CSCE 587 / STAT 587
Intro to R/RStudio

Part 3

Transferring files from the VM to your computer

1) If you're on a Mac, open a terminal window. If you're on a Windows computer, open Windows PowerShell
2) Type the following at the prompt:

   sftp -P555 student@vm-hadoop-XX.cse.sc.edu
3) Enter your password
4) Type the following at the sftp prompt

   get *filename*
5) Type exit at the sftp prompt

Subsetting: More Tricks

Changing values with subsetting. Example:

```
> fib[1]=3; fib
[1] 3 2 3 5 8

> fib[2:4] = c(3,1,4); fib
[1] 3 3 1 4 8
```

Subsetting: More Tricks

What about fib[]

```
> fib[]
[1] 3 3 1 4 8  # same as fib
```

Consider fib[] = 7: what will this do?

```
> fib[] = 7; fib
[1] 7 7 7 7 7
```

Consider fib[] = 7 vs fib=7?

Data Types in R

Type	**Definition**
logical	boolean: TRUE or FALSE
numeric	integer or floating point
complex	complex number
character	character string
function	encapsulated r expressions

Data Classes in R

Class	**Definition**
vector	1D array of elements of same type
matrix	2D array of elements of the same type
array	nD array of elements of the same type
data.frame	2D array; elements within a column must be of same type
List	1D array of elements allowing mixing of types
factor	categorical variable with defined values

Type Coercion

Type coercion: converting one type to another type

Automatic conversion examples:

```
> fib = c(1,2,3,5,8); x = c(TRUE, FALSE, TRUE, FALSE); c(fib,x)
[1] 1 2 3 5 8 1 0 1 0

> c(fib, "fib")
[1] "1"   "2"   "3"   "5"   "8"   "fib "

> c(c(TRUE, FALSE, TRUE, FALSE), "fib")
[1] "TRUE"  "FALSE" "TRUE"  "FALSE" "fib"
```

Surprising Coercions

First define cars:

```
> cars = factor( c("red", "blue", "green", "blue", "red"))
> cars
[1] red   blue  green blue  red
Levels: blue green red
```

Surprising Coercions

```
> cars
[1] red   blue  green blue  red
Levels: blue green red
```

Unexpected results:

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

```
> c(cars, FALSE)
[1] 3 1 2 1 3 0
```

```
> c(cars,"red")
[1] "3"   "1"   "2"   "1"   "3"   "red"
```

Explicit Coercion

Don’t be a victim of default coercion!

Use “as” functions to force your intent.

```
> as.numeric("3.142")
[1] 3.142
> as.complex("3.142")
[1] 3.142+0i
> as.factor(c("red", 7, "5"))
[1] red 7   5
Levels: 5 7 red
> as.character(c("red", 7, "7"))
[1] "red" "7"   "7"
```

Special values: NA, NULL, NaN, and Inf

Value	Description
NA	Represents the missing value
NULL	Represents the null/empty value
NaN	Represents a value that is NOT a valid number
Inf	Represents the infinite value

Special values: NA, NULL, NaN, and Inf

Value	**Test**
NA	is.na()
NULL	is.null()
NaN	is.nan()
Inf	is.infinite()

Examples of Tests for NA

```
> a= NA
> is.na(a)
[1] TRUE

> a==NA
[1] NA                  # The value of “a” is missing/can not evaluate

> a==7
[1] NA                  # The value of “a” is missing/can not evaluate
```

Examples of Tests for NULL

```
> a = c()
> is.null(a)
[1] TRUE

> a==c()
[1] logical(0)          The argument is an empty value

> a==7
[1] logical(0)          The argument is an empty value
```

Character Objects

Character objects are ***immutable strings***

Can not be modified

Not a character array

```
> length("This is a string")
[1] 1

> nchar("This is a string")
[1] 16
```

➔ Build the string you want! Use paste()

Character Objects

```
> paste("a", "b", "c")
[1] "a b c "

> paste("a", "b", "c", sep="+")
[1] "a+b+c "

> paste("x", 1:5, sep="")
[1] "x1" "x2" "x3" "x4" "x5"

> paste("x", 1:5, sep="", collapse=" + ")
[1] "x1 + x2 + x3 + x4 + x5"
```

Character Functions

Name	**Function**
substr(string, start, end)	extract/replace substring

```
> x = "12345"
> substr(x,3,3)
[1] "3"

> substr(x,3,3); x
[1] "3"
[1] "12345“

> substr(x,3,3)="c"; x
[1] "12c45"
```

Character Functions

Name	Function
strsplit(string, split, fixed=F)	split a string based on a pattern

```
> strsplit("String with white space", " ")
[[1]]
[1] "String" "with"   "white"  "space"
```

More Character Functions

Name	**Function**
toupper()	make characters upper case
tolower()	make characters lower case
grep()	global regular expression string matching
gsub()	global regular expression string replacement

Arrays

Name	**Data Structure**
Vector	1D array
Matrix	2D array
3D array	3D array
4D array	4D array

etc….

Use the array() function to create

Syntax array(data, dim)

Arrays

```
> array(1:6, c(2,3))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Arrays

```
> array(1:12, 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
```

Note: same as array(1:6, c(2,3)) from preceding slide

Note: same as array(7:12, c(2,3))

Data Frame

Data Frame – 2D data structure. Requires columns to be of same type

Use data.frame() to create

Syntax data.frame(columns)

```
> col1 = c(1,3,5)
> col2 = c("x","y","z")
> col3 = c(TRUE, TRUE, FALSE)
> (d = data.frame(col1, col2, col3))
  col1 col2  col3
1    1    x  TRUE
2    3    y  TRUE
3    5    z FALSE
```

Data Frame from the web

1) In the terminal window (not console) use:

wget https://cse.sc.edu/~bhipp/587/scores.csv --no-certificate-check

to download csv file to vm

2) Use ``Import Data Set''tab (Environment) to load data frame

A. Use ``From Text (base)'' (first choice in pulldown menu)
B. Select scores.csv as file to load
C. In preview, click Yes button for ``Heading"
D. Click on Import button

Data Frame from the web

```
> scores
    Name      id Quiz1 Exam1 Exam2 Quiz2 Exam3
1  Susan  123123    34    65    54     8    34
2   John  234234    35    47    65     7    32
3    Bob  345345    32    62    56     9    31
4   Bill  456456    15    61    46     6    36
5   Mary  675677    64    58    71     8    32
6   Paul  678678    36    63    29     7    38
7   Nepo 2354567    53    57    48     9    34
```

Data Frame: Column Selection

Method 1: use $

```
> scores$Name
[1] Susan John  Bob   Bill  Mary  Paul  Nepo
Levels: Bill Bob John Mary Nepo Paul Susan
```

Method 2: use [[]]

```
> scores[["Name"]]
[1] Susan John  Bob   Bill  Mary  Paul  Nepo
Levels: Bill Bob John Mary Nepo Paul Susan
```

Method 3: use index [], i.e. traditional indexing

```
> scores[,1]
[1] Susan John  Bob   Bill  Mary  Paul  Nepo
Levels: Bill Bob John Mary Nepo Paul Susan
```

Data Frame: Column Selection

Example: traditional indexing to retrieve only names and exams

```
> scores[,c(1,4,5,7)]
    Name Exam1 Exam2 Exam3
1  Susan    65    54    34
2   John    47    65    32
3    Bob    62    56    31
4   Bill    61    46    36
5   Mary    58    71    32
6   Paul    63    29    38
7   Nepo    57    48    34
```

Creating/Modifying Data Frames

Basically, like matrices when adding a column

```
> d
  col1 col2  col3
1    1    x  TRUE
2    3    y  TRUE
3    5    z FALSE
> d$col4 = as.factor(c("red", "green","blue"))
> d
  col1 col2  col3  col4
1    1    x  TRUE   red
2    3    y  TRUE green
3    5    z FALSE  blue
```

Creating/Modifying Data Frames

Basically, like matrices when adding a row

```
> d
  col1 col2  col3  col4
1    1    x  TRUE   red
2    3    y  TRUE green
3    5    z FALSE  blue
> d[4,] = list(7,as.factor("y"), TRUE, as.factor("blue")); d
```

#Note: factor levels can not be new

```
  col1 col2  col3  col4
1    1    x  TRUE   red
2    3    y  TRUE green
3    5    z FALSE  blue
4    7    y  TRUE  blue
```

Lists

Like vectors, but allow mixing of types/structures

Actually, lists are generic vectors

Lists

```
> col1
[1] 1 3 5
> m
     [,1] [,2]
[1,]    5    8
[2,]    6    9
[3,]    7   10
> d
  col1 col2  col3  col4
1    1    x  TRUE   red
2    3    y  TRUE green
3    5    z FALSE  blue
4    7    y  TRUE  blue
```

```
> myList = list(e1=42, e2=col1, e3=m, e4=d); myList
$e1
[1] 42

$e2
[1] 1 3 5

$e3
     [,1] [,2]
[1,]    5    8
[2,]    6    9
[3,]    7   10

$e4
  col1 col2  col3  col4
1    1    x  TRUE   red
2    3    y  TRUE green
3    5    z FALSE  blue
4    7    y  TRUE  blue
```

Accessing List elements

Use $, [[]], or [] indexing to access list elements

```
> myList$e1
[1] 42
> myList[["e2"]]
[1] 1 3 5
> myList[3]
$e3
     [,1] [,2]
[1,]    5    8
[2,]    6    9
[3,]    7   10
```

Return Type from List Access

Use class() to determine object type

```
> class(myList$e1)
[1] "numeric"
> class(myList[["e2"]])
[1] "numeric"
> class(myList[3])          # using [] will always return a list
[1] "list"
```

List Element Names

Accessing names in list: use names()

```
> names(myList)
[1] "e1" "e2" "e3" "e4"
```

Changing a name: use names()

```
> names(myList) = c("SL1", "SL2", "SL3", "SL4")
> names(myList)
[1] "SL1" "SL2" "SL3" "SL4"
```

Names in General

Can also add names to other objects. Example:

```
> fib = c(1,1,2,3,5,8,13)
> names(fib)
NULL
> names(fib) = c("f1", "f2", "f3", "f4","f5","f6","f7")
> names(fib)
[1] "f1" "f2" "f3" "f4" "f5" "f6" "f7"
> fib
f1 f2 f3 f4 f5 f6 f7
 1  1  2  3  5  8 13
> fib[c("f3", "f5")]
f3 f5
 2  5
```