

# Introduction to R

Ista Zahn

Harvard MIT Data Center



**The Institute**  
*for* Quantitative Social Science  
at Harvard University

# Outline

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions
- 4 Getting data into R
- 5 Data Manipulation
- 6 Basic Statistics and Graphs
- 7 Wrap-up

# Topic

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions
- 4 Getting data into R
- 5 Data Manipulation
- 6 Basic Statistics and Graphs
- 7 Wrap-up

# Materials and setup

Everyone should have R installed on their laptop—if not:

- Open a web browser and go to <http://cran.r-project.org> and download and install it
- Also helpful to install RStudio (download from <http://rstudio.com>)

Materials for this workshop include slides, example data sets, and example code.

- Download materials from <http://j.mp/intro-r>
- Extract the zip file containing the materials to your desktop

Workshop notes are available in .html and .pdf format. Navigate to your desktop and open either Rintro.pdf or Rintro.html.

# What is R?

R is a programming language designed for statistical computing. Notable characteristics include:

- Vast capabilities, wide range of statistical and graphical techniques
- Very popular in academia, growing popularity in business:  
<http://r4stats.com/articles/popularity/>
- Written primarily by statisticians
- FREE (no cost, open source)
- 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/~hiebelier/comp/matlabR.pdf>

**SciPy** <http://mathesaurus.sourceforge.net/matlab-python-xref.pdf>

**SAS/SPSS** <http://www.et.bs.ehu.es/~etptupaf/pub/R/RforSAS&SPSSusers.pdf>

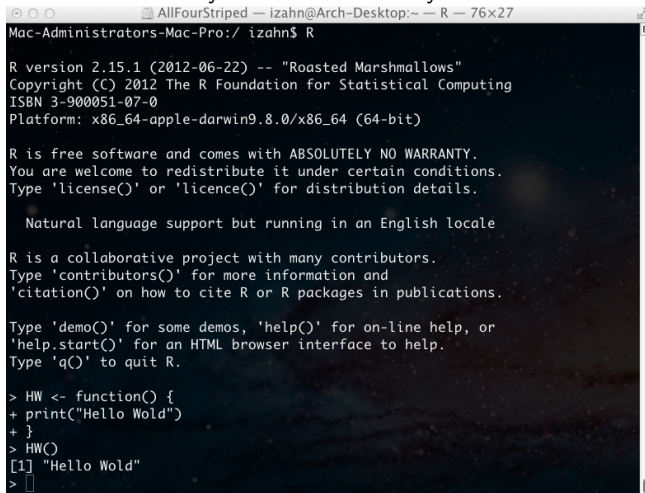
**Stata** <http://www.dss.princeton.edu/training/RStata.pdf>

# Topic

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions
- 4 Getting data into R
- 5 Data Manipulation
- 6 Basic Statistics and Graphs
- 7 Wrap-up

# R GUI alternatives (no GUI)

The old-school way is to run R directly in a terminal

A screenshot of a terminal window titled "AllFourStriped — izahn@Arch-Desktop:~ — R — 76x27". The prompt is "Mac-Administrators-Mac-Pro:/ izahn\$ R". The output shows the R version (2.15.1), copyright (2012 The R Foundation), platform (x86\_64-apple-darwin9.8.0/x86\_64), and a disclaimer. It then lists various help commands like 'license()', 'demo()', and 'help.start()'. Finally, a function 'HW' is defined and called, resulting in the output "[1] \"Hello Wold\"".

```
Mac-Administrators-Mac-Pro:/ izahn$ R

R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

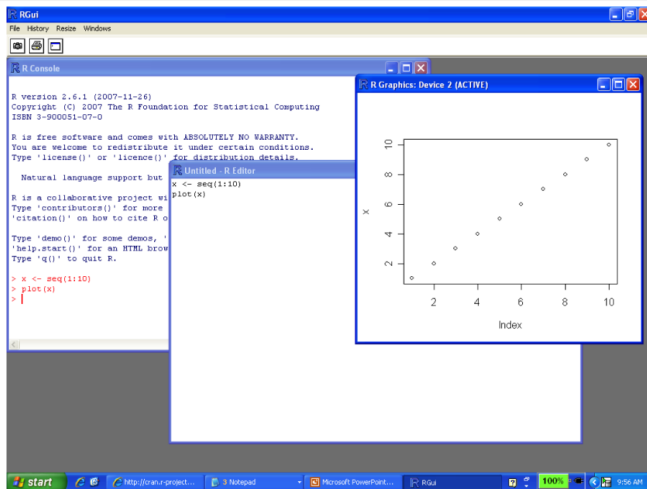
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> HW <- function() {
+ print("Hello Wold")
+ }
> HW()
[1] "Hello Wold"
>
```

But hardly anybody does it that way anymore!



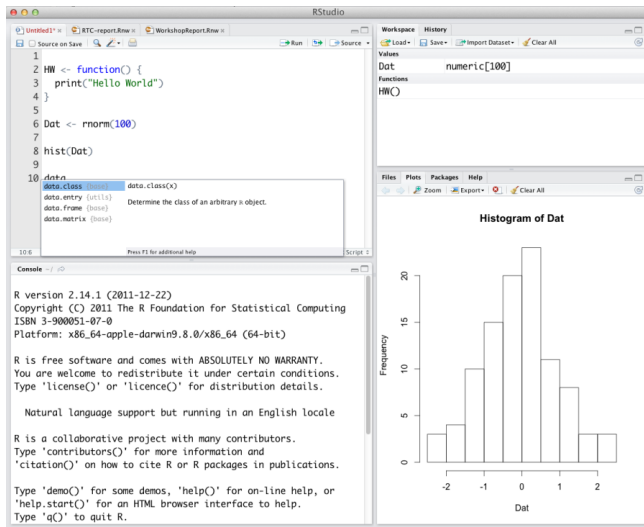
# R GUI alternatives (Windows default)



The default windows GUI is not very good

- No parentheses matching or syntax highlighting
- No work-space browser

# R GUI Alternatives (Rstudio on Mac)



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

# R GUI Alternatives (Emacs with ESS)

The screenshot shows the Emacs editor interface with the ESS package running R. The main editor window contains the following R code:

```
HW <- function() {
  print("Hello World")
}

Dat <- rnorm(100)

png(file = "myHist.png")
hist(Dat)
dev.off()
```

On the right side, a variable list shows:

Var	mode	length
Dat	numeric	100
HW	function	1

A help window for the `data.frame` function is open, showing its description: "This function creates data frames."

At the bottom, a histogram titled "Histogram of Dat" is displayed, showing the frequency distribution of the generated data. The x-axis is labeled "Dat" and ranges from -3 to 3. The y-axis is labeled "Frequency" and ranges from 0 to 35. The histogram shows a bell-shaped distribution centered around 0.

Emacs + ESS is a very powerful combination, but can be difficult to set up

# Components of R GUIs

- The R console
  - Displays command history and results
  - Commands can be typed directly in the console
  - R Console work disappears once session is closed
- A text editor
  - A plain text editor for writing R code
  - Good ones will have syntax highlighting, parentheses matching etc.
  - Anything that modifies your data should be done in a text editor
- Graphics windows
  - View, re-size, and save graphics
  - A good GUI will allow you to cycle through graph history
- Work-space viewer
  - Some GUIs have work-space browsers that allow you to see stored objects
  - Very helpful if you are absentminded like me and frequently forget what names you gave your data!

# Asking R for help

- Start html help, search/browse using web browser

- at the R console:

```
help.start()
```

- or use the help menu from you GUI

- Look up the documentation for a function

```
help(topicName)
```

```
?topicName
```

- Look up documentation for a package

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

- Search documentation from R (not always the best way... google often works better)

```
help.search("topicName")
```

# R packages and libraries

There are thousands of R packages that extend R's capabilities.

- 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, like Stata (unlike SAS)
- 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 forward slash ("/") instead of backslash in path names, even on Windows

# Exercise 0

- 1 Try to get R to add 2 plus 2
- 2 See if you can find the help page for the "mean" topic
- 3 Using any means available, try to figure out how to run a linear regression model in R
- 4 Go to <http://cran.r-project.org/web/views/> and skim the topic closest to your field/interests



# Topic

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions**
- 4 Getting data into R
- 5 Data Manipulation
- 6 Basic Statistics and Graphs
- 7 Wrap-up

# 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 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] "tmp" "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] "tmp" "y"
```

# Functions

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

- take **variable(s)** as input **argument(s)**
- perform operations
- **return** values which can be **assigned**
- optionally perform side-effects such as writing a file to disk or opening a graphics window

The general form for calling R functions is

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

Arguments can be matched by position or name

Examples:

```
> #?sqrt
> a <- sqrt(y) # Call the sqrt function with argument x=y
> round(a, digits = 2) # Call round() with arguments x=a 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
```

# Topic

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions
- 4 Getting data into R**
- 5 Data Manipulation
- 6 Basic Statistics and Graphs
- 7 Wrap-up

# The gss dataset

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

```
> head(read.csv("dataSets/gssInfo.csv"))
      var                description
1 marital          marital status
2   age            age of respondent
3  educ highest year of school completed
4   sex            respondents sex
5   inc            respondents income
6  happy            general happiness
> #see gssInfo.csv for rest of the variable descriptions
>
```

# The "working directory" and listing files

R knows the directory it was started in, and refers to this as the "working directory". Since our workshop examples are in the Rintro folder on the desktop, we should all take a moment to set that as our working directory: We can also set the working directory using paths relative to the current working directory:

```
> getwd() # get the current working directory
[1] "/Users/izahn/Documents/Work/IQSS/Classes/Rintro/New"
> setwd("dataSets") # set wd to the dataSets folder
> getwd()
[1] "/Users/izahn/Documents/Work/IQSS/Classes/Rintro/New/dataSets"
> setwd("../") # set wd to enclosing folder ("up")
>
```

It can be convenient to list files in a directory without leaving R

```
> list.files("dataSets") # list files in the dataSets folder
[1] "gss.csv"          "gss.dta"          "gss.rds"
[4] "gss.sas7bdat"     "gss.sav"          "gss.xlsx"
[7] "gssInfo.csv"      "states.csv"       "states.dta"
[10] "states.xlsx"      "statesCodebook.txt"
> # list.files("dataSets", pattern = ".csv") # restrict to .csv files
>
```

# Importing data from files

In order to read data from a file, you have to know what kind of file it is. The table below lists the functions needed to import data from common file formats.

data type	function	package
comma separated (.csv)	read.csv()	utils (default)
other delimited formats	read.table()	utils (default)
Stata (.dta)	read.dta()	foreign
SPSS (.sav)	read.spss()	foreign
SAS (.sas7bdat)	read.sas7bdat()	sas7bdat
Excel (.xls, .xlsx)	readWorksheetFromFile()	XLConnect

Examples:

```
> # read gss data from the gss.rds R file
> datGSS <- readRDS("dataSets/gss.rds")
> # read gss data from the gss.csv comma separated file
> gss.data <- read.csv("dataSets/gss.csv") # read gss data
> # read a Stata dataset from gss.dta
> library(foreign) # load foreign data functions
> datGSS <- read.dta(file="dataSets/gss.dta")
>
```

# Checking imported data

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) # how many rows and columns?
[1] 1419    35
> names(datGSS)[1:10] # first 10 column names
[1] "age"      "educ"      "emailhrs"  "hrs1"      "sex"      "usecomp"
[7] "usemail"  "useweb"    "webhrs"    "hapmar"
> str(datGSS[1:5]) # more details about the first 5 columns
'data.frame': 1419 obs. of  5 variables:
 $ age      : num  69 27 19 21 19 87 42 19 78 70 ...
 $ educ     : num  12 10 11 9 11 8 11 11 7 9 ...
 $ emailhrs : num  -1 -1 0 -1 50 -1 3 -1 -1 -1 ...
 $ hrs1     : num  -1 60 32 20 -1 -1 -1 -1 -1 22 ...
 $ sex      : Factor w/ 2 levels "Male","Female": 1 1 1 1 1 2 1 2 2 2 ...
```



# Saving and loading R workspaces

In addition to importing individual datasets, R can save and load entire workspaces

- Save our entire workspace

```
> ls() # list objects in our workspace
[1] "a"          "datGSS"     "gss.data"   "tmp"        "y"
> save.image(file="myWorkspace.RData") # save workspace
> rm(list=ls()) # remove all objects from our workspace
> ls() # list stored objects to make sure they are deleted
character(0)
```

- Load the "myWorkspace.RData" file and check that it is restored

```
> load("myWorkspace.RData") # load myWorkspace.RData
> ls() # list objects
[1] "a"          "datGSS"     "gss.data"   "tmp"        "y"
```

When you close R you will be asked if you want to save your workspace – if you choose yes then your workspace will be restored next time you start R

## Exercise 2: loading and manipulating data

- 1 Load the foreign package if you haven't already done so (`library(foreign)`)
- 2 Look at the help page for the `read.spss` function
- 3 Read the SPSS data set in `dataSets/gss.sav` and assign the result to an R data object named `GSS.sav`
- 4 Make sure that the data loaded in step 2 is a `data.frame` (hint: check the arguments documented in the help page)
- 5 Display the dimensions of the `GSS.sav`.
- 6 BONUS: figure out how to read the Excel file `"gss.xlsx"` into R

# Topic

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions
- 4 Getting data into R
- 5 Data Manipulation**
- 6 Basic Statistics and Graphs
- 7 Wrap-up

# data.frame objects

- Usually data read into R will be stored as a **data.frame**
- A data.frame is a list of vectors of equal length
  - Each vector in the list forms a column
  - Each column can be a different type of vector
  - Often the columns are variables and the rows are observations
- A data.frame has two dimensions corresponding the number of rows and the number of columns (in that order)

# data.frame meta-data

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

```
> # row and column names
> head(names(datGSS)) # variable names in datGSS
[1] "age"      "educ"      "emailhrs" "hrs1"      "sex"      "usecomp"
> head(rownames(datGSS)) # first few rownames of datGSS
[1] "1" "2" "3" "4" "5" "6"
> # dimensions
> dim(datGSS)
[1] 1419  35
> # structure
> #str(datGSS) # get structure
>
```

# Logical operators

It is often useful to select just those rows of your data where some condition holds—for example select only rows where sex is 1 (male). The following operators allow you to do this:

- `==` equal to
- `!=` not equal to
- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to
- `&` and
- `|` or

Note the double equals signs for testing equality. The following example show how to use some of these operators to extract and replace elements matching specific conditions.

# Extracting subsets of data.frames

You can flexibly extract subsets of data.frames using single brackets

- The first index corresponds to rows, the second to columns
- Empty index means "all"

```
> # extracting subsets
> # datGSS[c(1,2), ] # rows 1 and 2, all columns
> # datGSS[ , c(1,2)] # all rows, columns 1 and 2
> # datGSS[ , c("age", "educ")] # same as above
>
> datGSS[1:5, 1] # rows 1 through 5, column 1
[1] 69 27 19 21 19
> datGSS[1:5, "educ"] # rows 1-5, column "educ"
[1] 12 10 11 9 11
> datGSS[datGSS[, "age"] > 90, c("sex", "age")] # rows where age > 90
      sex age
315 Female 99
665  Male 99
```

Note the use of the `c()` function to combine arguments

# Replacing subsets of data.frames

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

```
> # creating new variable mean centered age
> datGSS[ , "ageC"] <- datGSS[ , "age"] - mean(datGSS[ , "age"])
>
> #education difference between wives and husbands
> datGSS[ , "educ.diff"] <- datGSS[ , "wifeduc"] - datGSS[ , "husbeduc"]
>
> # replacing subsets to create young/old variable
> datGSS[ , "young"] <- "no" # all values of young = "no"
> datGSS[datGSS[ , "age"] < 30, "young"] <- "yes" # change to "yes" if age < 30
>
> datGSS[1:4, c("age", "ageC", "young", "wifeduc", "husbeduc", "educ.diff")]
```

	age	ageC	young	wifeduc	husbeduc	educ.diff
1	69	22.36364	no	NA	NA	NA
2	27	-19.63636	yes	13	10	3
3	19	-27.63636	yes	NA	NA	NA
4	21	-25.63636	yes	NA	NA	NA



# Exporting Data

Now that we have made some changes to our GSS data set, we might want to save those changes to a file. Everything we have done so far has only modified the data in R; the files have remained unchanged.

```
> # write data to a .csv file
> write.csv(datGSS, file = "gss.csv")
> # write data to a Stata file
> write.dta(datGSS, file = "gss.dta")
> # write data to an R file
> saveRDS(datGSS, file = "gss.rds")
>
```

# Exercise 3: Data manipulation

Use the gss.rds data set

- ❶ 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
- ❷ Create a subset of the data containing only rows where "usecomp" = "Yes"
- ❸ Examine the data.frame created in step 2, and answer the following questions:
  - How many rows does it have?
  - How many columns does it have?
  - Is the "satjob" variable numeric?

# Topic

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions
- 4 Getting data into R
- 5 Data Manipulation
- 6 Basic Statistics and Graphs**
- 7 Wrap-up

# 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

```
> with(datGSS,
+       c(Lowest = min(educ),
+         Average = mean(educ),
+         Highest = max(educ))
+       )
      Lowest  Average  Highest
0.000000 13.47498 99.00000
```

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

# 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
```

sex	happy						DK	NA
	NAP	VERY	HAPPY	PRETTY	HAPPY	NOT TOO		
Male	0		189		350		73	0
Female	0		246		447		84	1

# 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(educ ~ sex + age, data = datGSS)
> summary(m1)
```

Call:

```
lm(formula = educ ~ sex + age, data = datGSS)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-13.434	-1.785	-0.688	1.955	86.049

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	14.652702	0.425691	34.421	< 2e-16
sexFemale	-0.275235	0.289290	-0.951	0.34156
age	-0.021938	0.008238	-2.663	0.00783

Residual standard error: 5.377 on 1416 degrees of freedom

Multiple R-squared: 0.006056, Adjusted R-squared: 0.004652

F-statistic: 4.214 on 2 and 1416 DF, p-value: 0.01256



# Save R output to a file

Earlier we learned how to write a data set to a file. But what if we want to write something that isn't in a nice rectangular format, like the results of our regression model? For that we can use the `sink()` function:

```
> sink(file="output.txt", split=TRUE) # start logging
> print("This is the result from model 1\n")
[1] "This is the result from model 1\n"
> print(summary(m1))
```

Call:

```
lm(formula = educ ~ sex + age, data = datGSS)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-13.434	-1.785	-0.688	1.955	86.049

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	14.652702	0.425691	34.421	< 2e-16
sexFemale	-0.275235	0.289290	-0.951	0.34156
age	-0.021938	0.008238	-2.663	0.00783

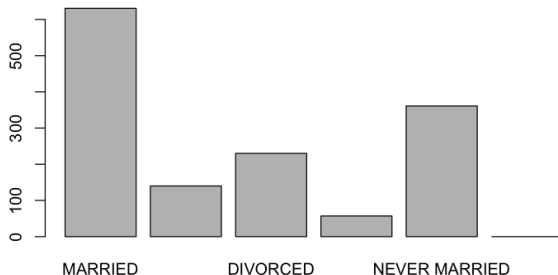
Residual standard error: 5.377 on 1416 degrees of freedom

Multiple R-squared: 0.006056, Adjusted R-squared: 0.004652

# Basic graphics: Frequency bars

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

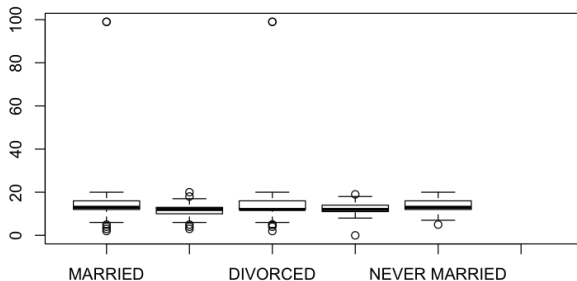
```
plot(datGSS[ , "marital"]) # Plot a factor
```



# Basic graphics: Boxplots by group

Thanks to classes and methods, you can `plot()` many 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 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 Save the results of the previous two calculations to a file
- 4 Create a scatter plot with `educ` on the x-axis and `incomdol` on the y-axis

# Topic

- 1 Workshop Materials and Introduction
- 2 Graphical User Interfaces
- 3 Data and Functions
- 4 Getting data into R
- 5 Data Manipulation
- 6 Basic Statistics and Graphs
- 7 Wrap-up**

# 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](http://projects.iq.harvard.edu/rtc/filter_by/workshops)
- IQSS statistical consulting: <http://rtc.iq.harvard.edu>
- 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/>
- Online tutorials
  - <http://www.codeschool.com/courses/try-r>
  - <http://www.datamind.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>