Introduction to R

Siny Tsang, M. A.

January 30, 2014
Statlab Workshop

Overview of Workshop

1. What is R?
   - R programming language
   - Advantages & Disadvantages
   - Setting up R & Rstudio
2. Introduction to the R Environment
   - Basic commands
   - Basic operations
   - Getting help
   - R packages
3. Working in R
   - Working with data
   - Simple data analysis & graphing
   - Reading & Writing data

What is R?

`http://cran.r-project.org`

- A statistical package (like SPSS, Stata, SAS, etc.)
- A programming language
- **An environment within which you can do statistical programming & graphics**

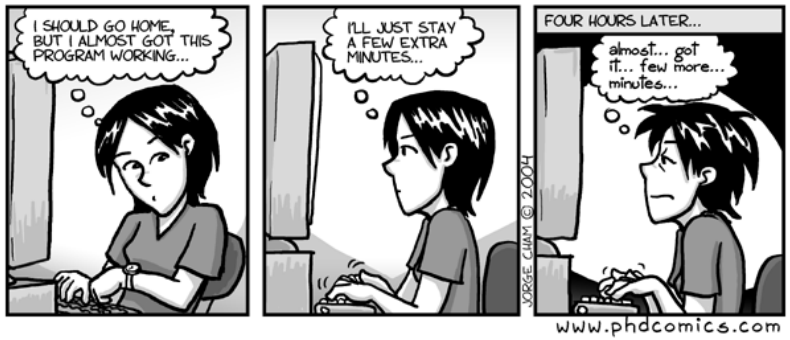
- Object-oriented programming language
  - Everything is an object in R
    - E.g., Statistical models, functions, graphics, output
  - Objects can be manipulated, saved, and reused

Why use R?

- R is **free**
- R runs on all major operating systems
- R is open-source
  - Access and modify source code – completely legal!
- R is highly customizable and extendible
- R is constantly improving and up-to-date
- R can easily produce beautiful publication-quality graphics
- Command-line interface
  - Work with data interactively
  - Replicate previous work easily

But...

- Depends on your needs and preferences
- Learning a new langauge is hard!



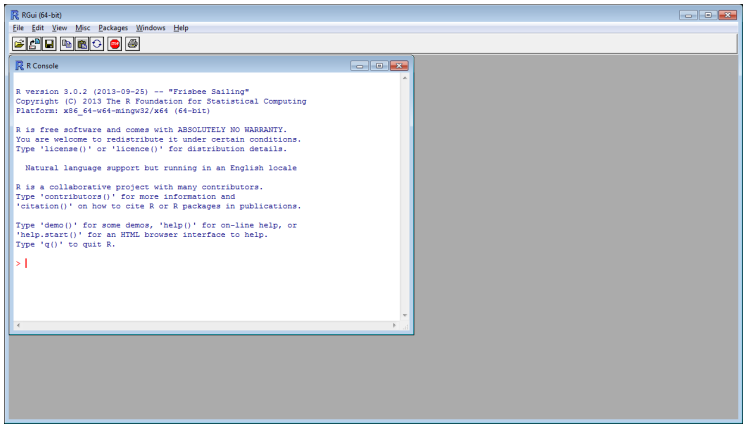
Setting up R and Rstudio

- Download R 3.0.2 for windows or mac
  - Go to the main R website `http://www.r-project.org`
  - Click on the CRAN button on the left
  - Select a mirror near you
  - Select which type of machine you have
- Install R studio `http://www.rstudio.org`
- Double click Rstudio to launch R

The R Environment

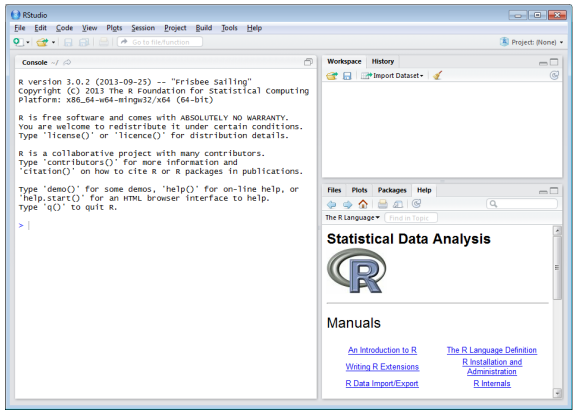
What happens at R startup

Just R



What happens at R startup

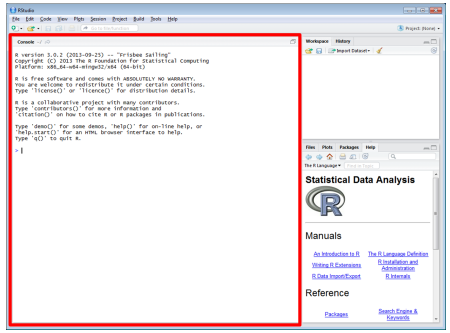
With Rstudio



Rstudio

Console

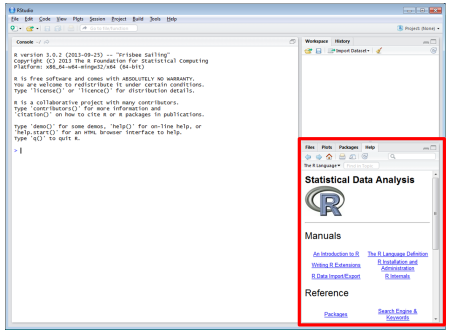
- Run code here
- Type commands after the “>” – the R prompt
- Hit `Return` / `Enter` to run



Rstudio

Output

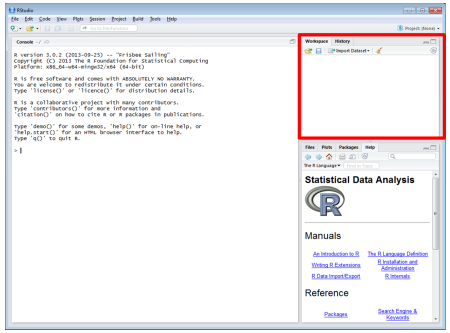
- Plots
- Help pages



Rstudio

Workspace

- Objects in the R Environment



Let's try typing into the console

R as a calulator

Type after the ">"

```
> 15+10
[1] 25
> 12/6
[1] 2
> .7*5
[1] 3.5
```

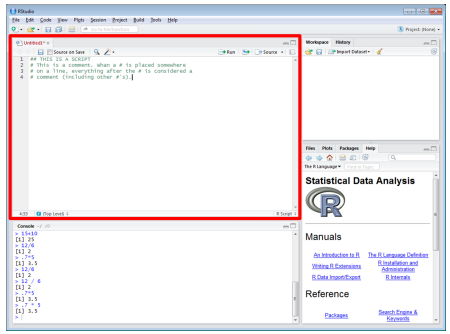

R ignores spaces

```
> 12/6
[1] 2
> 12 / 6
[1] 2
> .7*5
[1] 3.5
> .7 * 5
[1] 3.5
```


Wait, how do I save these codes?

Save codes in scripts

- File | New | R script
- `Ctrl`/`Cmd` + `Shift` + `N`



Shortcuts

Back and forth the Script & Console

- In Script
  - `Ctrl/Cmd` + `Enter`: Send code to console
  - `Ctrl` + `2`: Move cursor to console
- In Console
  - Up arrow: Retrieve previous command
  - `Ctrl` + up arrow: Search commands

Use the Script to...

save work as a .R file

- A text version of the commands we can type after the ">"
- Keep a record of what you are doing
- Look for errors (esp. when you have a large chunk of codes)
- Replicate previous analysis
- Strongly recommend writing R codes in the script!!

Open `.R` file

`IntroR_2014SpringUVAIntroRWorkshop.R`

- Download from:
  - Statlab: `http://static.lib.virginia.edu/statlab/materials/workshops/Intro_to_R.zip`
  - My website: `http://people.virginia.edu/~st2sb/introR.html`
- Follow with the script
  - Point cursor to the line of code
  - `Ctrl/Cmd` + `Enter` to run
- Look at my slides

Comments in R scripts

- **COMMENTS** are very important!
- Comments start with a "**#**" sign

```
> # This is a comment.  When a # is placed somewhere
> # on a line, everything after the # is considered a
> # comment (including other #'s).
>
```

- R ignores white spaces
- White space & comments are great tools to organize your codes

Basic Commands

Basic commands

3 main components

1. **Objects**
   - Used for storing things
   - E.g., variables, datasets, results of analyses
2. **Operations or Operators**
   - `+`, `-`, `*`, `\`
   - Exponents: `^`
3. **Functions**
   - Act on objects
   - E.g., `mean()`, `sd()`, `summary()`, `plot()`, `reshape()`

Objects

assignment: **<-**

- `value1 <- 22` stores **22** in **value1**
- `value2 <- 22/7` stores the result of **22/7** in **value2**
- Type the name to view the stored object

```
> value1
[1] 22
> value2
[1] 3.143
```


Assignment: <-

- Stored objects can be used in subsequent operations

```
> value1/2
[1] 11
```

- Results can be stored in a new object

```
> new.value1 <- value1/2
> new.value1
[1] 11
```

Assignment: <-

Naming conventions

- Cannot begin with a number
  - `3value <- 100` will not work
- Cannot have spaces
  - `value 3 <- 100` will not work
- R is case sensitive

```
> value3 <- 100
> value3

[1] 100

> Value3

Error: object 'Value3' not found
```


Assignment: <-

Overwriting

- Objects of the same name will be overwritten

```
> value4 <- 5^2
> value4

[1] 25

>
> value4 <- sqrt(100)
> value4

[1] 10
```

- Avoid names that are functions
  - e.g., `mean`, `sd`, `t.test`

Concatenation

`c()`

- Assign multiple values to an object, resulting in a vector

```
> my.vector1 <- c(1, 3, 2, 2, 4)
> my.vector1

[1] 1 3 2 2 4

> my.vector2 <- c(5, 6, 8, 1, 4)
> my.vector2

[1] 5 6 8 1 4
```

- Operations are conducted on an element-by-element basis

```
> my.vector1 + 2

[1] 3 5 4 4 6

> my.vector1 + my.vector2

[1]  6  9 10  3  8
```


Concatenation `c()`

Non-numeric data

- Characters are enclosed in " "

```
> my.names <- c("Rob", "Tess", "Megan", "Kevin", "Liam")
> my.names

[1] "Rob"   "Tess"  "Megan" "Kevin" "Liam"
```

Data Types

3 types of data

- Numeric

```
> my.vector1

[1] 1 3 2 2 4
```

- Character

```
> my.names

[1] "Rob"   "Tess"  "Megan" "Kevin" "Liam"
```

- Factor

```
> my.factor <- factor(c(0, 1, 1, 0, 0),
+                     labels = c("Male", "Female"))
> my.factor

[1] Male   Female Female Male   Male
Levels: Male Female
```


Data Types

`str()`

- `num`: a numeric vector
- `chr`: a character vector
- `Factor w/ x levels`: a factor with *x* levels

```
> str(my.vector1)

 num [1:5] 1 3 2 2 4

> str(my.names)

 chr [1:5] "Rob" "Tess" "Megan" "Kevin" ...

> str(my.factor)

 Factor w/ 2 levels "Male","Female": 1 2 2 1 1
```


Data Formats

3 main formats

- Matrix
  - Vectors must be of equal length
  - Can only have one type of data
- Data Frame
  - Vectors must be of equal length
  - Can have different types of data
- List
  - Vectors can be of different lengths
  - Can have different types of data

Matrix

One type of data

- Create matrix with `matrix()`
- Combine vectors `my.vector1`, `my.names`, & `my.factor`

```
> my.matrix <- matrix(c(my.vector1, my.names, my.factor),
+     ncol = 3)  # Matrix with 3 columns
> my.matrix

     [,1] [,2]    [,3]
[1,] "1"  "Rob"   "1"
[2,] "3"  "Tess"  "2"
[3,] "2"  "Megan" "2"
[4,] "2"  "Kevin" "1"
[5,] "4"  "Liam"  "1"
```

Matrix

Forces data to be of one type

- This is a character matrix

```
     [,1] [,2]    [,3]
[1,] "1"  "Rob"   "1"
[2,] "3"  "Tess"  "2"
[3,] "2"  "Megan" "2"
[4,] "2"  "Kevin" "1"
[5,] "4"  "Liam"  "1"
```

```
> str(my.matrix)
```

```
 chr [1:5, 1:3] "1" "3" "2" "2" "4" "Rob" ...
```


Data Frame

Combine different types of data

- Create data frame with `data.frame()`
- Specify column names with =

```
> my.dat <- data.frame(Score = my.vector1,
+                      Names = my.names,
+                      Gender = my.factor)
> my.dat
```

```
  Score Names Gender
1     1   Rob   Male
2     3  Tess Female
3     2 Megan Female
4     2 Kevin   Male
5     4  Liam   Male
```


Data Frame

Can have different types of data

- By default, a character vector is converted to a factor vector

```
> str(my.dat)
```

```
'data.frame': 5 obs. of  3 variables:
 $ Score : num  1 3 2 2 4
 $ Names : Factor w/ 5 levels "Kevin","Liam",..: 4 5 3 1 2
 $ Gender: Factor w/ 2 levels "Male","Female": 1 2 2 1 1
```


Data Frame

Can have different types of data

- Change default with `stringsAsFactors = FALSE`

```
> my.dat1 <- data.frame(Score = my.vector1,
+                       Names = my.names,
+                       Gender = my.factor,
+                       stringsAsFactors = FALSE)
> str(my.dat1)
```

```
'data.frame': 5 obs. of  3 variables:
 $ Score : num  1 3 2 2 4
 $ Names : chr  "Rob" "Tess" "Megan" "Kevin" ...
 $ Gender: Factor w/ 2 levels "Male","Female": 1 2 2 1 1
```

List

Can have different data types & lengths

- Create list with `list()`

```
> my.list <- list(Numeric = my.vector1,
+                 Character = my.names,
+                 Numeric.new = c(1.5, 2.8, 3.3))
> my.list
```

```
$Numeric
[1] 1 3 2 2 4

$Character
[1] "Rob"   "Tess"  "Megan" "Kevin" "Liam"

$Numeric.new
[1] 1.5 2.8 3.3
```

- Vectors are presented in rows

List

Can have different data types & lengths

- Assumes elements are vectors

```
> str(my.list)
```

```
List of 3
 $ Numeric    : num [1:5] 1 3 2 2 4
 $ Character  : chr [1:5] "Rob" "Tess" "Megan" "Kevin" ...
 $ Numeric.new: num [1:3] 1.5 2.8 3.3
```


Basic Operations

Basic Operations

Sequence of numbers

- Generate a sequence of consequtive numbers: `:`

```
> 1:10
```

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

- `seq()` has more flexibility
  - `seq(from = , to = , by = )`

```
> seq(1, 5)
```

```
[1] 1 2 3 4 5
```

```
> seq(1, 5, by = 0.5)
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Basic Operations

Repeat sequence

- Use `rep()` to create repeat number of sequence

```
> # times
> rep("x", 5)
[1] "x" "x" "x" "x" "x"
> # repeat c(1:5) 3 times
> rep(c(1:5), 3)
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> # repeat seq(1,3 by=.5) 2 times
> rep(seq(1, 3, by = 0.5), 2)
 [1] 1.0 1.5 2.0 2.5 3.0 1.0 1.5 2.0 2.5 3.0
```


Basic Operations

Combine repeat sequence with characters

- Use `paste()`

```
> paste("condition", seq(1:5), sep = "")
[1] "condition1" "condition2" "condition3" "condition4"
[5] "condition5"
> paste("condition", c("A", "B", "C"), sep = ".")
[1] "condition.A" "condition.B" "condition.C"
```


Basic Operations

Logical operations

- >, >=, <, <=

```
> x <- rep(c(1:3), 2)
> x
[1] 1 2 3 1 2 3
> x >= 3
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE
```

- Equal: ==, Not equal: !=,

```
> x == 1
[1]  TRUE FALSE FALSE  TRUE FALSE FALSE
> x != 1
[1] FALSE  TRUE  TRUE FALSE  TRUE  TRUE
```


Basic Operations

Logical operations

- And: &, Or: |

```
> x
[1] 1 2 3 1 2 3
> x == 1 & x == 2
[1] FALSE FALSE FALSE FALSE FALSE FALSE
> x == 1 | x == 2
[1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE
```

Basic Functions

in R

- List all the objects in workspace: `ls()`
- Remove objects from workspace: `rm()`

```
> ls()

 [1] "my.dat"      "my.dat1"     "my.factor"   "my.list"
 [5] "my.matrix"   "my.names"    "my.vector1"  "my.vector2"
 [9] "new.value1"  "value1"      "value2"      "value3"
[13] "value4"      "x"

> rm(my.dat)
> ls()

 [1] "my.dat1"     "my.factor"   "my.list"     "my.matrix"
 [5] "my.names"    "my.vector1"  "my.vector2"  "new.value1"
 [9] "value1"      "value2"      "value3"      "value4"
[13] "x"
```


Basic Functions

in R

- Remove (almost) everything from workspace: `rm(list=ls())`
  - Except objects that begin with "."
- Terminate a R session, type `quit()` or `q()`

Basic Functions

in `base` package

```
> mean(x)   # mean

[1] 2

> sd(x)   # standard deviation

[1] 0.8944

> summary(x)   # Basic descriptive statistics

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.25    2.00    2.00    2.75    3.00

> min(x)   # Minimum value

[1] 1
```


Getting Help

Getting help
in R

- Function name known
  - `help()`
  - `?`
  - `help(matrix)` or `?matrix`
- Function name unknown
  - `help.search("")`
  - `??`
  - `help.search("regression")` or `??anova`

Getting help
online

- CRAN website: `http://cran.r-project.org/`
- Quick R: `http://www.statmethods.net/`
- R Tutor: `http://www.r-tutor.com/`
- UCLA: `http://www.ats.ucla.edu/stat/r/`
- Google (*Tip: use [R] as a filter*)

R Packages

Install Packages

- Think of R as your Operating System (e.g., Windows, Mac)
- Packages are the "applications" (e.g., MS word, Firefox)
- We need to click on the "applications" to use them
- Initial R download includes `base`, `graphics`, `stats`, `utils`
- Install add-on packages: `install.packages("packagename")`
- Load an installed package: `library(packagename)` or `require(packagename)`

Install package: `car`

- `install.packages("car")`
- Choose the CRAN mirror close to your location to speed up the download process
  - e.g., `USA(PA1)` is a good mirror for UVA
- You do not need to install the package again, until you update your R version
- Load the package to use it: `library(car)` or `require(car)`
  - No need to re-load package within the current R session
  - Need to load the package again after terminating a R session

Installed packages

- View all installed packages `library()`
  - Will show up in a separate tab
- View currently loaded packages `search()`
  - *Note: Yours will look different because I have more packages installed*

```
> search()

 [1] ".GlobalEnv"        "package:foreign"
 [3] "package:psych"     "package:car"
 [5] "package:knitr"     "package:stats"
 [7] "package:graphics"  "package:grDevices"
 [9] "package:utils"     "package:datasets"
[11] "package:methods"   "Autoloads"
[13] "package:base"
```


Working with Data

Working with data frame

`Vocab` data

- Respondents to U.S. General Social Surveys, 1972-2004
- Load the `Vocab` data from the `car` package

```
> data(Vocab)
```

- What are the dimensions (rows x columns) of the data frame?

```
> dim(Vocab)

[1] 21638     4
```

- What are the variables?

```
> names(Vocab)

[1] "year"       "sex"        "education"  "vocabulary"
```

Beginning / end of data

```
> head(Vocab)
```

```
         year    sex education vocabulary
20040001 2004 Female         9          3
20040002 2004 Female        14          6
20040003 2004   Male        14          9
20040005 2004 Female        17          8
20040008 2004   Male        14          1
20040010 2004   Male        14          7
```

```
> tail(Vocab)
```

```
         year    sex education vocabulary
19982818 1998   Male        14          4
19982819 1998 Female        12          4
19982821 1998   Male        12          5
19982822 1998   Male        12          5
19982825 1998   Male        13          8
19982828 1998 Female        14          6
```


Index (one dimension)

Use square brackets: `[]`

- Subset a range of elements

```
> x
```

```
[1] 1 2 3 1 2 3
```

```
> # Select the first 3 elements
> x[1:3]
```

```
[1] 1 2 3
```

```
> # Select the 4th and 5th elements
> x[c(4, 5)]
```

```
[1] 1 2
```

```
> # Unselect the 4th and 5th elements
> x[-c(4, 5)]
```

```
[1] 1 2 3 3
```


Index (more than one dimension)

Use square brackets: `[row, column]`

- Use the `Vocab` data

```
         year    sex education vocabulary
20040001 2004 Female         9          3
20040002 2004 Female        14          6
20040003 2004   Male        14          9
20040005 2004 Female        17          8
20040008 2004   Male        14          1
20040010 2004   Male        14          7
```

```
> # Select the element in row 1, column 3
> Vocab[1, 3]
```

```
[1] 9
```

```
> # Select elements in rows 2 & 3, columns 2, 3, 4
> Vocab[2:3, 2:4]
```

```
            sex education vocabulary
20040002 Female        14          6
20040003   Male        14          9
```


Subset

- Subset the first 10 rows of the data

```
> vocab.10 <- Vocab[1:10, ]
> vocab.10
```

```
         year    sex education vocabulary
20040001 2004 Female         9          3
20040002 2004 Female        14          6
20040003 2004   Male        14          9
20040005 2004 Female        17          8
20040008 2004   Male        14          1
20040010 2004   Male        14          7
20040012 2004 Female        12          6
20040013 2004   Male        10          6
20040016 2004   Male        11          5
20040017 2004 Female         9          1
```

Subset

by variable values

- Subset data where "sex == Female"

```
> vocab.10.female <- vocab.10[which(vocab.10$sex ==
+     "Female"), ]
> vocab.10.female
```

```
         year    sex education vocabulary
20040001 2004 Female         9          3
20040002 2004 Female        14          6
20040005 2004 Female        17          8
20040012 2004 Female        12          6
20040017 2004 Female         9          1
```


Modify data frame

Change variable names

- Show column names

```
> colnames(vocab.10)  # or
```

```
[1] "year"       "sex"        "education"  "vocabulary"
```

```
> names(vocab.10)
```

```
[1] "year"       "sex"        "education"  "vocabulary"
```

- Assign new variable names

```
> names(vocab.10) <- c("time", "gender", "edu", "vocab")
> names(vocab.10)
```

```
[1] "time"   "gender" "edu"    "vocab"
```

- The same applies for row names

Modify data frame

Print columns

- By column position: `dataname[ ,3]`
- By variable name: `dataname$variablename`

```
> vocab.10[, 3]
```

```
 [1]  9 14 14 17 14 14 12 10 11  9
```

```
> vocab.10$edu
```

```
 [1]  9 14 14 17 14 14 12 10 11  9
```


Modify data frame

Sort data: `order()`

- Original data

```
         time gender edu vocab
20040001 2004 Female   9     3
20040002 2004 Female  14     6
20040003 2004   Male  14     9
20040005 2004 Female  17     8
20040008 2004   Male  14     1
```

- Sort by education

```
> vocab.10.edu <- vocab.10[order(vocab.10$edu), ]
> head(vocab.10.edu, 5)
```

```
         time gender edu vocab
20040001 2004 Female   9     3
20040017 2004 Female   9     1
20040013 2004   Male  10     6
20040016 2004   Male  11     5
20040012 2004 Female  12     6
```

Modify data frame

Sort data: `order()`

- Sort by education, decreasing: `decreasing = TRUE`

```
> vocab.10.edu <- vocab.10[order(vocab.10$edu,
+                                decreasing = TRUE), ]
> head(vocab.10.edu, 5)
```

```
         time gender edu vocab
20040005 2004 Female  17     8
20040002 2004 Female  14     6
20040003 2004   Male  14     9
20040008 2004   Male  14     1
20040010 2004   Male  14     7
```


Modify data frame

Create new variable

- New variable: vocab x 10
- Assign name “vocab10” in dataframe “vocab.10.edu”

```
> vocab.10.edu$vocab10 <- vocab.10.edu$vocab * 10
>
> head(vocab.10.edu, 5)
```

```
         time gender edu vocab vocab10
20040005 2004 Female  17     8      80
20040002 2004 Female  14     6      60
20040003 2004   Male  14     9      90
20040008 2004   Male  14     1      10
20040010 2004   Male  14     7      70
```


Simple Data Analysis & Graphing

Descriptive statistics

- `mean()`, `sd()`

```
> mean(Vocab$education)
[1] 12.8
> sd(Vocab$education)
[1] 3.041
```

- Use `with()` so we don’t have to keep typing data name

```
> with(Vocab, c(mean(education), sd(education)))
[1] 12.795  3.041
```

Summary statistics

from other packages

- Install the `psych` package

```
> install.packages("psych")
```

- Load the `psych` package

```
> library(psych)   # or
> require(psych)
```


Summary statistics

describe() in the psych package

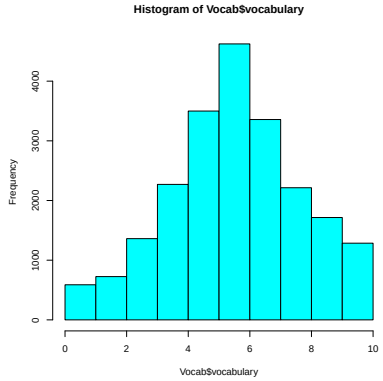
```
> describe(Vocab)
```

```
           var     n    mean   sd median trimmed   mad  min
year         1 21638 1988.85 8.65   1989 1988.90 10.38 1974
sex*         2 21638    1.43 0.50      1    1.41  0.00    1
education    3 21638   12.80 3.04     12   12.84  2.97    0
vocabulary   4 21638    6.00 2.17      6    6.04  1.48    0
             max range  skew kurtosis   se
year        2004    30 -0.11    -0.97 0.06
sex*           2     1  0.28    -1.92 0.00
education     20    20 -0.22     0.86 0.02
vocabulary    10    10 -0.20    -0.19 0.01
```


Plot histogram

hist()

```
> hist(Vocab$vocabulary, breaks = 10, col = "cyan")
```



Summary statistics by group

describeBy()

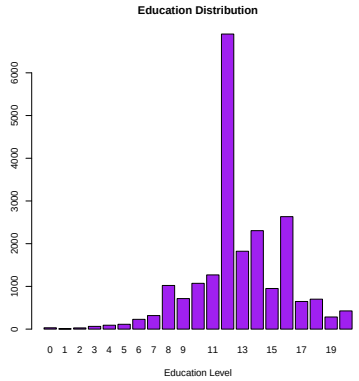
```
> vocab.summary <- describeBy(Vocab, group = Vocab$sex,
+     mat = TRUE)
> vocab.summary[1:4, ]  # Show only the first 2 variables
```

```
      item group1 var     n mean    sd median trimmed   mad
year1    1 Female   1 12312 1989 8.580   1989    1989 10.38
year2    2   Male   1  9326 1989 8.748   1989    1989 10.38
sex*1    3 Female   2 12312    1 0.000      1       1  0.00
sex*2    4   Male   2  9326    2 0.000      2       2  0.00
       min  max range    skew kurtosis      se
year1 1974 2004    30 -0.1123  -0.9551 0.07732
year2 1974 2004    30 -0.1116  -0.9897 0.09058
sex*1    1    1     0     NaN      NaN 0.00000
sex*2    2    2     0     NaN      NaN 0.00000
```

Plot bar chart

`barplot()`

```
> barplot(table(Vocab$education),
+     main = "Education Distribution",
+     xlab = "Education Level", col = "purple")
```



Frequency table

`table(row, column)`

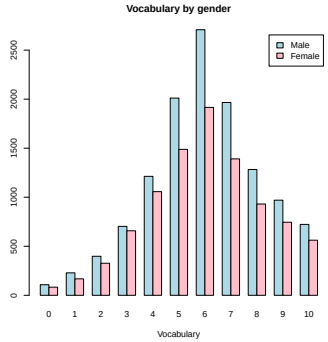
```
> my.table <- with(Vocab, table(sex, year))
> my.table
```

```
        year
sex      1974 1976 1978 1982 1984 1987 1988 1989 1990 1991
  Female  774  791  861  999  828  955  513  559  481  565
  Male    672  643  623  724  574  724  407  409  371  396
        year
sex      1993 1994 1996 1998 2000 2004
  Female  560 1085 1050  760  730  801
  Male    453  755  816  541  581  637
```


Plot bar chart, by group

`barplot()`

```
> barplot(table(Vocab$sex, Vocab$vocabulary),
+     main = "Vocabulary by gender", xlab = "Vocabulary",
+     col = c("lightblue", "pink"), legend = c("Male",
+         "Female"), beside = TRUE)
```



2 group t-test

`t.test()`

- `sat.act` data: 3 measures of ability

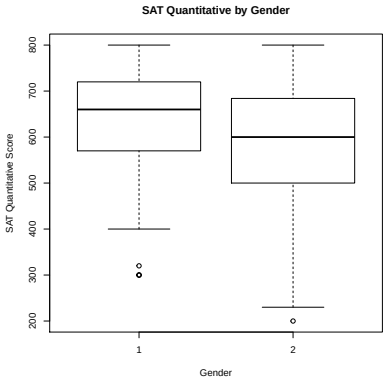
```
> data(sat.act)
> with(sat.act, t.test(SATQ ~ gender))
```

```
Welch Two Sample t-test

data:  SATQ by gender
t = 4.354, df = 492.9, p-value = 1.624e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 21.88 57.87
sample estimates:
mean in group 1 mean in group 2
          635.9           596.0
```

Box plot

`box.plot()`

```
> boxplot(SATQ ~ gender, data = sat.act,
+      main = "SAT Quantitative by Gender",
+      xlab = "Gender", ylab = "SAT Quantitative Score")
```



Correlation

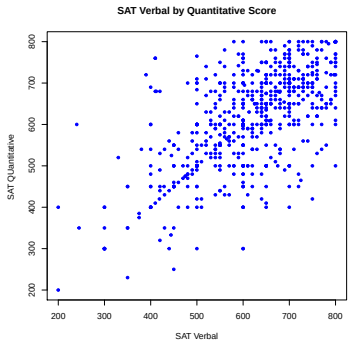
`cor.test()`

```
> cor.test(x = sat.act$SATQ, y = sat.act$SATV,
+          alternative="two.sided", method="pearson"
+          )


Pearson's product-moment correlation

data:  sat.act$SATQ and sat.act$SATV
t = 22.05, df = 685, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.5983 0.6860
sample estimates:
   cor
0.6443
```


Scatterplot

`plot()`

```
> with(sat.act, plot(SATV, SATQ,
+      main = "SAT Verbal by Quantitative Score",
+      xlab = "SAT Verbal", ylab = "SAT QUantitative ",
+      pch = 20, col = "blue"))
```



Linear Regression

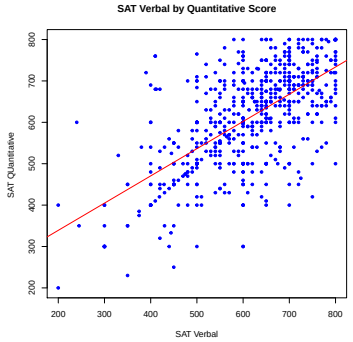
`lm()`

```
> lm.1 <- lm(SATQ ~ SATV, data = sat.act)
> summary(lm.1)


Call:
lm(formula = SATQ ~ SATV, data = sat.act)

Residuals:
   Min     1Q Median     3Q    Max
-302.1  -46.5    2.4   51.3  282.9

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 207.5253    18.5725    11.2   <2e-16 ***
SATV          0.6576     0.0298    22.1   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 88.5 on 685 degrees of freedom
  (13 observations deleted due to missingness)
Multiple R-squared:  0.415, Adjusted R-squared:  0.414
F-statistic:  486 on 1 and 685 DF,  p-value: <2e-16
```

Add regression line to scatterplot

`abline()`

```
> with(sat.act, plot(SATV, SATQ, main = "SAT Verbal by Quantitative Score",
+     xlab = "SAT Verbal", ylab = "SAT QUantitative ",
+     pch = 20, col = "blue"))
> abline(lm(SATQ ~ SATV, data = sat.act), col = "red")
```



Read & Write/Save Data

Read data

`read.table()`, `read.csv()`

- `read.table("filename", header=TRUE, sep="")`
- Works for `.txt`, `.csv`, `.dat` files
- Set `header = TRUE` if variable names are in the first row
- Specify separator with `sep = " "` (e.g., `sep = ";"`)

- `read.csv("filename", header=TRUE)`
- Works for Comma Separated (`.csv`) files

Read data

`read.table()`

```
 > Dosage <- read.table(
"https://personality-project.org/r/datasets/R.appendix2.text",
header = TRUE)

> head(Dosage)

  Observation Gender Dosage Alertness
1           1      m      a         8
2           2      m      a        12
3           3      m      a        13
4           4      m      a        12
5           5      m      b         6
6           6      m      b         7
```

Other data formats

- `foreign` package
  - SPSS: `read.spss`
  - Stata: `read.dta`
  - SAS: `read.ssd`
- View functions in the package

```
> library(help = foreign)
```


Set working directory

An example

- I have a data file: `myawesomedata.txt`
- The data is stored in "`C:/Users/My Name/Documents/R Stuff/`"
- To read the data, I'll have to type:

```
> mydata <- read.table("C:/Users/My Name/Documents/R Stuff/myawseomedata.txt",
header=TRUE)
```

- Easy to make mistakes!

Set working directory

through the menu

- Windows:
  - `Session | Choose Directory...` or `Ctrl + Shift + k`
  - Select the appropriate folder/directory
- Mac:
  - `Tools | Change working directory` or `Ctrl + Shift + k`
  - Select the appropriate folder/directory

Set working directory

`setwd()`

- Windows:
  - Must use forward slash: `/`, or double backslash: `\\`
  - `setwd("C:/Users/My Name/Documents/R Stuff/")`
  - This works too: `setwd("C:\\Users\\My Name \\Documents\\R Stuff")`
- Mac:
  - `setwd("/Users/My Name/Documents/R Stuff/")`

Get working directory

`getwd()`

- Check if working directory is correctly set: `getwd()`
- Should return the path you previously set
- Now I can read my data just by typing:

```
> mydata <- read.table("myawesomedata.txt",
header = TRUE)
```

- Always set working directory when you start a new R session
- Read data, saved files from specific location
- Write data, save files to specific location

Write data

`write.table(), write.csv()`

- `write.table()`
  - Writes row names by default
  - Puts quotes around character values by default

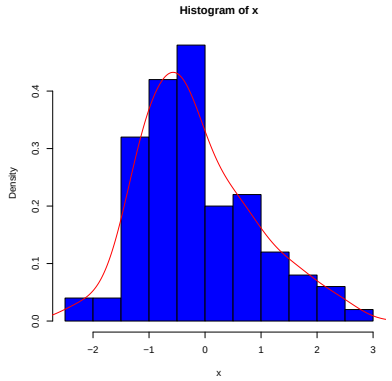
```
> write.table(Dosage, "NewDosage.txt")
> # Removes row names and quotes
> write.table(Dosage, "NewDosage_new.txt")
```

- `write.csv()`
  - No quotes by default

```
> write.csv(Dosage, "NewDosage.csv")
```


Save graphs

```
> # Generate some data
> x <- rnorm(100)
> hist(x, probability = TRUE, col = "blue")
> lines(density(x), col = "red")
```



Save graphs

- Output `.pdf` graph: `pdf(graphnameinquotes.pdf)`
- Turn off graphic output: `dev.off()`

```
> pdf("randomhistogram.pdf")
> hist(x, probability = TRUE, col = "blue")
> lines(density(x), col = "red")
> dev.off()  # Turn off graphic output
```

Save graphs

other formats

Function	Output
pdf("mygraph.pdf")	pdf file
png("mygraph.png")	png file
jpeg("mygraph.jpg")	jpeg file
bmp("mygraph.bmp")	bmp file
postscript("mygraph.ps")	postscript file
win.metafile("mygraph.wmf")	windows metafile

THANK YOU!

Introduction to R

Siny Tsang, M. A.
st2sb@virginia.edu

2014 Spring Statlab Workshop