CSCE 587

Introduction to R and RStudio

Introduction to R and RStudio

- R is a programing language for statistical analysis
- Created by statisticians at The University of Auckland (in New Zealand) in the 1990's
- It's open source
- It's an interpreted language
- There is an enormous library of packages that can be installed
- Introduce R using RStudio
  - Install on your computer, if you'd like: https://posit.co/download/rstudio-desktop/

Introduction to R and RStudio

- RStudio is an Integrated Development Environment (IDE) for R
- You can install R and RStudio on your Apple, Linux or Windows computer, if you'd like: https://posit.co/download/rstudio-desktop/

R – Variables

- Variables are created by providing a name and assigning a value with either the **=** or **<-** operator
- For example,

```
> x = 3
```

  creates a variable named x and assigns it the value 3

- Similarly

```
> y <- 1.4
```

  creates a variable named y and assigns it the value 1.4

R – Variables

- Once a variable is assigned, it can be called by name to use its value

```
> x = 3
> x
[1] 3
> x + 5
[1] 8
```

R – Variables

- The type of value a variable holds can change from one command to the next

```
> x = 3
> x = “three”
```

R – Variables

- The type of value a variable holds can change from one command to the next.
- The class function can be called on an object to show the type of value(s) it holds at any point in time

```
> x = 3
> class(x)
[1] “numeric”
> x = “three”
> class(x)
[1] “character”
```

R – Help

- If a function has documentation associated with it, you can view that documentation by typing ? in front of the function name

```
> ? class
```

R – Vectors

- A vector is a sequence of elements of the same type
- Can create via concatenation with the c function

```
>  x = c( 1, 1, 2, 3, 5, 8, 13 )
```

- Can create with an arithmetic sequence with the seq function

```
>  y = seq( from = 1, to = 13, by = 2)
```

- Can create a repeated sequence with rep

```
>  z = rep( x = 4,  times = 9)
```

R – Vectors

- Note: If a function has named parameters, you can send them in any order using the parameter names, or you can send them in the expected order without using the names
- For example, each of the following creates the same vector

```
> seq( from = 1, to = 13, by = 2)
> seq(1, 13, 2)
> seq( by = 2, to = 13, from = 1 )
```

R – Some Functions to use with Vectors

- length – for the number of values
- max – for the largest value
- min – for the smallest value
- mean – for the mean value

```
> x = seq( 2, 11, 1 )
> length(x)
[1] 10
> max(x)
 [1] 11
> min(x)
[1] 2
> mean(x)
 [1] 6.5
```

R – Subsetting a Vector

- Use the [] operator with a single integer or a range *from:to*

```
> x = c("eh", "bee", "see", "dee", "eee")
> x[3]
[1] "see"
> x[2:4]
[1] "bee" "see" "dee"
```

R – Subsetting a Vector

- We can also use - to exclude elements

```
> x = c("eh", "bee", "see", "dee", "eee")
> x[-3]
[1] "eh" "bee" "dee" "eee"
> x[-(2:4)]
[1] "eh"  "eee"
```

R – Subsetting a Vector

- We can also use a vector as the as the selection to pick out particular elements from the vector

```
> x = c("eh", "bee", "see", "dee", "eee")
> x[c(2,4)]
[1] "bee" "dee"
> x[c(4,2)]
[1] "dee" "bee"
> x[-(c(2,4))]
[1] "eh" "see" "eee"
```

R – Subsetting a Vector

- We can also use a vector as the as the selection… So suppose we wanted to get the values in reverse order

```
> x = c("eh", "bee", "see", "dee", "eee")
> x[seq(from=length(x), to=1, by=-1)]
[1] "eee" "dee" "see" "bee" "eh"
```

R – Relational Operators

- Relational operators compare values and return TRUE or FALSE
- == returns true if the values are the same, false if they're different
- != returns true if the values are different, false if they're the same
- Examples:

```
> y = 7
> y == 7
[1] TRUE
> y == 4
[1] FALSE
> y != 4
[1] TRUE
```

R – Relational Operators

- When used with a vector, these operators will return a vector of logical values
- Examples:

```
> x = seq(6, 18, 3)
> x
[1] 6  9  12  15  18
> x == 15
[1] FALSE  FALSE  FALSE  TRUE  FALSE
> x != 9
[1] TRUE  FALSE  TRUE  TRUE  TRUE
```

R – which Function

- The which function returns a vector of indices where an expression evaluates to true
- Examples:

```
> x = c( 5, 6, 7, 6, 8, 6, 1, 3 )
> x == 6
[1] FALSE  TRUE  FALSE  TRUE  FALSE  TRUE FALSE FALSE
> which(x == 6)
[1]  2  4  6
> which(x == max(x))
[1]  5
```

R – Matrices

- A matrix is like a two-dimensional vector
- All elements must be of the same type
- Create using the matrix function
  - matrix(data=NA, nrow=1, ncol=1, byrow=FALSE, dimnames=NULL)
- Example:

```
> matrix(10:29, 4, 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   14   18   22   26
[2,]   11   15   19   23   27
[3,]   12   16   20   24   28
[4,]   13   17   21   25   29
```

R – Matrices

- matrix(data=NA, nrow=1, ncol=1, byrow=FALSE)
- Example:

```
> matrix(10:29, 4, 5, TRUE)
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
[3,]   20   21   22   23   24
[4,]   25   26   27   28   29
```

R – Matrix Functions

- The t function will transpose a matrix (swap the rows and columns)

```
> x = matrix(10:19, 2, 5, TRUE)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> t(x)
     [,1] [,2]
[1,]   10   15
[2,]   11   16
[3,]   12   17
[4,]   13   18
[5,]   14   19
```

R – Multiplication

- The * operator does pairwise multiplication

```
> x = matrix(1:6, 2, 3, TRUE)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x * x
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

R – Multiplication

- The %* % operator is for matrix multiplication

```
> x = matrix(1:6, 2, 3, TRUE)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x %*% t(x)
     [,1] [,2]
[1,]   14   32
[2,]   32   77
```

R – Matrix Functions

- The dim function returns the matrix dimensions

```
> x = matrix(10:19, 2, 5, TRUE)
> dim(x)
[1] 2 5
```

- The nrow function returns the number of rows

```
> nrow(x)
[1] 2
```

- The ncol function returns the number of columns

```
> ncol(x)
[1] 5
```

R – Subsetting a Matrix

- Use the [] operator with row and/or column indices

```
> x = matrix(10:19, 2, 5, TRUE)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> x[1, 3]
[1] 12
> x[1, ]
[1] 10 11 12 13 14
> x[,2]
[1] 11 16
```

R – Subsetting a Matrix

- We can use a vector for particular rows or columns to include

```
> x = matrix(10:19, 2, 5, TRUE)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> x[ ,c(2,4,5)]
     [,1] [,2] [,3]
[1,]   11   13   14
[2,]   16   18   19
```

R – Subsetting a Matrix

- We can use a vector for particular rows or columns to include

```
> x = matrix(10:19, 2, 5, TRUE)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> x[ ,c(5,2)]
     [,1] [,2]
[1,]   14   11
[2,]   19   16
```

R – Subsetting a Matrix

- We can use a vector for particular rows or columns or exclude (using -)

```
> x = matrix(10:19, 2, 5, TRUE)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> x[ ,-c(5,2)]
     [,1] [,2] [,3]
[1,]   10   12   13
[2,]   15   17   18
```

R – which with Matrices

- The which function will return the position, sequentially from left to right, row by row, in a matrix

```
> x = matrix(10:19, 2, 5, TRUE)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> x == 17
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] FALSE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE  TRUE FALSE FALSE
> which(x == 17)
[1] 6
```

R – Building Matrices from Vectors

- We can bind vectors together as rows in a matrix with rbind

```
> v = c(1, 8, 7)
> m = rbind(v, v)
  [,1] [,2] [,3]
v    1    8    7
v    1    8    7
```

R – Building Matrices from Vectors

- We can bind vectors together as columns in a matrix with cbind

```
> v = c(1, 8, 7)
> m = cbind(v, v)
     v v
[1,] 1 1
[2,] 8 8
[3,] 7 7
```

Matrix Example Problem

- Create a matrix with 5 rows and 10 columns containing the odd integers from 1 to 99, shown below

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
[1,]	1	11	21	31	41	51	61	71	81	91
[2,]	3	13	23	33	43	53	63	73	83	93
[3,]	5	15	25	35	45	55	65	75	85	95
[4,]	7	17	27	37	47	57	67	77	87	97
[5,]	9	19	29	39	49	59	69	79	89	99

- Compute the sum of the values in the even indexed columns of row three, highlighted below

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
[1,]	1	11	21	31	41	51	61	71	81	91
[2,]	3	13	23	33	43	53	63	73	83	93
[3,]	5	15	25	35	45	55	65	75	85	95
[4,]	7	17	27	37	47	57	67	77	87	97
[5,]	9	19	29	39	49	59	69	79	89	99

Matrix Example Problem

- Create a matrix with 5 rows and 10 columns containing the odd integers from 1 to 99, shown below
- Compute the sum of the values in the even indexed columns of row three, highlighted below

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
[1,]	1	11	21	31	41	51	61	71	81	91
[2,]	3	13	23	33	43	53	63	73	83	93
[3,]	5	15	25	35	45	55	65	75	85	95
[4,]	7	17	27	37	47	57	67	77	87	97
[5,]	9	19	29	39	49	59	69	79	89	99

```
> m = matrix(seq(1,100,2), 5, 10)
> m
> sum( m[ 3, seq(2,10,2) ) 
```