Welcome to
SI 618: Exploratory Data Analysis

Fall 2016

Instructor: Dr. Chris Teplovs (cteplovs@umich.edu)

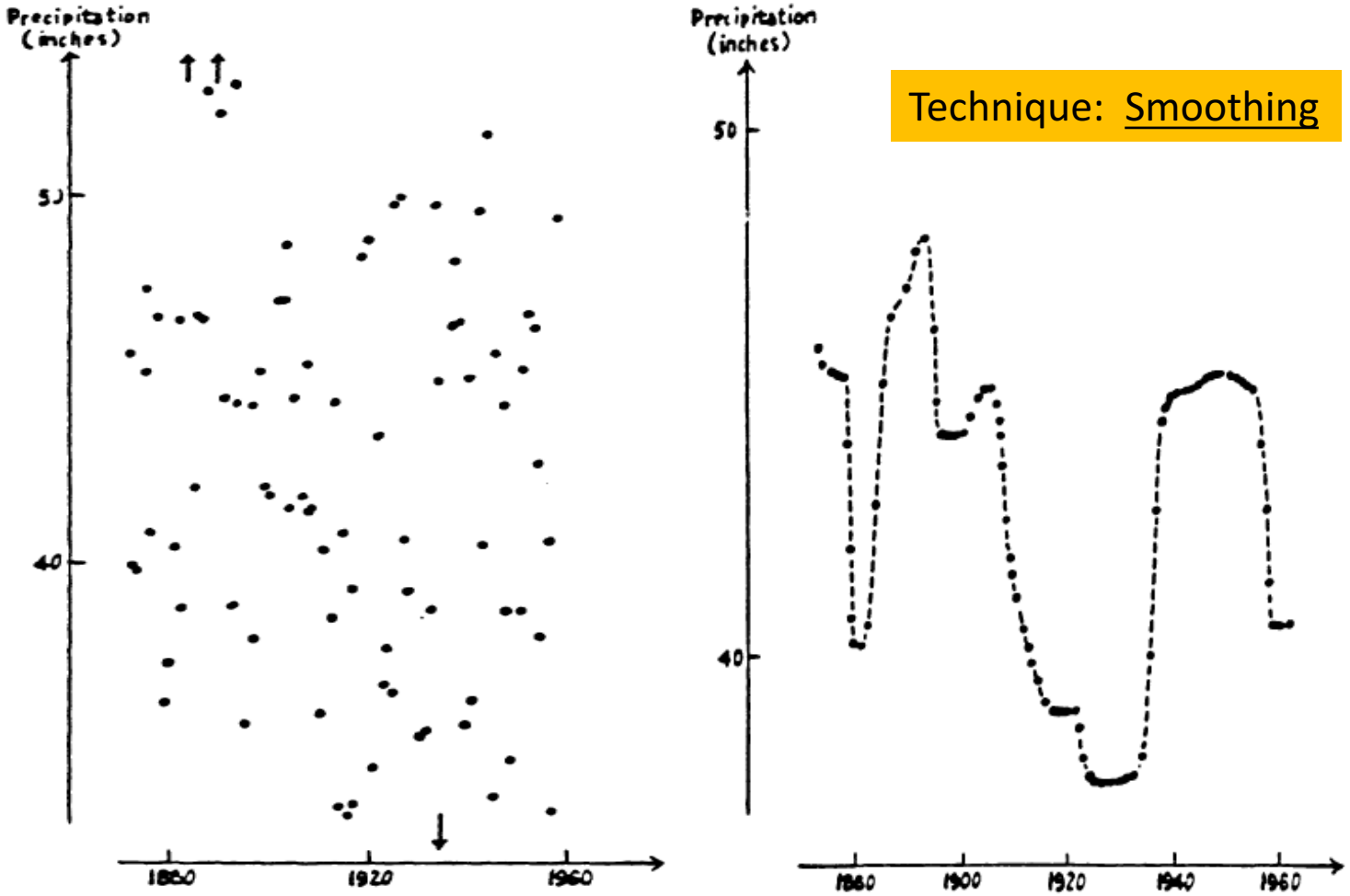
Lead Developer, Digital Innovation Greenhouse, Office of Academic Innovation

Adjunct Lecturer of Information, School of Information

GSI: SungJin Nam (sjnam@umich.edu)

Some material courtesy of: Kevyn Collins-Thompson and Yuhang Wang

Exploratory data analysis is detective work



Technique: Smoothing

Source: J. Tukey, Exploratory Data Analysis

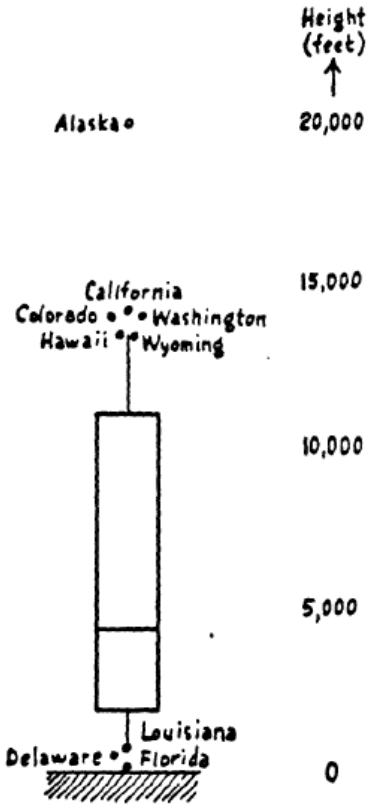
Data exploration makes you a detective

- Exploratory data analysis = Detective work
  - Tools to find clues, put together a story, identify suspects
  - Sometimes clues are accidental or misleading
  - But it's critical to find them anyway.. *Overlooking* potential clues could be even worse!
- Goals:
  - Simplified descriptions of what's happening in possibly complex data
  - Look below the surface to get new insights
  - See interesting behavior through visualization

Exploration involves trying different techniques from your toolkit

Technique: Boxplot

Heights of 50 states

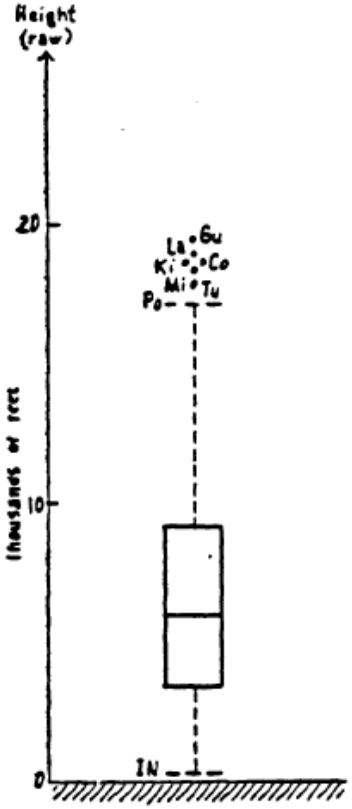


Source: J. Tukey, Exploratory Data Analysis

Different tools can reveal different insights

What are the heights of 217 active volcanoes?

Height

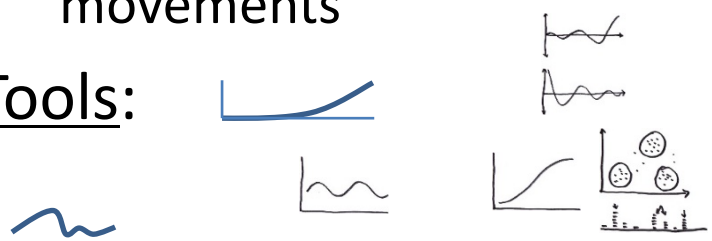


Technique:
Change of scale

Source: J. Tukey, Exploratory Data Analysis

Exploration vs. Interpretation

- Exploration uncovers clues about what is there
- Finds possible relationships, trends
- Example:
  - Find unusual stock price movements
- Tools:



Our focus in SI 618

- Interpretation seeks to formulate and confirm hypotheses to explain why
- Example:
  - Create a financial model that attempts to explain and predict stock price movement
- Tools:
  - Regression models
  - Classification
  - Confidence testing

Exploration and Interpretation often proceed side-by-side

SI 618 Data Exploration: Class Schedule

(Some curriculum details subject to change)

Date	Topic	Assignments Due
Week 1	Course introduction Basics of Programming with R	
Week 2	Basics of ggplot2 Computing summary statistics. Manipulating data frames. Using qplot, plyr.	Homework/Lab 1
Week 3	Smoothing and Trend-finding. Building ggplots layer by layer	Homework/Lab 2
Week 4	Finding relationships between variables Time series and autocorrelation	Homework/Lab 3
Week 5	Clustering and outlier detection	Homework/Lab 4
Week 6	Principal Component Analysis and Exploratory Factor Analysis	Homework/Lab 5
Week 7	Machine Learning **Project Presentations**	Project Due

Texts

- **Required:**
  - Hadley Wickham, ggplot2: Elegant graphics for data analysis, Springer (2009) http://www.springerlink.com.proxy.lib.umich.edu/content/978-0-387-98140-6/contents/
- **Recommended:**
  - Phil Spector, Data Manipulation with R, Springer (2008) http://www.springer.com/statistics/computanional+statistics/book/978-0-387-74730-9
  - Leland Wilkinson, The Grammar of Graphics, Springer (2005) http://www.springerlink.com.proxy.lib.umich.edu/content/978-0-387-24544-7/contents/
  - Wes McKinney (2012). Python for Data Analysis. O'Reilly Media. ISBN: 978-1-4493-1979-3, Ebook ISBN: 978-1-4493-1978-6
- **Other related works:**
  - Peter Dalgaard, Introductory Statistics with R, Springer (2008) http://link.springer.com/book/10.1007%2F978-0-387-79054-1

- **NOTE: Most of these are in the Resources folder on Canvas**

Grading

- **Homework (75 %)** - There will be 4 x 100 point homework assignments
  - Assignments will be posted on Canvas.
  - Some future homework may be broken into separately-graded lab and assignment steps.
- **Project (25 %)** – Individual project worth 100 points. This will involve doing cool exploratory data analysis to reveal something surprising.
  - Something you can add to your portfolio to show off your skills
- **Bonus (10%)** – Homework #5 is optional. If completed if may contribute up to 10% bonus of your final grade

Note: Minor adjustments may be made to this rubric

Late Policy

- You have 3 penalty-free late days
  - One late day = one 24hr period after due date
  - No fractional late days: all or nothing
- 20% penalty per day after late days used up
- You don’t need to explain late days
  - We track them for you
- Submit late assignments via Canvas
  - Late submission is possible after due date

Original work policy

- Unless otherwise specified in an assignment, all submitted work must be your own, original work.
- You may discuss general approaches with others on individual assignments, but you may not copy code or other work and must indicate on your turned-in assignment who you worked with.
- Any excerpts from the work of others must be clearly identified as a quotation, and a proper citation provided.
- Any violation of the School's policy on Academic and Professional Integrity (stated in the Master's and Doctoral Student Handbooks) will result in severe penalties, which might range from:
  - failing an assignment
  - failing the course
  - being expelled from the program

  at discretion of the instructor & Senior Associate Dean for Academic Affairs.

Short version:

Any code and writing must be your creation, and yours alone

- Discuss high-level approaches with classmates
- But: do the programming on your own.

Instructor contact details

- Instructor: Chris Teplovs
- Email: **`cteplovs@umich.edu`**
- Phone: 734-615-2132
- Office: **NQ 1278 during office hours only**
- Office hours: Tuesday 1:30pm-3:30pm or by appointment
- GSI: SungJin Nam (sjnam@umich.edu)
  Office hours: **NQ1270** Tuesdays 3-5pm, Fridays 10am-noon

Questions?

- For (personal) questions about course materials, assignments, or anything else related to the course, come talk to me during my office hours. You can also send email. Please include 618 in subject line.
- For technical help with assignments or lecture material, SungJin is also available during his office hours.
- For general questions about assignments, use **Piazza** on Canvas.

Class Format

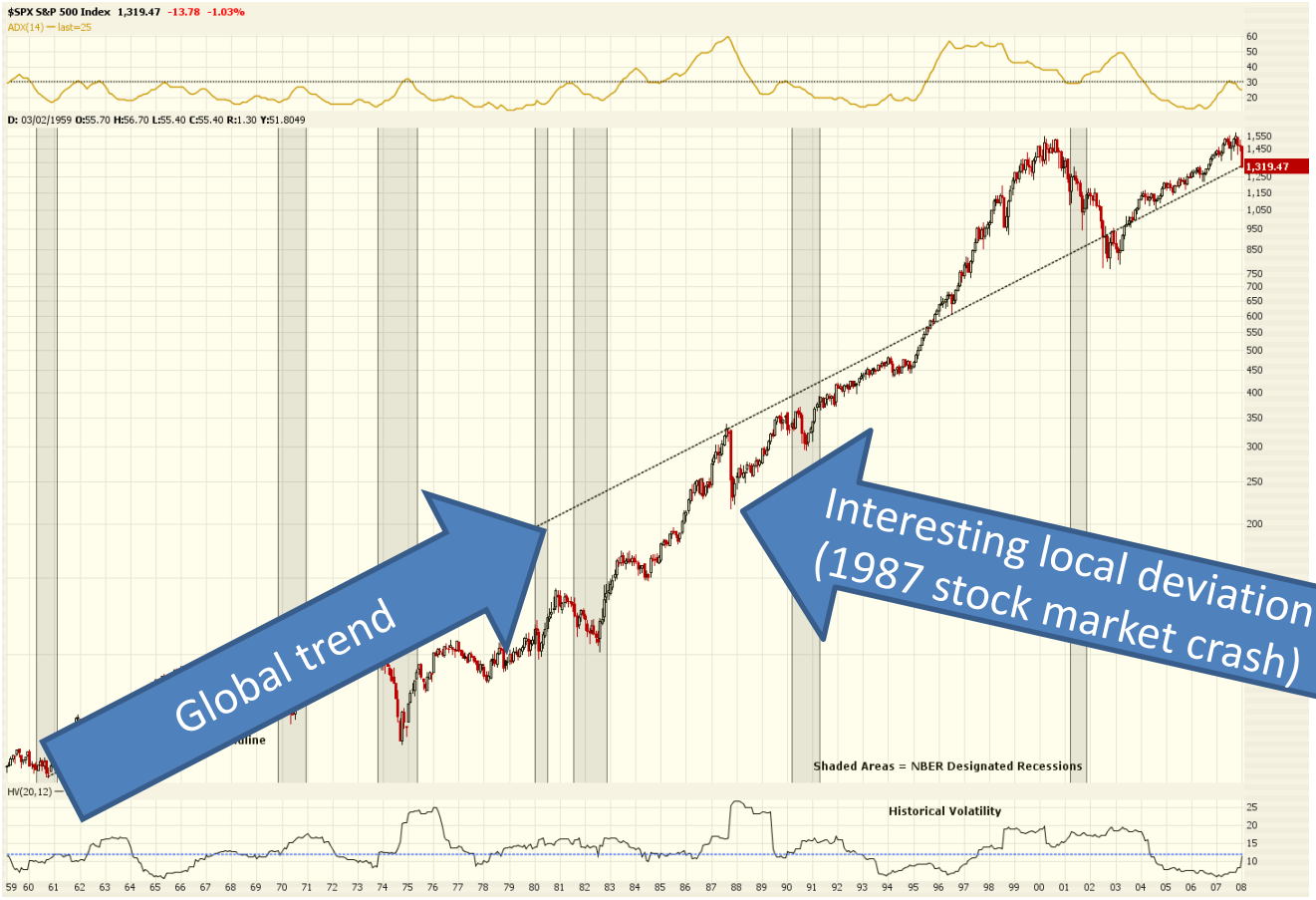
- Lecture (75-80 min)
- 15 min break
- Lab/Homework (75 min)
  - Goal: start applying what was covered in lecture
  - Interactive demos, homework review
  - Instructor & GSI will be available to help you
- Lecture/lab ratio may vary depending on topic

We want surprises!
So exploration and visualization are closely linked

- *Pictures force us to notice what we never expected to see.* - Tukey
- We need <u>pictures with impact</u>
  - Avoid too much detail
  - Avoid too obvious or simple
- Make comparisons easy

Source: http://eparity.wordpress.com/2008/01/18/50-year-monthly-chart-of-the-sp-500-index/

Data = global + local



Source: http://eparity.wordpress.com/2008/01/18/50-year-monthly-chart-of-the-sp-500-index/

Key principle of data exploration

1. First find or define "normal"
2. Then compute and visualize difference from normal

- If studying an event or property in a dataset
  - What frequency would you "normally" expect?
  - What was the actual frequency observed, compared to normal?
- Pick visualizations that emphasize those differences
  - Use the same x, y scale across facets
  - Show data (points) together with trends (smoothed fit)
  - Use graph aesthetics: color, shape, to ease comparison

Text example: what words are typical of Grade 1 stories?

Word	Frequency (Count)	Probability in Grade 1 text
the	358	0.0298
a	178	0.0148
to	153	0.0127
and	135	0.0112
in	112	0.0093
how	105	0.0087
are	95	0.0079
many	90	0.0075
on	87	0.0072
of	86	0.0071

Here are the most frequent words in "Normal" English

Grade 1

Word	Frequency (Count)	Probability in Grade 1 text
the	358	0.0298
a	178	0.0148
to	153	0.0127
and	135	0.0112
in	112	0.0093
how	105	0.0087
are	95	0.0079
many	90	0.0075
on	87	0.0072
of	86	0.0071

"Normal" English

Word	Frequency (Count)	Probability in English text
the	6187927	0.0659
of	2941790	0.0313
and	2682878	0.0286
to	2560346	0.0273
a	2150885	0.0229
in	1883295	0.0201
that	1115382	0.0119
it	1089559	0.0116
is	998867	0.0106
was	923975	0.0098

Boring! How could we find something more interesting about Grade 1 words?

One answer: Find the Grade 1 words that are much more likely than would be expected in general English

Word	Probability in Grade 1 text
the	0.0298
a	0.0148
to	0.0127
and	0.0112
in	0.0093
how	0.0087
are	0.0079
many	0.0075
on	0.0072
of	0.0071

Word	Probability in English text
the	0.0659
of	0.0313
and	0.0286
to	0.0273
a	0.0229
in	0.0201
that	0.0119
it	0.0116
is	0.0106
was	0.0098

$\frac{P_{GRADE}(w)}{P_{ENGLISH}(w)}$
0.7092
0.8285
0.7192
0.6675
0.7177
2.4812
1.2200
2.4492
0.9741
0.5272

"Surprise" factor

The words with top "surprise factor" in Grade 1 texts relative to their probability in general English

word	$\frac{P_{GRADE}(w)}{P_{ENGLISH}(w)}$
color	20.201
kittens	12.735
camels	11.988
grandpa	11.179
blanks	10.757
grownup	10.551
tots	10.551
crayons	10.486
pancakes	10.443

Which words are most ‘distinctive’ of each grade in these language models (larger corpus)?

Grade 1

grownup	2.485
ram	2.425
planes	2.411
pig	2.356
jimmy	2.324
toad	2.237
shelf	2.192
cover	2.184
spot	2.174
fed	2.164

Grade 4

desert	1.787
crew	1.765
habitat	1.763
butterflies	1.758
rough	1.707
slept	1.659
bowling	1.643
ribs	1.610
grows	1.606
entrance	1.604

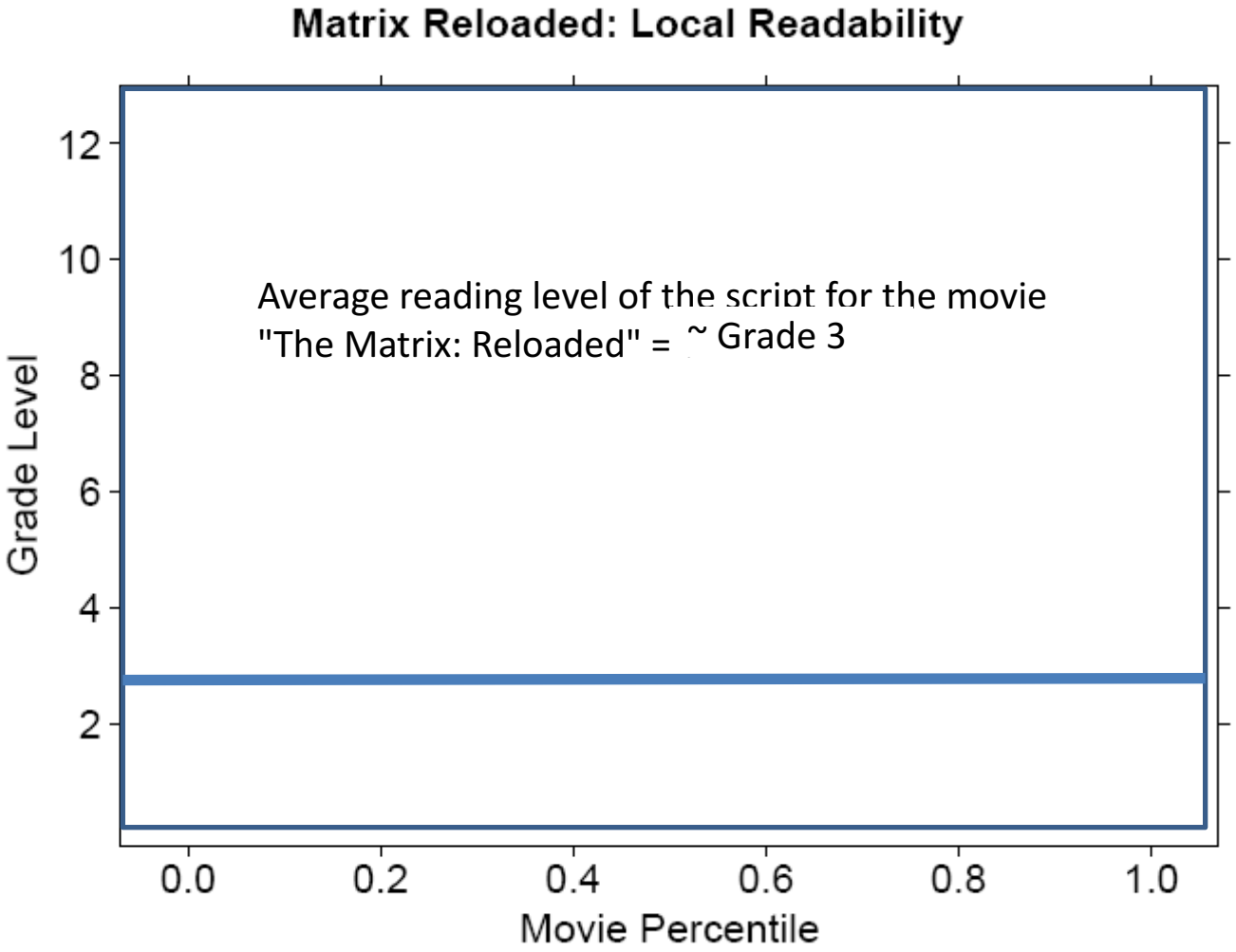
Grade 8

acidic	1.425
soda	1.425
acid	1.408
typical	1.379
angle	1.362
press	1.318
radio	1.284
flash	1.231
levels	1.229
pain	1.220

Grade 12

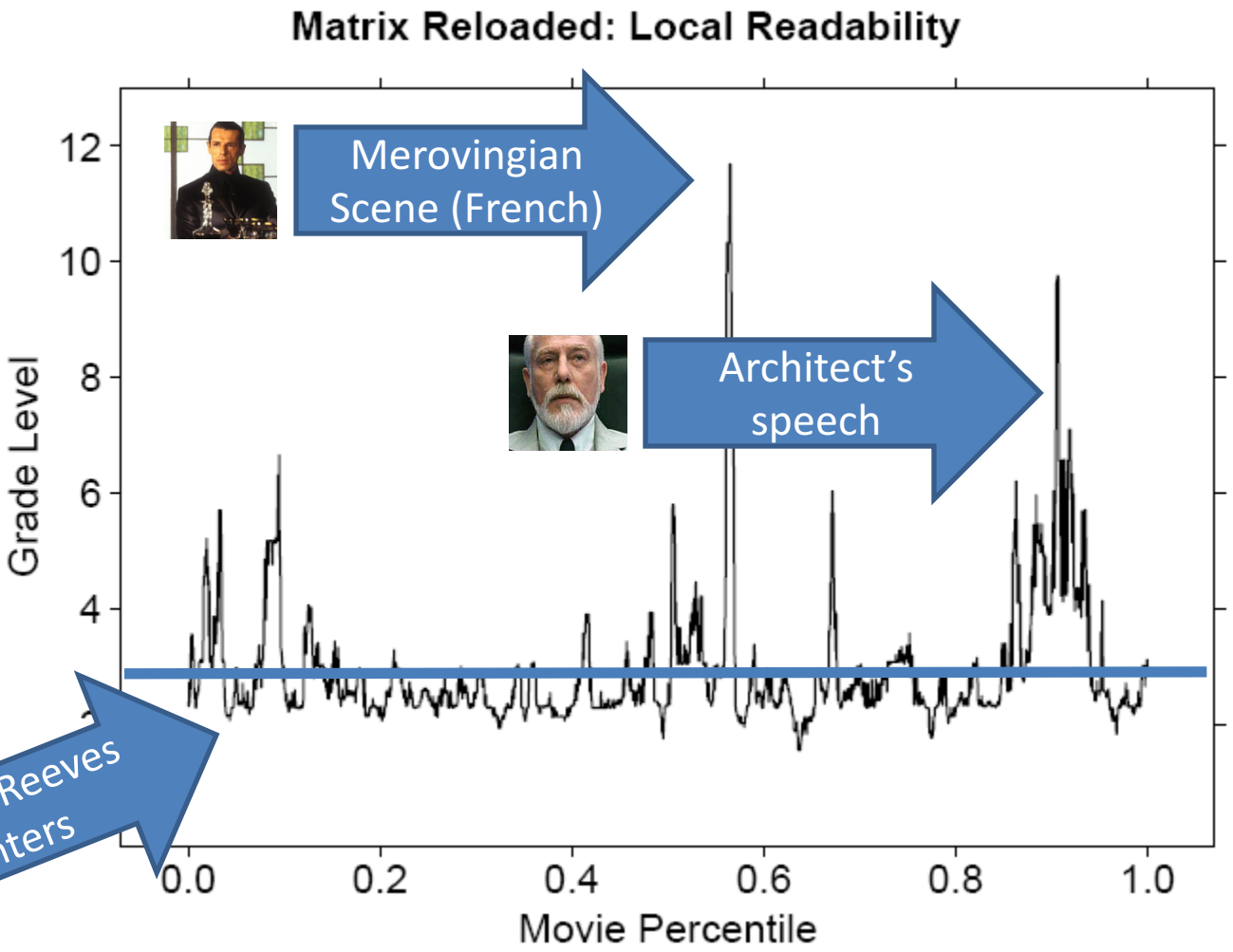
essay	2.441
literary	2.383
technology	2.363
analysis	2.301
fuels	2.296
senior	2.292
analyze	2.279
management	2.269
issues	2.248
tested	2.226

Time series surprises



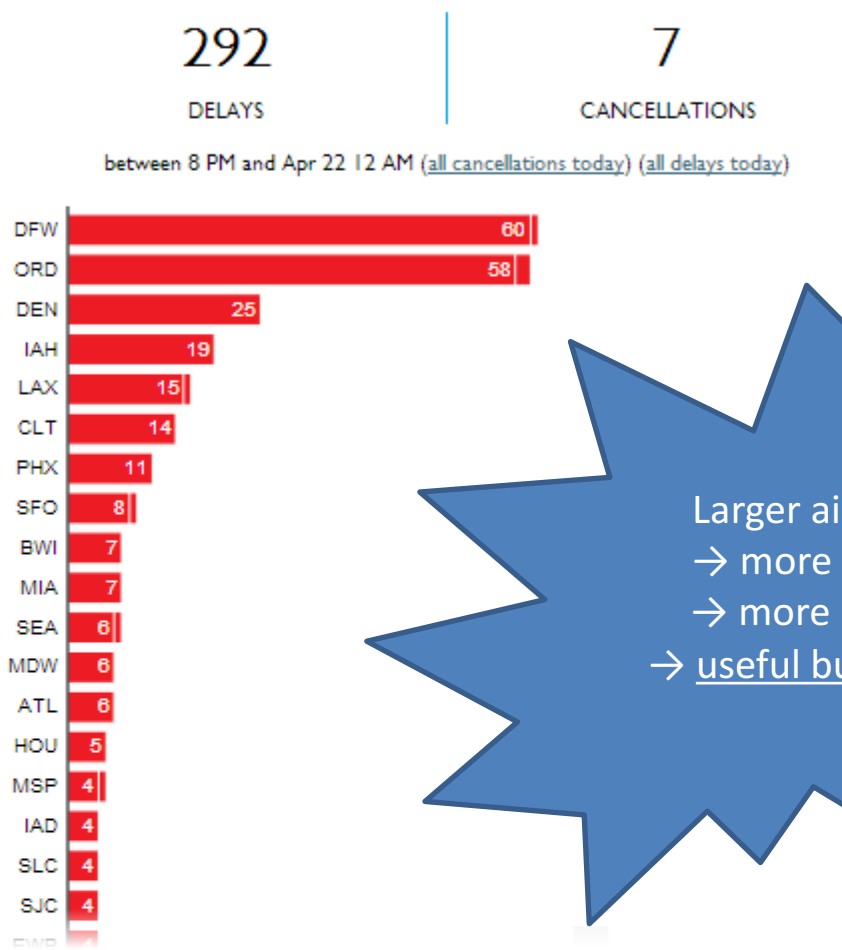
Local readability for documents with varying difficulty

Movie dialogue in "**The Matrix: Reloaded**"



[*Kidwell, Lebanon, **Collins-Thompson**. J. Am. Stats. 2011*]

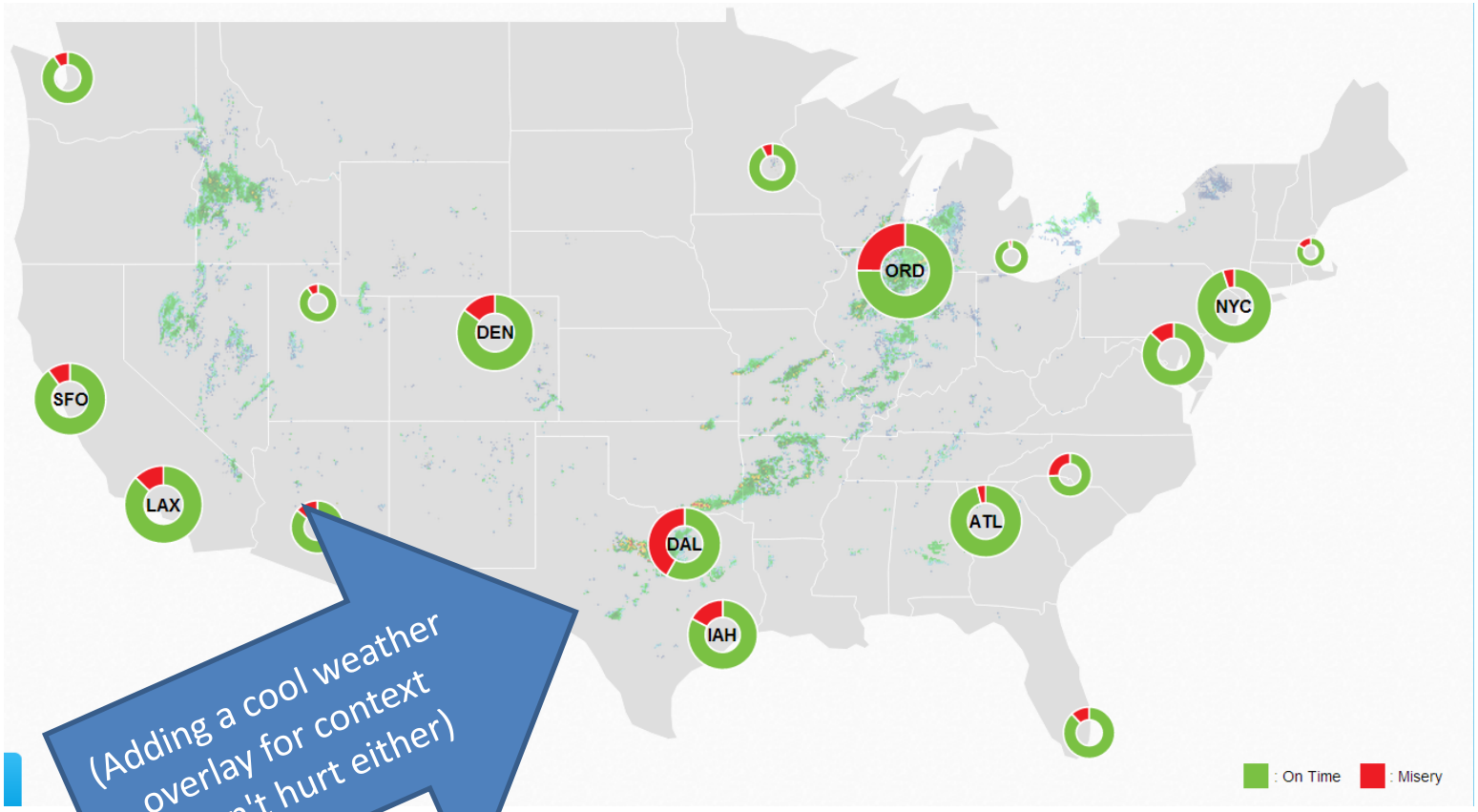
Numeric surprises: flight delays



Larger airports
→ more flights
→ more delays
→ useful but boring

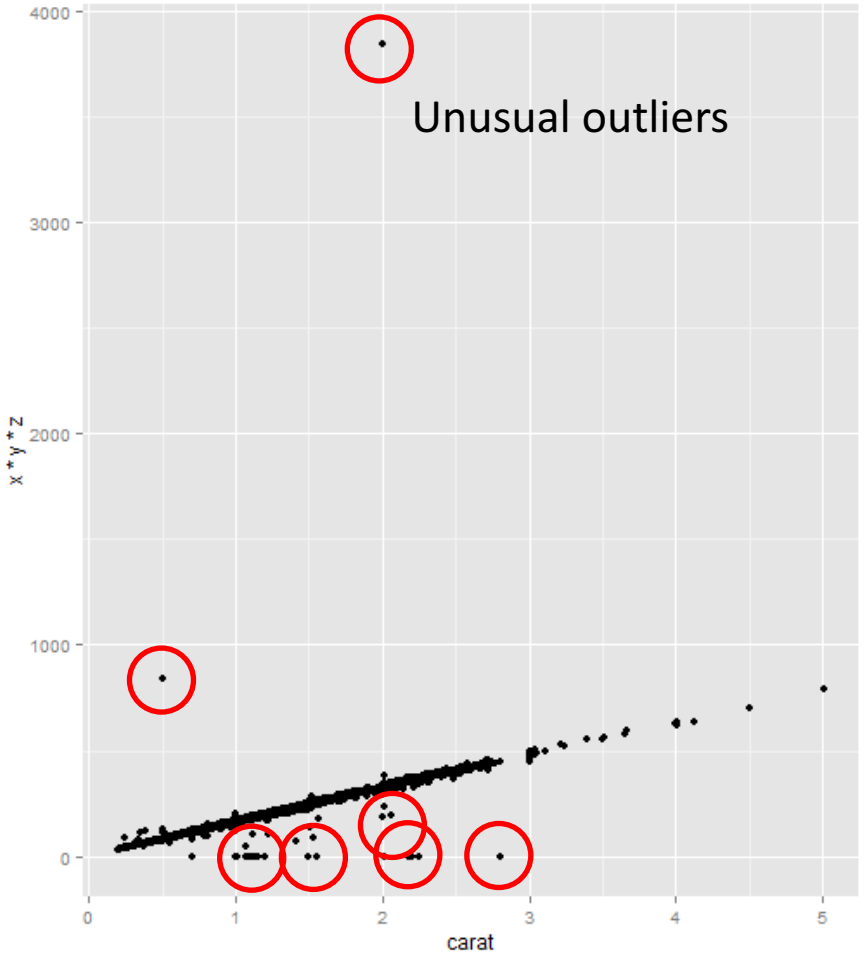
Source: http://flightaware.com

Show flight delay counts, but relative to total airport traffic



Source: http://flightaware.com

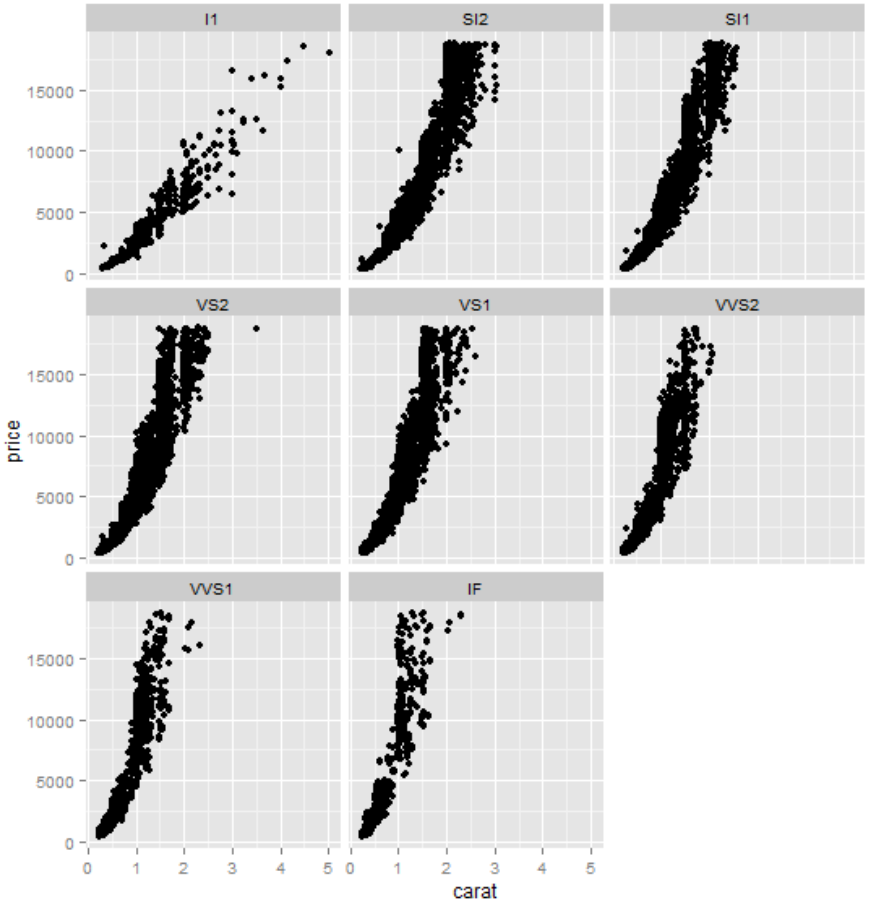
Expected: All diamonds have same density
Observed: Surprise! Some outliers differ from norm



Use faceted plots to show differences in a relationship as it varies over facets of the data

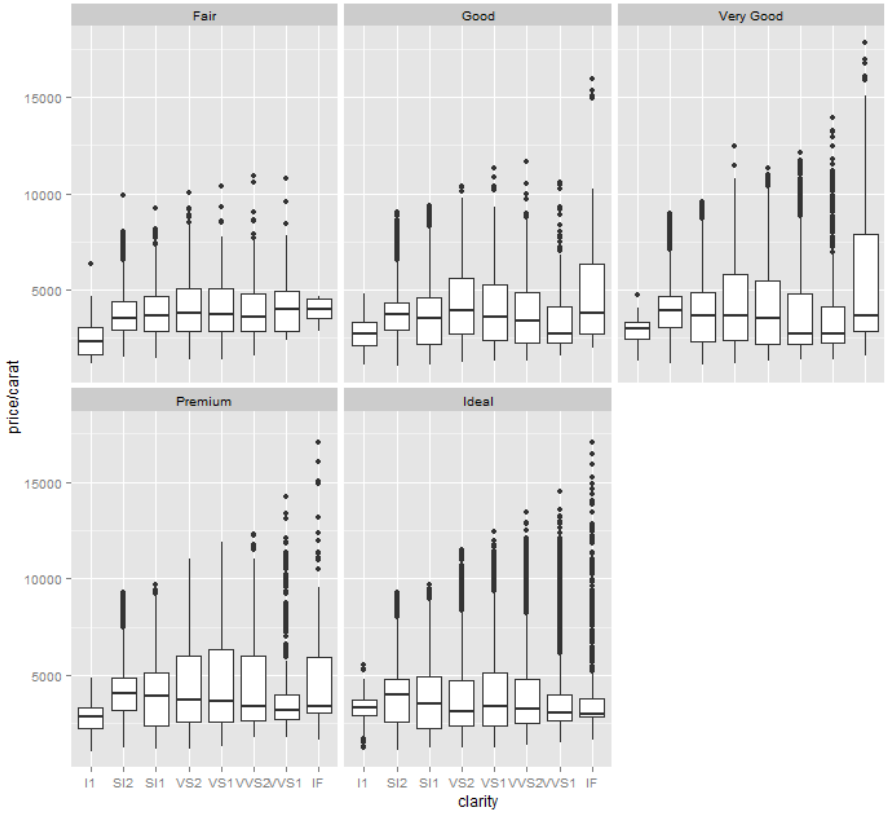
```
qplot(carat, price, data=diamonds, facets=~clarity)
```

One facet:
Diamond clarity

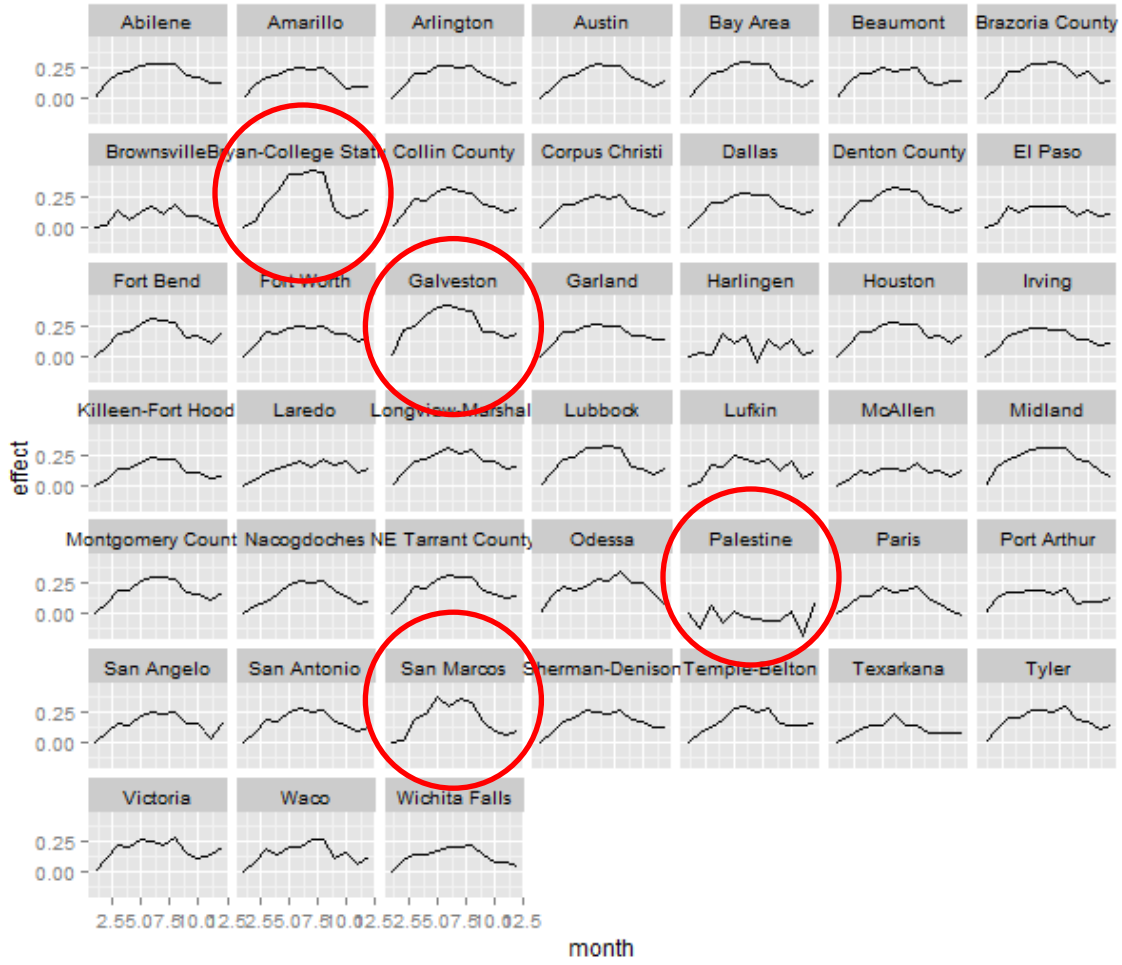


How is price/carat affected by clarity, for each cut type?

```
> qplot(clarity, price/carat, data = diamonds, geom =
"boxplot", facets=~cut)
```



What are the outliers? Use facets

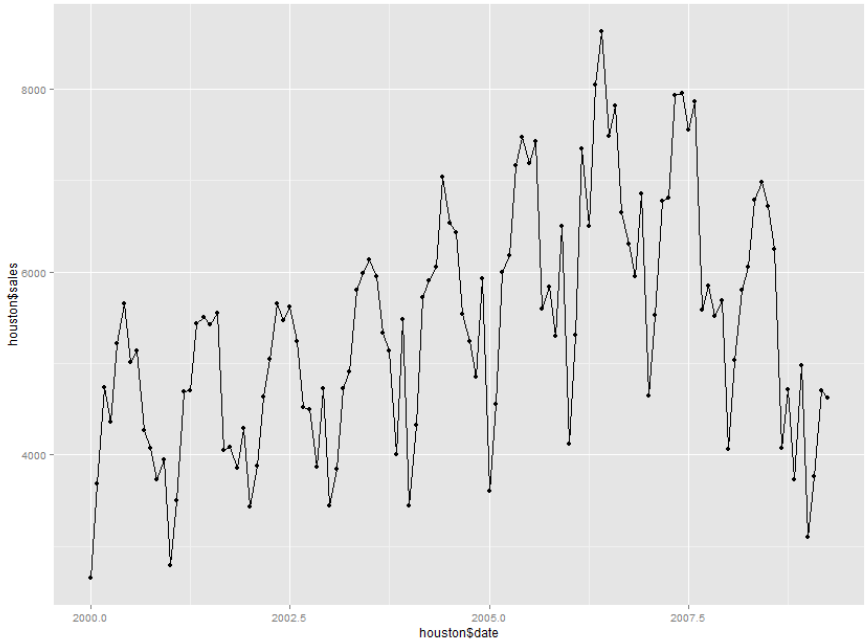



```
qplot(month, effect, data = coef2, group = city, geom = "line") + facet_wrap(~city)
```

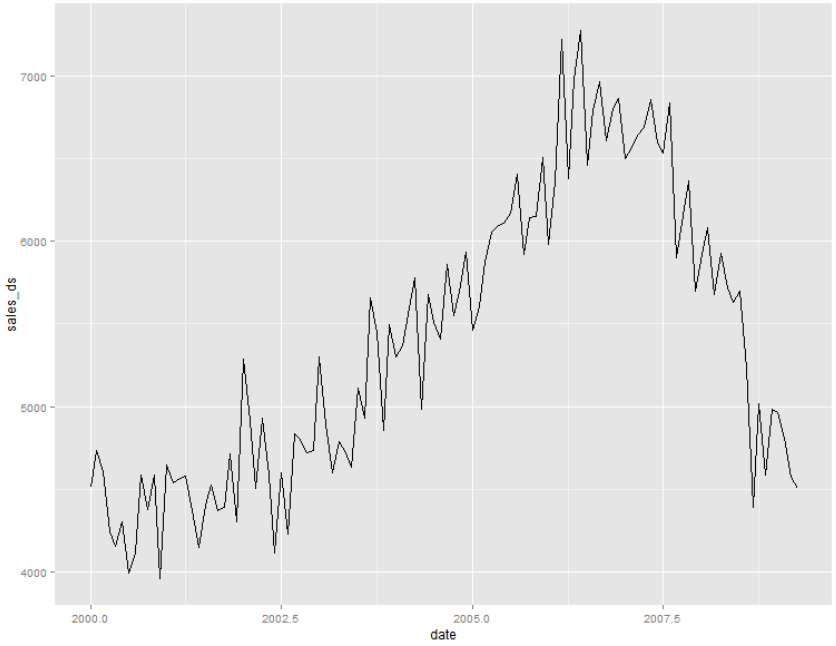
Map data attributes to graph aesthetics to visualize differences in groups


```
qplot(carat, price, data=diamonds, colour=color)
```

Remove short term variation and trends to see longer term trends more clearly



Original sales price data



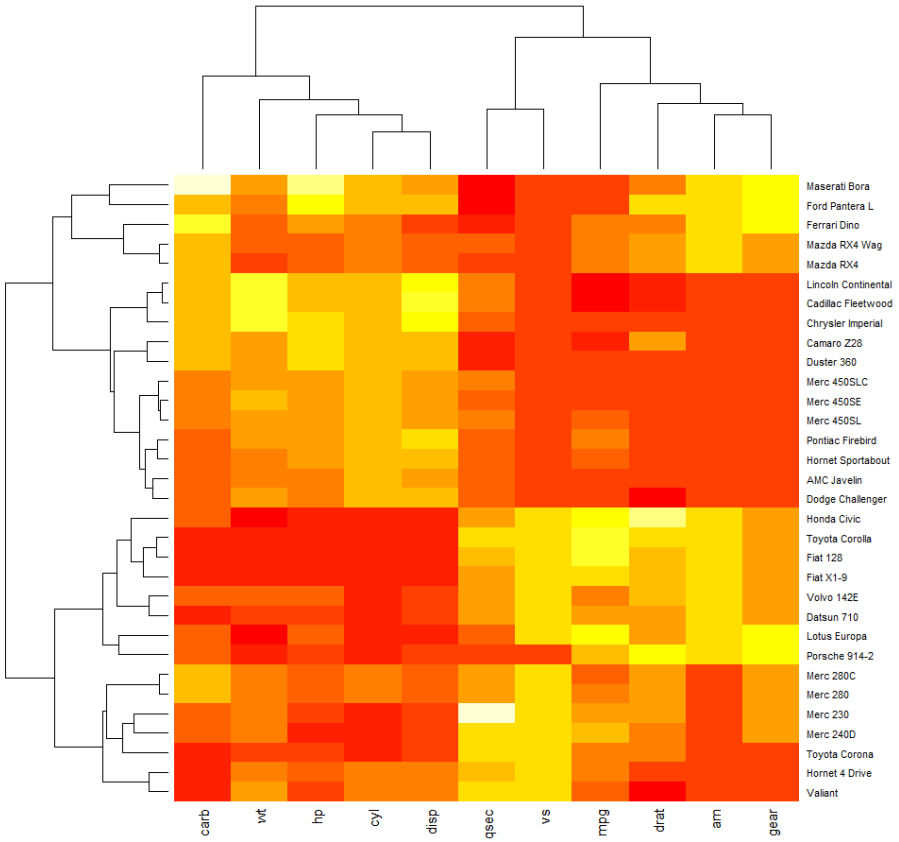
De-seasonalized sales price data

Use clustering to group records (rows)…or group features (columns)

```
> mtscaled <- as.matrix(scale(mtcars
> heatmap(mtscaled, Colv=F,
scale='none')
```

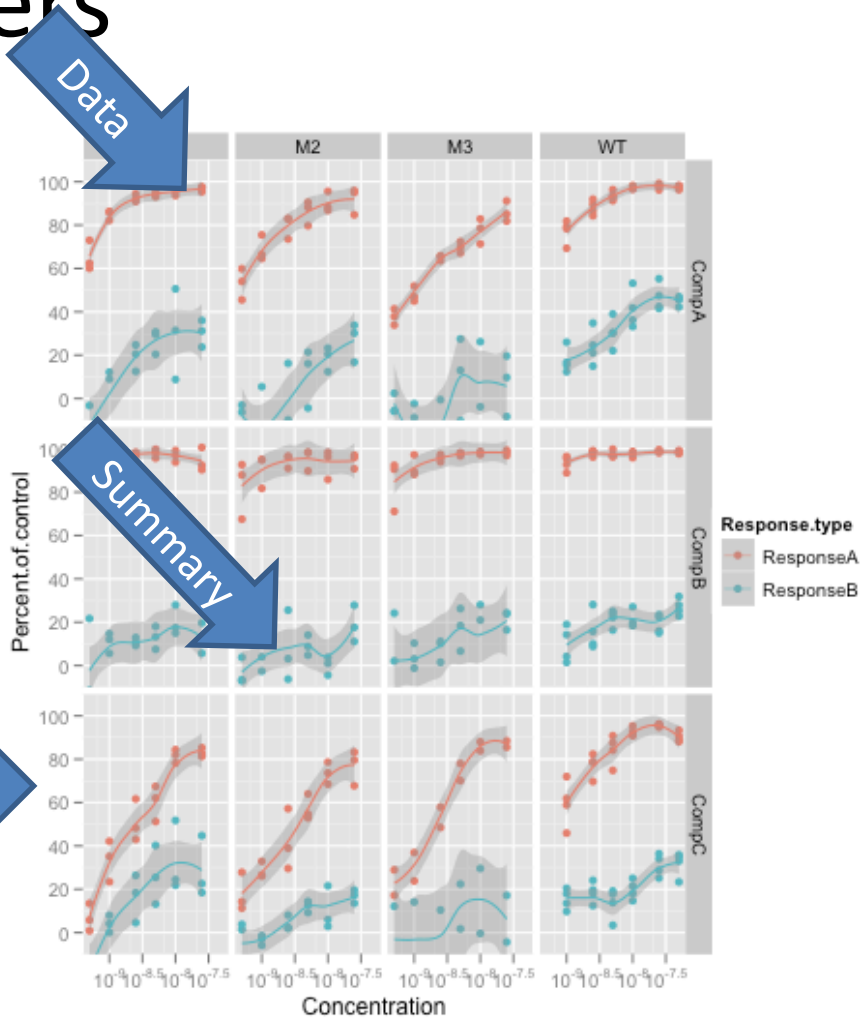
Clustering columns:

- Some info is highly correlated
- e.g. displacement, hp, # cylinders very similar



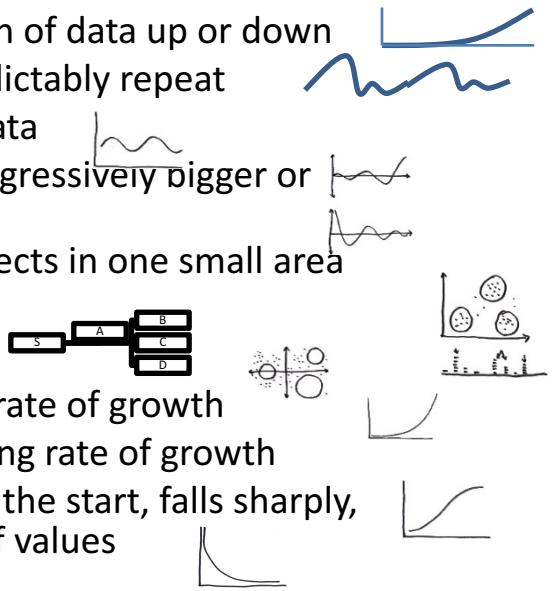
Show differences from the norm with layers

1. Display data (points)
2. Display statistical summary of the data (trend)
3. Additional metadata, context, annotations
   - Facets
   - e.g. map as geospatial background layer
   - Highlight or label important



What can we expect to see in our tour of exploratory data analysis..?

- Trends: the gradual, general progression of data up or down
- Repetitions: a series of values that predictably repeat
- Cycles: a regularly recurring series of data
- Feedback systems: a cycle that gets progressively bigger or smaller because of some influence
- Clusters: a concentration of data or objects in one small area
- Pathways: a sequential pattern of data
- Gaps: an area devoid of observations
- Exponential growth: rapidly increasing rate of growth
- Diminishing returns: there is a decreasing rate of growth
- Long tail: a pattern that rises steeply at the start, falls sharply, and then levels off over a large range of values



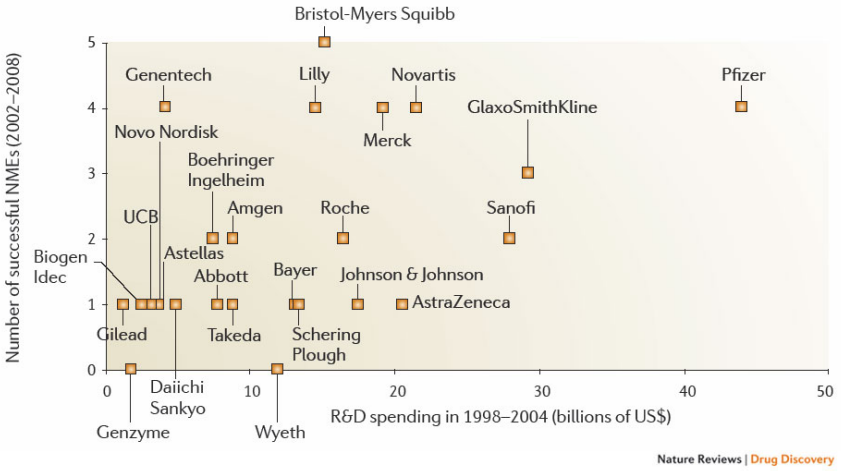
Source: http://uxmatters.com/mt/archives/2009/02/patterns-in-ux-research.php

How is exploration done?

- Use tools to get different points of view…
  - in order to find a simpler view
- Many things you try might not work!
- There's no single right answer.
  - There are many ways to approach a body of data
  - Some better than others
- NOT finding any patterns one level deeper can still be an important finding!

How do we find interesting patterns?

- Visualization tools + human pattern recognition (first part of course)
  - a.k.a. “eyeballing”
- Automatic methods (later in course)
  - Statistical information theory, originally used for compressing signals
  - Machine learning



A warning for the future about 'big data'

Classical database:

Name	Height	Weight	Income	Age
Bruce A.				
Kevin C.				
Amy Z.		<data>		
Jerry R.				
Leopold L.				

A few thousand records

A warning for the future about 'big data'

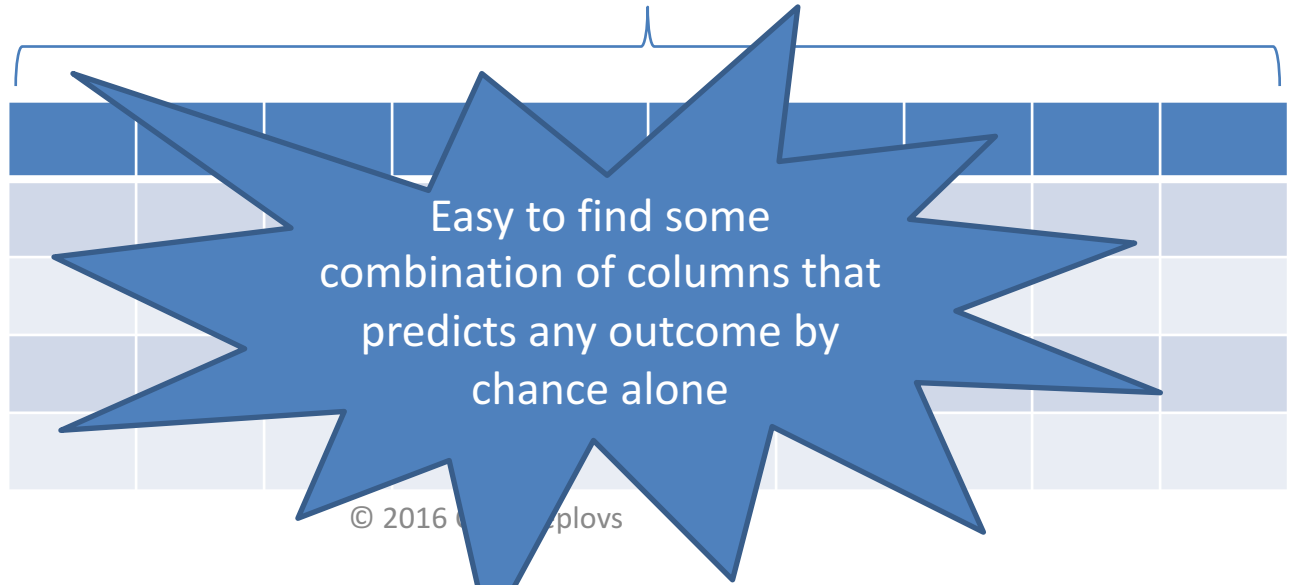
Classical database:

Name	Height	Weight	Income	Age
Bruce A.				
Kevin C.				
Amy Z.		<data>		
Jerry R.				
Leopold L.				

A few thousand records

Modern database:

Thousands of columns



Millions of records

Overfitting

'So it’s like having billions of monkeys typing. One of them will write Shakespeare.'

- Michael Jordan (Machine Learning)

What you'll learn in SI 618:
How to conduct and present an exploratory data analysis

Skills:

- Import messy data into a form that can be analyzed in R
- Compute and visualize a dataset's key summary statistics
- Explore relationships between variables
- Find trends over time
- Discover clusters and outliers
- Use factors to analyze underlying variables in data
- Produce polished presentations for publication/display
- How to apply R coding and packages to solve the above problems
- .. And much more!

Tools:

- The R language
- RStudio integrated development environment
- RMarkdown authoring tool
- ggplot2 visualization package
- Several other useful R packages

Programming with R

What is R?

- A programming language designed to make statistics and graphics easier.

```
> x <- c(1,2,3,4,5,6)   # Create ordered collection (vector)
> y <- x^2              # Square the elements of x
> print(y)              # print (vector) y
[1]  1  4  9 16 25 36
> mean(y)        # Calculate average (arithmetic mean) of (vector) y; result is scalar
[1] 15.16667
> var(y)                # Calculate sample variance
[1] 178.9667
> lm_1 <- lm(y ~ x)  # Fit a linear regression model "y = f(x)" or "y = B0 + (B1 * x)"
                        # store the results as lm_1
> print(lm_1)           # Print the model from the (linear model object) lm_1

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     -9.333        7.000
```

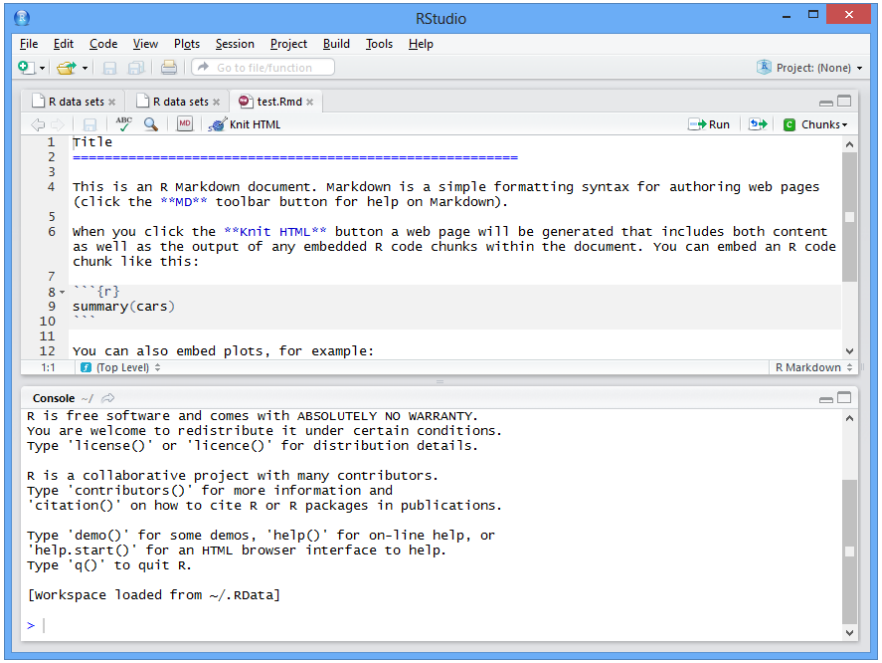
Why R?

- R has powerful data analysis and visualization capabilities
- R is widely used in both academia and industry
  - Political science, statistics, econometrics, finance, sociology, etc.
- R is free and open-source
- R has a very active user community
- Many companies, especially technology startups, are hiring people with R skills
- Why not SAS, SPSS, or Python with scipy, numpy, pandas?
  - Pandas (Python Data Analysis Library) may be a future alternative.
    - But limited modeling abilities: for now R has many more packages.
  - SAS? Proprietary, and oriented toward statisticians.
  - SPSS? Well-suited to survey data analysis
    - But we want more flexibility than menu-driven software provides.

Applications for programming in R

- We'll use Rstudio
- Free, runs on win/mac/linux

http://www.rstudio.com/ide/download/desktop

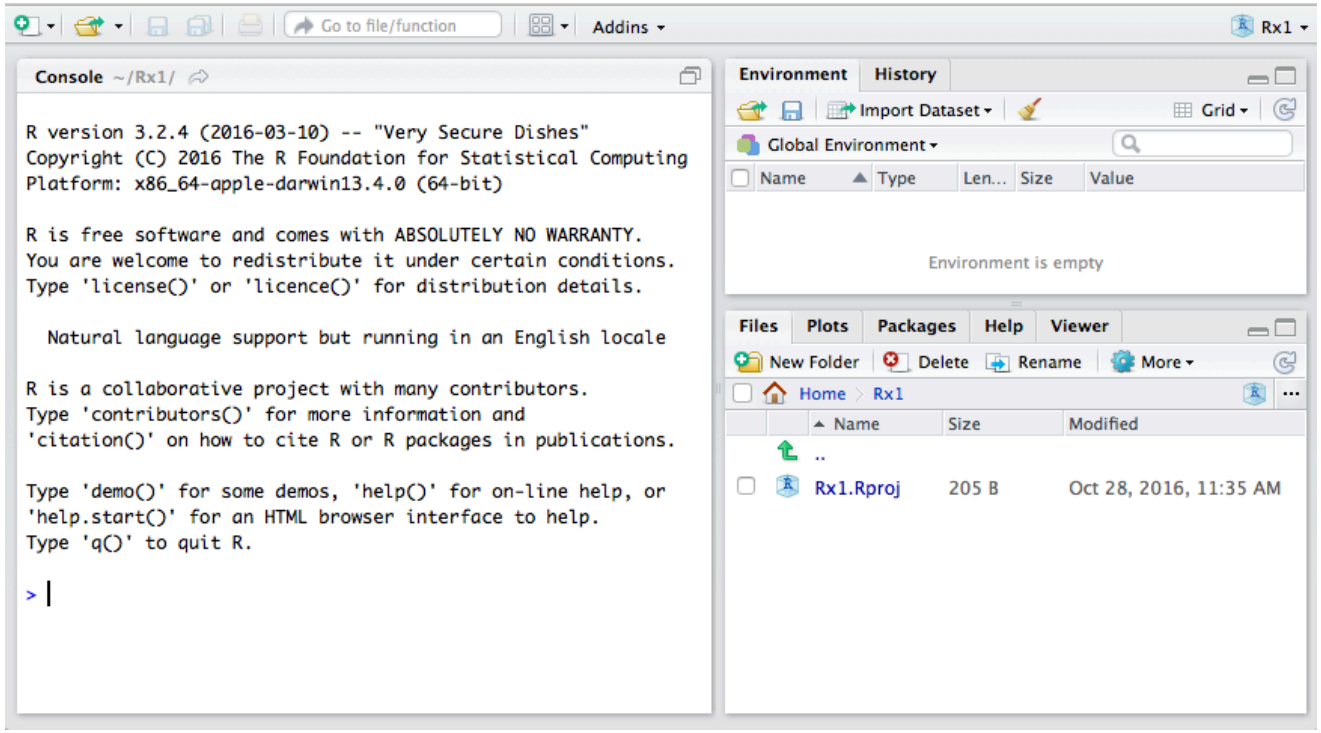


Roadmap to R

- How to use RStudio
- Fundamentals of the R language
- How to read CSV/TSV data into R
- How to create simple plots with R
- How to create reports using R Markdown
- http://en.wikibooks.org/wiki/R_Programming/Sample_Session

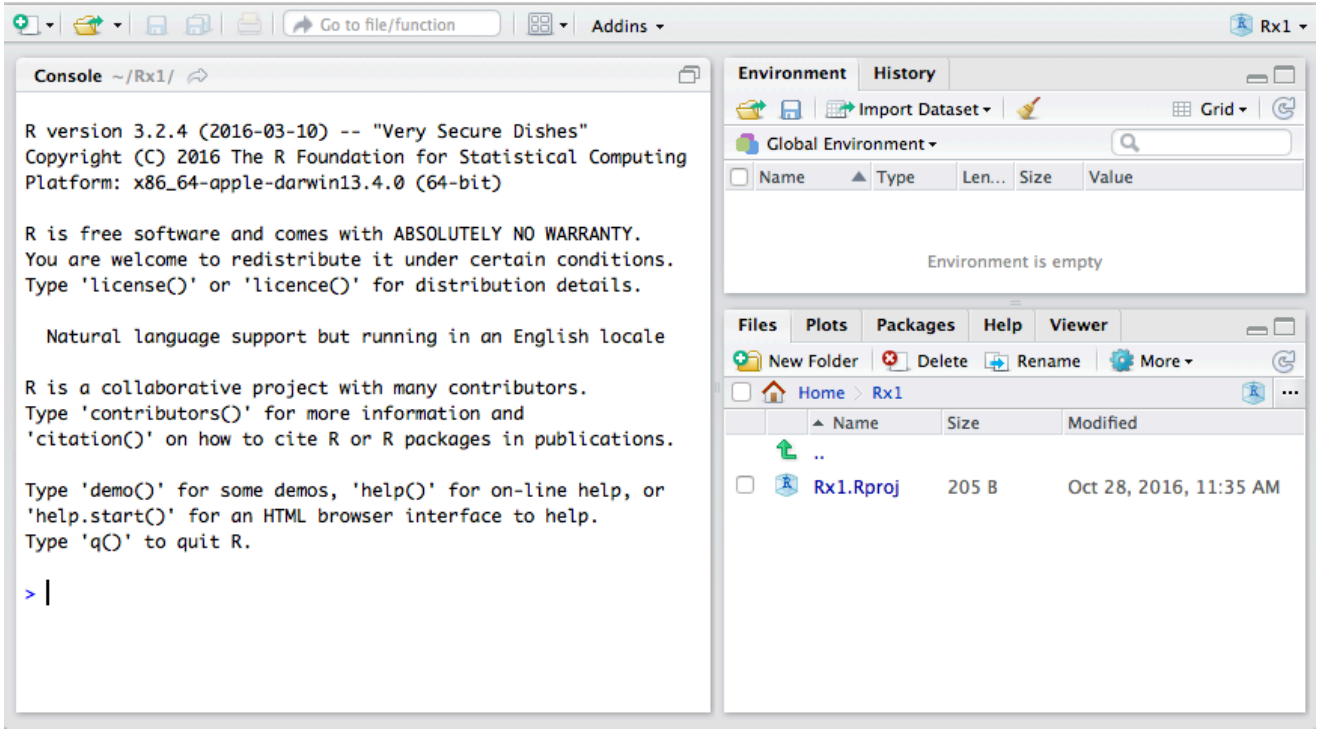
If you have installed R (and possibly Rstudio)

- Feel free to follow along



If you have **not** installed R (and possibly Rstudio)

- Feel free to follow along with someone who has ☺



A sample R program

```
# Sample code: These are comments

# load the R libraries you will use at the start of the script
# The qplot function used below comes from ggplot2
library(ggplot2)

# set the working directory: replace string with your own directory path
# containing the file.
setwd("c:/temp")

# create a data frame to hold the csv table
data <- read.csv("countrydata_test.csv")

# plot the area variable vs population variable
qplot(data$area, data$population)

# group table rows by region, and sum the areas
dataG <- aggregate(data$area, by=list(data$region), FUN=sum)

pie(dataG$x, labels=dataG$Group.1)

# create a new data frame
data_new <- data.frame(country=data$country, log_area=sqrt(data$area),
log_popu=(log(data$population)))

barplot(data_new$log_area, names.arg=data_new$country)
```

Three things that make R different from Python

1. You can use both '<−' and '=' to do assignment.
2. Dot (.) can be used in variable/function names.
3. In R, a vector index starts with 1, not 0.
4. There is no scalar variable in R. The only way to represent a single number in a variable is to use a vector of length one.

For identifier naming rules in R see
http://cran.r-project.org/doc/manuals/R-lang.html#Identifiers

```
> x <- c(1,2,3,4,5,6)    # Create ordered collection (vector)
> y.result <- x^2              # Square the elements of x
> print(y.result)              # print (vector) y
[1]  1  4  9 16 25 36
> mean(y.result)         # Calculate average of (vector) y
[1] 15.16667
```

http://cran.r-project.org/doc/contrib/R_language.pdf

Basic R types

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

- Vector
- List
- Matrix
- Data frame

- An R program is more like a series of *transforms*
- In contrast to Python, loops are rare
  - Because you operate on entire list or matrix at once

Use c to create vectors

- c – concatenate or combine

```
> x = c(1,2,14,29)
> x
[1] 1 2 14 29
> Names <- c("robin", "bluebird", "eagle")
> Names
[1] "robin"    "bluebird" "eagle"
```

A vector is one of the basic R objects.

Vectors must have all elements of the same type
(type is called 'mode' in R: don't ask me why)

Some simple R interactions

```
> 1+1
[1] 2
> a = 2
> a
[1] 2
> a[1]
[1] 2
> v = c(1,2,3)
> v
[1] 1 2 3
> length(v)
[1] 3
```

Names of objects

- In R, you can associate names with vector elements.

```
> prices = c(1.29, 2.50, 0.99)
> names(prices) = c("Banana", "Apple", "Orange")
> prices["Apple"]
Apple
  2.5
```

Adding vectors

Case 1. Vectors are same length

- Just add the individual elements

```
> c(1,2,3) + c(4,5,6)
[1] 5 7 9
```

Case 2. Adding two vectors with different lengths

- Elements of the shorter vector are recycled as often as necessary to create a vector the length of the longer vector.

```
> someNumbers = 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> someNumbers + c(1,2,3)
[1] 2 4 6 5 7 9 8 10 12 11
Warning message: In someNumbers + c(1, 2, 3) :
   longer object length is not a multiple of shorter object
   length
```

More vector math: element-wise operations

```
> v1 = c(1,2,3)
> v2 = c(4,5,6)
> v1/v2
[1] 0.25 0.40 0.50
> v1+v2
[1] 5 7 9
> v1-v2
[1] -3 -3 -3
> v1*v2
[1]  4 10 18
```

Types and type conversion

- Types refer to the underlying storage format
- Types must be one of:

  logical, integer, double, complex, character, or *raw*.
- All elements of a vector must have the same underlying type. (Does not apply to lists.)
  - Type of a vector: type of the elements it contains
- Type conversion functions use naming convention `as.T` which converts its argument to type T.
- `as.integer(3.2)` returns the integer 3
- `as.character(3.2)` returns the string "3.2".

Types and type conversion

```
> v1 = c(10,20,30,40,50)
> typeof(v1)
[1] "double"
> class(v1)
[1] "numeric"
> mode(v1)
[1] "numeric"
> v2 = c("a", "b", "c")
> v2
[1] "a" "b" "c"
> typeof(v2)
[1] "character"
> class(v2)
[1] "character"
> mode(v2)
[1] "character"
> as.character(v1)
[1] "10" "20" "30" "40" "50"
```

Infinity, missing values and NaNs: Inf, -Inf, NaN, NA

```
> 1/0
[1] Inf
> -1/0
[1] -Inf
> 0/0
[1] NaN
> v1 = c("1", "2", "")
> as.integer(v1)
[1]  1  2 NA
> is.nan(0/0)
[1] TRUE
> v2 =as.integer(v1)
> is.na(v2[3])
[1] TRUE
> sum(as.integer(v1))
[1] NA
> sum(as.integer(v1), na.rm=TRUE)
[1] 3
> a <- c(1, 2)
> a[3]
[1] NA
```

Boolean operators

- You can input T or TRUE for true values and F or FALSE for false values.
- The operators & and | apply element-wise on vectors.
- The operators && and || are often used in conditional statements.

```
> 1 == 1
[1] TRUE
> 1 == 2
[1] FALSE
> a = TRUE
> a
[1] TRUE
> (1 == 1) && (2 >= 3)
[1] FALSE
> (1 == 1) || (2 >= 3)
[1] TRUE
```

Lists

- Lists are like vectors, except elements do not need to have the same type.
- Lists are created using the list() function.
- Single elements can be accessed by position using double brackets `[[...]]`.
- Indexing by vectors can be done using single brackets `[...]`
- Named elements may be accessed either by position or by name.

```
> a = list(1, 2, "hello")
> a
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] "hello"

> a[[1]]
[1] 1

> a[1:2]
[[1]]
[1] 1

[[2]]
[1] 2

> a[[1:2]]
Error in a[[1:2]] : subscript out of bounds
```

Use the $ operator to access named elements in lists

```
> a[[1]]
[1] 1
> a[[3]]
[1] "hello"
> a <- list(name="Joe", 1, goodies=c(3,4,5))
> a$name
[1] "Joe"
> a$goodies
[1] 3 4 5
```

Sorting vectors with order()

```
> a <- c(45, 50, 10, 96)
> order(a)
[1] 3 1 2 4
> a[order(a)]
[1] 10 45 50 96
> a[order(a, decreasing=T)]
[1] 96 50 45 10
```

Matrices

- You can **edit** the dimension of a vector, making it a matrix.
- By default, R fills matrices by column.
- To fill by row instead, add the argument byrow = TRUE to the call to the matrix() function.

```
> v = c(1,2,3,4,5,6)
> dim(v) = c(2,3)
> v
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m <- array( c(1,2,3,4,5,6), dim=c(2,3) )
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> A = matrix(c(1,2,3,4,5,6), nrow=2, ncol=3, byrow = TRUE)
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Building a matrix using cbind and rbind

```
# 3 column matrix with column dimnames
x <- cbind(Index = c(1:3), Age = c(30, 45, 34),
                   Salary = c(500, 600, 550))

# Combining two named vectors
mil1 <- c(c=3,d=4,e=5)
mil2 <- c(f=6,g=7,h=8)

> cbind(mil1, mil2)
  mil1 mil2
c    3    6
d    4    7
e    5    8
> cbind(mil2, mil1)
  mil2 mil1
f    6    3
g    7    4
h    8    5
> rbind(mil1, mil2)
     c d e
mil1 3 4 5
mil2 6 7 8
> rbind(mil2, mil1)
     f g h
mil2 6 7 8
mil1 3 4 5
```

Index	Age	Salary
1	30	500
2	45	600
3	34	550

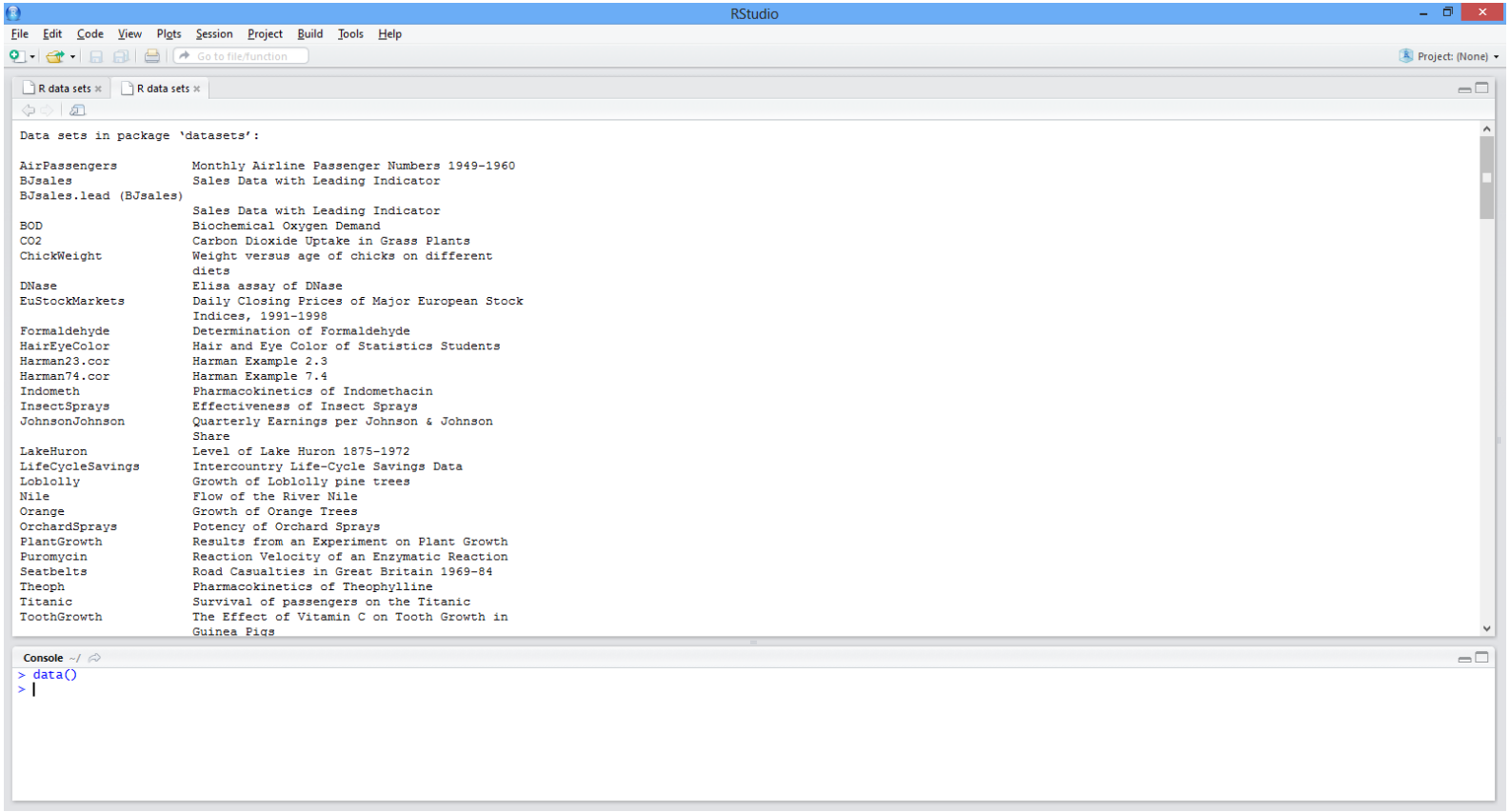
data.frame
The most important structure for storing data in R

- rbind and cbind can't combine vectors with different types
  - e.g. numeric and string
- Data frames are a powerful way to keep related variables together in a package

```
> df <- data.frame(a = c(1:5),
+                  b = c("cat", "dog", "emu", "fish", "giraffe"))
> df
  a       b
1 1     cat
2 2     dog
3 3     emu
4 4    fish
5 5 giraffe
> df$a
[1] 1 2 3 4 5
> df$b
[1] cat     dog     emu     fish    giraffe
Levels: cat dog emu fish giraffe
```

R comes with dozens of built-in data.frames

- Type data() to see built-in datasets



Example: mtcars

```
> head(mtcars)
                   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
```

Creating data.frames by importing data using read.table() and read.csv()

Tab-separated format, tabular data:

```
data <- read.table("yourfile.txt", sep="\t",
       col.names=c("length", "deviation", "file"))
```

Comma-separated format file:

```
data = read.csv("spam.csv", header = TRUE)
```

Handy tip: after reading a file for the first time, check that it has the correct column names using `names(data)`

Get information about a variable's structure with str()

```
> str(mtcars)
'data.frame':  32 obs. of  11 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 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

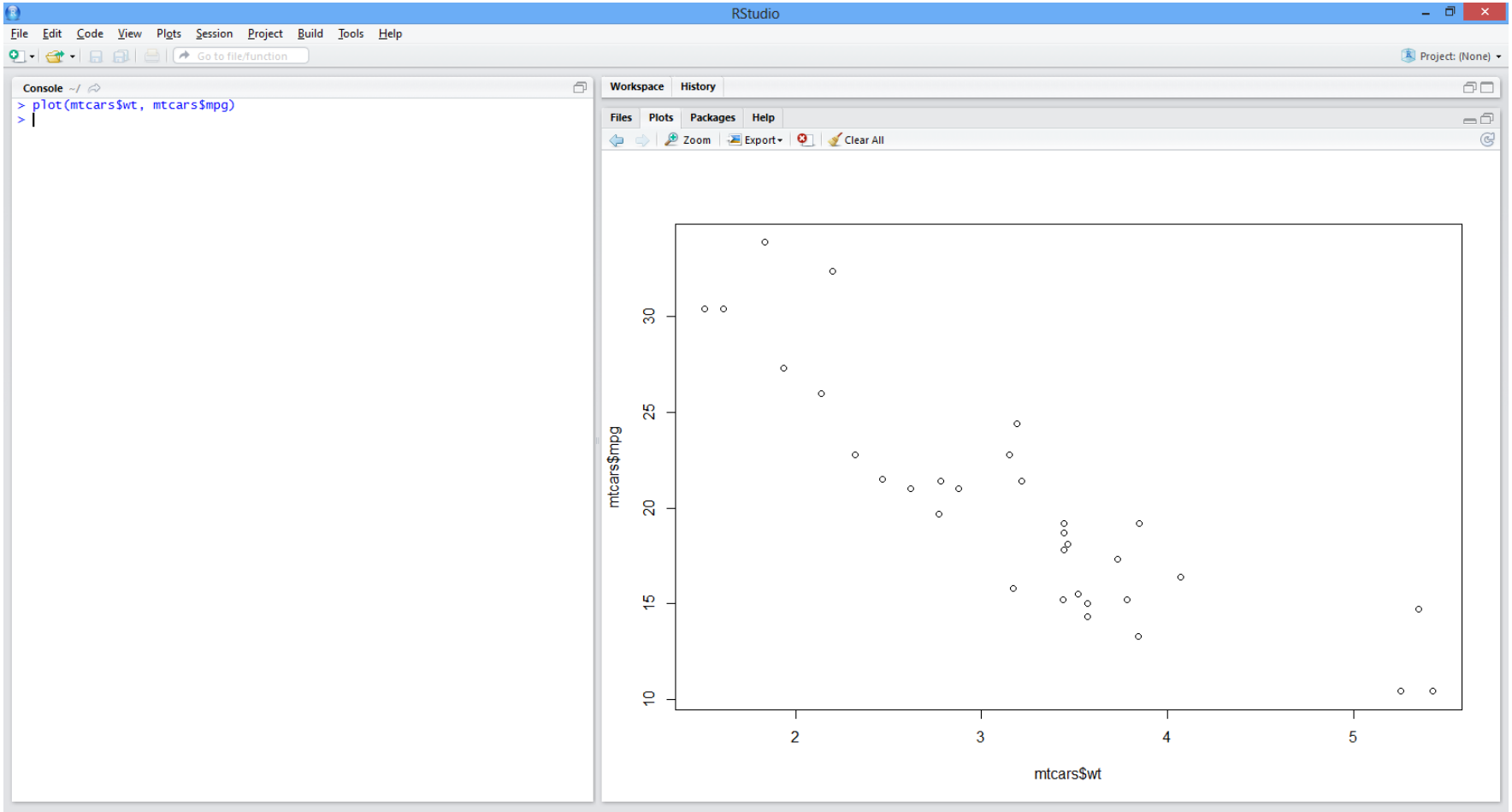
aggregate(): group rows

- Like GROUP BY in SQL

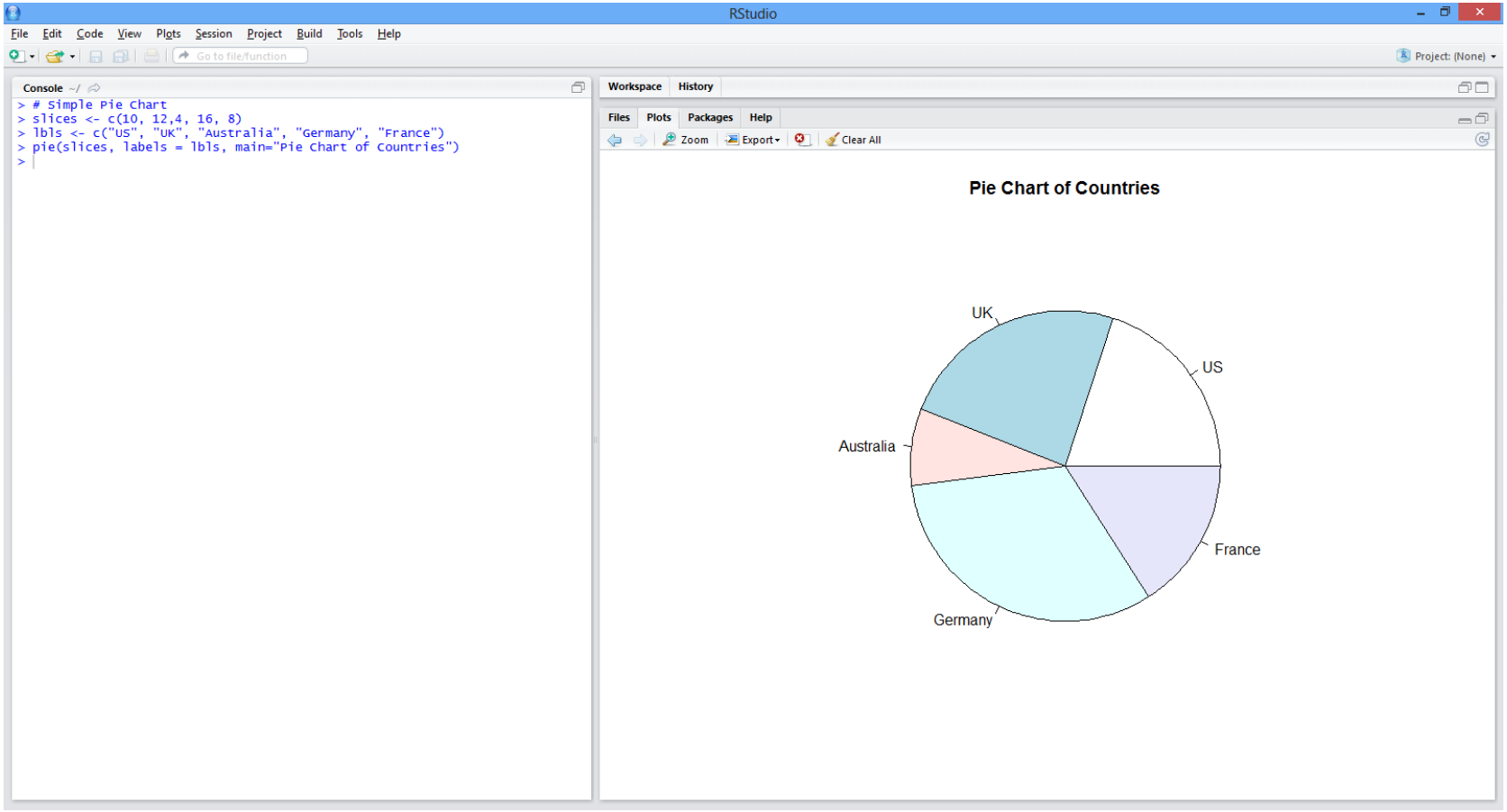
```
> aggregate(mtcars$mpg, list(cyl = mtcars$cyl), mean )
  cyl        x
1   4 26.66364
2   6 19.74286
3   8 15.10000
> aggregate(mtcars$qsec, list(cyl = mtcars$cyl), mean )
  cyl        x
1   4 19.13727
2   6 17.97714
3   8 16.77214
```

- Data frames can be combined with cbind and rbind
  - Column-wise: # rows must match, row names ignored
  - Row-wise: number and names of columns must match

plot()

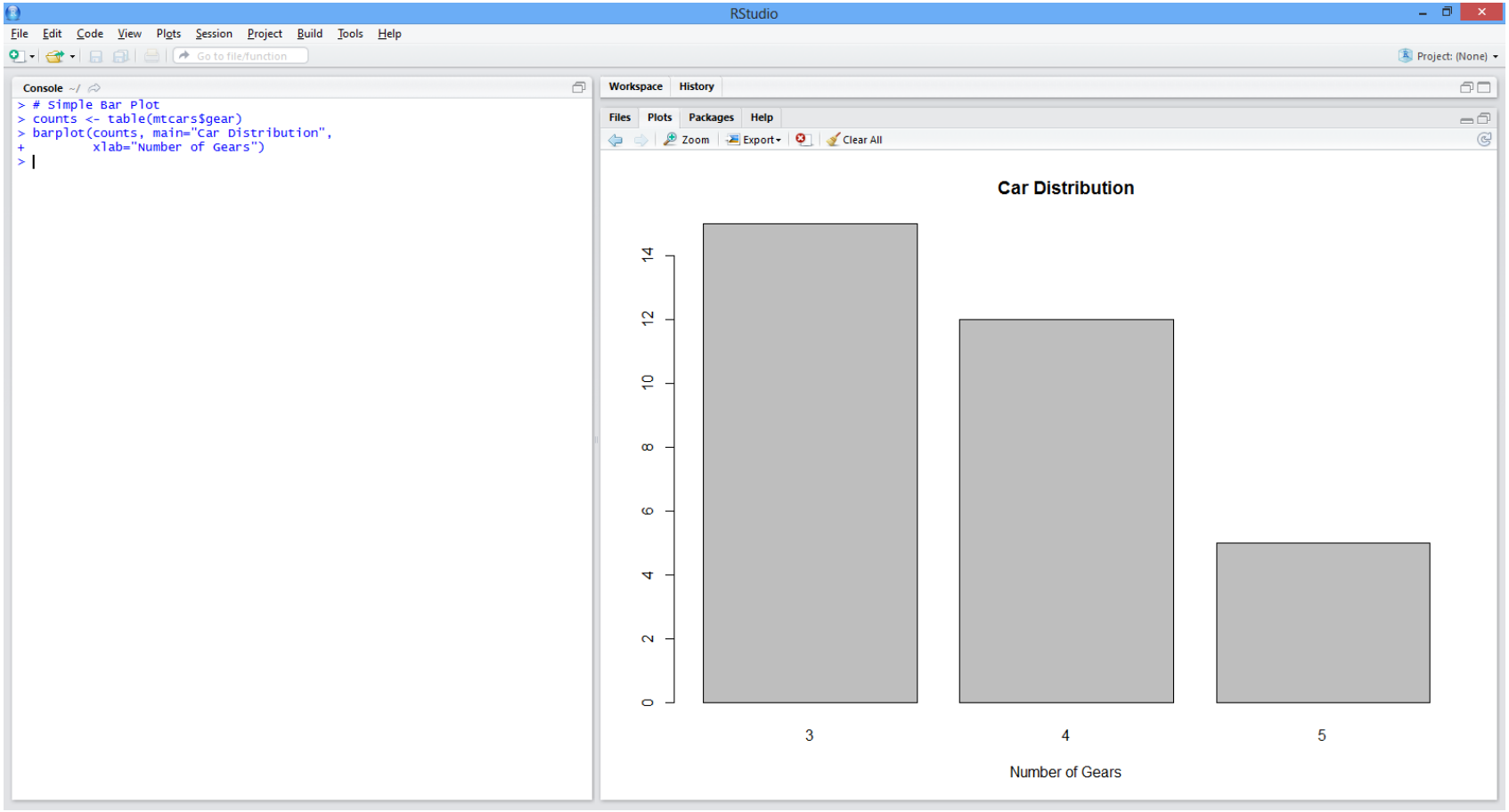


pie()



Source: http://www.statmethods.net/graphs/pie.html

barplot()



Source: http://www.statmethods.net/graphs/bar.html

Review reading and reference

- ComputerWorld Beginner's Guide
- http://bit.ly/11GQtLe
- Beginner's Guide to R (pdf)
- http://bit.ly/16DVFZN
- R tutorial http://www.cyclismo.org/tutorial/R/

R authoring tools

Report authoring tools that allow you to embed R code and results

R Markdown (this course)

http://www.rstudio.com/ide/docs/r_markdown

- Uses Markdown: a plain text format for web content

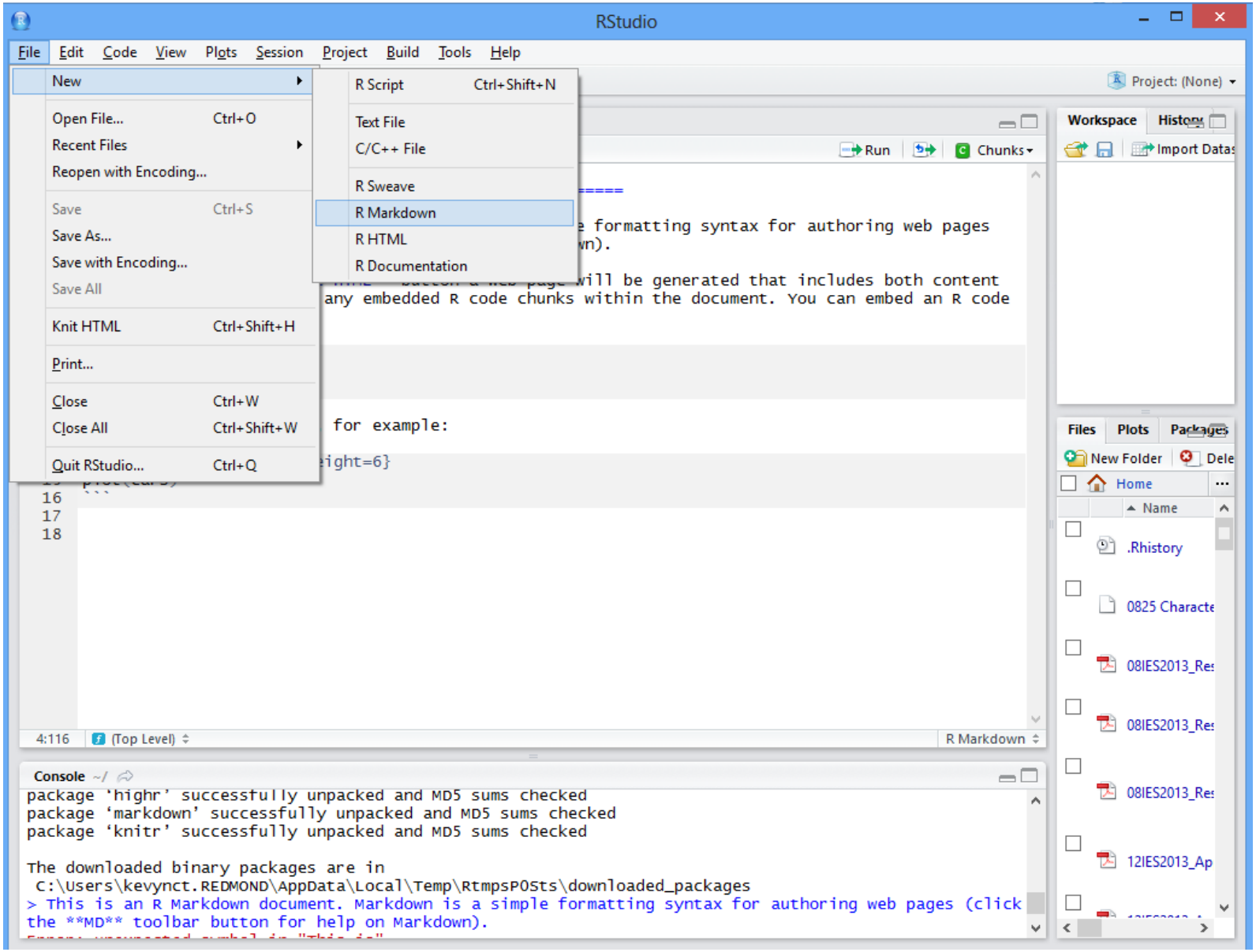
SWeave:

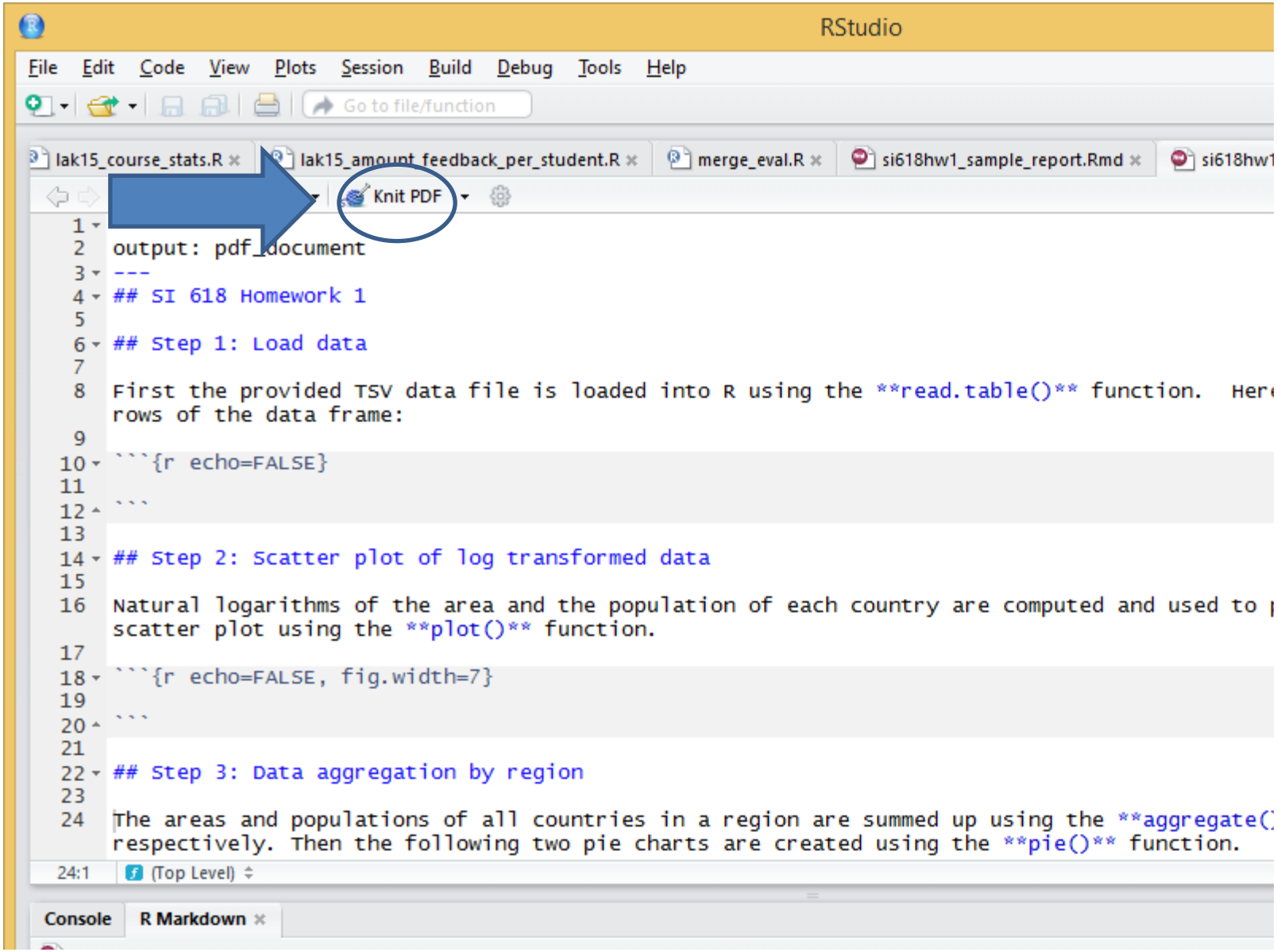
- Uses LaTeX : a tag-based formatting language
- Part of every R installation
- `help("Sweave", package="utils")`
- http://www.statistik.lmu.de/~leisch/Sweave/

R Markdown Demo

- R Markdown enables easy authoring of reproducible reports from R. It will be used in all lab/homework.
- Before using Rmarkdown you need to first install the "knitr" package. From Rstudio:
  ```
  > install.packages("knitr")
  ```
- Then you can use File/New/R Markdown document.

- http://www.rstudio.com/ide/docs/authoring/using_markdown
- https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet





RStudio: Preview HTML

Preview: ~/test.html | Log | Save As | Publish | Find

Title

This is an R Markdown document. Markdown is a simple formatting syntax for authoring web pages (click the **MD** toolbar button for help on Markdown).

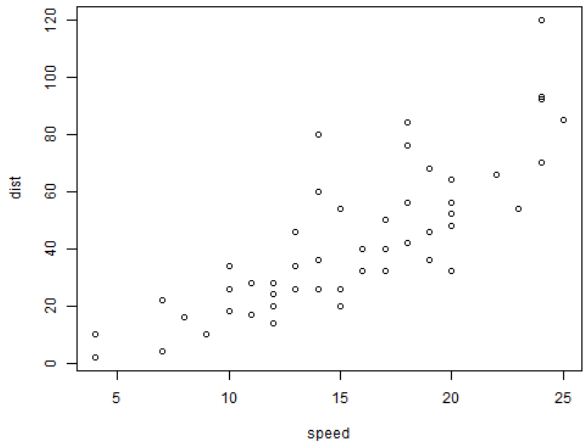
When you click the **Knit HTML** button a web page will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

You can also embed plots, for example:

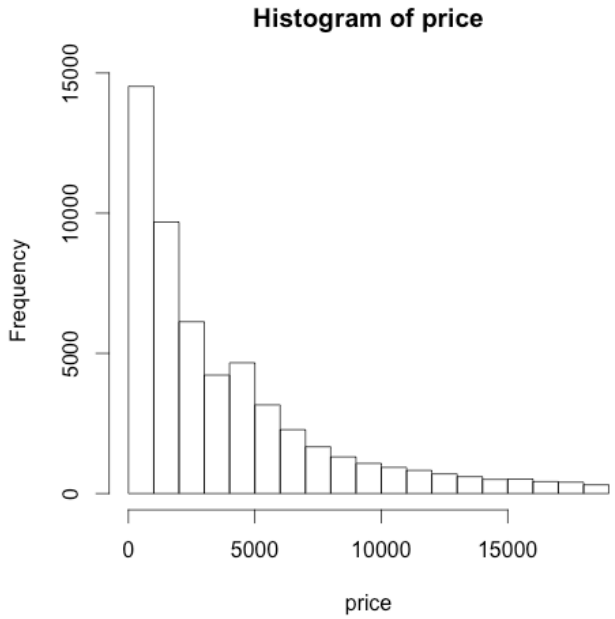
```
plot(cars)
```



ggplot2

ggplot2 basics

```
> install.packages("ggplot2")
> library(ggplot2)
> data(diamonds)
> head(diamonds)
  carat price color clarity
1    0.23   326     E     SI2
2    0.21   326     E     SI1
3    0.23   327     E     VS1
4    0.29   334     I     VS2
5    0.31   335     J     SI2
6    0.24   336     J    VVS2
7    0.24   336     I    VVS1
8    0.26   337     H     SI1
9    0.22   337     E     VS2
10   0.23   338     H     VS1
> with(diamonds, hist(price))
```



Source: http://docs.ggplot2.org/current/

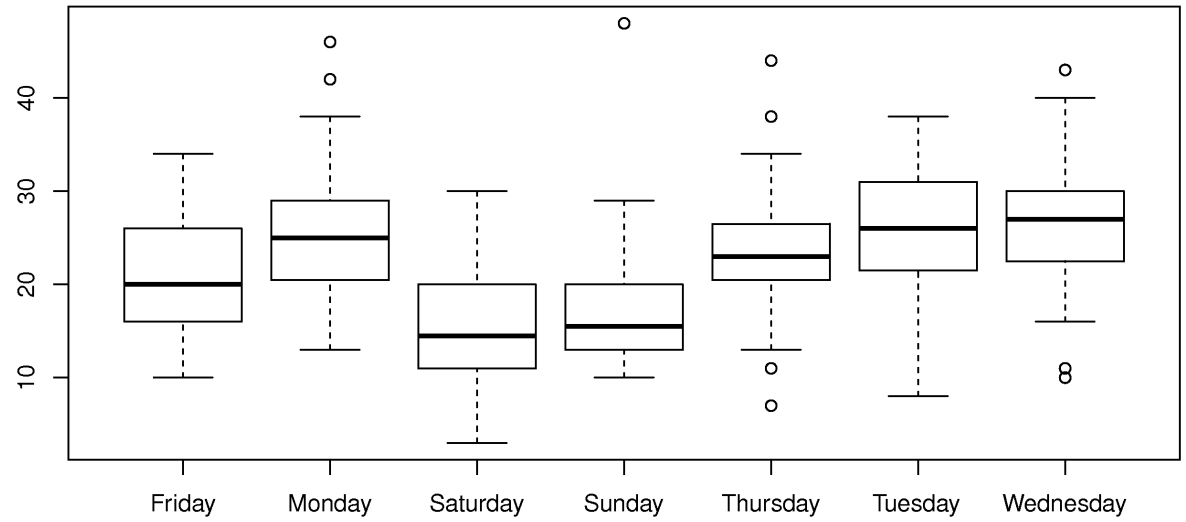
Supplemental: http://gettinggeneticsdone.blogspot.com/2010/01/ggplot2-tutorial-scatterplots-in-series.html

Datasets that come with ggplot2

`diamonds`
Prices of 50,000 round cut diamonds
`economics`
US economic time series.
`midwest`
Midwest demographics.
`movies`
Movie information and user ratings from IMDB.com.
`mpg`
Fuel economy data from 1999 and 2008 for 38 popular models of car
`msleep`
An updated and expanded version of the mammals sleep dataset.
`presidential`
Terms of 10 presidents from Eisenhower to Bush W.
`seals`
Vector field of seal movements.

Next week sneak peek

- You'll learn how to make and read these important boxplot summary graphics:



Lab time

- If you took 601
  - And haven't yet filled out the course evaluation
  - Please do this today…
- R installation
- Programming exercises in R
- Starting on homework 1

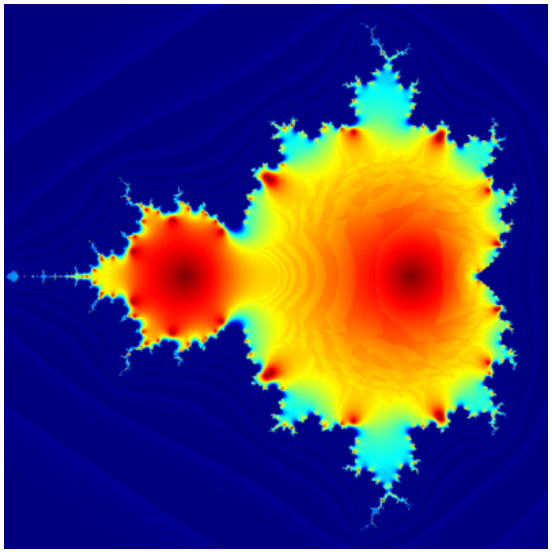
RStudio Usage Demo

- RStudio is a nice integrated user interface to R
- Main functionality:
  - How to edit & run R code
  - How to change working directory
  - How to load data into workspace
  - How to clear variables in workspace
  - How to find help on R function usage
  - How to manage R packages

Fun Extras

What is this an image of?

```
library(caTools)          # external package providing write.gif function
jet.colors <- colorRampPalette(c("#00007F", "blue", "#007FFF", "cyan", "#7FFF7F",
                                 "yellow", "#FF7F00", "red", "#7F0000"))
m <- 1200                 # define size
C <- complex( real=rep(seq(-1.8,0.6, length.out=m), each=m ),
              imag=rep(seq(-1.2,1.2, length.out=m), m ) )
C <- matrix(C,m,m)        # reshape as square matrix of complex numbers
Z <- 0                    # initialize Z to zero
X <- array(0, c(m,m,20)) # initialize output 3D array
for (k in 1:20) {         # loop with 20 iterations
  Z <- Z^2+C              # the central difference equation
  X[,,k] <- exp(-abs(Z)) # capture results
}
write.gif(X, "M.gif", col=jet.colors, delay=1000)
```



What's the difference between = and <- ?

If you use <- then you assign a variable that you will be able to use in your current environment.

In a function call you can't assign an object with = because = means assigning arguments there.

Consider:

```
matrix(1,nrow=2)
```

This just makes a 2 row matrix. Now consider:

```
matrix(1,nrow <- 2)
```

This also gives you a two-row matrix, but now we also have an object called nrow which evaluates to 2!

What happened is that in the second use we didn't assign the argument nrow 2, we assigned an object nrow 2 and send that to the second argument of matrix, which happens to be nrow.

R objects have basic properties: mode and length

- Mode: the basic type of object's fundamental elements
  - *numeric*: numbers (internally, double-precision real)
  - *complex*: complex number
  - *logical*: possible values TRUE, FALSE, and NA
  - *character*: string
  - *raw:* byte array
- Length: the number of elements in the object
- Empty objects can still have a mode:
  - Empty numeric vector: numeric(0)
  - Empty character vector: character(0)

R objects also belong to classes (in the object-oriented sense)

- Class of a vector: the mode of the object
- Other classes:
  - "matrix", "array", "factor" and "data.frame"
- class controls how certain R functions get applied to an object
  - plot()
  - summarize() etc.

Creating sequences

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1,10,2)
[1] 1 3 5 7 9
> seq(1,10,length=5)
[1]  1.00  3.25  5.50  7.75 10.00
> v1 = c(10,20,30,40,50)
> v1[1:3]
[1] 10 20 30
```

Creating repetitions

```
> a <- seq(from = 1, to = 4, by = 1)
> a
[1] 1 2 3 4
> rep(a, 8)
 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1
2 3 4 1 2 3 4 1 2 3 4
> rep(a, each = 8)
[1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3
3 3 4 4 4 4 4 4 4 4
> Names <- c("robin", "bluebird", "eagle")
> rep(Names, each = 4)
 [1] "robin"    "robin"    "robin"    "robin"
  "bluebird" "bluebird" "bluebird" "bluebird"
  "eagle"    "eagle"    "eagle"    "eagle"
```

ggobi (optional)

An interactive visualization toolkit

First install ggobi
http://www.ggobi.org/downloads/

Then from R:

```
install.packages("rggobi")
```

http://www.ggobi.org/rggobi/introduction.pdf

ggmap (optional)

http://cran.r-project.org/web/packages/ggmap/ggmap.pdf

```
install.packages("ggmap")
library(ggmap)

um <- get_map('university of michigan')
ggmap(um)
```

```
um <- get_map('university of michigan')
```