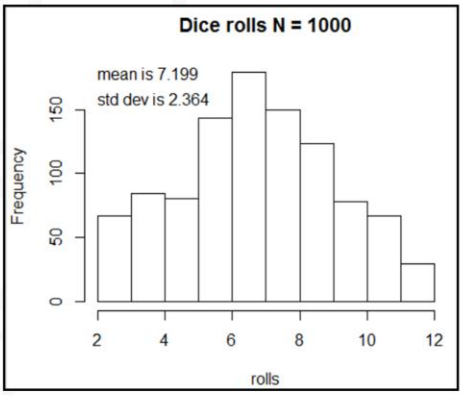
Here we see a summary of the before and after of our measurements for heights and shoe sizes.

The strong correlation is quite evident in the revised data for shoe sizes shown on the right.

Lab Exercises

Using functions

1. simulate rolling of dice
   - graph the results
2. Is there a correlation between stock prices and interest rates?
3. Use linear regression to predict the registrations for Big Data University next year
4. Prepare data (recode) for analysis using lapply()



There are many lab exercises that you can work with following this lesson including: dice simulations, correlations, and recoding of datasets using the apply family of functions in R.

Thank you for watching this lesson on R and enjoy working on the labs.