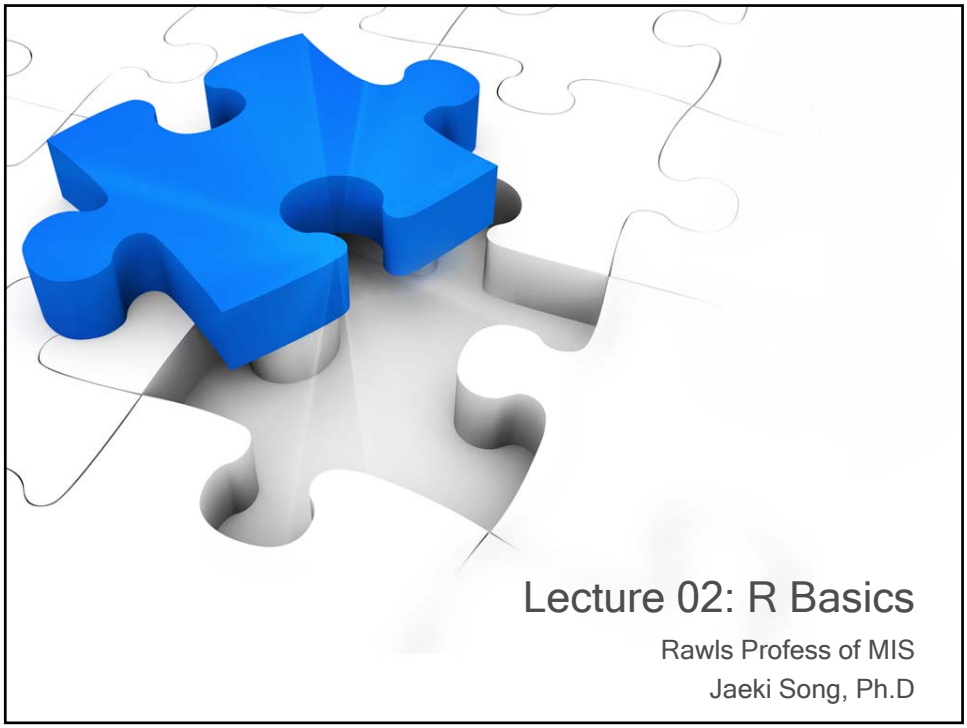
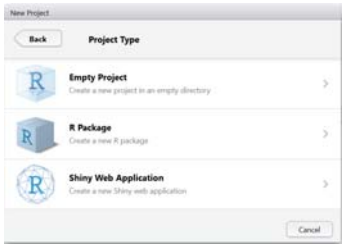


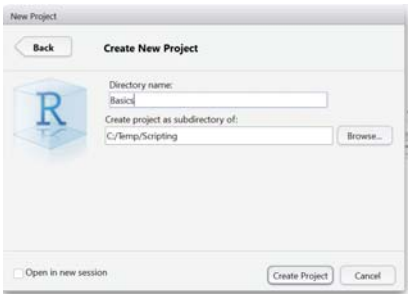
Lecture 02: R Basics

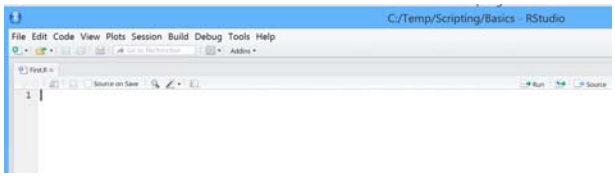
Rawls Profess of MIS
Jaeki Song, Ph.D

Overview: A First R Session

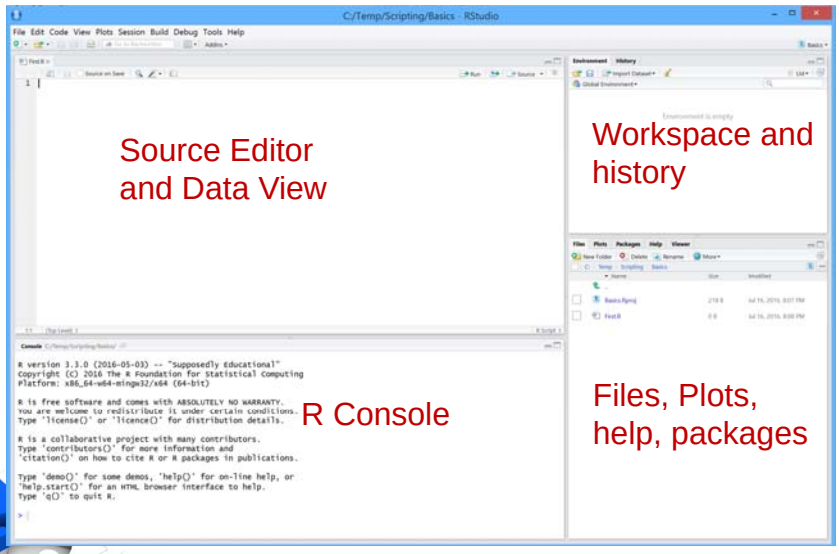


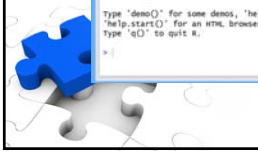
Overview: A First R Session





Overview: A First R Session





Variables

- Name
  - a, b, a1, a2, .x
- Variable assignment

```
a <-3
b = 4.5
c <- a+b
print (c)

d <- "hello"
print (d)
```

Variable

- Character data
  - categorical data

```
> X <- factor (c("a", "b", "c"))
> X
[1] a b c
Levels: a b c
```

Vectors

- A collection of elements
  - all of the same type
  - vectors in R do not have a dimension
    - not like the mathematical vector where there is a difference between row and column orientation
  - the most common way to create a vector is with *c*

```
> x <- c (1,2,3,4,5)
> x
[1] 1 2 3 4 5
```

Vectors

- seq( ) function

```
> seq(1,5)
[1] 1 2 3 4 5
> seq(1,5,2)
[1] 1 3 5
```

- rep ( ) function

```
> rep (1:2, 5)
 [1] 1 2 1 2 1 2 1 2 1 2
> rep(1:2, each =5)
 [1] 1 1 1 1 1 2 2 2 2 2
```

Lists

- A container is needed to hold arbitrary objects of either the same type of varying types.
  - R accomplishes this through list
  - Store any number of items of any type

```
> list (1,2,3)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3
```

```
> list(c(1,2,3))
[[1]]
[1] 1 2 3
```

Lists

```
> x <- list (name ="song", height = 70)
> x
$name
[1] "song"

$height
[1] 70
```

```
> x <- list (name ="song", height = c(1,3,5))
> x
$name
[1] "song"

$height
[1] 1 3 5
```

```
> list (a =list(val=c(1,2,3)), b=list(val=c(1,2,3,4)))
$a
$a$val
[1] 1 2 3


$b
$b$val
[1] 1 2 3 4
```

Matrix

```
> matrix(c(1,2,3,4,5,6,7,8,9), nrow=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> matrix(c(1,2,3,4,5,6,7,8,9), ncol=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
> matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Matrix

```
> x<-matrix(c(1,2,3,4,5,6,7,8,9), nrow=3)
> x[2,3]
[1] 8
```

```
> x[1:2,]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

```
> x[-3,]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

```
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x%*%x
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

```
> y<-matrix(c(1,2,3,4), ncol=2)
> y
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> solve(y)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

Arrays

- essentially a multidimensional vector
  - must all be of the same type
  - individual elements are accessed in a similar fashion using square brackets

```
> matrix(1:12, ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> array(1:12, dim=c(3,4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Arrays

```
> array(1:12, dim=c(2,2,3))
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

, , 3

     [,1] [,2]
[1,]    9   11
[2,]   10   12
```

```
> array(1:12, dim=c(2,3,2))
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Data Frame

- Like an excel spreadsheet
  - has columns and rows
  - data.frame( )

```
> x <- 5:1
> y <- -3:1
> z <- c ("Footbal1", "Baseball", "Rugby", "Basketball", "Soccer")
> theDF <-data.frame(x,y,z)
> theDF
  x  y          z
1 5 -3   Footbal1
2 4 -2   Baseball
3 3 -1      Rugby
4 2  0 Basketball
5 1  1     Soccer
```

Data Frame

- Functions
  - str( )

```
> str(d)
'data.frame':   5 obs. of  2 variables:
 $ x: num  1 2 3 4 5
 $ y: num  2 4 6 8 10
```

  - data transformation
    - class( )
      - factor(), as.factor( )

```
> class(c(1,2))
[1] "numeric"
```

Control Statements

- Allow to control the flow of programming
  - cause different things to happen depending on the values of tests

```
if (TRUE){
  print('hello')
}
else
{
  print ('world')
}
```

Control Statement

- *for*- statement
  - the most commonly used loop
    - iterates over an index
      - provided as a vector

```
for (i in 1:10){
  print (i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Control Statement

- *while* - statement
  - simple to implement
    - runs the code inside the braces repeatedly as long as the condition is true

```
i<-0
while(i<10){
  print (i)
  i <-i +1
}
```

```
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

Function

- Reduce redundancy whenever possible
  - running the same code repeatedly, it is a good idea turn it into a function
  - function.name <- function ( )

```
say.hello <-function ()
{print ("hello world!")}

say.hello()
```

```
[1] "hello world!"
```

```
fibo <- function (n){
  if (n==1 || n==2){
    return (1)
  }
  return (2)
}

fibo (1)

fibo (5)
```

```
> fibo (1)
[1] 1
>
> fibo (5)
[1] 2
```