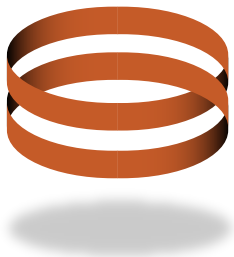
Introduction to R

Alex Storer

Harvard MIT Data Center

January 16, 2013



The Institute
for Quantitative Social Science
at Harvard University

Outline

Topic

Introductions

- Who am I?
- IQSS Research Consulting Team
- IQSS Services
  - OpenScholar
  - Dataverse Network
  - Workshops and Trainings
- Slide Notes
  - Thanks to Ista Zahn

Materials and setup

- Find class materials at

http://projects.iq.harvard.edu/rtc/event/introduction-r-computefest-2013

- **Download the zip file at the bottom of the page and unzip on your desktop!**

Workshop description

This is an INTRODUCTION to R. We assume no/very little knowledge of R!
Learning objectives:

- Install and load R packages
- Find package and function documentation
- Load external data into R
- Create and query vectors, matrices, and data frames
- Compute descriptive statistics and regression models
- Construct basic graphical displays

Organization

- Please feel free to ask questions at any point if they are relevant to the current topic (or if you are lost!)
- There will be exercises to check for comprehension and get practice
- Collaboration with your neighbors is encouraged

What to expect from R

When talking about user friendliness of computer software I like the analogy of cars vs. busses: [. . .] Using this analogy programs like SPSS are busses, easy to use for the standard things, but very frustrating if you want to do something that is not already preprogrammed. R is a 4-wheel drive SUV (though environmentally friendly) with a bike on the back, a kayak on top, good walking and running shoes in the passenger seat, and mountain climbing and spelunking gear in the back. R can take you anywhere you want to go if you take time to learn how to use the equipment, but that is going to take longer than learning where the bus stops are in SPSS. – Greg Snow R-help (May 2006)

Why R?

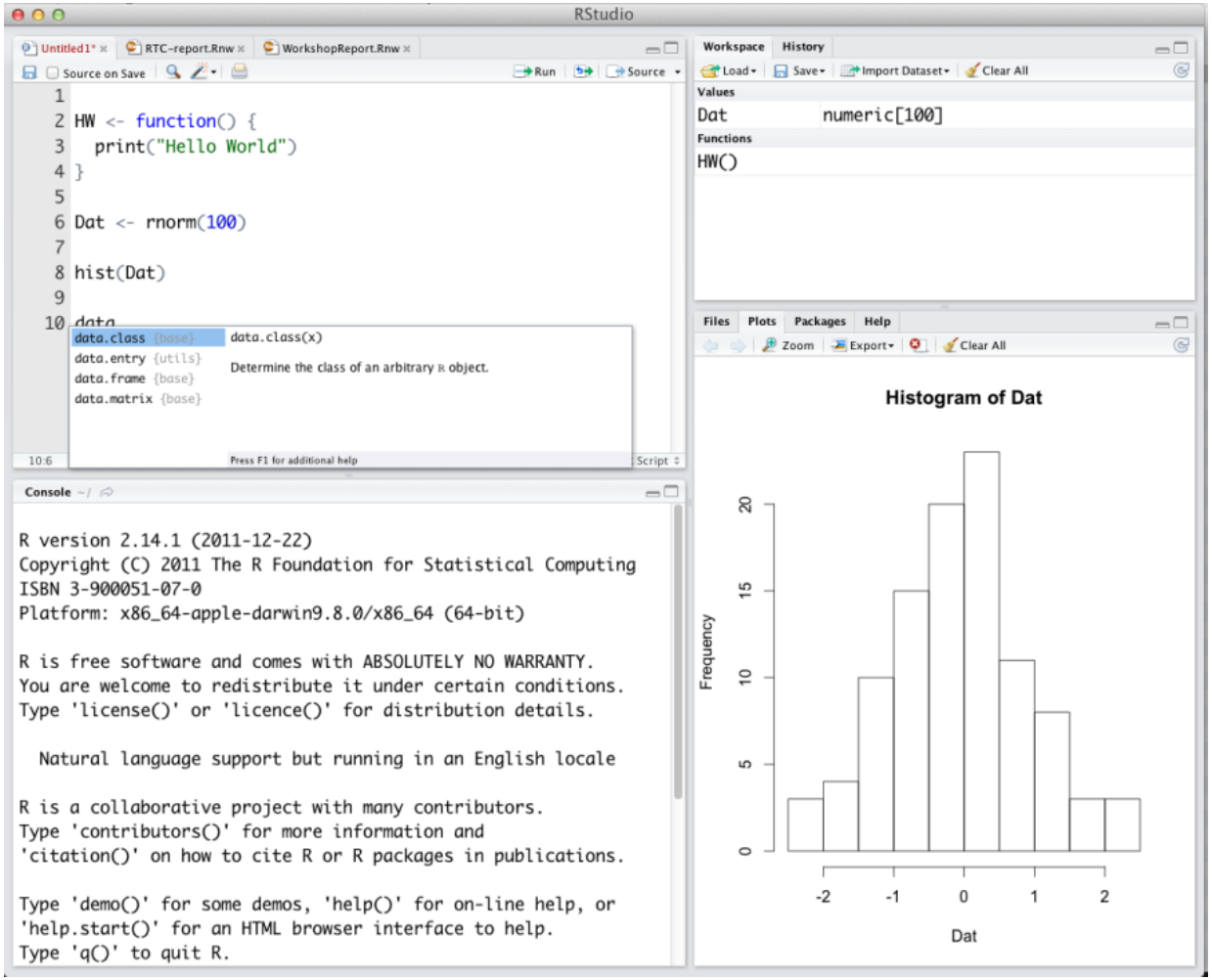
- Vast capabilities, wide range of statistical and graphical techniques
- Written primarily by statisticians
- FREE as in free beer: no cost
- FREE as in free speech: collaborative development
- Excellent community support: mailing list, blogs, tutorials
- Easy to extend by writing new functions

Coming to R

- Coming from Matlab?
  `http://www.math.umaine.edu/~hiebeler/comp/matlabR.pdf`
- Coming from SciPy?
  `http://mathesaurus.sourceforge.net/matlab-python-xref.pdf`
- Coming from SAS/SPSS?
  `http://www.et.bs.ehu.es/~etptupaf/pub/R/RforSAS&SPSSusers.pdf`
- Coming from Stata? `http://dss.princeton.edu/training/RStata.pdf`

Topic

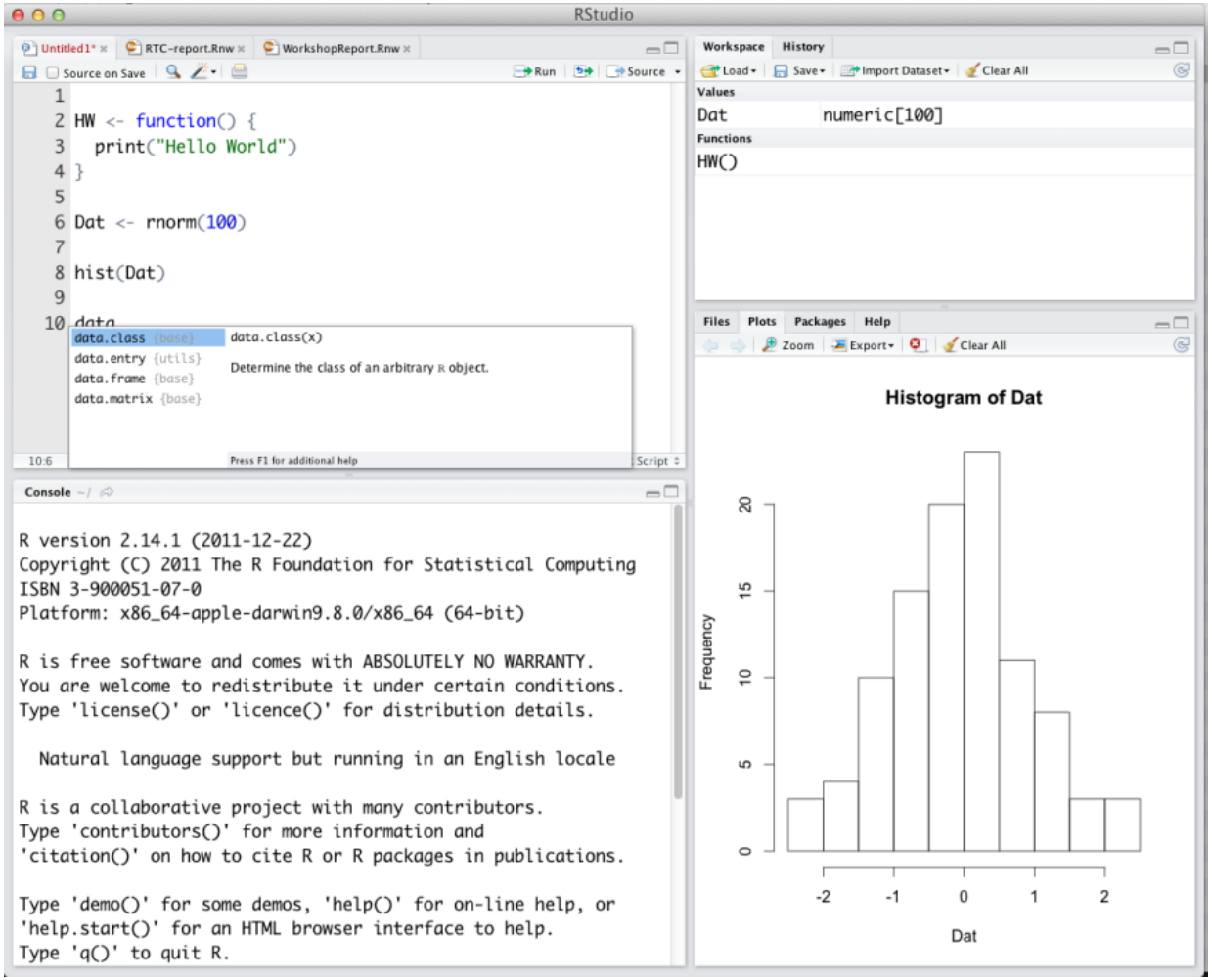
R Studio



Rstudio has many useful features, including parentheses matching and auto-completion

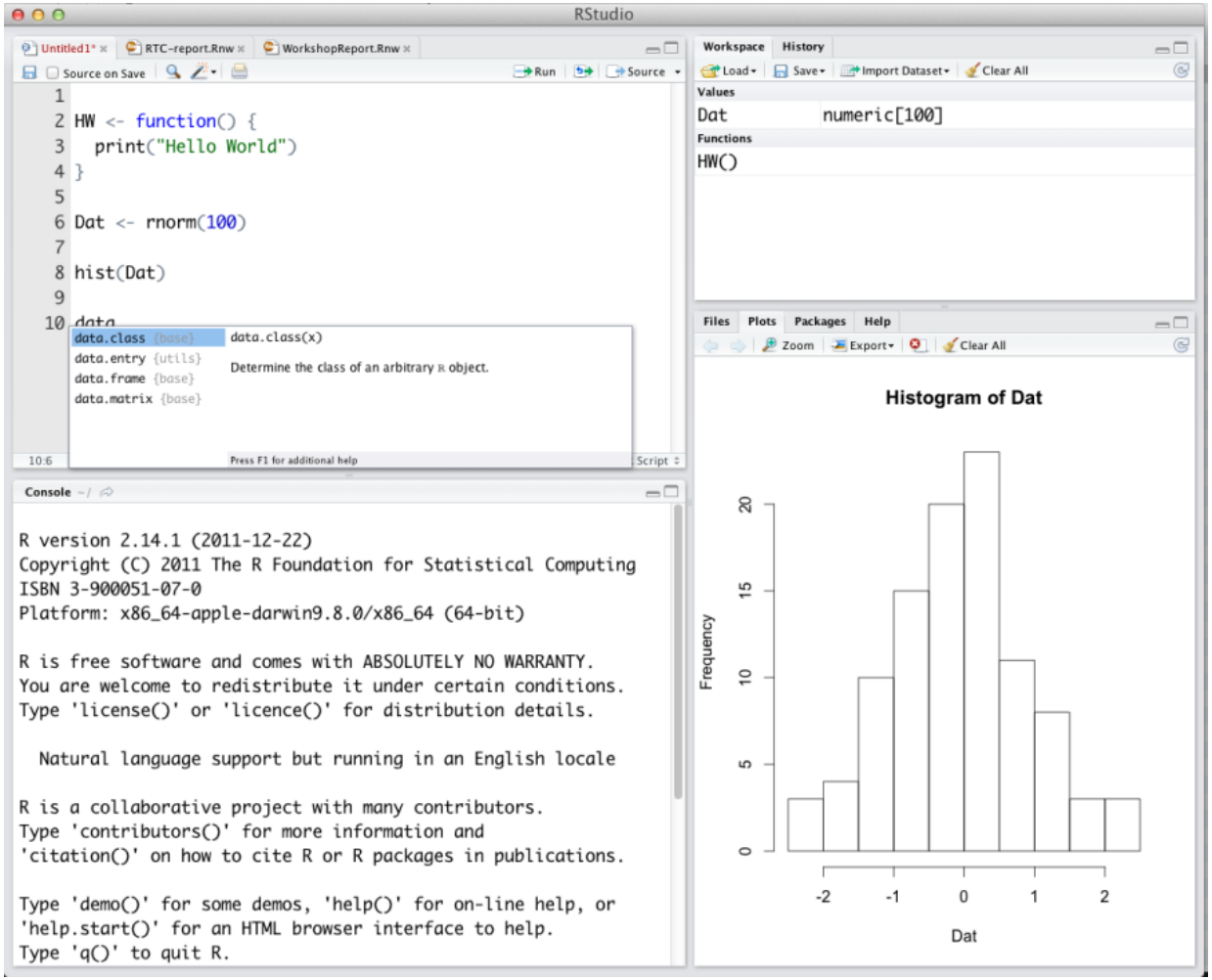
If you use another editor extensively (e.g., Emacs, Eclipse) you can find R extensions for it - otherwise, it's probably best to use R Studio

R Studio - Console



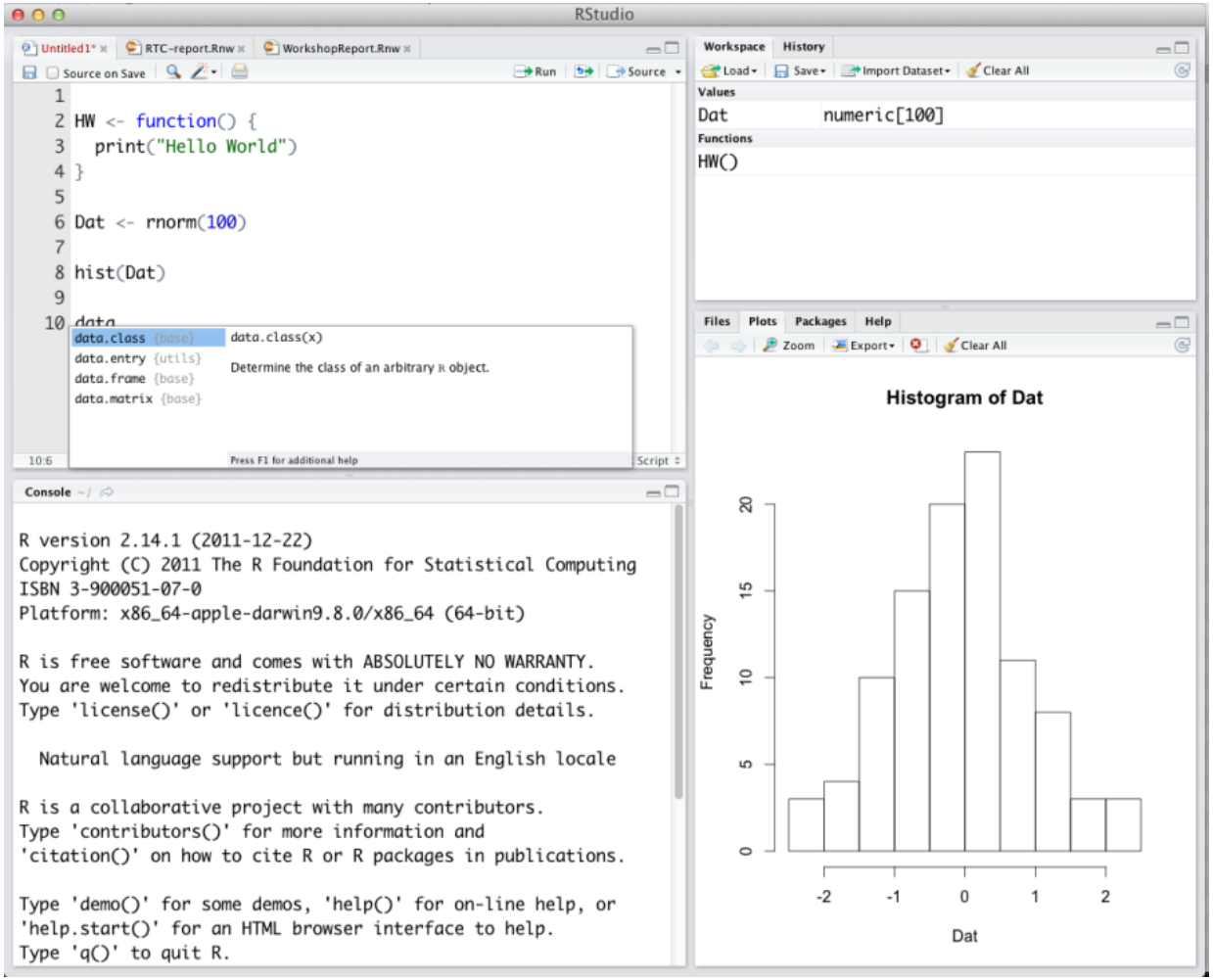
- The R console
  - Displays command history and results
  - Commands can be typed directly in the console
  - R Console work disappears once session is closed

R Studio - Editor



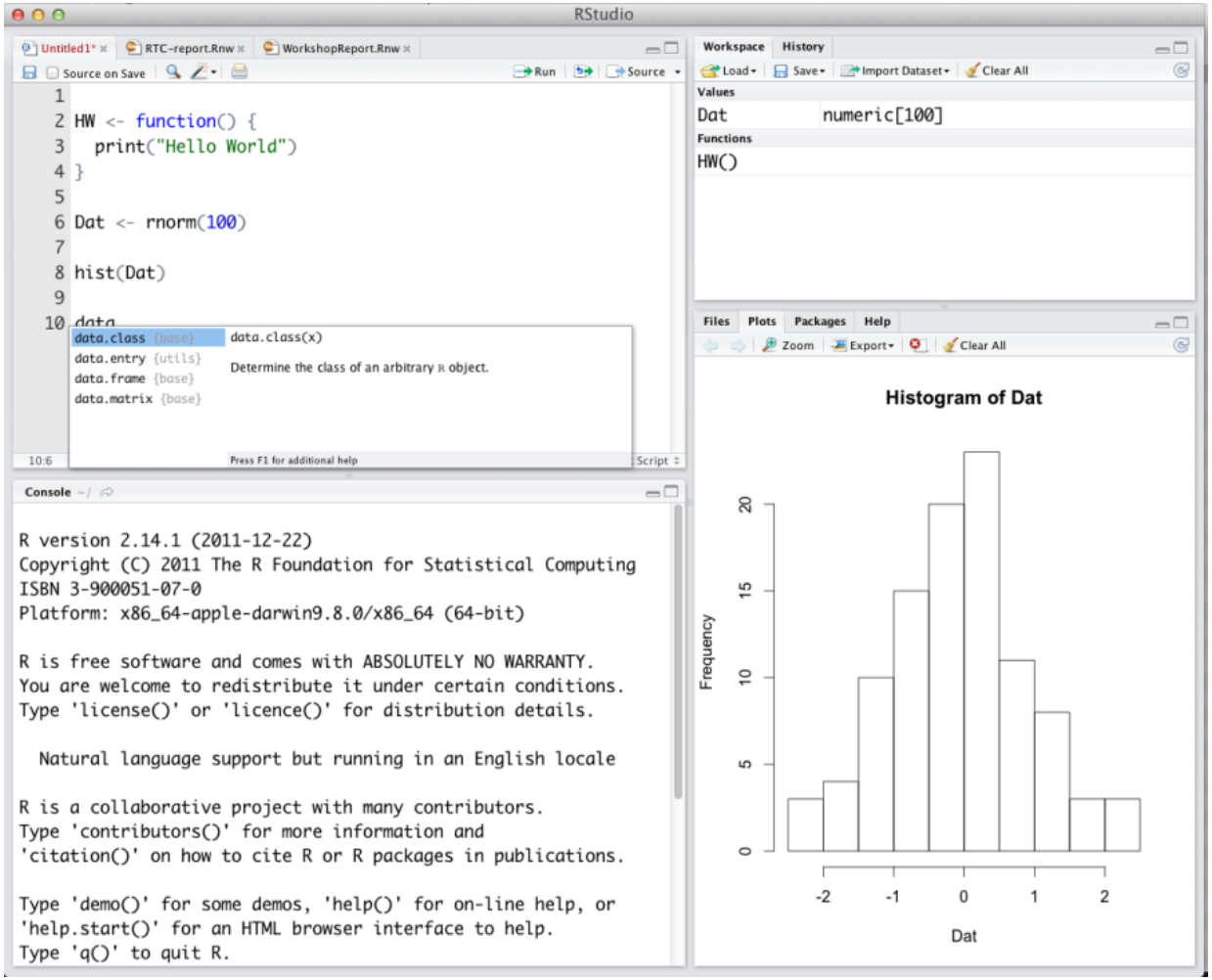
- A text editor
  - A plain text editor for writing R code
  - Syntax highlighting, parentheses matching etc.
  - **Anything that modifies your data should be done in a text editor**

R Studio - Graphics



- Graphics windows
  - View, re-size, and save graphics
  - Cycle through graph history

R Studio - Workspace



- Work-space viewer
  - Allow you to see stored objects
  - Very helpful if you are absentminded like me and frequently forget what names you gave your data!

Topic

Getting help in R

- Start html help, search/browse using web browser
  - at the R console:

    ```
    help.start()
    ```

  - or use the help menu from R Studio
- Look up the documentation for a function

  ```
  help(topicName)
  ```

  ```
  ?topicName
  ```

- Look up documentation for a package

  ```
  help(package="packageName")
  ```

- When in doubt, just google it!

Reading R documentation

Beginners sometimes find the docs intimidating. It is much easier once you know the structure:

Description	What does this function do?
Usage	Generic example showing default arguments
Arguments	What are the inputs to this function?
Value	What is the output (or result) of this function?
Details	Additional details to help you understand how it works
References	Citations for, e.g., the algorithms used in the function
See Also	Other related functions
Examples	Examples of the function in action; type example(functionName) to run them

In many cases just looking at the usage section is enough

R packages and libraries

There are thousands of R packages that extend R's capabilities. To get started, check out http://cran.r-project.org/web/views/.

- To view available packages:

  ```
  library()
  ```

- To see what packages are loaded:

  ```
  search()
  ```

- To load a package:

  ```
  library("packageName")
  ```

- Install new package:

  ```
  install.packages("packageName")
  ```

Things to keep in mind

- Case sensitive
- Comments can be put almost anywhere, starting with a hash mark ('#'); everything to the end of the line is a comment
- The command prompt ">" indicates that R is ready to receive commands
- If a command is not complete at the end of a line, R will give a different prompt, '+' by default
- Parentheses must always match (first thing to check if you get an error)
- R Does not care about spaces between commands or arguments
- Names should start with a letter and should not contain spaces
- Can use "." in object names (e.g., "my.data")
- Use / instead of \ in path names
- R counts from 1, not from 0

Exercise 0

1. Find the R console prompt and type '2+2 <enter>'
2. Look up the help page for the "mean" topic
3. Use google and search for "R linear model". Click on the first link.
4. Go to http://cran.r-project.org/web/views/ and skim the topic closest to your field/interests

Topic

Assignment

Values can be assigned names and used in subsequent operations

- The <- operator (less than followed by a dash) is used to save values
- The name on the left gets the value of the value on the right.

```
> x <- 11 # Assign the value 10 to a variable named x
> x + 1 # Add 1 to x
[1] 12
> y <- x + 1 # Assign y the value x + 1
> y
[1] 12
```

Saved variables can be listed, overwritten and deleted

```
> ls() # List variables in workspace
[1] "x" "y"
> x # Print the value of x
[1] 11
> x <- 100 # Overwrite x. Note that no warning is given!
> x
[1] 100
> rm(x) # Delete x
> ls()
[1] "y"
> y # y has the same value
[1] 12
```

Functions

Using R is mostly about applying **functions** to **variables**. Functions

- take **variable** as input **arguments**
- perform operations
- **return** values which can be **assigned**

The general form for calling R functions is

```
FunctionName(arg.1, arg.2, ... arg.n)
```

Examples

```
> a <- sqrt(y) # Call the sqrt function with argument x=y
> round(a, digits = 2) # Call round() with arguments x=x and digits=2
[1] 3.46
> # Functions can be nested so an alternative is
> round(sqrt(y), digits = 5) # Take sqrt of a and round
[1] 3.4641
```

Vectors and class

Values can be combined into vectors using the c() function

```
> num.var <- c(1, 2, 3, 4) # numeric vector
> char.var <- c("1", "2", "3", "4") # character vector
> log.var <- c(TRUE, TRUE, FALSE, TRUE) # logical vector
> char.var2 <- c(num.var, char.var) # numbers coverted to character
>
```

Vectors have a *class* which determines how functions treat them

```
> class(num.var)
[1] "numeric"
> mean(num.var) # take the mean of a numeric vector
[1] 2.5
> class(char.var)
[1] "character"
> mean(char.var) # cannot average characters
[1] NA
> class(char.var2)
[1] "character"
```

Vector conversion and info

Vectors can be converted from one class to another

```
> class(char.var)
[1] "character"
> num.var2 <- as.numeric(char.var) # convert to numeric
> class(num.var)
[1] "numeric"
> mean(as.numeric(char.var)) # now we can calculate the mean
[1] 2.5
> as.numeric(c("a", "b", "c")) # cannot convert letters to numeric
[1] NA NA NA
```

In addition to class, you can examine the `length()` and `str()` ucture of vectors

```
> ls() # list objects in our workspace
[1] "a"        "char.var"  "char.var2" "log.var"   "num.var"
[6] "num.var2"  "y"
> length(char.var) # how many elements in char.var?
[1] 4
> str(num.var2) # what is the structure of num.var2?
 num [1:4] 1 2 3 4
```

Factor vectors

Factors are stored as numbers, but have character labels. Factors are useful for

- Modeling
- Sorting/presenting values in arbitrary order

Examples:

- Gender
- Low/Medium/High

```
> lmh.var <- c("l", "m", "h")
> sort(lmh.var) # not in order!
[1] "h" "l" "m"
> lmh.factor.var <- factor(lmh.var,
+                          levels=c("l", "m", "h"),
+                          labels=c("low", "medium", "high"),
+                          ordered=TRUE)
> sort(lmh.factor.var) # sorted correctly
[1] low    medium high
Levels: low < medium < high
```

Date/time vectors

Date/time classes are a little complicated, but briefly:

```
> date.char.var <- c("June-22-2012", "July-22-2012")
> sort(date.char.var) # sorted as character
[1] "July-22-2012" "June-22-2012"
> date.var <- as.Date(date.char.var, #convert to date
+                     format="%B-%d-%Y")
> class(date.var)
[1] "Date"
> sort(date.var) # sorted as date
[1] "2012-06-22" "2012-07-22"
> date.var + 10 # you can add and subtract dates
[1] "2012-07-02" "2012-08-01"
```

See `?strptime` for more details

Exercise 1: vectors and classes

1. Create a new vector called "test" containing five numbers of your choice [`c()`, `<-`]
2. Create a second vector called "students" containing five common names of your choice [`c()`, `<-`]
3. Determine the class of "students" and "test" [`class()` or `str()`]
4. Convert "test" to character class, and confirm that you were successful [`as.numeric()`, `<-`, `str()`]
5. Remove the objects "students" and "test" and check that they were properly removed [`rm()`, `ls()`]
6. Create a factor with levels "high school", "college", "university" (in that order), and check that you were successful [`factor()`, `c()`, `str()`]
7. Using the help for `strptime()` and the `as.Date()` function, convert `c("01/30/2012", "03/04/2011")` to the Date class [`?strptime`, `as.Date()`]

Topic

Data Frame

A **data frame** is one of the most important data structures in R

- Shaped like a matrix
- Can contain multiple data types
- Both rows and columns can have human readable names

```
> data(mtcars) # load the included "MT Cars" data
> head(mtcars) # look at the first few rows
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

data.frame metadata

A number of functions are available for inspecting data.frame objects:

```
> # row and column names
> head(names(mtcars)) # variable names in mtcars
[1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"
> head(rownames(mtcars)) # first few rownames of mtcars
[1] "Mazda RX4"          "Mazda RX4 Wag"      "Datsun 710"
[4] "Hornet 4 Drive"     "Hornet Sportabout" "Valiant"
>
> # dimensions
> dim(mtcars)
[1] 32 11
>
> # structure
> str(mtcars[, 1:5]) # get structure of first 5 cols
'data.frame':   32 obs. of  5 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
```

Extracting subsets of data.frames

You can select single columns of a data.frame using the dollar sign

```
> mean(mtcars$mpg)
[1] 20.09062
```

You can flexibly extract subsets of data.frames using bracket notation

```
> # extracting subsets
> mtcars[1:5 ,1] # rows 1 through 5, column 1
[1] 21.0 21.0 22.8 21.4 18.7
> mtcars[1:5 ,"hp"] # rows 1-5, column "hp"
[1] 110 110  93 110 175
> mtcars[mtcars$mpg < 15, c("mpg", "gear")] # rows where MPG < 15
                   mpg gear
Duster 360          14.3    3
Cadillac Fleetwood  10.4    3
Lincoln Continental 10.4    3
Chrysler Imperial   14.7    3
Camaro Z28          13.3    3
> mtcars[c(1,2), ] # rows 1 and 2, all columns
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

Modifying Data Frames

Maybe you think that the log of the horsepower will be an important predictor of something. You can add it to your dataframe:

```
> mtcars$loghp <- log(mtcars$hp) # Make a new variable and assign it
>
```

You can flexibly replace subsets of data.frames using bracket notation

```
> # replacing subsets
> mtcars$guzzler <- "no"
> mtcars$guzzler[mtcars$mpg<20] <- "yes"
>
```

We can use the `table` command to tabulate values across one or more variables. The `with` command means that the variables are being referred to within a data frame.

```
> with(mtcars,table(guzzler,cyl)) # Are cylinders related to gas guzzling?
       cyl
guzzler  4  6  8
    no  11  3  0
    yes  0  4 14
```

Exercise

1. Load the Morley speed of light observation data `data(morley)`
2. What is the `class` of the `morley` object?
3. The `Expt` column should be a factor. Use `as.factor` to make this change.
4. Make a new column which is the distance from the mean of the `Speed` (Hint: use the `abs` function and the `mean` function)

Topic

The "working directory" and listing files

- R knows the directory it was started in, and refers to this as the "working directory".
- In RStudio, you can click on **Files**, and navigate your folder structure

just like a normal file browser.

- Click **More ==> Set as Working Directory**
  - This can also be done with an R Command

```
> getwd() # get the current working directory
[1] "/Users/astorer/Work/presentations/ista/Rintro"
> setwd("dataSets") # set wd to the dataSets folder
> getwd()
[1] "/Users/astorer/Work/presentations/ista/Rintro/dataSets"
> setwd("..")
>
```

We can also list files in a directory without leaving R

```
> list.files("dataSets") # list files in the dataSets folder
[1] "NewGSS.csv"   "gss.csv"      "gss.dta"      "gss.rds"
[5] "gss.sas7bdat" "gss.sav"      "gss.xlsx"     "gssInfo.csv"
> list.files("dataSets", pattern = ".csv") # restrict to .csv files
[1] "NewGSS.csv"  "gss.csv"     "gssInfo.csv"
```

Reading and writing delimited plain-text data

Delimited files can be read and written to file with the `read.*` and `write.*` functions

```
> list.files("dataSets", # list .csv files in the dataSets folder
+            pattern = ".csv")
[1] "NewGSS.csv"  "gss.csv"     "gssInfo.csv"
> gss.data <- read.csv("dataSets/gss.csv") # read gss data
> write.csv(gss.data, # write gss data to a new csv file
+           file = "dataSets/NewGSS.csv")
>
```

For details see

`?read.table`

and

`?write.table`

Reading data from other programs

R has tools for bringing in data from SPSS, Stata, SAS, etc. First, you need to load the R package, "foreign"

```
> # Read a Stata dataset and assign the results to a variable named "datGSS"
> library(foreign) # load foreign data functions
> # read Stata data
> datGSS <- read.dta(file="dataSets/gss.dta")
>
```

Always a good idea to examine the imported data set–usually we want the results to be a `data.frame`

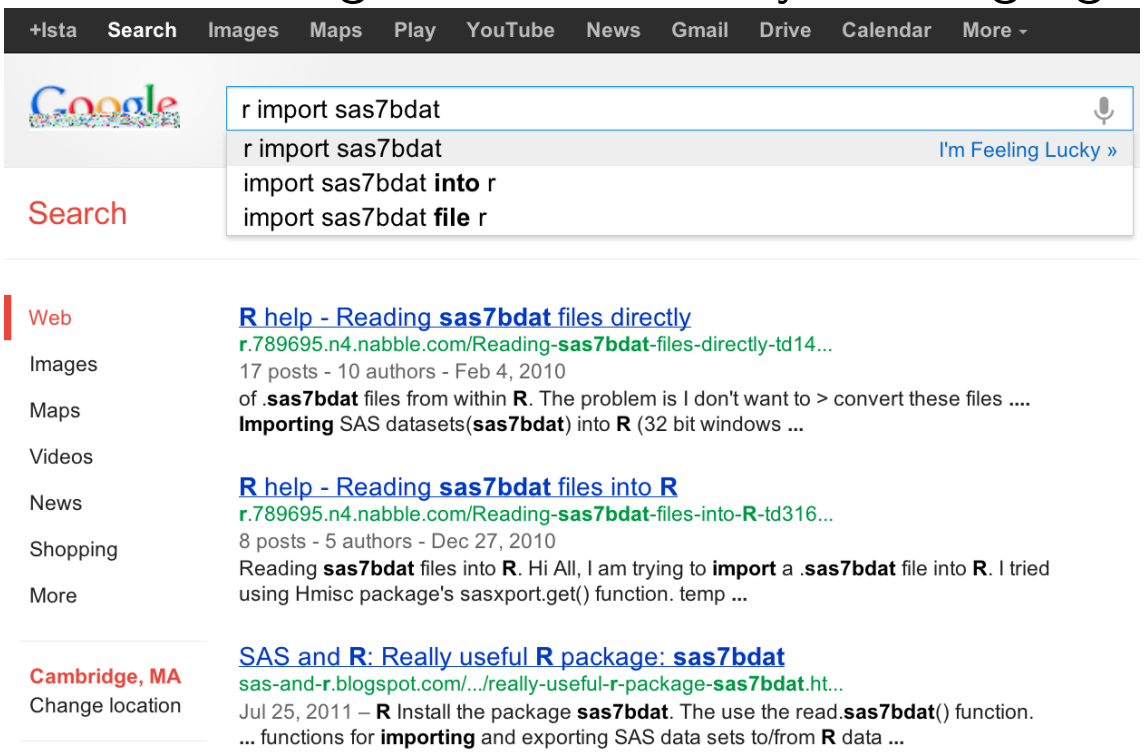
```
> class(datGSS) # check to see that test is what we expect it to be
[1] "data.frame"
> dim(datGSS)
[1] 1419   35
> names(datGSS)[1:10] # first 10 column names
 [1] "age"      "educ"     "emailhrs" "hrs1"     "sex"      "usecomp"
 [7] "usemail"  "useweb"   "webhrs"   "hapmar"
```

For information about other formats, see

```
help(package="foreign")
```

What if my data cannot be read by any of the foreign package functions?

Web search engines are increadibly useful: google for "R import sas7bdat"




```
> # read data from sas dataset
> # install.packages("sas7bdat")
> library(sas7bdat)
> dat.sas <- read.sas7bdat("dataSets/gss.sas7bdat")
>
```

Exercise 2: loading and manipulating data

1. Look at the help for the foreign package and determine which function you need to read data in SPSS format
2. Read the SPSS data set in dataSets/gss.sav and assign the result to an R data object named GSS.sav
3. Make sure that the data loaded in step 2 is a data.frame
4. Display the dimensions of the GSS.sav.
5. Write out a .csv file containing the values in GSS.sav
6. BONUS: figure out how to read the Excel file "gss.xlsx" into R

Topic

The gss dataset

The next few examples use a subset of the General Social Survey data set. The variables in this subset include

```
> library(foreign)
> datGSS <- read.dta("dataSets/gss.dta")
>
```

Stata stores lots of metadata about the data - in R, this is loaded into the `attributes` section:

```
> names(attributes(datGSS))
 [1] "datalabel"   "time.stamp"  "names"       "formats"
 [5] "types"       "val.labels"  "var.labels"  "row.names"
 [9] "version"     "label.table" "class"
> head(data.frame(var=attributes(datGSS)$names,
+                 label = attributes(datGSS)$var.labels))
       var                                        label
1      age                          Age of respondent
2     educ            Highest year of school completed
3 emailhrs Hours of e-mail per week for Internet users
4     hrs1            Number of hours worked last week
5      sex                              Respondent sex
6  usecomp                               Use computer?
```

Basic statistics

Descriptive statistics of single variables are straightforward:

```
> mean(datGSS$educ) # calculate mean of x
[1] 13.47498
> sd(datGSS$educ) # calculate standard deviation of x
[1] 5.389476
> summary(datGSS$educ) # calculate min, max, quantiles, mean
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   12.00   13.00   13.47   16.00   99.00
```

If you get tired of typing the data.frame name over and over, use `with()` instead

```
> stats <- with(datGSS, {
+   c(Lowest = min(educ),
+     Average = mean(educ),
+     Highest = max(educ))
+ })
> print(stats)
  Lowest  Average  Highest
 0.00000 13.47498 99.00000
```

Some of these functions (e.g., summary) will also work with data.frames and other types of objects

Missing values in R

R represents missing values with "NA"

```
> x.missing <- c(1, 2, NA, 4, 5)
> is.na(x.missing)
[1] FALSE FALSE  TRUE FALSE FALSE
```

R does not exclude missing values by default

```
> mean(x.missing)
[1] NA
> mean(x.missing, na.rm=TRUE)
[1] 3
```

Counts and proportions

Start by using the `table()` function to tabulate counts, then perform additional computations if needed

```
> sex.counts <- table(datGSS[, "sex"]) # tabulate sex categories
> sex.counts

  Male Female
   622    797
> prop.table(sex.counts) # convert to proportions

     Male    Female
0.4383369 0.5616631
```

Add variables for crosstabs

```
> table(datGSS[, c("sex", "happy")]) # crosstab marital X happy
        happy
sex       NAP VERY HAPPY PRETTY HAPPY NOT TOO HAPPY  DK  NA
  Male      0        189          350             73   0  10
  Female    0        246          447             84   1  19
```

Statistics by classification factors

The by() function can be used to perform a calculation separately for each level of a classifying variable

```
> by(datGSS[, c("income", "educ")],
+    INDICES=datGSS["sex"],
+    FUN=summary)
sex: Male
              income          educ
 $40000 TO 49999: 59   Min.   : 4.00
 $50000 TO 59999: 56   1st Qu.:12.00
 $60000 TO 74999: 49   Median :13.00
 $35000 TO 39999: 48   Mean   :13.68
 REFUSED        : 48   3rd Qu.:16.00
 $110000 OR OVER: 43   Max.   :99.00
 (Other)        :319
------------------------------------------------------
sex: Female
              income          educ
 REFUSED        : 76   Min.   : 0.00
 $60000 TO 74999: 62   1st Qu.:12.00
 $40000 TO 49999: 60   Median :12.00
 $50000 TO 59999: 52   Mean   :13.32
 $30000 TO 34999: 49   3rd Qu.:15.00
 $25000 TO 29999: 42   Max.   :99.00
```

Correlations

Let's look at correlations among between age, income, and education

```
> cor(datGSS[ , c("age", "incomdol", "educ")])
                 age   incomdol        educ
age        1.00000000 -0.1186564 -0.07362454
incomdol -0.11865641  1.0000000  0.21013267
educ     -0.07362454  0.2101327  1.00000000
```

For significance tests, use cor.test()

```
> with(datGSS,
+      cor.test(age, educ))

        Pearson's product-moment correlation

data:  age and educ
t = -2.779, df = 1417, p-value = 0.005525
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.12518333 -0.02166916
sample estimates:
        cor
-0.07362454
```

Multiple regression

Modeling functions generally use the *formula* interface with DV on left followed by "~" followed by predictors–for details see

```
help("formula")
```

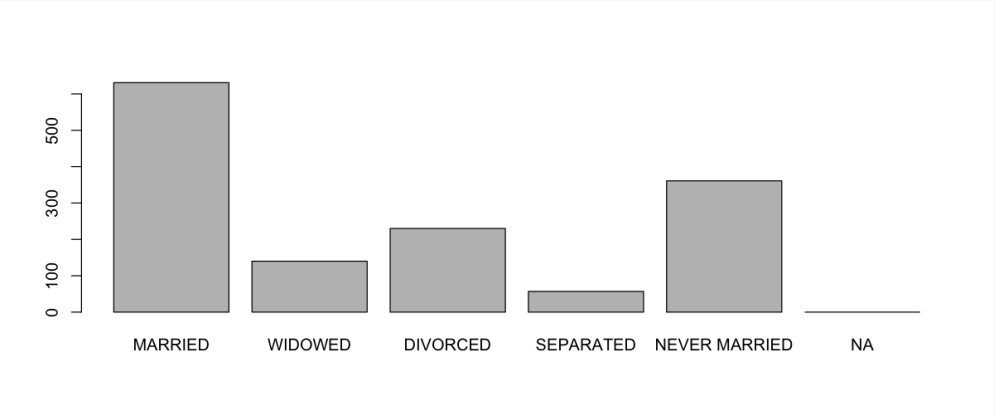
- Predict the number of hours individuals spend on email (emailhrs)

```
> m1 <- lm(emailhrs ~ sex + age, data = datGSS)
> summary(m1)

Call:
lm(formula = emailhrs ~ sex + age, data = datGSS)

Residuals:
   Min     1Q Median     3Q    Max
-5.832 -2.485 -1.394 -0.151 58.970

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.783762   0.451428   8.382  < 2e-16
sexFemale    0.077198   0.306781   0.252    0.801
age         -0.051470   0.008736  -5.891 4.78e-09

Residual standard error: 5.702 on 1416 degrees of freedom
Multiple R-squared: 0.02402,    Adjusted R-squared: 0.02264
```

Basic graphics: Frequency bars

Thanks to classes and methods, you can `plot()` many different kinds of objects:

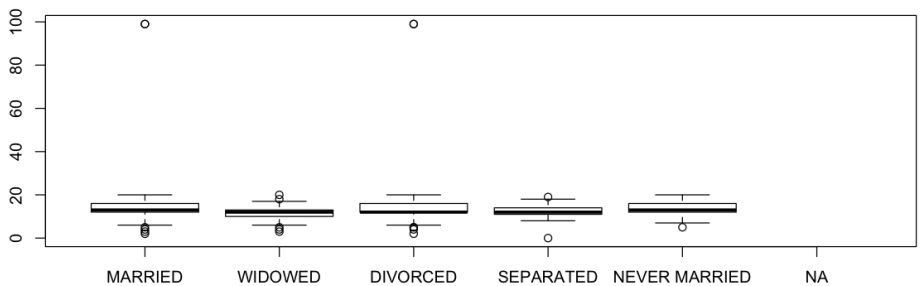
```
plot(datGSS$marital) # Plot a factor
```



Basic graphics: Boxplots by group

Thanks to classes and methods, you can `plot()` many different kinds of objects:

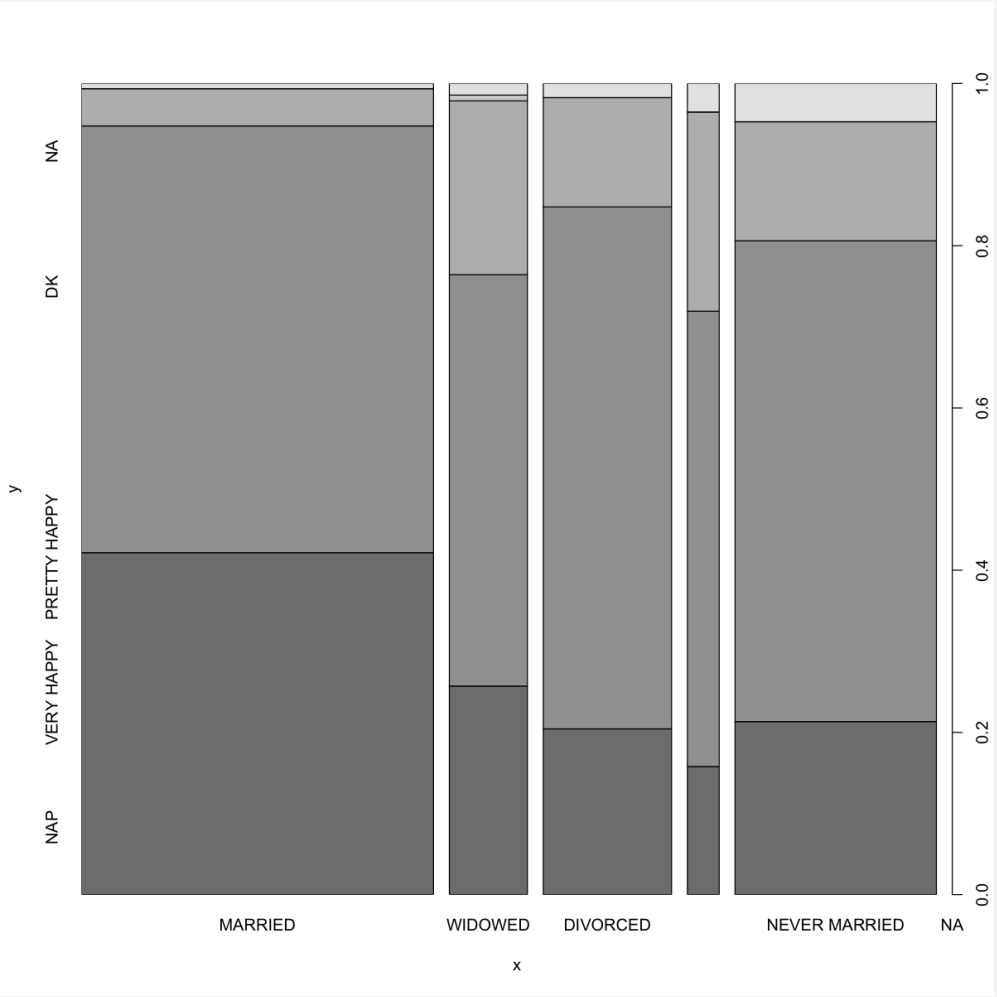
```
with(datGSS,
     plot(marital, educ)) # Plot ordinal by numeric
```



Basic graphics: Mosaic chart

Thanks to classes and methods, you can `plot()` many different kinds of objects:

```
with(datGSS, # Plot factor X factor
     plot(marital, happy))
```



Exercise 3

Using the datGSS data.frame

1. Cross-tabulate sex and emailhrs
2. Calculate the mean and standard deviation of incomdol by sex
3. The educ variable is coded such that the values 97, 98 and 99 are different types of missing data. Replace them with `NA`.
4. Create a scatter plot with educ on the x-axis and incomdol on the y-axis
5. Create a subset of the datGSS data frame that only includes female respondents
6. Generate the following variables:
   - "rich" equal to 0 if rincdol is less than 100000, and 1 otherwise
   - "sinc" equal to incomdol - rincdol
   - "dual.earn" equal to 1 if wkftwife = 1 and wkfthusb = 1, and zero otherwise
7. Create a subset of the data containing only rows where "usecomp" = "Yes"
8. Examine the data.frame created in step 7, and answer the following questions:
   - How many rows does it have?
   - How many columns does it have?
   - Is the "satjob" variable numeric?

Topic

Help us make this workshop better!

- Please take a moment to fill out a very short feedback form
- These workshops exist for you – tell us what you need!
- http://tinyurl.com/R-intro-feedback

Additional resources

- IQSS workshops: http://projects.iq.harvard.edu/rtc/filter_by/workshops
- Software (all free!):
  - R and R package download: http://cran.r-project.org
  - Rstudio download: http://rstudio.org
  - ESS (emacs R package): http://ess.r-project.org/
- Getting help:
  - Documentation and tutorials: http://cran.r-project.org/other-docs.html
  - Recommended R packages by topic: http://cran.r-project.org/web/views/
  - Mailing list: https://stat.ethz.ch/mailman/listinfo/r-help
  - StackOverflow: http://stackoverflow.com/questions/tagged/r