

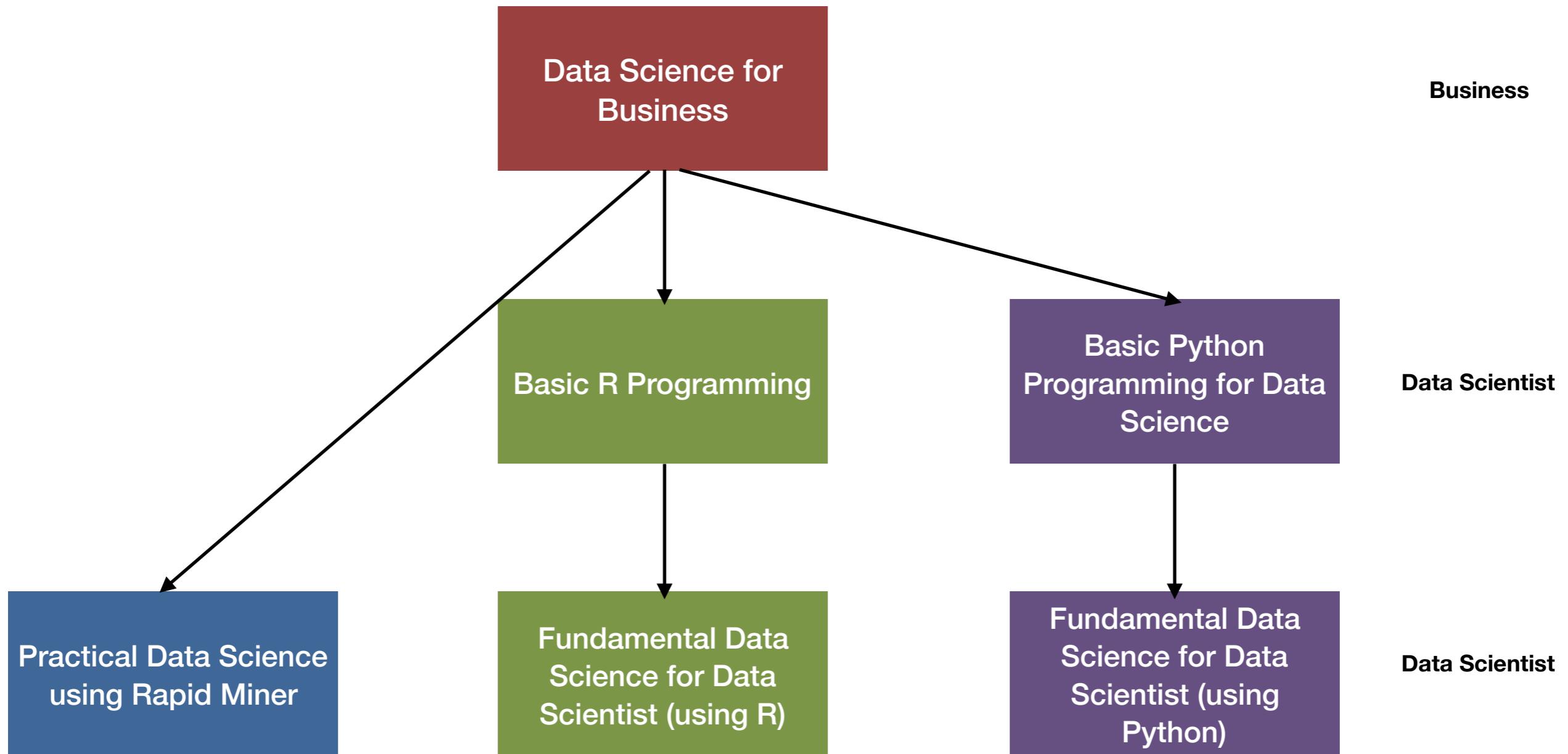
# BASIC R PROGRAMMING

VEERASAK KRITSANAPRAPHAN - SOFTWARE PARK THAILAND



# Data Science Series

Target



# Agenda

## Day 1

- Basics of R
- Basics of RStudio
- Basic data types and operations: numbers, characters and composites
- Vectors, creating sequences, common functions
- Importing Data
- Manipulating data
- Lists
- Functions

## Day 2

- Data Wrangling
- Loops
- Apply functions
- Data summaries
- Visualization on Standard Graphics

## Day 3

- Visualization on ggplot2
- Hypothesis Testing
- Anova
- Linear Regression

# About myself?

- Graduated from San Francisco State University in Master of Science in Computer Information System, 1997
- PhD. Candidate at Chulalongkorn University, research focus on Data Science, Big Data, Mobile Computing and Internet of Thing (IOT)
- Assistant Chief Information Officer at Bangkok Hospital
- Instructor for Software Park in Data Science, Requirement Discovery and Practical Enterprise Integration
- Instructor for Chulalongkorn University in Data Mining

# Introduce yourself?

- What is your name/position/role in your organization?
- Tell you a bit about yourself?
- What is your passion?
- What is your expectation for this class?



# Slide and Sample Data

<https://github.com/vkrit/r-programming>



# What is R?

- R is a system for statistical computation and graphics.
- It is heavily influenced by the **S** language
- **R** was initially written by **Ross Ihaka** and **Robert Gentleman** at the Department of Statistics of the University of Auckland in Auckland, New Zealand.
- The “**R Core Team**” maintain the source code for the software and release regular updates

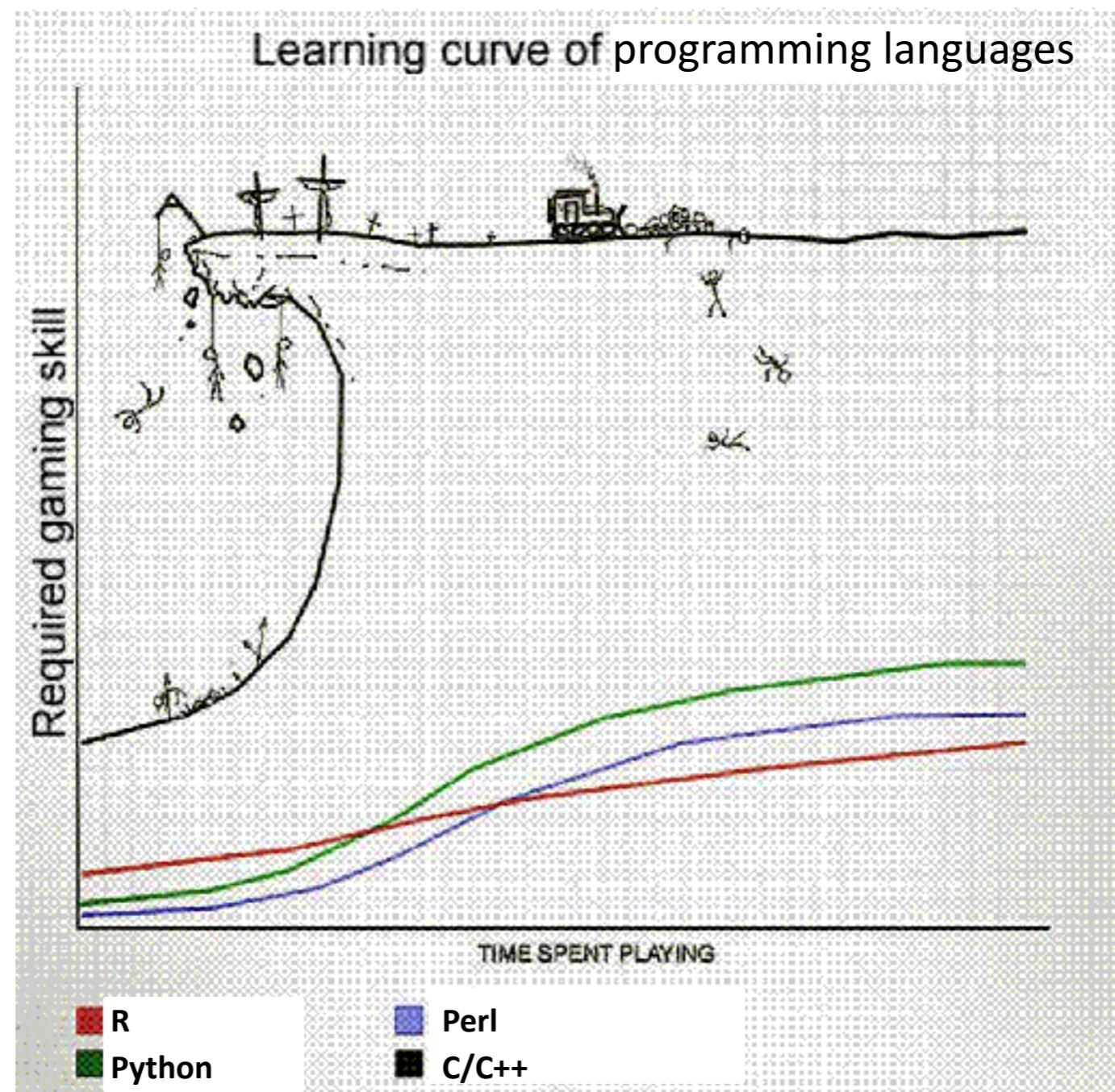
# What is R?

- In addition, the R project is added to by many of its users, who write source code for many different types of analytical procedures
- Everything from analytical chemistry to epidemiology to linguistics
- Currently 10,000+ different user--written libraries available (<http://cran.r-project.org>)

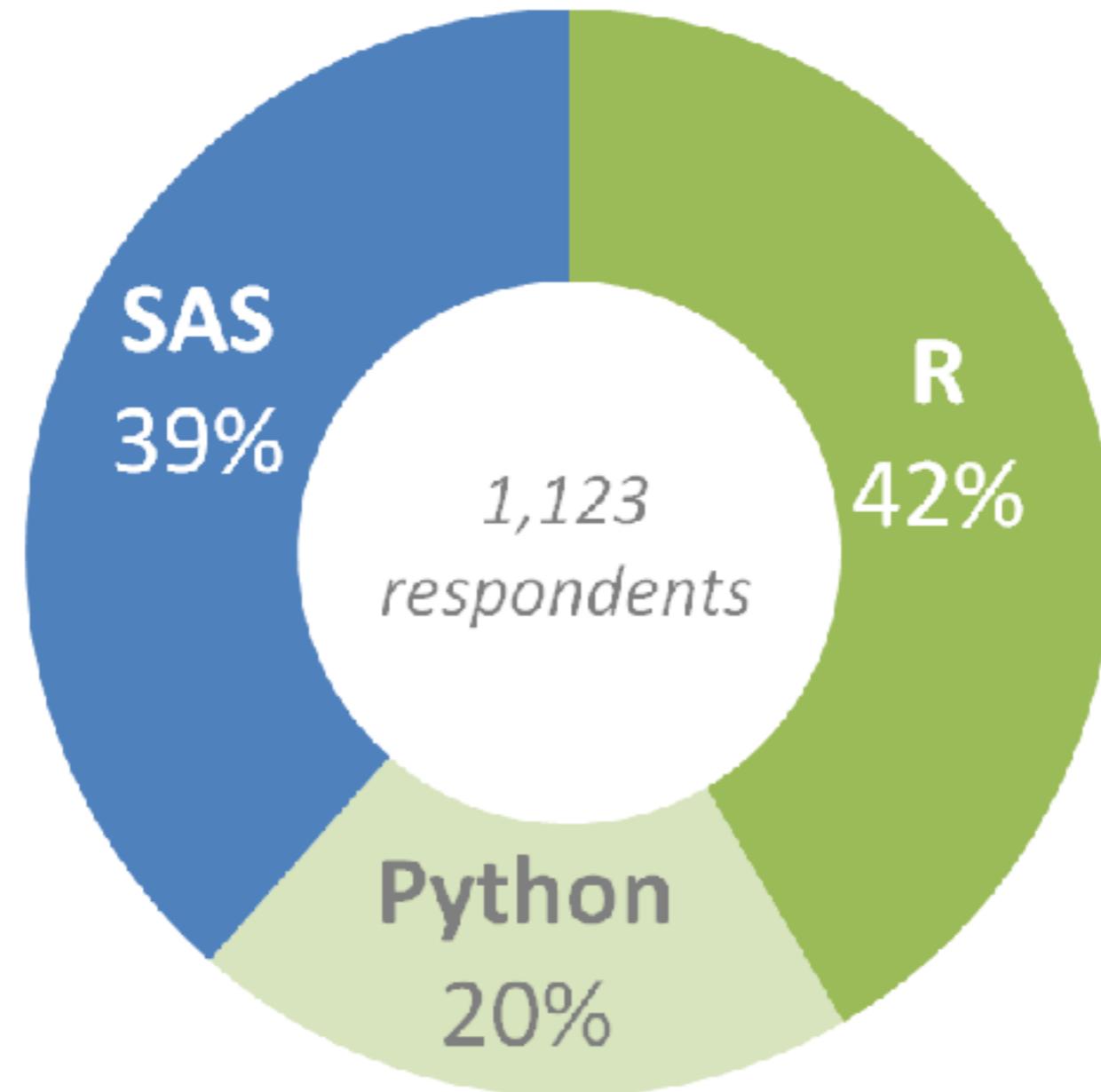
# Why use R?

- R is Open-Source Software
- Many built-in functions and installable packages that will cover nearly every possible need
- R is an interpreted language
- Code doesn't have to be compiled
- Interactive console makes testing and debugging easy
- Cons to using R
  - Slower than compiled languages
  - Can have runtime errors

# Why use R?

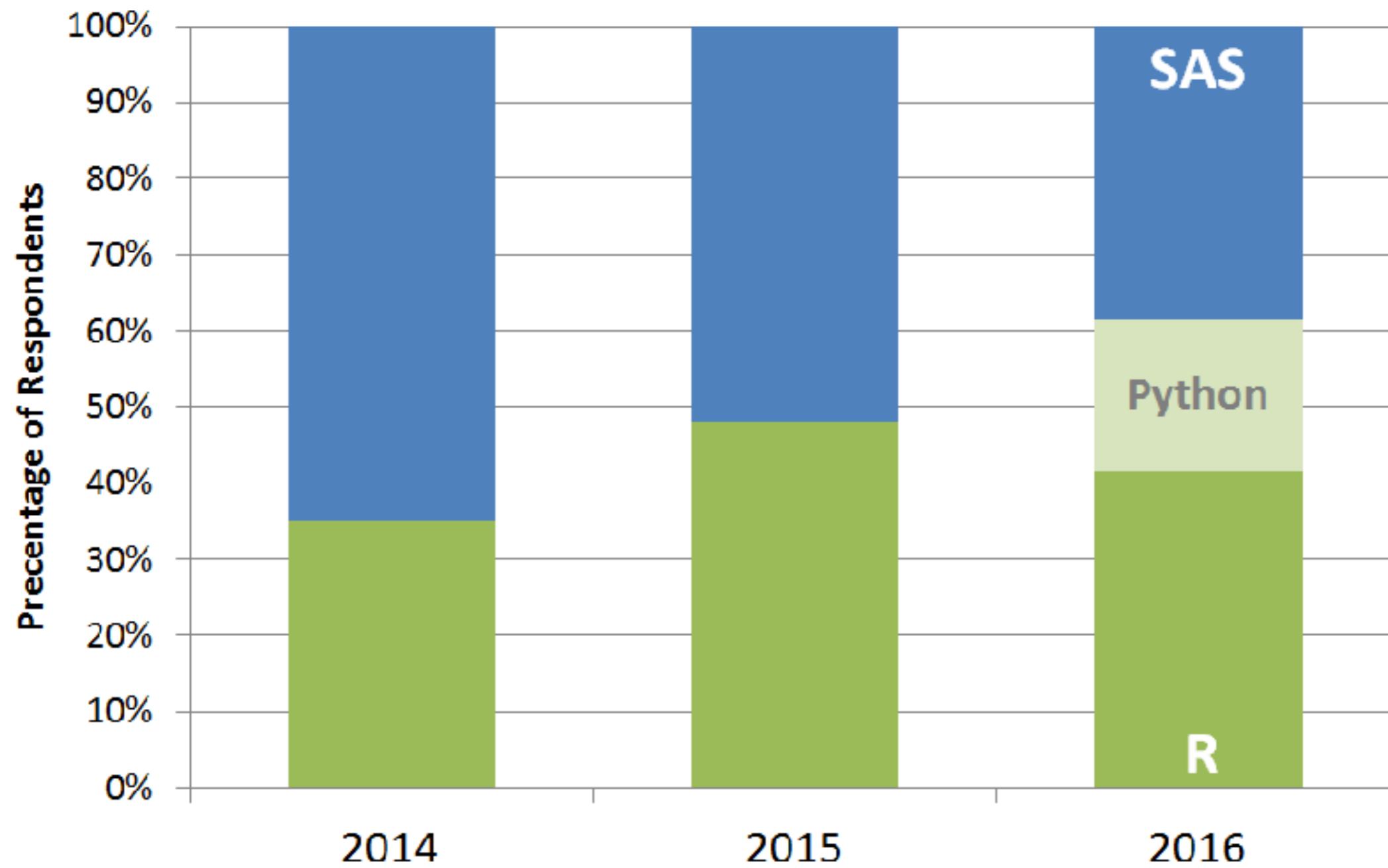


# R vs SAS vs Python

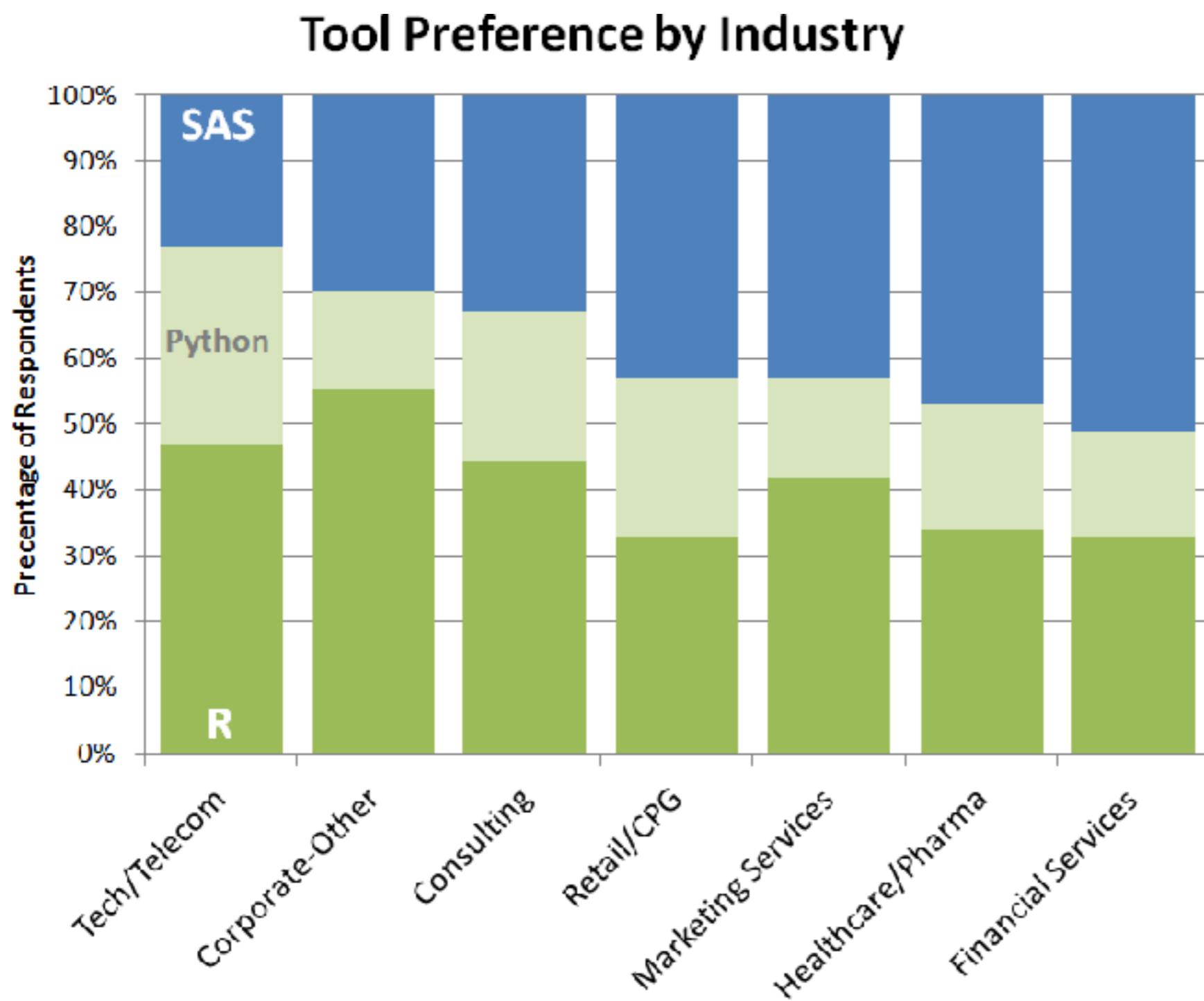


# SAS vs R vs Python over time

**SAS, R, Python Preference Over Time**

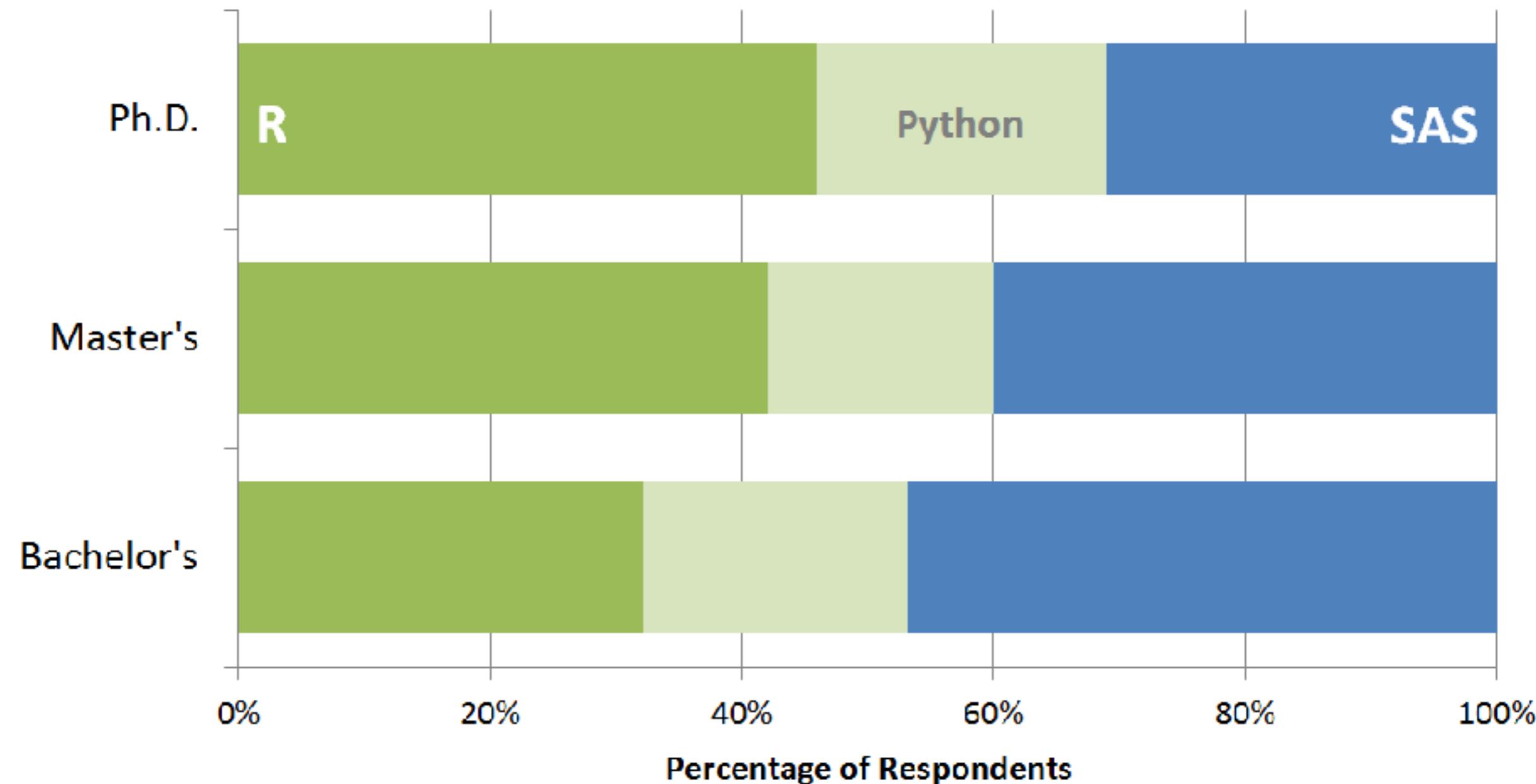


# SAS vs Python vs R by Industry



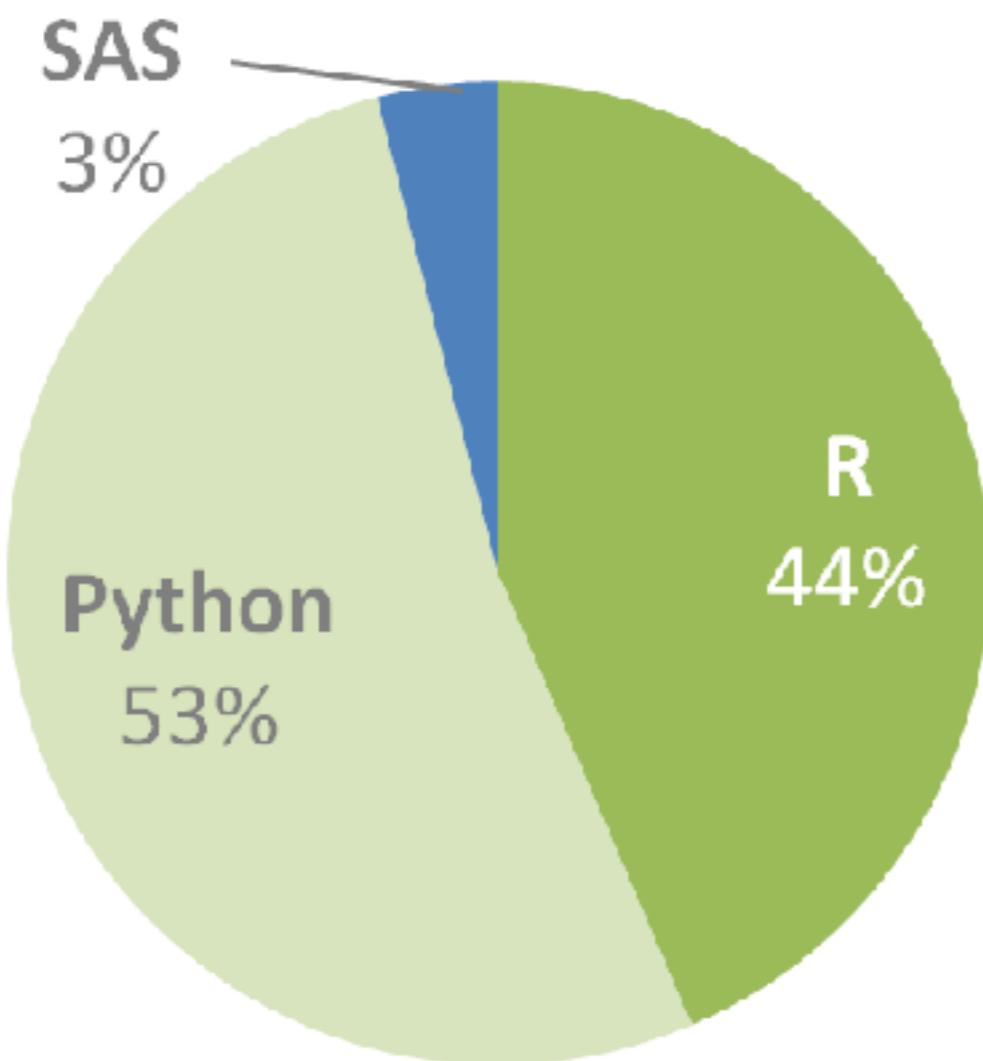
# by Education

Tool Preference by Education

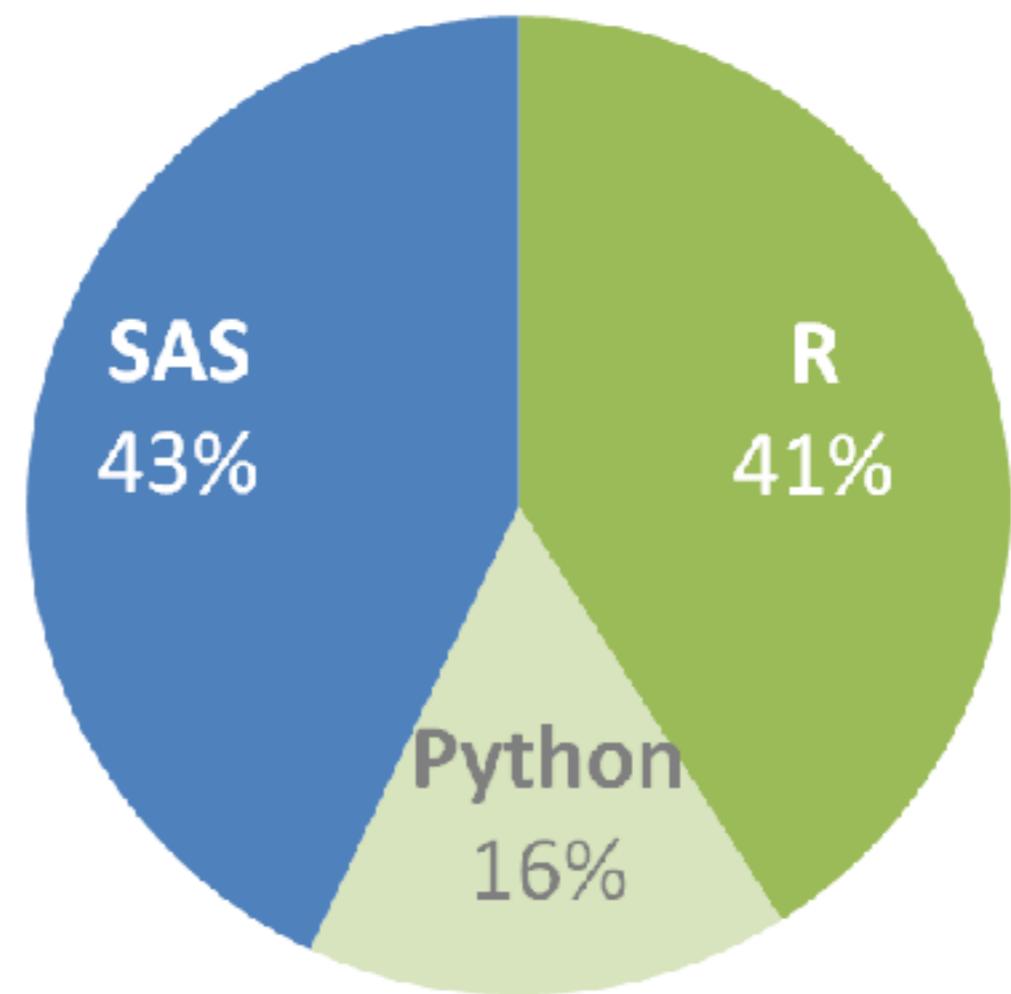


# Data Science vs Predictive Analysis

**Data Scientist**



**Predictive Analytics**



Source : <http://www.kdnuggets.com/2016/07/burttworks-sas-r-python-analytics-pros-prefer.html>

# Tools



**R**  
[www.r-project.org](http://www.r-project.org)

The engine\*



**RStudio**  
[www.rstudio.org](http://www.rstudio.org)

The pretty face\*\*

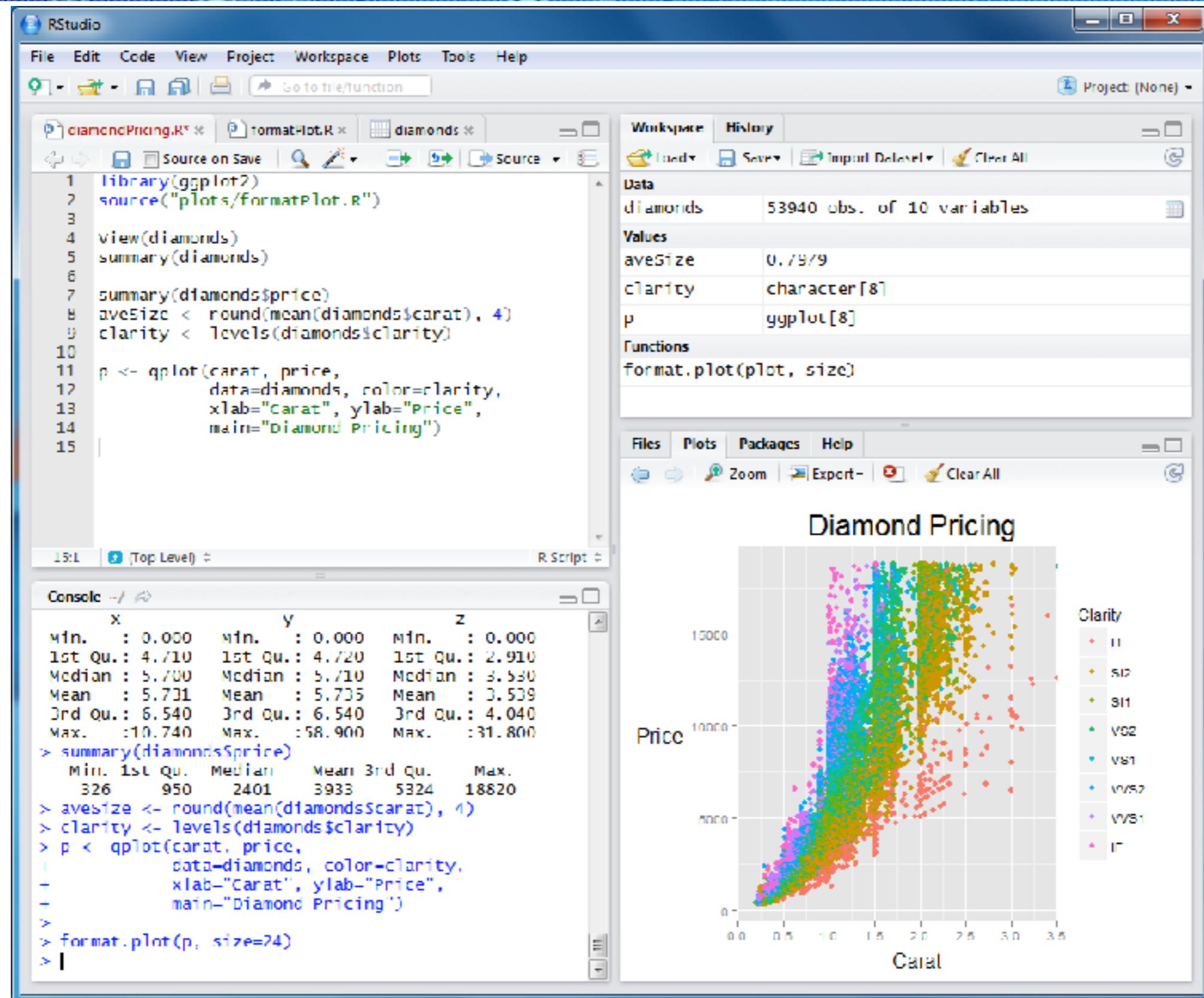
\* Many alternatives exist. Smallest learning curve.

\*\* A few alternatives exist. This happens to be the easiest at the moment.

# RStudio Features

- Code completion
- Command history search
- Command history to R script / file
- Function extraction from Rscript
- Sweave and Knitr support

# Introducing R-Studio



# RStudio Panes

RStudio has 4 main windows ('panes'):

- **The Source pane:** create a file that you can save and run later
- **The Console pane:** type or paste in commands to get output from R
- **The Workspace/History pane:** see a list of variables or previous commands
- **The Files/Plots/Packages/Help pane:** see plots, help pages, and other items in this window.

# Source and Console Panes

**Source window:** Create a file here, so that you can save and run it later (or turn in as homework)

Any non-command line  
should start with a #

**Console:** Type or paste commands in here to get results from R

If you are loading a data file, you will need to be in the correct directory

> denotes that R is waiting for a command

+ denotes that R is waiting for you to finish the previous command (not shown here)

The screenshot shows the RStudio interface with two main panes: the Source pane and the Console pane.

**Source pane:** The title bar says "hw1.R x". The code content is as follows:

```
1 #  
2 # David Choi  
3 # HW 1  
4 #  
5  
6 # Here are the steps to accomplish problem 1:  
7 x = sqrt(5)  
8 y = sqrt(2)  
9 z = x + y  
10 z  
11
```

**Console pane:** The title bar says "Console ~/Dropbox/teaching/DataVis mini 4/homework 1". The history shows:

```
11:1 type <ctrl> c to quit R.  
[Workspace loaded from ~/.RData]  
> setwd("~/Dropbox/teaching/DataVis mini 4/homework 1")  
> x = sqrt(5)  
> y = sqrt(2)  
> z = x + y  
> z  
[1] 3.650282  
>
```

# Console Panes

## Variables

- Save the results of a command in memory by **giving it a name**:

`z` uses `x` and `y`: →

Values are not linked;  
updating `x` doesn't →  
change `z`

Redefine `z` using the new →  
value for `x`

Use “ “ or ‘ ’ to distinguish →  
between variable names  
and regular text

**Note:**

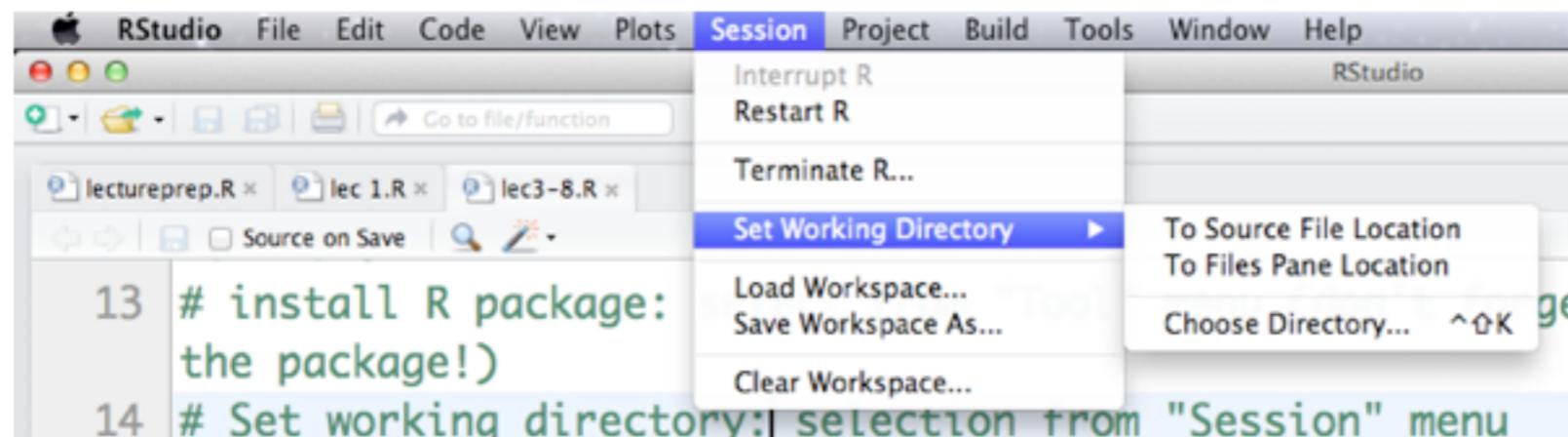
`x <- 3`: same as `x=3`

```
Console ~ / ↗
> x = 3
> y = 2
> x           ← Create x and y
[1] 3
> y
[1] 2
> z = sqrt(x^2+y^2)
> z
[1] 3.605551
> x = 5
> z
[1] 3.605551
> z = sqrt(x^2+y^2)
> z
[1] 5.385165
> str = 'Hi there'
> str
[1] "Hi there"
> str2 = "It's cold outside"
> str2
[1] "It's cold outside"
```

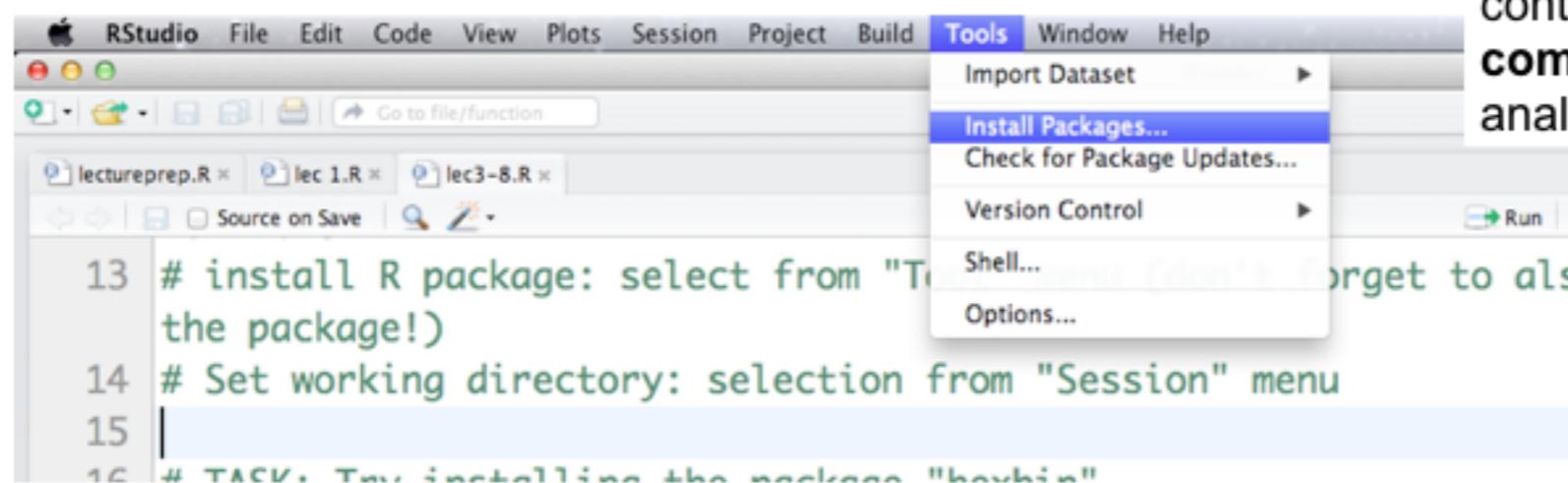
# RStudio Menu

## Two helpful menu items in Rstudio

- Set the current directory:



- Install a package



Packages are extensions to R, containing **new commands** for analysis or graphics

# R Markdown

The image shows the RStudio interface with two main panes. The left pane is titled "example.Rmd" and contains the R Markdown code. The right pane is titled "RStudio: Preview HTML" and shows the rendered output.

**Code (example.Rmd):**

```
1 Header 1
2 -----
3 This is an R Markdown document. Markdown is a
4 simple formatting syntax for authoring web pages.
5 Use an asterisk mark, to provide emphasis such as
6 *italics* and **bold**.
7 Create lists with a dash:
8 - Item 1
9 - Item 2
10 - Item 3
11
12 You can write `in-line` code with a back-tick.
13 ...
14
15 Code blocks display
16 with fixed-width font
17 ...
18
19 > Blockquotes are offset
20
```

**Preview (RStudio: Preview HTML):**

**Header 1**

This is an R Markdown document. Markdown is a simple formatting syntax for authoring web pages.

Use an asterisk mark, to provide emphasis such as *italics* and **bold**.

Create lists with a dash:

- Item 1
- Item 2
- Item 3

You can write `in-line` code with a back-tick.

Code blocks display  
with fixed-width font

Blockquotes are offset

# R Markdown

chunks.Rmd x

ABC MD Knit HTML Chunks

```
1 R Code Chunks
2 -----
3
4 With R Markdown, you can insert R code
5 chunks including plots:
6
7 ````{r qplot, fig.width=4, fig.height=3,
8 | message=FALSE}
9 # quick summary and plot
10 library(ggplot2)
11 summary(cars)
12 qplot(speed, dist, data=cars) +
13   geom_smooth()
```

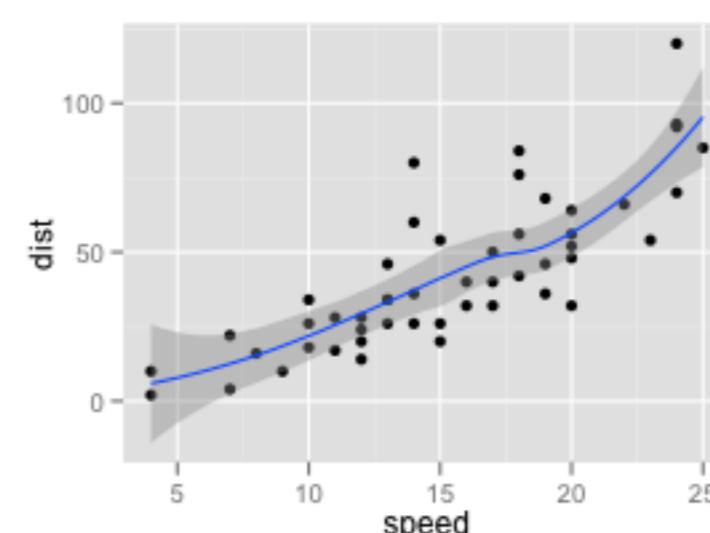
R Code Chunks

With R Markdown, you can insert R code chunks including plots:

```
# quick summary and plot
library(ggplot2)
summary(cars)
```

```
##       speed          dist
##  Min.   : 4.0   Min.   :  2
##  1st Qu.:12.0   1st Qu.: 26
##  Median :15.0   Median : 36
##  Mean   :15.4   Mean   : 43
##  3rd Qu.:19.0   3rd Qu.: 56
##  Max.   :25.0   Max.   :120
```

```
qplot(speed, dist, data = cars) + geom_smooth()
```



# Workshop 1: Hello World

Open RStudio on your machine

- File > New File > R Markdown ...
- Change summary(cars) in the first code block to print("Hello world!")
- Click Knit HTML to produce an HTML file.
- Save your Rmd file as helloworld.Rmd

# Data Types

You'll encounter different kinds of data types

- **Booleans** Direct binary values: TRUE or FALSE in R
- **Integers**: whole numbers (positive, negative or zero)
- **Characters** fixed-length blocks of bits, with special coding; strings = sequences of characters
- **Floating point numbers**: a fraction (with a finite number of bits) times an exponent, like  $1.87 * 10^6$
- **Missing** or ill-defined values: NA, NaN, etc.

# Operators (Functions)

Command	Description
<code>+, -, *, \</code>	add, subtract, multiply, divide
<code>^</code>	raise to the power of
<code>%%</code>	remainder after division (ex: <code>8 %% 3 = 2</code> )
<code>( )</code>	change the order of operations
<code>log(), exp()</code>	logarithms and exponents (ex: <code>log(10) = 2.302</code> )
<code>sqrt()</code>	square root
<code>round()</code>	round to the nearest whole number (ex: <code>round(2.3) = 2</code> )
<code>floor(), ceiling()</code>	round down or round up
<code>abs()</code>	absolute value

# Operators

```
7 + 5 # Addition
```

```
[1] 12
```

```
7 - 5 # Subtraction
```

```
[1] 2
```

```
7 * 5 # Multiplication
```

```
[1] 35
```

```
7 ^ 5 # Exponentiation
```

```
[1] 16807
```

```
7 / 5 # Division
```

```
[1] 1.4
```

```
7 %% 5 # Modulus
```

```
[1] 2
```

```
7 %/% 5 # Integer division
```

```
[1] 1
```

# Operators

**Comparisons** are also **binary operators**; they take two objects, like numbers, and give a Boolean

```
7 > 5
```

```
[1] TRUE
```

```
7 < 5
```

```
[1] FALSE
```

```
7 >= 7
```

```
[1] TRUE
```

```
7 <= 5
```

```
[1] FALSE
```

```
7 == 5
```

```
[1] FALSE
```

```
7 != 5
```

```
[1] TRUE
```

# Boolean Operators

Basically “and” and “or”:

```
(5 > 7) & (6*7 == 42)
```

```
[1] FALSE
```

```
(5 > 7) | (6*7 == 42)
```

```
[1] TRUE
```

# Basic Math / Basic Logic

- + addition
- subtraction
- \* multiplication
- / division
- % modulus (remainder)
- ^ to the power

?Arithmetic

- ! NOT
- & bitwise AND
- | bitwise OR
- && short circuit AND
- || short circuit OR
- == equality
- != NOT equality

?Logic

Also try ?Syntax, ?Comparison

# More types

**typeof()** function returns the type

**is.foo()** functions return Booleans for whether the argument is of type foo

**as.foo()** (tries to) “cast” its argument to type foo — to translate it sensibly into a foo-type value

```
typeof(7)
```

```
[1] "double"
```

```
is.numeric(7)
```

```
[1] TRUE
```

```
is.na(7)
```

```
[1] FALSE
```

```
is.character(7)
```

```
[1] FALSE
```

```
is.character("7")
```

```
[1] TRUE
```

```
is.character("seven")
```

```
[1] TRUE
```

```
is.na("seven")
```

```
[1] FALSE
```

# Variables

We can give names to data objects; these give us **variables**

A few variables are built in:

```
pi
```

```
[1] 3.141593
```

Variables can be arguments to functions or operators, just like constants:

```
pi*10
```

```
[1] 31.41593
```

```
cos(pi)
```

```
[1] -1
```

# Assignment Operator

Most variables are created with the **assignment operator**, `<-` or `=`

```
approx.pi <- 22 / 7  
approx.pi
```

```
[1] 3.142857
```

```
diameter.in.cubits = 10  
approx.pi*diameter.in.cubits
```

```
[1] 31.42857
```

# Variables

- Using names and variables makes code: easier to design, easier to debug, less prone to bugs, easier to improve, and easier for others to read
- Avoid “magic constants”; use named variables
- Use descriptive variable names
  - Good: num.students <- 35
  - Bad: ns <- 35

# Workspace

What names have you defined values for?

```
ls()
```

```
[1] "approx.pi"           "circumference.in.cubits"  
[3] "diameter.in.cubits"
```

Getting rid of variables:

```
rm("circumference.in.cubits")  
ls()
```

```
[1] "approx.pi"           "diameter.in.cubits"
```

# Data Structure : Vector

- Group related data values into one object, a **data structure**
- A **vector** is a sequence of values, all of the same type
- `c()` function returns a vector containing all its arguments in order

```
students <- c("Sean", "Louisa", "Frank", "Farhad", "Li")
midterm <- c(80, 90, 93, 82, 95)
```

- Typing the variable name at the prompt causes it to display

```
students
```

```
[1] "Sean"    "Louisa"  "Frank"   "Farhad" "Li"
```

# Indexing Vector

- `vec[1]` is the first element, `vec[4]` is the 4th element of `vec`

```
students
```

```
[1] "Sean"    "Louisa"  "Frank"   "Farhad"  "Li"
```

```
students[4]
```

```
[1] "Farhad"
```

- `vec[-4]` is a vector containing all but the fourth element

```
students[-4]
```

```
[1] "Sean"    "Louisa"  "Frank"   "Li"
```

# Vector Arithmetic

Operators apply to vectors “pairwise” or “elementwise”:

```
final <- c(78, 84, 95, 82, 91) # Final exam scores  
midterm # Midterm exam scores
```

```
[1] 80 90 93 82 95
```

```
midterm + final # Sum of midterm and final scores
```

```
[1] 158 174 188 164 186
```

```
(midterm + final)/2 # Average exam score
```

```
[1] 79 87 94 82 93
```

```
course.grades <- 0.4*midterm + 0.6*final # Final course grade  
course.grades
```

```
[1] 78.8 86.4 94.2 82.0 92.6
```

# Pairwise Comparison

Is the final score higher than the midterm score?

```
midterm
```

```
[1] 80 90 93 82 95
```

```
final
```

```
[1] 78 84 95 82 91
```

```
final > midterm
```

```
[1] FALSE FALSE TRUE FALSE FALSE
```

Boolean operators can be applied elementwise:

```
(final < midterm) & (midterm > 80)
```

```
[1] FALSE TRUE FALSE FALSE TRUE
```

# Functions on Vectors

Command	Description
<code>sum(vec)</code>	sums up all the elements of vec
<code>mean(vec)</code>	mean of vec
<code>median(vec)</code>	median of vec
<code>min(vec), max(vec)</code>	the largest or smallest element of vec
<code>sd(vec), var(vec)</code>	the standard deviation and variance of vec
<code>length(vec)</code>	the number of elements in vec
<code>pmax(vec1, vec2), pmin(vec1, vec2)</code>	example: <code>pmax(quiz1, quiz2)</code> returns the higher of quiz 1 and quiz 2 for each student
<code>sort(vec)</code>	returns the vec in sorted order
<code>order(vec)</code>	returns the index that sorts the vector vec
<code>unique(vec)</code>	lists the unique elements of vec
<code>summary(vec)</code>	gives a five-number summary
<code>any(vec), all(vec)</code>	useful on Boolean vectors

# Function on Vectors

```
course.grades
```

```
[1] 78.8 86.4 94.2 82.0 92.6
```

```
mean(course.grades) # mean grade
```

```
[1] 86.8
```

```
median(course.grades)
```

```
[1] 86.4
```

```
sd(course.grades) # grade standard deviation
```

```
[1] 6.625708
```

# More on Functions

```
sort(course.grades)
```

```
[1] 78.8 82.0 86.4 92.6 94.2
```

```
max(course.grades) # highest course grade
```

```
[1] 94.2
```

```
min(course.grades) # lowest course grade
```

```
[1] 78.8
```

# Referencing elements of Vectors

```
students
```

```
[1] "Sean"    "Louisa"  "Frank"   "Farhad"  "Li"
```

Vector of indices:

```
students[c(2,4)]
```

```
[1] "Louisa" "Farhad"
```

Vector of negative indices

```
students[c(-1,-3)]
```

```
[1] "Louisa" "Farhad" "Li"
```

# More on Referencing

`which( )` returns the TRUE indexes of a Boolean vector:

```
course.grades
```

```
[1] 78.8 86.4 94.2 82.0 92.6
```

```
a.threshold <- 90 # A grade = 90% or higher  
course.grades >= a.threshold # vector of booleans
```

```
[1] FALSE FALSE TRUE FALSE TRUE
```

```
a.students <- which(course.grades >= a.threshold) # Applying which()  
a.students
```

```
[1] 3 5
```

```
students[a.students] # Names of A students
```

```
[1] "Frank" "Li"
```

# Workshop 2

Using the same Rmarkdown file from Workshop 1, do the following:-

- 1) Creating sequences

## Colon operator:

```
1:10 # Numbers 1 to 10
```

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

```
127:132 # Numbers 127 to 132
```

```
## [1] 127 128 129 130 131 132
```

# Workshop 2

**seq** function: `seq(from, to, by)`

```
seq(1,10) # Numbers 1 to 10
```

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

```
seq(1,10,2) # Odd numbers from 1 to 10
```

```
## [1] 1 3 5 7 9
```

```
seq(2,10,2) # Even numbers from 2 to 10
```

```
## [1] 2 4 6 8 10
```

## Workshop 2 : Use RStudio to answer questions below

- (a) Use `:` to output the sequence of numbers from 3 to 12
- (b) Use `seq()` to output the sequence of numbers from 3 to 30 in increments of 3
- (c) Save the sequence from (a) as a variable `x`, and the sequence from (b) as a variable `y`. Output their product `x*y`

# Difference between = and <-

- The operators <- and = assign into the environment in which they are evaluated.
- The operator <- can be used anywhere, whereas the operator = is only allowed at the top level (e.g., in the complete expression typed at the command prompt) or as one of the subexpressions in a braced list of expressions.

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

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

# Reading and Getting Data into R

## Combine Command

```
c(1, 2, 3, 4)  
c(item1, item2, item3, item4)  
c("item1", "item2", "item3")
```

## Scan Command

```
our.data = scan()  
  
scan(what = 'character')  
data5 = scan(sep = ';', what = 'char')  
data6 = scan(file = 'data.txt')
```

## Working Directory

```
getwd()
```

# Reading Bigger Data Files

```
read.csv()  
read.csv(file, sep = ';', header = TRUE, row.names)  
fw = read.csv(file.choose())  
  
my.ssv = read.table(file.choose(), header = TRUE)  
my.tsv = read.delim(file.choose())  
my.tsv = read.csv(file.choose(), sep = '\t')  
my.tsv = read.table(file.choose(), header = TRUE, sep = '\t')
```

# Importing Data

- To import tabular data into R, we use the `read.table()` command

```
1 survey <- read.table("survey_data.csv", header=TRUE, sep=",")  
2  
3
```

- Let's parse this command one component at a time
  - The data is in a file called `survey_data.csv`, which is an online file
  - The file contains a header as its first row
  - The csv format means that the data is comma-separated, so `sep=","`
- Could've also used `read.csv()`, which is just `read.table()` with the preset `sep=","`

# Exploring the Data

- R imports data into a `data.frame` object

```
class(survey)
```

```
[1] "data.frame"
```

- To view the first few rows of the data, use `head()`

```
head(survey, 3)
```

	Program	PriorExp	Rexperience	OperatingSystem	TVhours
1	MISM	Some experience	Never used	Windows	1
2	Other	Some experience	Basic competence	Windows	8
3	MISM	Extensive experience Editor	Basic competence	Windows	4
1	Microsoft Word				
2	Microsoft Word				
3	Microsoft Word				

- `head(data.frame, n)` returns the first n rows of the data frame
- In the Console, you can also use `View(survey)` to get a spreadsheet view

# Simple Summary

- Use the `str( )` function to get a simple summary of your data set

```
str(survey)
```

```
'data.frame': 31 obs. of 6 variables:  
 $ Program      : Factor w/ 3 levels "MISM","Other",...: 1 2 1 3 2 3 2 1 3 3 ...  
 $ PriorExp     : Factor w/ 3 levels "Extensive experience",...: 3 3 1 2 2 3 3 3  
 2 1 ...  
 $ Rexperience  : Factor w/ 3 levels "Basic competence",...: 3 1 1 2 3 3 3 3 3 3  
 ...  
 $ OperatingSystem: Factor w/ 2 levels "Mac OS X","Windows": 2 2 2 2 2 1 1 2 1 2  
 ...  
 $ TVhours       : int 1 8 4 5 3 10 0 10 15 4 ...  
 $ Editor        : Factor w/ 1 level "Microsoft Word": 1 1 1 1 1 1 1 1 1 1 ...
```

- This says that `TVhours` is a numeric variable, while all the rest are factors (categorical)

# Another Summary

```
summary(survey)
```

Program	PriorExp	Rexperience
MISM :10	Extensive experience : 5	Basic competence : 2
Other: 6	Never programmed before:12	Installed on machine: 7
PPM :15	Some experience :14	Never used :22

OperatingSystem	TVhours	Editor
Mac OS X:14	Min. : 0.000	Microsoft Word:31
Windows :17	1st Qu.: 1.000	
	Median : 4.000	
	Mean : 5.742	
	3rd Qu.: 9.000	
	Max. :40.000	

# Data Frames

- 2 Dimensional Objects, it has rows and columns. R treats the columns as separate samples or variables, rows represent the replicates or observations.
- To see what an R object is made up of, you can use `attributes()`

```
attributes(survey)
```

```
$names
[1] "Program"          "PriorExp"        "Rexperience"    "OperatingSystem"
[5] "TVhours"          "Editor"          "Experience"      "OperatingSystem"

$class
[1] "data.frame"

$row.names
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[24] 24 25 26 27 28 29 30 31
```

An R **data frame** is a *list* whose columns you can refer to by *name* or *index*

# Data Frame dimensions

- We can use `nrow()` and `ncol` to determine the number of survey responses and the number of survey questions

```
nrow(survey) # Number of rows (responses)
```

```
[1] 31
```

```
ncol(survey) # Number of columns (questions)
```

```
[1] 6
```

- When writing reports, you will often want to say how large your sample size was
- To do this *inline*, use the syntax:

```
`r nrow(survey)`
```

- This allows us to write “31 students responded to the survey”, and have the number displayed automatically change when `nrow(survey)` changes.

# Indexing Data Frame

- There are many different ways of indexing the same piece of a data frame

```
survey[["Program"]] # "Program" element
```

```
[1] MISM Other MISM PPM  Other PPM  Other MISM PPM  PPM  MISM  
[12] PPM   PPM   PPM   PPM   PPM   PPM   MISM  MISM  MISM  PPM  PPM  
[23] MISM Other PPM  PPM  MISM  PPM  Other Other MISM  
Levels: MISM Other PPM
```

```
survey$Program # "Program" element
```

```
[1] MISM Other MISM PPM  Other PPM  Other MISM PPM  PPM  MISM  
[12] PPM   PPM   PPM   PPM   PPM   MISM  MISM  MISM  PPM  PPM  
[23] MISM Other PPM  PPM  MISM  PPM  Other Other MISM  
Levels: MISM Other PPM
```

```
survey[,1] # Data from 1st column
```

```
[1] MISM Other MISM PPM  Other PPM  Other MISM PPM  PPM  MISM  
[12] PPM   PPM   PPM   PPM   PPM   MISM  MISM  MISM  PPM  PPM  
[23] MISM Other PPM  PPM  MISM  PPM  Other Other MISM  
Levels: MISM Other PPM
```

# More Indexing

- Note that single brackets and double brackets have different effects

```
survey[["Program"]]
```

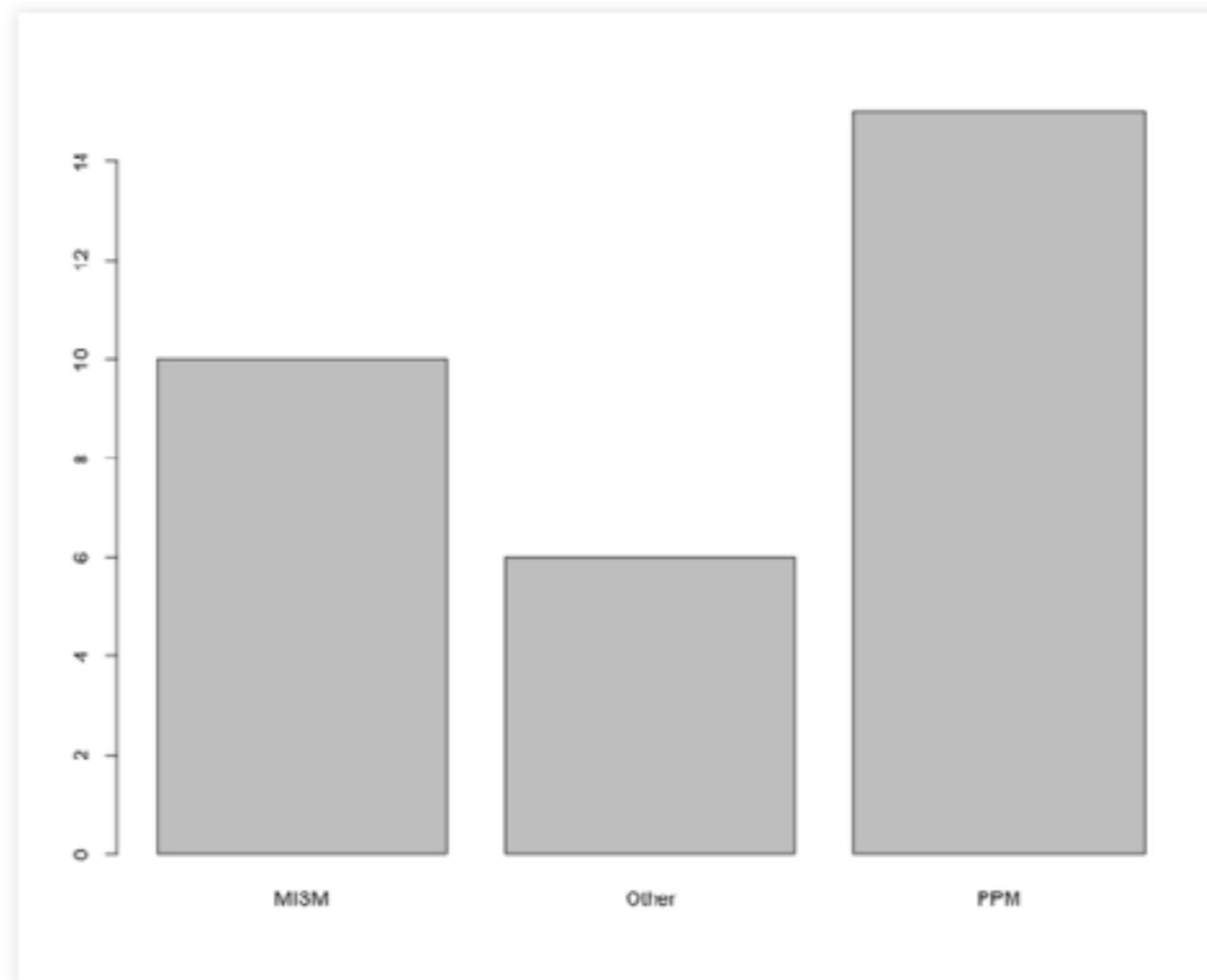
```
[1] MISM Other MISM PPM Other PPM Other MISM PPM PPM MISM  
[12] PPM PPM PPM PPM PPM PPM MISM MISM MISM PPM PPM  
[23] MISM Other PPM PPM MISM PPM Other Other MISM  
Levels: MISM Other PPM
```

```
survey["Program"] # sub-data frame containing only "Program"
```

```
Program  
1      MISM  
2      Other  
3      MISM  
4      PPM  
5      Other  
6      PPM  
7      Other  
8      MISM  
9      PPM  
10     PPM  
11     MISM  
12     PPM  
13     PPM  
14     PPM
```

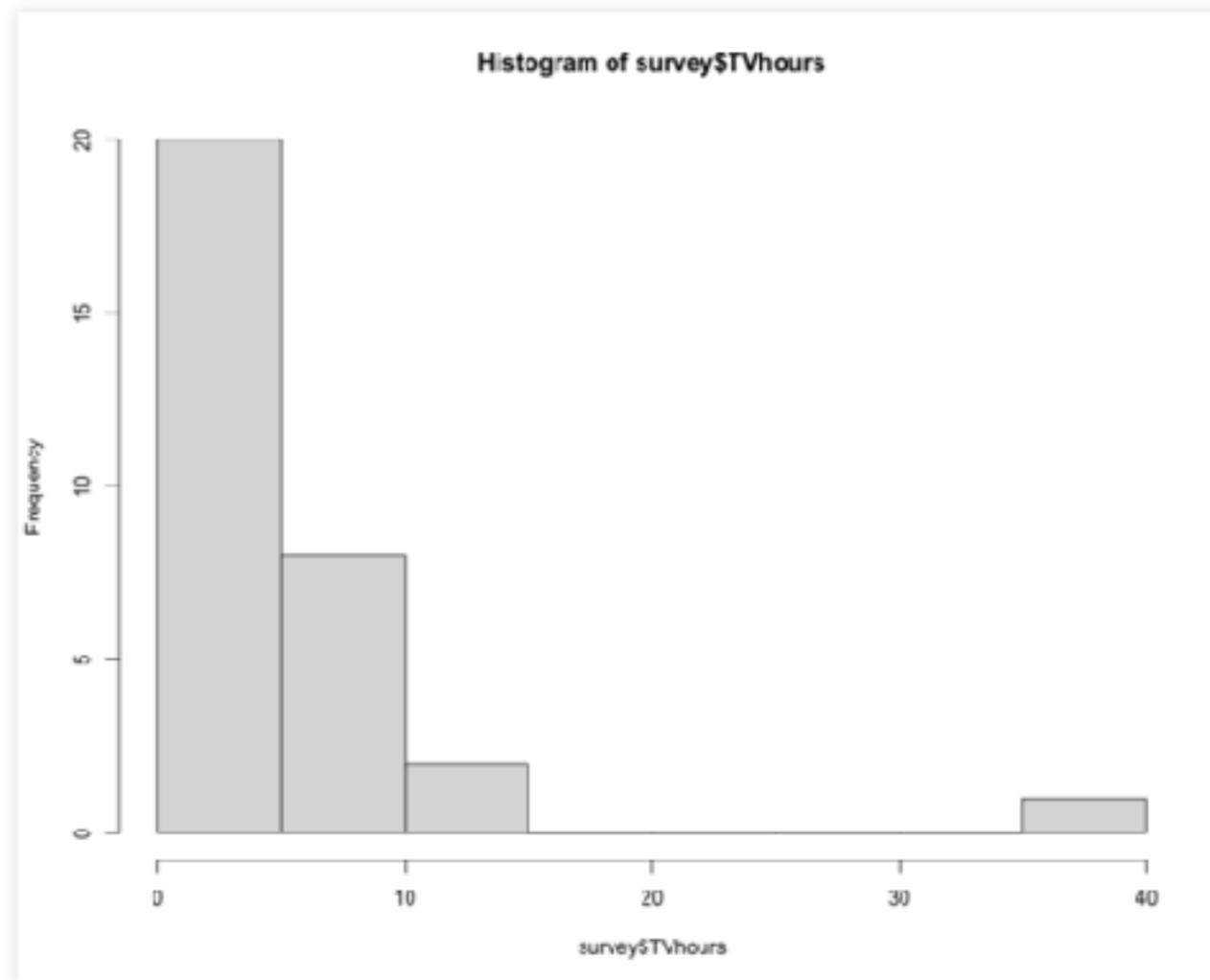
# Bar Plot (Categorical Data)

```
plot(survey[["Program"]])
```



# Histogram (Continuous Data)

```
hist(survey$TVhours, col="lightgray")
```



# Indexing Multiple Columns

```
head(survey[,c(1,5)]) # Data from 1st and 5th columns
```

```
Program TVhours
1 MISM 1
2 Other 8
3 MISM 4
4 PPM 5
5 Other 3
6 PPM 10
```

```
head(survey[c("Program", "Editor")]) # Data from "Program" and "Editor"
```

```
Program Editor
1 MISM Microsoft Word
2 Other Microsoft Word
3 MISM Microsoft Word
4 PPM Microsoft Word
5 Other Microsoft Word
6 PPM Microsoft Word
```

# Indexing Row and Column

- Data frames have two dimensions to index across

```
survey[6,] # 6th row
```

```
Program      PriorExp Rexperience OperatingSystem TVhours
6      PPM Some experience    Never used      Mac OS X      10
          Editor
6 Microsoft Word
```

```
survey[6,5] # row 6, column 5
```

```
[1] 10
```

```
survey[6, "Program"] # Program of 6th survey respondent
```

```
[1] PPM
Levels: MISM Other PPM
```

```
survey[["Program"]][6] # Program of 6th survey respondent
```

```
[1] PPM
Levels: MISM Other PPM
```

# More Indexing

- We can use this operator for indexing

```
survey[1:3,] # equivalent to head(survey, 3)
```

	Program	PriorExp	Rexperience	OperatingSystem	TVhours
1	MISM	Some experience	Never used	Windows	1
2	Other	Some experience	Basic competence	Windows	8
3	MISM	Extensive experience	Basic competence Editor	Windows	4
1	Microsoft Word				
2	Microsoft Word				
3	Microsoft Word				

```
survey[3:5, c(1,5)]
```

	Program	TVhours
3	MISM	4
4	PPM	5
5	Other	3

# Subsets of Data

- We are often interested in learning something a specific subset of the data

```
survey[survey$Program=="MISM",] # Data from the MISM students
```

	Program	PriorExp	Rexperience	OperatingSystem
1	MISM	Some experience	Never used	Windows
3	MISM	Extensive experience	Basic competence	Windows
8	MISM	Some experience	Never used	Windows
11	MISM	Extensive experience	Installed on machine	Mac OS X
18	MISM	Extensive experience	Never used	Mac OS X
19	MISM	Extensive experience	Never used	Windows
20	MISM	Some experience	Never used	Windows
23	MISM	Some experience	Never used	Windows
27	MISM	Some experience	Never used	Mac OS X
31	MISM	Some experience	Installed on machine	Windows
	TVhours	Editor		
1		1 Microsoft Word		
3		4 Microsoft Word		
8		10 Microsoft Word		
11		0 Microsoft Word		
18		3 Microsoft Word		
19		0 Microsoft Word		
20		1 Microsoft Word		
23		3 Microsoft Word		
27		0 Microsoft Word		
31		0 Microsoft Word		

# More subsets example

- Let's pull all of the PPM students who have never used R before

```
survey[survey$Program=="PPM" & survey$Rexperience=="Never used", ]
```

	Program	PriorExp	Rexperience	OperatingSystem	TVhours
6	PPM	Some experience	Never used	Mac OS X	10
9	PPM	Never programmed before	Never used	Mac OS X	15
10	PPM	Extensive experience	Never used	Windows	4
12	PPM	Never programmed before	Never used	Windows	0
13	PPM	Some experience	Never used	Mac OS X	10
14	PPM	Never programmed before	Never used	Mac OS X	4
15	PPM	Some experience	Never used	Windows	10
16	PPM	Some experience	Never used	Mac OS X	2
22	PPM	Never programmed before	Never used	Windows	7
25	PPM	Never programmed before	Never used	Mac OS X	6
		Editor			
6	Microsoft Word				
9	Microsoft Word				
10	Microsoft Word				
12	Microsoft Word				
13	Microsoft Word				
14	Microsoft Word				
15	Microsoft Word				
16	Microsoft Word				
22	Microsoft Word				
25	Microsoft Word				

# Subset

- When the subset conditions get long or messy, it is preferable to use the **subset()** function
- Here's an example of selecting the OperatingSystem and TVhours responses from all of the students who are either in PPM or Other and who listed their R experience as "Basic competence".

```
subset(survey, select=c("OperatingSystem", "TVhours"), subset=(Program == "PPM"  
| Program == "Other") & Rexperience == "Basic competence")
```

	OperatingSystem	TVhours
2	Windows	8

# Calculation from subset

```
mean(survey$TVhours[survey$Program == "PPM"]) # Average time PPM's spent watching TV
```

```
[1] 8.8
```

```
mean(survey$TVhours[survey$Program == "MISM"]) # Average time MISM's spent watching TV
```

```
[1] 2.2
```

```
mean(survey$TVhours[survey$Program == "Other"]) # Average time "Others" spent watching TV
```

```
[1] 4
```

# Coding Style

- Coding style (and code commenting) will become increasingly more important as we get into more advanced and involved programming tasks
- A few R “style guides” exist:
  - <http://r-pkgs.had.co.nz/style.html>
  - Google’s R Style Guide (<https://google.github.io/styleguide/Rguide.xml>)

# Example Style Guide

## Assignment operator. USE <-

```
student.names <- c("Eric", "Hao", "Jennifer") # Good  
student.names = c("Eric", "Hao", "Jennifer") # Bad
```

- Note: When specifying function arguments, only = is valid

```
sort(tv.hours, decreasing=TRUE) # Good  
sort(tv.hours, decreasing<-TRUE) # Bad!!
```

# Style Guide : Variable Name

- To make code easy to read, debug, and maintain, you should use **concise** but **descriptive** variable names
- Terms in variable names should be separated by `_` or `.`

```
# Accepted
day_one   day.one   day_1   day.1   day1

# Bad
d1      DayOne   dayone

# Can be made more concise:
first.day.of.the.month
```

- Avoid using variable names that are already pre-defined in R

```
# EXTREMELY bad:
c      T      pi      sum      mean
```

# Workshop 3: Importing and Indexing data

## Data practice

```
survey <- read.csv("survey_data.csv", header=TRUE);
```

Use command to answer these questions?

- (a) How many survey respondents are from MISM or Other?
- (b) What % of survey respondents are from PPM?

## Index practice

- (a) Use \$ notation to pull the OperatingSystem column from the survey data
- (b) Do the same thing with [,] notation, referring to OperatingSystem by name

# More on Data Frame

```
library(MASS)  
head(Cars93, 3)
```

	Manufacturer	Model	Type	Min.Price	Price	Max.Price	MPG.city
1	Acura	Integra	Small	12.9	15.9	18.8	25
2	Acura	Legend	Midsize	29.2	33.9	38.7	18
3	Audi	90	Compact	25.9	29.1	32.3	20
	MPG.highway		AirBags	DriveTrain	Cylinders	EngineSize	
1	31		None	Front	4	1.8	
2	25	Driver & Passenger		Front	6	3.2	
3	26	Driver only		Front	6	2.8	
	Horsepower	RPM	Rev.per.mile	Man.trans.avail	Fuel.tank.capacity		
1	140	6300	2890		Yes	13.2	
2	200	5500	2335		Yes	18.0	
3	172	5500	2280		Yes	16.9	
	Passengers	Length	Wheelbase	Width	Turn.circle	Rear.seat.room	
1	5	177	102	68	37	26.5	
2	5	195	115	71	38	30.0	
3	5	180	102	67	37	28.0	
	Luggage.room	Weight	Origin		Make		
1	11	2705	non-USA	Acura	Integra		
2	15	3560	non-USA	Acura	Legend		
3	14	3375	non-USA		Audi	90	

# Add column using “Transform”

- `transform( )` returns a new data frame with columns modified or added as specified by the function call

```
Cars93.metric <- transform(Cars93,
                           KMPL.city = 0.425 * MPG.city,
                           KMPL.highway = 0.425 * MPG.highway)
tail(names(Cars93.metric))
```

```
[1] "Luggage.room"   "Weight"        "Origin"       "Make"
[5] "KMPL.city"      "KMPL.highway"
```

- Our data frame has two new columns, giving the fuel consumption in km/l

# Another Approach

```
KMPL.city.2 <- 0.425 * Cars93$MPG.city  
# Add a new column called KMPL.city.2  
Cars93.metric$KMPL.city.2 <- KMPL.city.2  
tail(names(Cars93.metric))
```

```
[1] "Weight"          "Origin"         "Make"           "KMPL.city"  
[5] "KMPL.highway"   "KMPL.city.2"
```

- Let's check that both approaches did the same thing

```
identical(Cars93.metric$KMPL.city, Cars93.metric$KMPL.city.2)
```

```
[1] TRUE
```

# Changing level of a factor

```
manufacturer <- Cars93$Manufacturer  
head(manufacturer, 10)
```

```
[1] Acura      Acura      Audi       Audi       BMW        Buick      Buick  
[8] Buick      Buick      Cadillac  
32 Levels: Acura Audi BMW Buick Cadillac Chevrolet Chrysler ... Volvo
```

We'll use the `mapvalues(x, from, to)` function from the `plyr` library.

```
library(plyr)  
  
# Map Chevrolet, Pontiac and Buick to GM  
manufacturer.combined <- mapvalues(manufacturer,  
                                    from = c("Chevrolet", "Pontiac", "Buick"),  
                                    to = rep("GM", 3))  
  
head(manufacturer.combined, 10)
```

```
[1] Acura      Acura      Audi       Audi       BMW        GM        GM  
[8] GM         GM         Cadillac  
30 Levels: Acura Audi BMW GM Cadillac Chrysler Dodge ... Volvo
```

# Another example

- A lot of data comes with integer encodings of levels
- You may want to convert the integers to more meaningful values for the purpose of your analysis
- Let's pretend that in the class survey 'Program' was coded as an integer with 1 = MISM, 2 = Other, 3 = PPM
- Here's how we would get back the program codings using the `transform()`, `as.factor()` and `mapvalues()` functions

```
survey <- transform(survey, Program = as.factor(mapvalues(Program, c(1, 2, 3),
c("MISM", "Other", "PPM"))))  
head(survey)
```

	Program	PriorExp	Rexperience	OperatingSystem
1	MISM	Some experience	Never used	Windows
2	Other	Some experience	Basic competence	Windows
3	MISM	Extensive experience	Basic competence	Windows
4	PPM	Never programmed before	Installed on machine	Windows
5	Other	Never programmed before	Never used	Windows
6	PPM	Some experience	Never used	Mac OS X
	TVhours	Editor		
1	1	Microsoft Word		
2	8	Microsoft Word		
3	4	Microsoft Word		
4	5	Microsoft Word		
5	3	Microsoft Word		
6	10	Microsoft Word		

# table() function

- The `table()` function builds **contingency tables** showing counts at each combination of factor levels

```
table(Cars93$AirBags)
```

Driver & Passenger	16	Driver only	43	None	34
--------------------	----	-------------	----	------	----

```
table(Cars93$Origin)
```

USA	non-USA
48	45

```
table(Cars93$AirBags, Cars93$Origin)
```

	USA	non-USA
Driver & Passenger	9	7
Driver only	23	20
None	16	18

- Looks like US and non-US cars had about the same distribution of AirBag types

# more table()

- When `table()` is supplied a data frame, it produces contingency tables for all combinations of factors

```
head(Cars93[c("AirBags", "Origin")], 3)
```

```
      AirBags   Origin  
1       None non-USA  
2 Driver & Passenger non-USA  
3     Driver only non-USA
```

```
table(Cars93[c("AirBags", "Origin")])
```

AirBags	Origin	
	USA	non-USA
Driver & Passenger	9	7
Driver only	23	20
None	16	18

# Basic of Lists

A list is a **data structure** that can be used to store **different kinds** of data

- Recall: a vector is a data structure for storing *similar kinds of data*
- To better understand the difference, consider the following example.

```
my.vector.1 <- c("Michael", 165, TRUE) # (name, weight, is.male)
```

```
[1] "Michael" "165"      "TRUE"
```

```
typeof(my.vector.1) # All the elements are now character strings!
```

```
[1] "character"
```

# List vs Vector

```
my.vector.2 <- c(FALSE, TRUE, 27) # (is.male, is.citizen, age)  
typeof(my.vector.2)
```

```
[1] "double"
```

- Vectors expect elements to be all of the same type (e.g., Boolean, numeric, character)
- When data of different types are put into a vector, the R converts everything to a common type

# Lists

- To store data of different types in the same object, we use lists
- Simple way to build lists: use `list()` function

```
my.list <- list("Michael", 165, TRUE)  
my.list
```

```
[[1]]  
[1] "Michael"  
  
[[2]]  
[1] 165  
  
[[3]]  
[1] TRUE
```

```
sapply(my.list, typeof)
```

```
[1] "character" "double"    "logical"
```

# Named Elements

```
patient.1 <- list(name="Michael", weight=165, is.male=TRUE)  
patient.1
```

```
$name  
[1] "Michael"
```

```
$weight  
[1] 165
```

```
$is.male  
[1] TRUE
```

# Referencing element inside list

```
patient.1$name # Get "name" element (returns a string)
```

```
[1] "Michael"
```

```
patient.1[["name"]] # Get "name" element (returns a string)
```

```
[1] "Michael"
```

```
patient.1["name"] # Get "name" slice (returns a sub-list)
```

```
$name  
[1] "Michael"
```

```
c(typeof(patient.1$name), typeof(patient.1["name"]))
```

```
[1] "character" "list"
```

# Function

- We have used a lot of built-in functions: `mean()`, `subset()`, `plot()`, `read.table()`...
- An important part of programming and data analysis is to write custom functions
- Functions help make code **modular**
- Functions make debugging easier
- Remember: this entire class is about applying *functions* to *data*

# what is a function?

A function is a machine that turns **input objects** (arguments) into an **output object** (return value) according to a definite rule.

- Let's look at a really simple function

```
addOne <- function(x) {  
  x + 1  
}
```

- **x** is the **argument** or **input**
- The function **output** is the input **x** incremented by 1

```
addOne(12)
```

```
[1] 13
```

# Another example

- Here's a function that returns a % given a numerator, denominator, and desired number of decimal values

```
calculatePercentage <- function(x, y, d) {  
  decimal <- x / y # Calculate decimal value  
  round(100 * decimal, d) # Convert to % and round to d digits  
}  
  
calculatePercentage(27, 80, 1)
```

```
[1] 33.8
```

- If you're calculating several %'s for your report, you should use this kind of function instead of repeatedly copying and pasting code

# Function return a list

- Here's a function that takes a person's full name (FirstName LastName), weight in lb and height in inches and converts it into a list with the person's first name, person's last name, weight in kg, height in m, and BMI.

```
createPatientRecord <- function(full.name, weight, height) {  
  name.list <- strsplit(full.name, split=" ")[[1]]  
  first.name <- name.list[1]  
  last.name <- name.list[2]  
  weight.in.kg <- weight / 2.2  
  height.in.m <- height * 0.0254  
  bmi <- weight.in.kg / (height.in.m ^ 2)  
  list(first.name=first.name, last.name=last.name, weight=weight.in.kg,  
    height=height.in.m,  
    bmi=bmi)  
}
```

# Try out function

```
createPatientRecord("Michael Smith", 185, 12 * 6 + 1)
```

```
$first.name  
[1] "Michael"
```

```
$last.name  
[1] "Smith"
```

```
$weight  
[1] 84.09091
```

```
$height  
[1] 1.8542
```

```
$bmi  
[1] 24.45884
```

# Another example

- Calculate mean, median and standard deviation

```
threeNumberSummary <- function(x) {  
  c(mean=mean(x), median=median(x), sd=sd(x))  
}  
x <- rnorm(100, mean=5, sd=2) # Vector of 100 normals with mean 5 and sd 2  
threeNumberSummary(x)
```

mean	median	sd
4.926875	5.050984	1.953703

# If-else statement

- Oftentimes we want our code to have different effects depending on the features of the input
- Example: Calculating a student's letter grade
  - If grade  $\geq 90$ , assign A
  - Otherwise, if grade  $\geq 80$ , assign B
  - Otherwise, if grade  $\geq 70$ , assign C
  - In all other cases, assign F
- To code this up, we use if-else statements

# If-else example

```
calculateLetterGrade <- function(x) {  
  if(x >= 90) {  
    grade <- "A"  
  } else if(x >= 80) {  
    grade <- "B"  
  } else if(x >= 70) {  
    grade <- "C"  
  } else {  
    grade <- "F"  
  }  
  grade  
}  
  
course.grades <- c(92, 78, 87, 91, 62)  
sapply(course.grades, FUN=calculateLetterGrade)
```

```
[1] "A" "C" "B" "A" "F"
```

# return()

- In the previous examples we specified the output simply by writing the output variable as the last line of the function
- More explicitly, we can use the `return( )` function

```
addOne <- function(x) {  
  return(x + 1)  
}  
  
addOne(12)
```

```
[1] 13
```

- We will generally avoid the `return( )` function, but you can use it if necessary or if it makes writing a particular function easier.

# Workshop 4 : Transform

For the first two problems we'll use the Cars93 data set from the MASS library.

```
library(MASS)
```

## 1. Manipulating data frames

Use the **transform()** and **log()** functions to create a new data frame called Cars93.log that has MPG.highway and MPG.city replaced with log(MPG.highway) and log(MPG.city).

## 2. Functions, lists, and if-else

- (a) Write a function called **isPassingGrade** whose input x is a number, and which returns FALSE if x is lower than 50 and TRUE otherwise.
- (b) Write a function called **sendMessage** whose input x is a number, and which prints **Congratulations** if **isPassingGrade(x)** is TRUE and prints **Oh no!** if **isPassingGrade(x)** is FALSE.



# Data Wrangling and Visualization

# Common Problem

- One of the most common problems you'll encounter when importing manually-entered data is inconsistent data types within columns
- For a simple example, let's look at TVhours column in a messy version of the survey data

```
survey.messy <- read.csv("survey_messy.csv", header=TRUE)  
survey.messy$TVhours
```

```
[1] 1           8h          4           5  
[5] 3           ~10         0           10  
[9] 15 (incl movies) 4           0           0  
[13] 10          4           10          2hours  
[17] 5           3           0           1  
[21] 2           7           3           10  
[25] 6.5          40          0           12  
[29] adfjalkj     3           0           0  
16 Levels: ~10 0 1 10 12 15 (incl movies) 2 2hours 3 4 40 5 6.5 7 ... adfjalkj
```

# What are the problems?

```
str(survey.messy)
```

```
'data.frame': 31 obs. of 6 variables:  
 $ Program      : Factor w/ 3 levels "MISM","Other",...: 1 2 1 3 2 3 2 1 3 3 ...  
 $ PriorExp     : Factor w/ 3 levels "Extensive experience",...: 3 3 1 2 2 3 3 3  
 2 1 ...  
 $ Rexperience   : Factor w/ 3 levels "Basic competence",...: 3 1 1 2 3 3 3 3 3 3  
 ...  
 $ OperatingSystem: Factor w/ 2 levels "Mac OS X","Windows": 2 2 2 2 2 1 1 2 1 2  
 ...  
 $ TVhours       : Factor w/ 16 levels "-10","0","1",...: 3 15 10 12 9 1 2 4 6 10  
 ...  
 $ Editor        : Factor w/ 1 level "Microsoft Word": 1 1 1 1 1 1 1 1 1 1 ...
```

- Several of the entries have non-numeric values in them (they contain strings)
- As a result, TVhours is being imported as factor

# Fix the type

```
as.character(tv.hours.messy)[1:30]
```

```
[1] "1"          "8h"        "4"  
[4] "5"          "3"          "~10"  
[7] "0"          "10"         "15 (incl movies)"  
[10] "4"          "0"          "0"  
[13] "10"         "4"          "10"  
[16] "2hours"     "5"          "3"  
[19] "0"          "1"          "2"  
[22] "7"          "3"          "10"  
[25] "6.5"        "40"         "0"  
[28] "12"         "adfjalkj"   "3"
```

```
as.numeric(as.character(tv.hours.messy))[1:30]
```

```
[1] 1.0    NA    4.0   5.0   3.0   NA    0.0  10.0   NA   4.0   0.0  0.0  10.0  4.0  
[15] 10.0   NA    5.0   3.0   0.0   1.0   2.0  7.0    3.0  10.0  6.5  40.0  0.0  12.0  
[29]    NA    3.0
```

```
typeof(as.numeric(as.character(tv.hours.messy))) # Success!! (Almost...)
```

```
[1] "double"
```

# A small improvement

- All the corrupted cells now appear as NA, which is R's missing indicator
- We can do a little better by cleaning up the vector once we get it to character form

```
tv.hours.strings <- as.character(tv.hours.messy)  
tv.hours.strings
```

```
[1] "1"                 "8h"                "4"  
[4] "5"                 "3"                  "~10"  
[7] "0"                 "10"                "15 (incl movies)"  
[10] "4"                "0"                  "0"  
[13] "10"               "4"                  "10"  
[16] "2hours"           "5"                  "3"  
[19] "0"                 "1"                  "2"  
[22] "7"                 "3"                  "10"  
[25] "6.5"              "40"                "0"  
[28] "12"               "adfjalkj"         "3"  
[31] "0"
```

# Deleting non-numeric (or .) characters

```
tv.hours.strings
```

```
[1] "1"                 "8h"                "4"  
[4] "5"                 "3"                  "~10"  
[7] "0"                 "10"                "15 (incl movies)"  
[10] "4"                "0"                  "0"  
[13] "10"               "4"                  "10"  
[16] "2hours"           "5"                  "3"  
[19] "0"                 "1"                  "2"  
[22] "7"                 "3"                  "10"  
[25] "6.5"              "40"                "0"  
[28] "12"               "adfjalkj"          "3"  
[31] "0"
```

```
# Use gsub() to replace everything except digits and '.' with a blank ""  
gsub("[^0-9.]", "", tv.hours.strings)
```

```
[1] "1"    "8"    "4"    "5"    "3"    "10"   "0"    "10"   "15"   "4"    "0"  
[12] "0"   "10"   "4"    "10"   "2"    "5"    "3"    "0"    "1"    "2"    "7"  
[23] "3"   "10"   "6.5"  "40"   "0"    "12"   ""     "3"    "0"
```

# Redo

```
tv.hours.messy
```

```
[1] 1           8h          4           5
[5] 3           ~10         0           10
[9] 15 (incl movies) 4           0           0
[13] 10          4           10          2hours
[17] 5           3           0           1
[21] 2           7           3           10
[25] 6.5          40          0           12
[29] adfjalkj    3           0           ...
16 Levels: ~10 0 1 10 12 15 (incl movies) 2 2hours 3 4 40 5 6.5 7 ... adfjalkj
```

```
tv.hours.clean <- as.numeric(gsub("[^0-9.]", "", tv.hours.strings))
tv.hours.clean
```

```
[1] 1.0 8.0 4.0 5.0 3.0 10.0 0.0 10.0 15.0 4.0 0.0 0.0 10.0 4.0
[15] 10.0 2.0 5.0 3.0 0.0 1.0 2.0 7.0 3.0 10.0 6.5 40.0 0.0 12.0
[29] NA   3.0 0.0
```

# Another approach

- We can also handle this problem by setting `stringsAsFactors = FALSE` when importing our data.

```
survey.messy <- read.csv("survey_messy.csv", header=TRUE, stringsAsFactors=FALSE)
str(survey.messy)
```

```
'data.frame': 31 obs. of 6 variables:
 $ Program      : chr  "MISM" "Other" "MISM" "PPM" ...
 $ PriorExp     : chr  "Some experience" "Some experience" "Extensive
experience" "Never programmed before" ...
 $ Rexperience   : chr  "Never used" "Basic competence" "Basic competence"
"Installed on machine" ...
 $ OperatingSystem: chr  "Windows" "Windows" "Windows" "Windows" ...
 $ TVhours       : chr  "1" "8h" "4" "5" ...
 $ Editor        : chr  "Microsoft Word" "Microsoft Word" "Microsoft Word"
"Microsoft Word" ...
```

- Now everything is a character instead of a factor

# One-line Cleanup

- Let's clean up the `TVhours` column and cast it to numeric all in one command

```
survey <- transform(survey.messy, TVhours = as.numeric(gsub("[^0-9.]", "", TVhours)))
str(survey)
```

```
'data.frame': 31 obs. of 6 variables:
 $ Program      : chr  "MISM" "Other" "MISM" "PPM" ...
 $ PriorExp     : chr  "Some experience" "Some experience" "Extensive
experience" "Never programmed before" ...
 $ Rexperience   : chr  "Never used" "Basic competence" "Basic competence"
"Installed on machine" ...
 $ OperatingSystem: chr  "Windows" "Windows" "Windows" "Windows" ...
 $ TVhours       : num  1 8 4 5 3 10 0 10 15 4 ...
 $ Editor        : chr  "Microsoft Word" "Microsoft Word" "Microsoft Word"
"Microsoft Word" ...
```

# What about other character column?

```
table(survey[["Program"]])
```

MISM	Other	PPM
10	6	15

```
table(as.factor(survey[["Program"]]))
```

MISM	Other	PPM
10	6	15

- Having factors coded as characters may be OK for many parts of our analysis

# Let's fix it

```
# Figure out which columns are coded as characters  
chr.indexes <- sapply(survey, FUN = is.character)  
chr.indexes
```

Program	PriorExp	Rexperience	OperatingSystem
TRUE	TRUE	TRUE	TRUE
TVhours	Editor		
FALSE	TRUE		

```
# Re-code all of the character columns to factors  
survey[chr.indexes] <- lapply(survey[chr.indexes], FUN = as.factor)
```

# Here is the outcome

```
str(survey)
```

```
'data.frame': 31 obs. of 6 variables:  
 $ Program      : Factor w/ 3 levels "MISM","Other",...: 1 2 1 3 2 3 2 1 3 3 ...  
 $ PriorExp     : Factor w/ 3 levels "Extensive experience",...: 3 3 1 2 2 3 3 3  
 2 1 ...  
 $ Rexperience   : Factor w/ 3 levels "Basic competence",...: 3 1 1 2 3 3 3 3 3 3  
 ...  
 $ OperatingSystem: Factor w/ 2 levels "Mac OS X","Windows": 2 2 2 2 2 1 1 2 1 2  
 ...  
 $ TVhours       : num  1 8 4 5 3 10 0 10 15 4 ...  
 $ Editor        : Factor w/ 1 level "Microsoft Word": 1 1 1 1 1 1 1 1 1 1 ...
```

- Success!

# Another example

- When data is entered manually, misspellings and case changes are very common
- E.g., a column showing life support mechanism may look like,

```
life.support <- as.factor(c("dialysis", "Ventilation", "Dialysis", "dialysis",
  "none", "None", "nnone", "dyalysis", "dialysis", "ventilation", "none"))
summary(life.support)
```

dialysis	Dialysis	dyalysis	nnone	none	None
3	1	1	1	2	1
ventilation	Ventilation				
1	1				

```
summary(life.support)
```

dialysis	Dialysis	dyalysis	nnone	none	None
3	1	1	1	2	1
ventilation	Ventilation				
1	1				

- This factor has 8 levels even though it should have 3 (dialysis, ventilation, none)
- We can fix many of the typos by running spellcheck in Excel before importing data, or by changing the values on a case-by-case basis later

# [l/s/t/]apply() functions?

- These are all efficient ways of applying a function to margins of an array or elements of a list
- Before we talk about the details of `apply()` and its relatives, we should first understand loops
- **loops** are ways of iterating over data
- The `apply()` functions can be thought of as good *alternatives* to loops

# Loop

```
for(i in 1:4) {  
  print(i)  
}
```

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

```
phrase <- "Good Night, "  
for(word in c("and", "Good", "Luck")) {  
  phrase <- paste(phrase, word)  
  print(phrase)  
}
```

```
[1] "Good Night, and"  
[1] "Good Night, and Good"  
[1] "Good Night, and Good Luck"
```

# Loop Syntax

A **for loop** executes a chunk of code for every value of an **index variable** in an **index set**

- The basic syntax takes the form

```
for(index.variable in index.set) {  
    code to be repeated at every value of index.variable  
}
```

- The index set is often a vector of integers, but can be more general

# Example

```
index.set <- rnorm(4) # 4 random standard normal variables  
index.set
```

```
[1] -1.9912818 0.7119512 0.3999575 0.5872188
```

```
for(i in index.set) {  
  print(abs(i))  
}
```

```
[1] 1.991282  
[1] 0.7119512  
[1] 0.3999575  
[1] 0.5872188
```

# Example

```
index.set <- list(name="Michael", weight=185, is.male=TRUE) # a list
for(i in index.set) {
  print(c(i, typeof(i)))
}
```

```
[1] "Michael"    "character"
[1] "185"        "double"
[1] "TRUE"       "logical"
```

# Example

```
fake.data <- matrix(rnorm(500), ncol=5) # create fake 100 x 5 data set  
head(fake.data,2) # print first two rows
```

```
 [,1]      [,2]      [,3]      [,4]      [,5]  
[1,] 0.9551196 -0.3985364 -0.2338078 -1.018674 -0.684313  
[2,] 0.6483192 -0.5487889  0.2091660  2.016935 -1.030668
```

```
col.sums <- numeric(ncol(fake.data)) # variable to store running column sums  
for(i in 1:nrow(fake.data)) {  
  col.sums <- col.sums + fake.data[i,] # add ith observation to the sum  
}  
print(col.sums)
```

```
[1] -12.808804 -11.552563   3.211727   4.049877 -9.901475
```

```
colSums(fake.data) # A better approach (see also colMeans())
```

```
[1] -12.808804 -11.552563   3.211727   4.049877 -9.901475
```

# While loop

- **while loops** repeat a chunk of code while the specified condition remains true

```
day <- 1
num.days <- 365
while(day <= num.days) {
  day <- day + 1
}
```

- We won't really be using while loops in this class

# Various apply() function

Command	Description
<code>apply(x, MARGIN, FUN)</code>	Obtain a vector/array/list by applying <code>FUN</code> along the specified <code>MARGIN</code> of an array or matrix <code>x</code>
<code>lapply(x, FUN)</code>	Obtain a list by applying <code>FUN</code> to the elements of a list <code>x</code>
<code>sapply(x, FUN)</code>	Simplified version of <code>lapply</code> . Returns a vector/array instead of list.
<code>tapply(x, INDEX, FUN)</code>	Obtain a table by applying <code>FUN</code> to each combination of the factors given in <code>INDEX</code>

- These functions are (good!) alternatives to loops
- They are typically *more efficient* than loops (often run considerably faster on large data sets)
- Take practice to get used to, but make analysis easier to debug and less prone to error when used effectively

# Example: apply()

```
colMeans(fake.data)
```

```
[1] -0.12808804 -0.11552563  0.03211727  0.04049877 -0.09901475
```

```
apply(fake.data, MARGIN=2, FUN=mean) # MARGIN = 1 for rows, 2 for columns
```

```
[1] -0.12808804 -0.11552563  0.03211727  0.04049877 -0.09901475
```

```
# Function that calculates proportion of vector indexes that are > 0
propPositive <- function(x) mean(x > 0)
apply(fake.data, MARGIN=2, FUN=propPositive)
```

```
[1] 0.51 0.47 0.51 0.51 0.46
```

# Example: lapply(), sapply()

```
lapply(survey, is.factor) # Returns a list
```

```
$Program  
[1] TRUE  
  
$PriorExp  
[1] TRUE  
  
$Rexperience  
[1] TRUE  
  
$OperatingSystem  
[1] TRUE  
  
$TVhours  
[1] FALSE  
  
$Editor  
[1] TRUE
```

```
sapply(survey, FUN = is.factor) # Returns a vector with named elements
```

Program	PriorExp	Rexperience	OperatingSystem
TRUE	TRUE	TRUE	TRUE
TVhours	Editor		
FALSE	TRUE		

# Example: apply, lapply, sapply

```
apply(cars, 2, FUN=mean) # Data frames are arrays
```

```
speed dist  
15.40 42.98
```

```
lapply(cars, FUN=mean) # Data frames are also lists
```

```
$speed  
[1] 15.4
```

```
$dist  
[1] 42.98
```

```
sapply(cars, FUN=mean) # sapply() is just simplified lapply()
```

```
speed dist  
15.40 42.98
```

# tapply()

- Think of tapply( ) as a generalized form of the table( ) function

```
library(MASS)
# Get a count table, data broken down by Origin and DriveTrain
table(Cars93$Origin, Cars93$DriveTrain)
```

	4WD	Front	Rear
USA	5	34	9
non-USA	5	33	7

```
# Calculate average MPG.City, broken down by Origin and Drivetrain
tapply(Cars93$MPG.city, INDEX = Cars93[c("Origin", "DriveTrain")], FUN=mean)
```

Origin	DriveTrain		
	4WD	Front	Rear
USA	17.6	22.14706	18.33333
non-USA	23.4	24.93939	19.14286

# Example: tapply()

- Let's get the average horsepower by car Origin and Type

```
tapply(Cars93[["Horsepower"]], INDEX = Cars93[c("Origin", "Type")], FUN=mean)
```

Origin	Type					
	Compact	Large	Midsize	Small	Sporty	Van
USA	117.4286	179.4545	153.5000	89.42857	166.5000	158.40
non-USA	141.5556	NA	189.4167	91.78571	151.6667	138.25

- What's that NA doing there?

```
any(Cars93$Origin == "non-USA" & Cars93>Type == "Large")
```

```
[1] FALSE
```

- None of the non-USA manufacturers produced Large cars!

# with()

- Thus far we've repeatedly typed out the data frame name when referencing its columns
- This is because the data variables don't exist in our working environment
- Using **with**(*data*, *expr*) lets us specify that the code in *expr* should be evaluated in an environment that contains the elements of *data* as variables

```
with(Cars93, table(Origin, Type))
```

Origin	Type					
	Compact	Large	Midsize	Small	Sporty	Van
USA	7	11	10	7	8	5
non-USA	9	0	12	14	6	4

# Example: with()

```
any(Cars93$Origin == "non-USA" & Cars93>Type == "Large")
```

```
[1] FALSE
```

```
with(Cars93, any(Origin == "non-USA" & Type == "Large")) # Same effect!
```

```
[1] FALSE
```

```
with(Cars93, tapply(Horsepower, INDEX = list(Origin, Type), FUN=mean))
```

	Compact	Large	Midsize	Small	Sporty	Van
USA	117.4286	179.4545	153.5000	89.42857	166.5000	158.40
non-USA	141.5556	NA	189.4167	91.78571	151.6667	138.25

- Using `with()` makes code simpler, easier to read, and easier to debug

# Workshop 5: Loop

## Loop practice

- (a) Write a function called `calculateRowMeans` that uses a `for` loop to calculate the row means of a matrix `x`.
- (b) Try out your function on the random matrix `fake.data` defined below.
- (b) Use the `apply()` function to calculate the row means of the matrix `fake.data`
- (c) Compare this to the output of the `rowMeans()` function to check that your calculation is correct.



# Data Summary and Standard Graphic

# Let's get started

- We're going to start by operating on the `birthwt` dataset from the MASS library
- Let's get it loaded and see what we're working with

```
library(MASS)
str(birthwt)
```

```
'data.frame': 189 obs. of 10 variables:
 $ low   : int  0 0 0 0 0 0 0 0 0 ...
 $ age   : int  19 33 20 21 18 21 22 17 29 26 ...
 $ lwt   : int  182 155 105 108 107 124 118 103 123 113 ...
 $ race  : int  2 3 1 1 1 3 1 3 1 1 ...
 $ smoke : int  0 0 1 1 1 0 0 0 1 1 ...
 $ ptl   : int  0 0 0 0 0 0 0 0 0 ...
 $ ht    : int  0 0 0 0 0 0 0 0 0 ...
 $ ui   : int  1 0 0 1 1 0 0 0 0 0 ...
 $ ftv   : int  0 3 1 2 0 0 1 1 1 0 ...
 $ bwt   : int  2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
```

# Renaming columns

- The dataset doesn't come with very descriptive variable names
- Let's get better column names (use `help(birthwt)` to understand the variables and come up with better names)

```
colnames(birthwt)
```

```
[1] "low"    "age"    "lwt"    "race"   "smoke"  "ptl"    "ht"    "ui"  
[9] "ftv"    "bwt"
```

```
# The default names are not very descriptive  
  
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",  
  "race", "mother.smokes", "previous.prem.labor", "hypertension",  
  "uterine.irr",  
  "physician.visits", "birthwt.grams")  
  
# Better names!
```

# Renaming the variables

- The dataset doesn't come with very descriptive variable names
- Let's get better column names (use `help(birthwt)` to understand the variables and come up with better names)

```
colnames(birthwt)
```

```
[1] "low"     "age"      "lwt"      "race"     "smoke"    "ptl"      "ht"       "ui"  
[9] "ftv"     "bwt"
```

```
# The default names are not very descriptive  
  
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",  
  "race", "mother.smokes", "previous.prem.labor", "hypertension",  
  "uterine.irr",  
  "physician.visits", "birthwt.grams")  
  
# Better names!
```

# Renaming the factors

- All the factors are currently represented as integers
- Let's use the `transform()` and `mapvalues()` functions to convert variables to factors and give the factors more meaningful levels

```
library(plyr)
birthwt <- transform(birthwt,
                      race = as.factor(mapvalues(race, c(1, 2, 3),
                                                  c("white", "black", "other"))),
                      mother.smokes = as.factor(mapvalues(mother.smokes,
                                                 c(0,1), c("no", "yes"))),
                      hypertension = as.factor(mapvalues(hypertension,
                                                 c(0,1), c("no", "yes"))),
                      uterine.irr = as.factor(mapvalues(uterine.irr,
                                                 c(0,1), c("no", "yes"))),
                      birthwt.below.2500 = as.factor(mapvalues(birthwt.below.2500,
                                                 c(0,1), c("no", "yes"))))
)
```

# Summary of the data

- Now that things are coded correctly, we can look at an overall summary

```
summary(birthwt)
```

```
birthwt.below.2500    mother.age      mother.weight     race
no :130                 Min.   :14.00     Min.   : 80.0    black:26
yes: 59                1st Qu.:19.00    1st Qu.:110.0   other:67
                           Median :23.00    Median :121.0   white:96
                           Mean   :23.24    Mean   :129.8
                           3rd Qu.:26.00    3rd Qu.:140.0
                           Max.   :45.00    Max.   :250.0
mother.smokes previous.prem.labor hypertension uterine.irr
no :115                 Min.   :0.0000    no :177       no :161
yes: 74                1st Qu.:0.0000    yes: 12      yes: 28
                           Median :0.0000
                           Mean   :0.1958
                           3rd Qu.:0.0000
                           Max.   :3.0000
physician.visits birthwt.grams
Min.   :0.0000    Min.   : 709
1st Qu.:0.0000    1st Qu.:2414
Median :0.0000    Median :2977
Mean   :0.7937    Mean   :2945
3rd Qu.:1.0000    3rd Qu.:3487
Max.   :6.0000    Max.   :4990
```

# A simple table

- Let's use the `tapply()` function to see what the average birthweight looks like when broken down by race and smoking status

```
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN =  
mean))
```

	no	yes
black	2854.500	2504.000
other	2815.782	2757.167
white	3428.750	2826.846

# Output table

- Let's use the header `{r, results='asis'}`, along with the `kable()` function from the `knitr` library

```
library(knitr)
kable(with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN
= mean)), format = "markdown")
```

	no	yes
black	2854.500	2504.000
other	2815.782	2757.167
white	3428.750	2826.846

- `kable()` outputs the table in a way that Markdown can read and nicely display
- Note: changing the CSS changes the table appearance

# aggregate() function

- Let's first recall what `tapply()` does
- Command: `tapply(X, INDEX, FUN)`
  - Applies `FUN` to `X` grouped by factors in `INDEX`
- `aggregate()` performs a similar operation, but presents the results in a form that is at times more convenient
- There are many ways to call the `aggregate()` function
- Analog of `tapply` call: `aggregate(X, by, FUN)`
  - Here, `by` is exactly like `INDEX`

# Example: tapply vs aggregate

```
library(MASS)
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN =
mean)) # tapply
```

	no	yes
black	2854.500	2504.000
other	2815.782	2757.167
white	3428.750	2826.846

```
with(birthwt, aggregate(birthwt.grams, by = list(race, mother.smokes), FUN =
mean)) # aggregate
```

	Group.1	Group.2	x
1	black	no	2854.500
2	other	no	2815.782
3	white	no	3428.750
4	black	yes	2504.000
5	other	yes	2757.167
6	white	yes	2826.846

# Different syntax

- Here's a convenient alternative way to call aggregate
- It uses the R formula syntax, which we'll learn more about when we discuss regression

```
aggregate(birthwt.grams ~ race + mother.smokes, FUN=mean, data=birthwt)
```

	race	mother.smokes	birthwt.grams
1	black	no	2854.500
2	other	no	2815.782
3	white	no	3428.750
4	black	yes	2504.000
5	other	yes	2757.167
6	white	yes	2826.846

- We'll see later that aggregate output can be more convenient for plotting

```
weight.smoke.tbl <- with(birthwt, table(birthwt.below.2500, mother.smokes))  
weight.smoke.tbl
```

		mother.smokes	
		no	yes
birthwt.below.2500	no	86	44
	yes	29	30

- The odds of low bwt among non-smoking mothers is

```
or.smoke.bwt <- (weight.smoke.tbl[2,2] / weight.smoke.tbl[1,2]) /  
(weight.smoke.tbl[2,1] / weight.smoke.tbl[1,1])  
or.smoke.bwt
```

```
[1] 2.021944
```

- So the odds of low birth weight are 2 times higher when the mother smokes

- Is the mother's age correlated with birth weight?

```
with(birthwt, cor(birthwt.grams, mother.age)) # Calculate correlation
```

```
[1] 0.09031781
```

- Does this change when we account for smoking status?

```
with(birthwt, cor(birthwt.grams[mother.smokes == "yes"],  
mother.age[mother.smokes == "yes"]))
```

```
[1] -0.1441649
```

```
with(birthwt, cor(birthwt.grams[mother.smokes == "no"], mother.age[mother.smokes  
== "no"]))
```

```
[1] 0.2014558
```

# by() function

- Think of the `by(data, INDICES, FUN)` function as a `tapply()` function that operates on data frames instead of just vectors
- When using `tapply(X, INDEX, FUN)`, `X` is generally a numeric vector
- To calculate correlations, we need to allow `X` to be a data frame or matrix

```
by(data = birthwt[c("birthwt.grams", "mother.age")],  
   INDICES = birthwt["mother.smokes"],  
   FUN = function(x) {cor(x[,1], x[,2])})
```

```
mother.smokes: no  
[1] 0.2014558  
-----  
mother.smokes: yes  
[1] -0.1441649
```

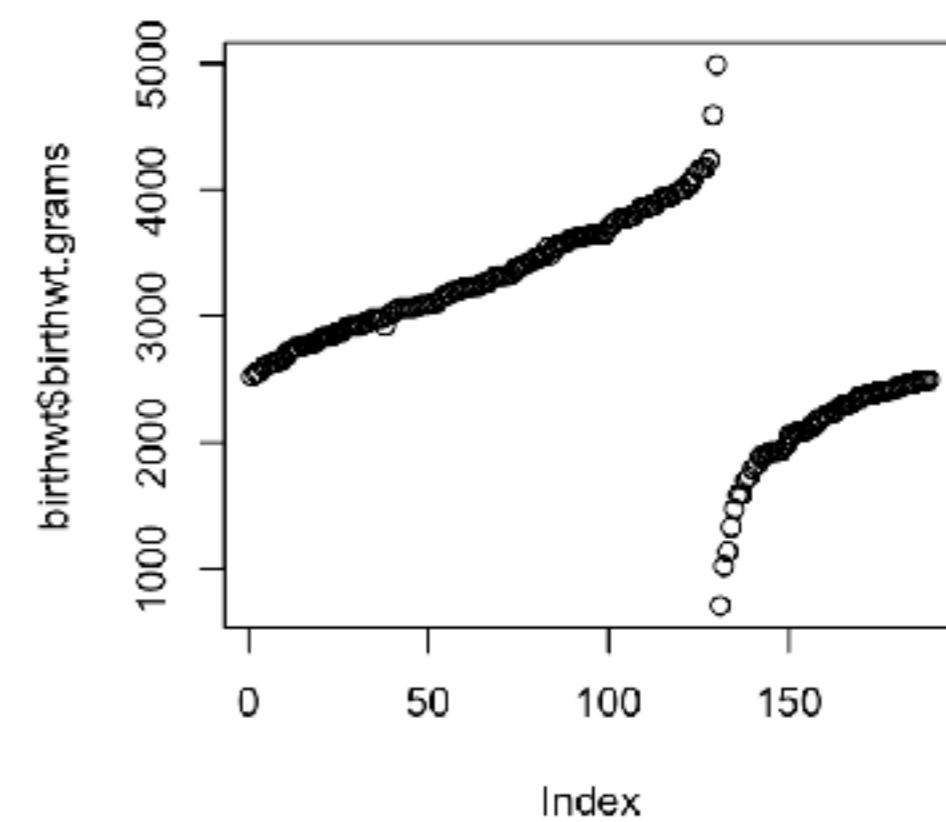
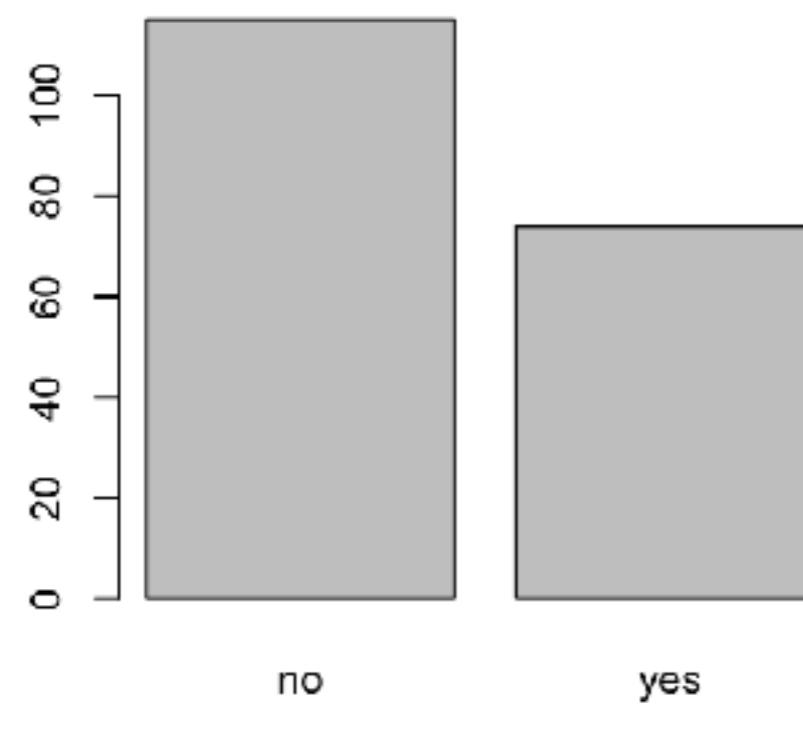
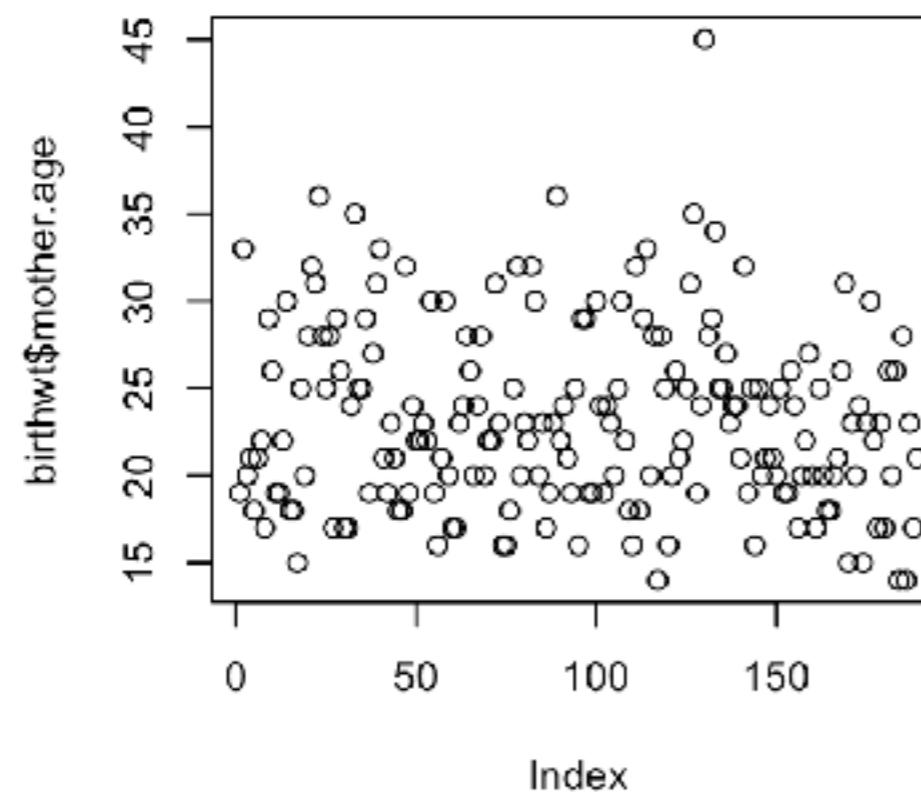
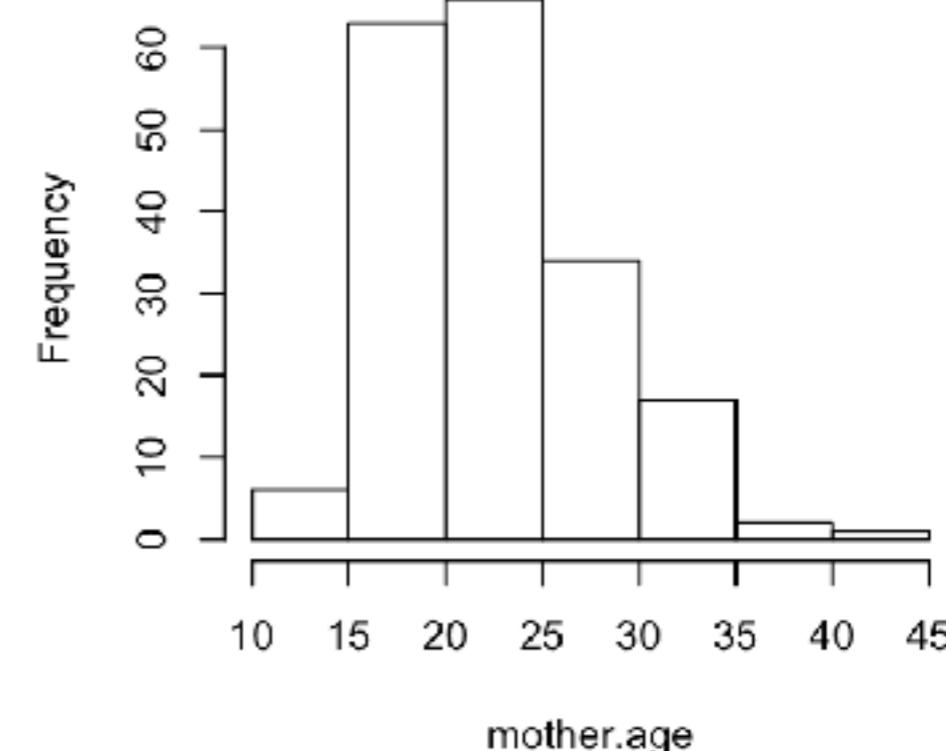
# Basic single plot

Let's continue with the `birthwt` data from the `MASS` library.

Here are some basic single-variable plots.

```
par(mfrow = c(2,2)) # Display plots in a single 2 x 2 figure
plot(birthwt$mother.age)
with(birthwt, hist(mother.age))
plot(birthwt$mother.smokes)
plot(birthwt$birthwt.grams)
```

Histogram of mother.age

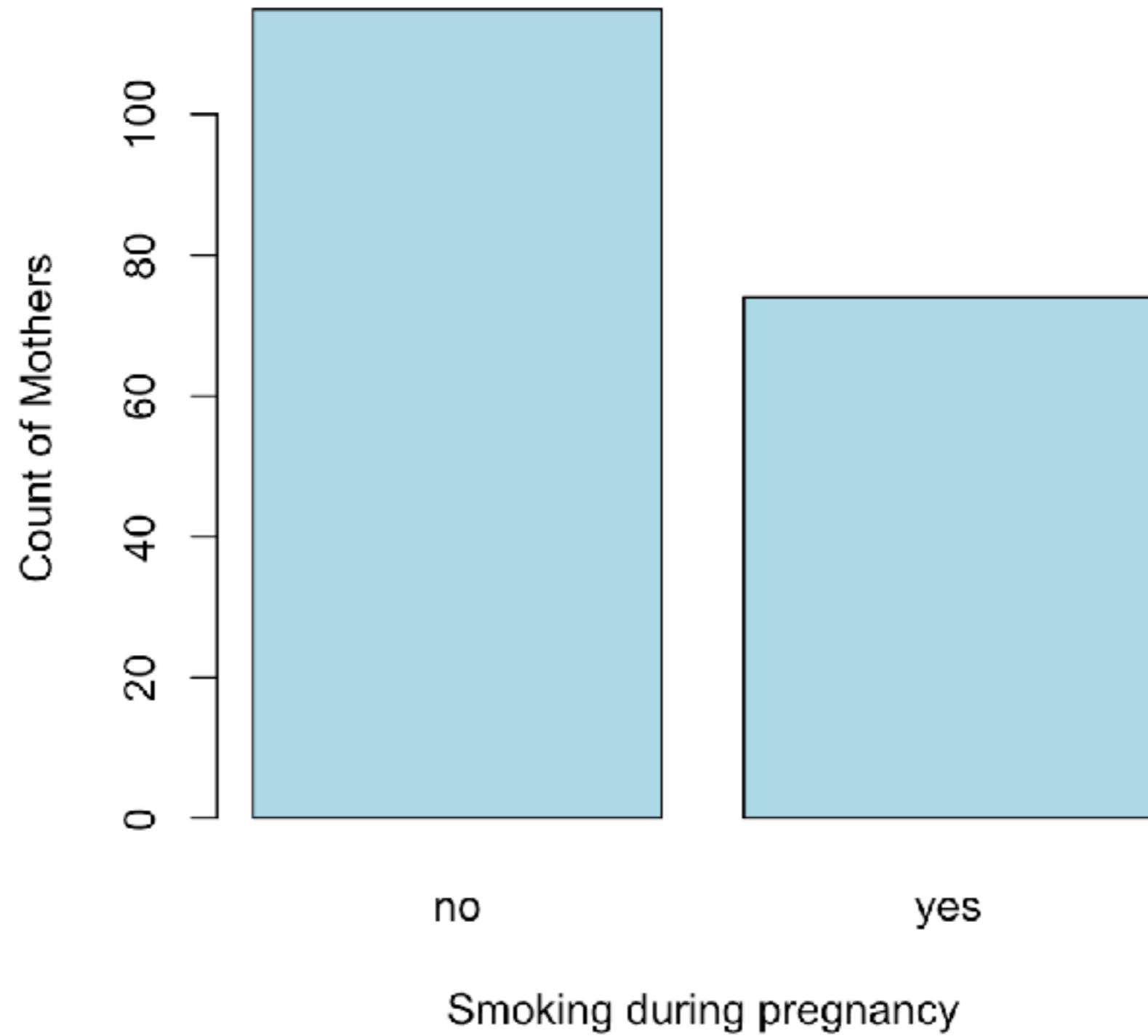


# Another example

Let's add more information to the smoking bar plot, and also change the color by setting the `col` option.

```
plot(birthwt$mother.smokes,  
      main = "Mothers Who Smoked In Pregnancy",  
      xlab = "Smoking during pregnancy",  
      ylab = "Count of Mothers",  
      col = 'lightblue')
```

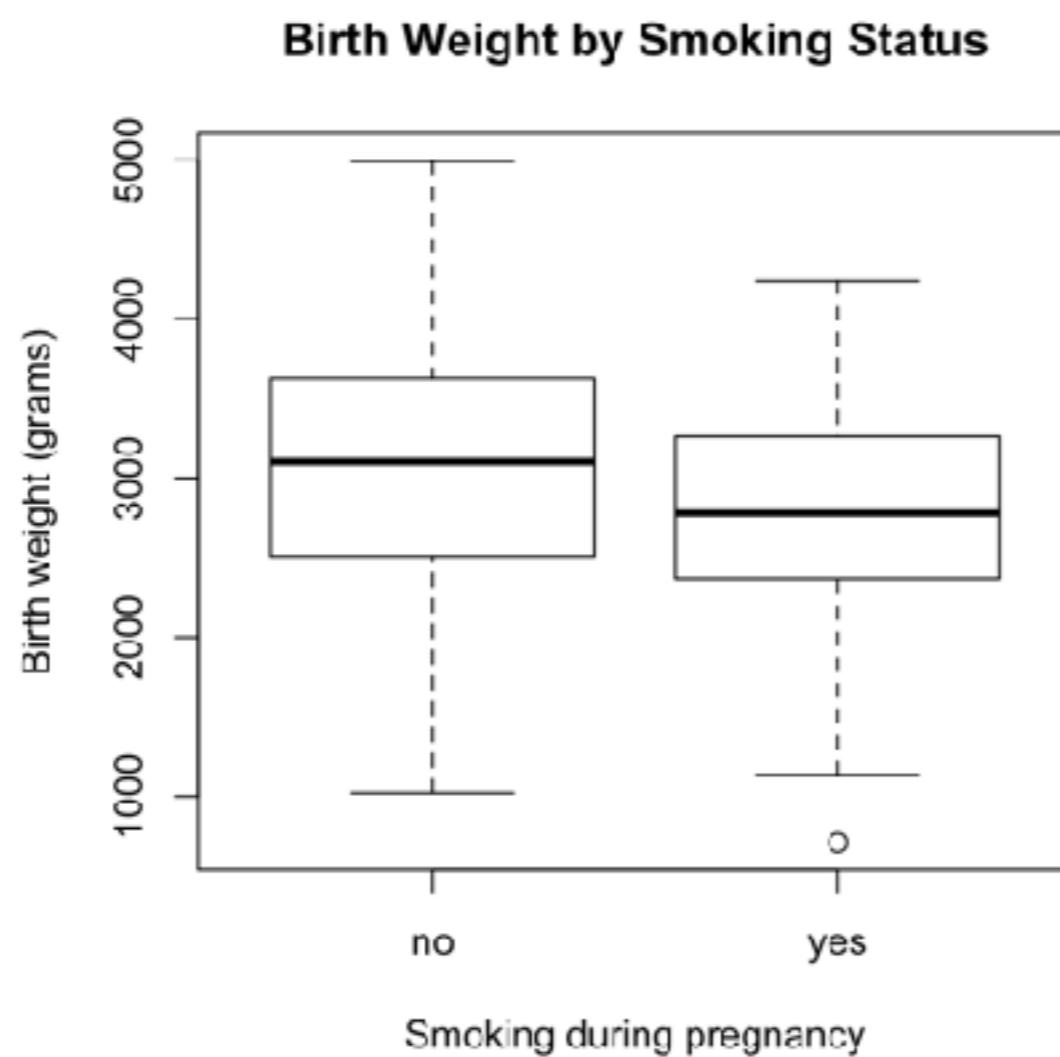
## Mothers Who Smoked In Pregnancy



# Plots with several variables

If we call `plot(x, y, ...)` with `x` a factor and `y` numeric, R will produce boxplots of `y` at every level of `x`.

```
with(birthwt, plot(mother.smokes, birthwt.grams,  
                    main = "Birth Weight by Smoking Status",  
                    xlab = "Smoking during pregnancy",  
                    ylab = "Birth weight (grams)"))
```



# Plotting using ggplot2

# What is ggplot2?

- An implementation of the Grammar of Graphics by Leland Wilkinson
- Written by Hadley Wickham (while he was a graduate student at Iowa State)
- A “third” graphics system for R (along with base and lattice)
- Available from CRAN via `install.packages()`
- Web site: <http://ggplot2.org> (better documentation)

# What is ggplot2?

- Grammar of graphics represents and abstraction of graphics ideas/objects
- Think “verb”, “noun”, “adjective” for graphics
- Allows for a “theory” of graphics on which to build new graphics and graphics objects
- “Shorten the distance from mind to page”

# Grammar of Graphics

- “In brief, the grammar tells us that a statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also contain statistical transformations of the data and is drawn on a specific coordinate system”
- from ggplot2 book

# The Basics: qplot()

- Works much like the plot function in base graphics system
- Looks for data in a data frame, similar to lattice, or in the parent environment
- Plots are made up of aesthetics (size, shape, color) and geoms (points, lines)

# The Basics: qplot()

Factors are important for indicating subsets of the data (if they are to have different properties); they should be labeled

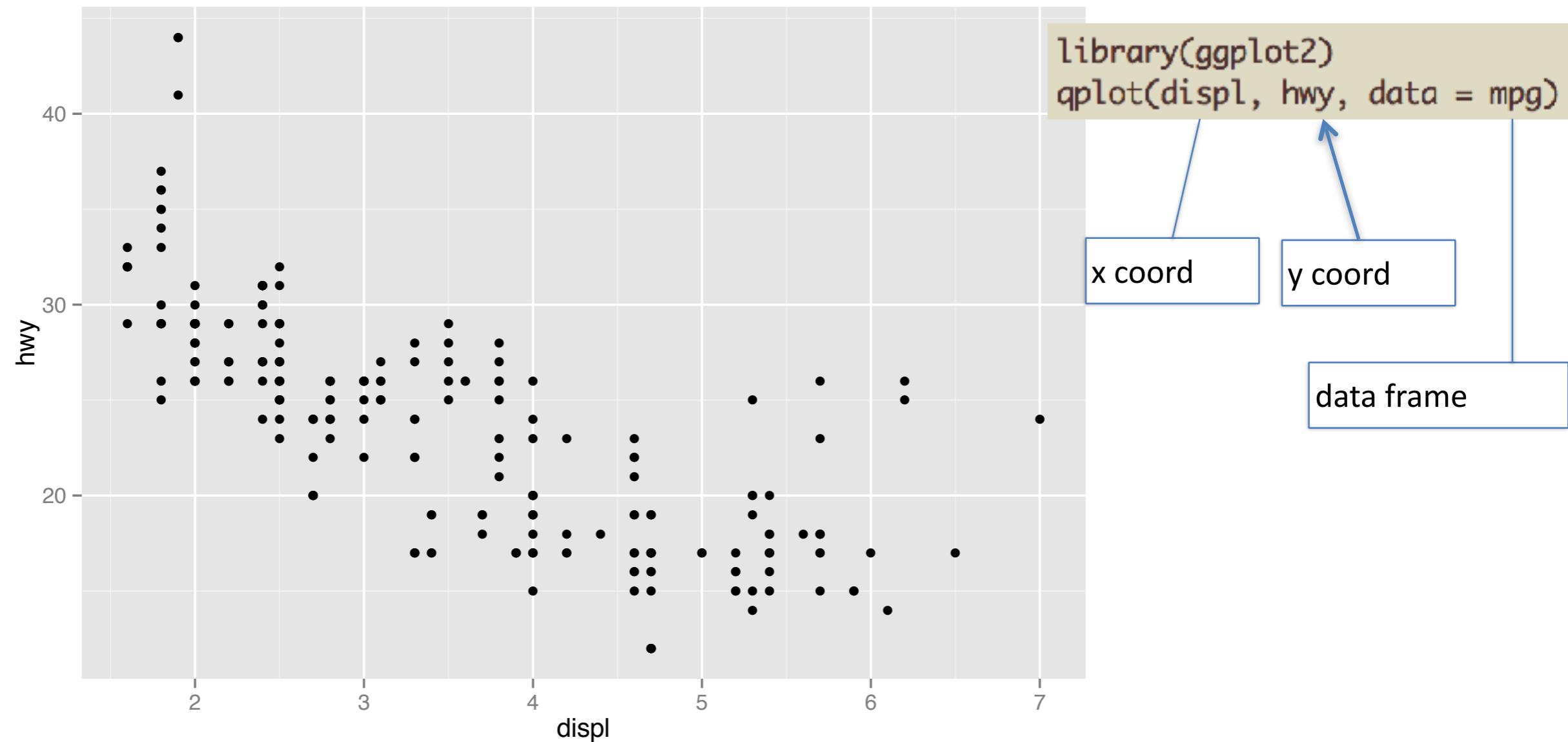
The qplot() hides what goes on underneath, which is okay for most operations

ggplot() is the core function and very flexible for doing things qplot() cannot do

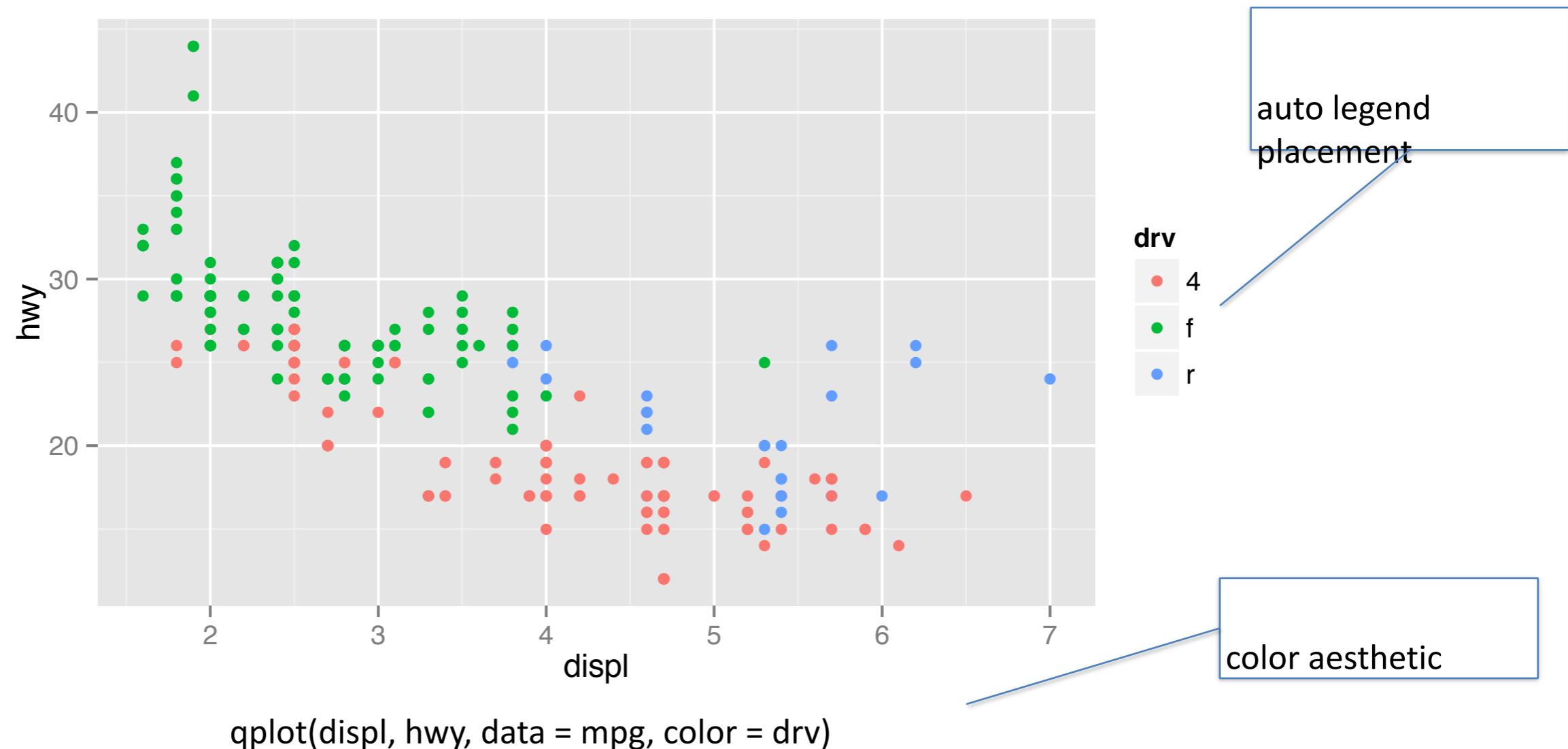
# Example Dataset

```
> library(ggplot2)
> str(mpg)
'data.frame': 234 obs. of 11 variables:
 $ manufacturer: Factor w/ 15 levels "audi","chevrolet",...: 1 1 1 1 1 1 1 1 1 ...
 $ model       : Factor w/ 38 levels "4runner 4wd",...: 2 2 2 2 2 2 2 3 3 3 ...
 $ displ        : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year         : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl          : int  4 4 4 4 6 6 6 4 4 4 ...
 $ trans        : Factor w/ 10 levels "auto(av)","auto(l3)",...: 4 9 10 1 4 9 1 9 4 10 ...
 ...
 $ drv          : Factor w/ 3 levels "4","f","r": 2 2 2 2 2 2 2 1 1 1 ...
 $ cty          : int  18 21 20 21 16 18 18 18 16 20 ...
 $ hwy          : int  29 29 31 30 26 26 27 26 25 28 ...
 $ fl           : Factor w/ 5 levels "c","d","e","p",...: 4 4 4 4 4 4 4 4 4 4 ...
 $ class        : Factor w/ 7 levels "2seater","compact",...: 2 2 2 2 2 2 2 2 2 2 ...
```

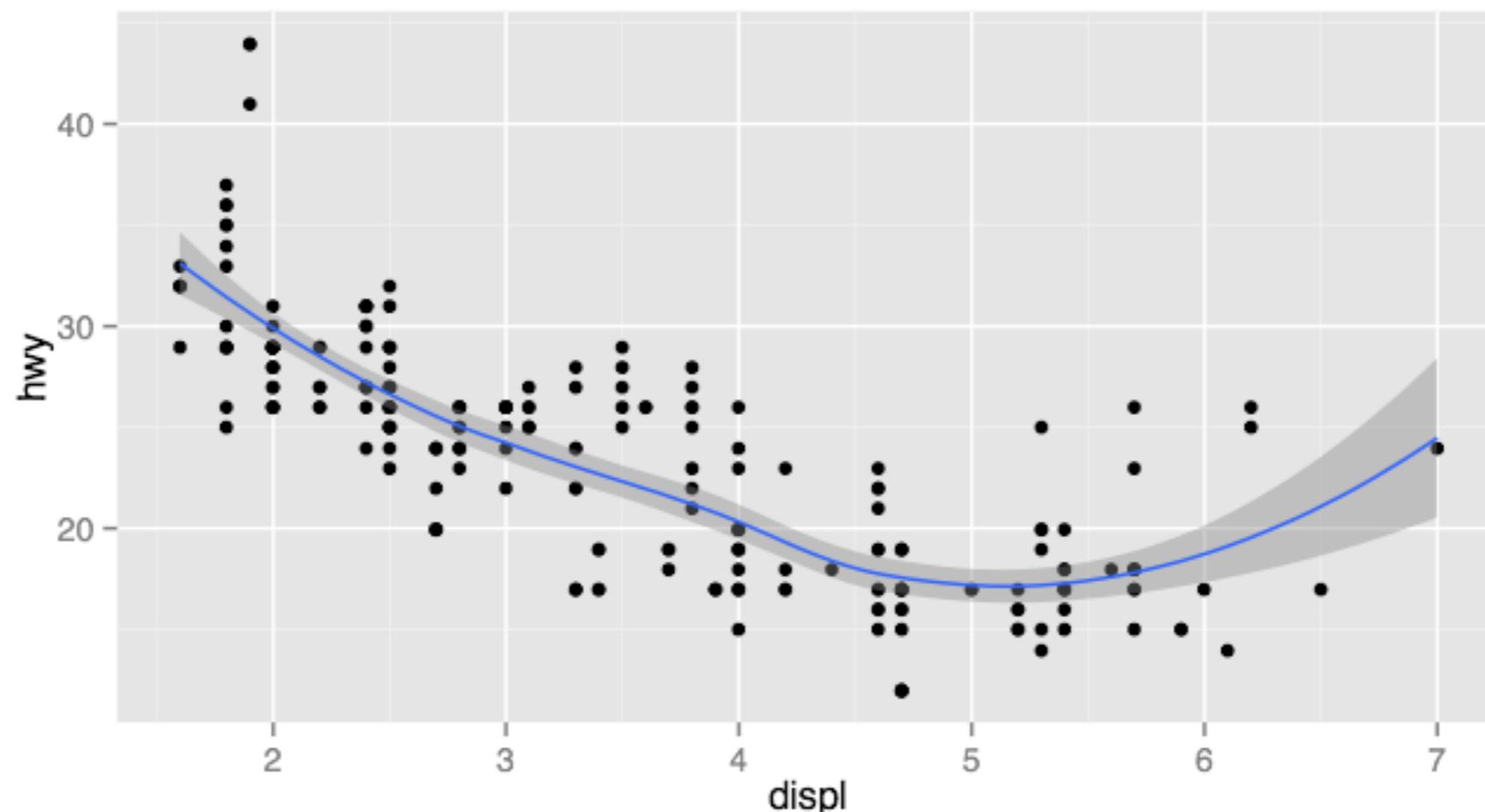
# ggplot2 “Hello, world!”



# Modifying aesthetics

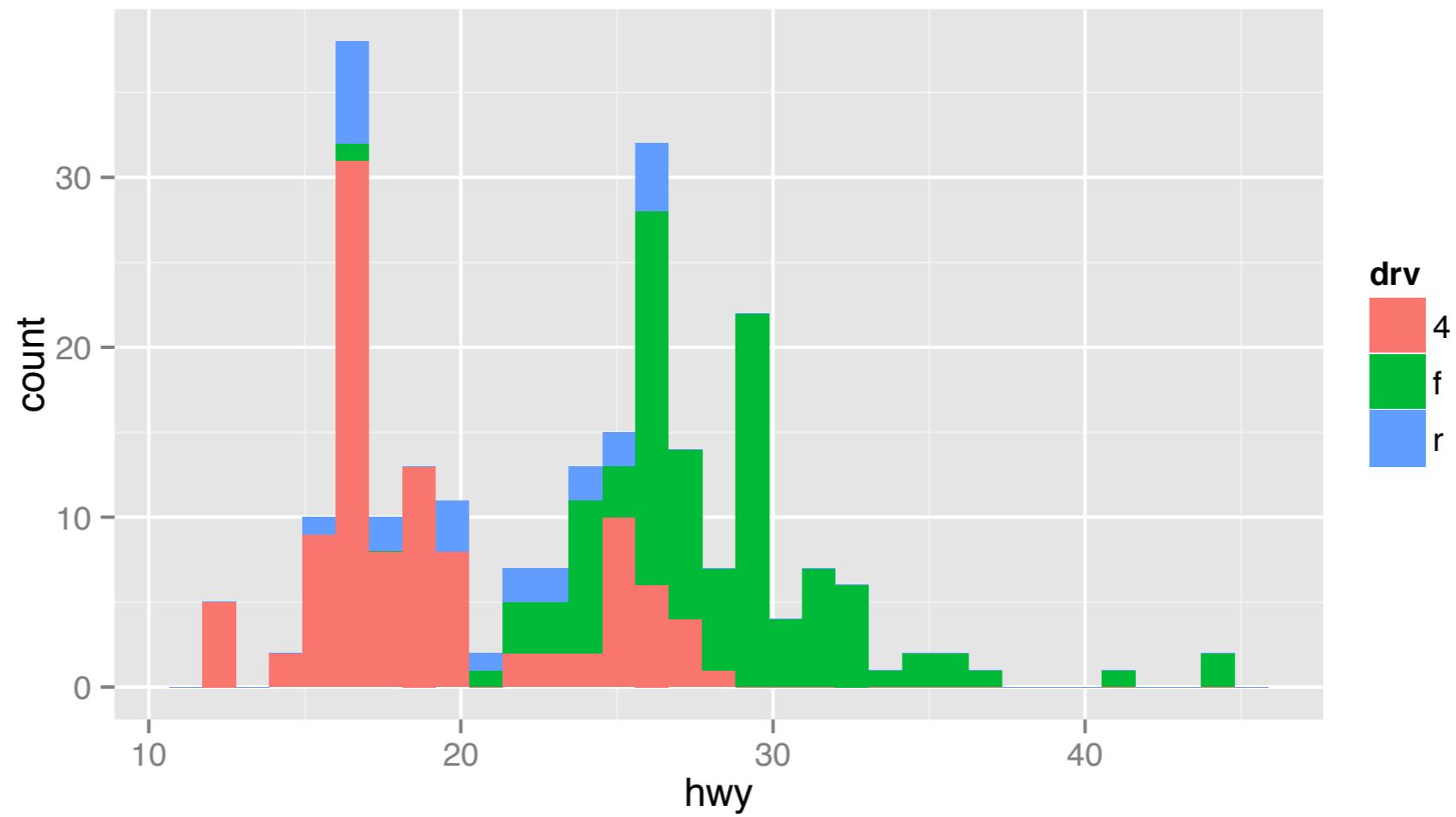


# Adding a geom



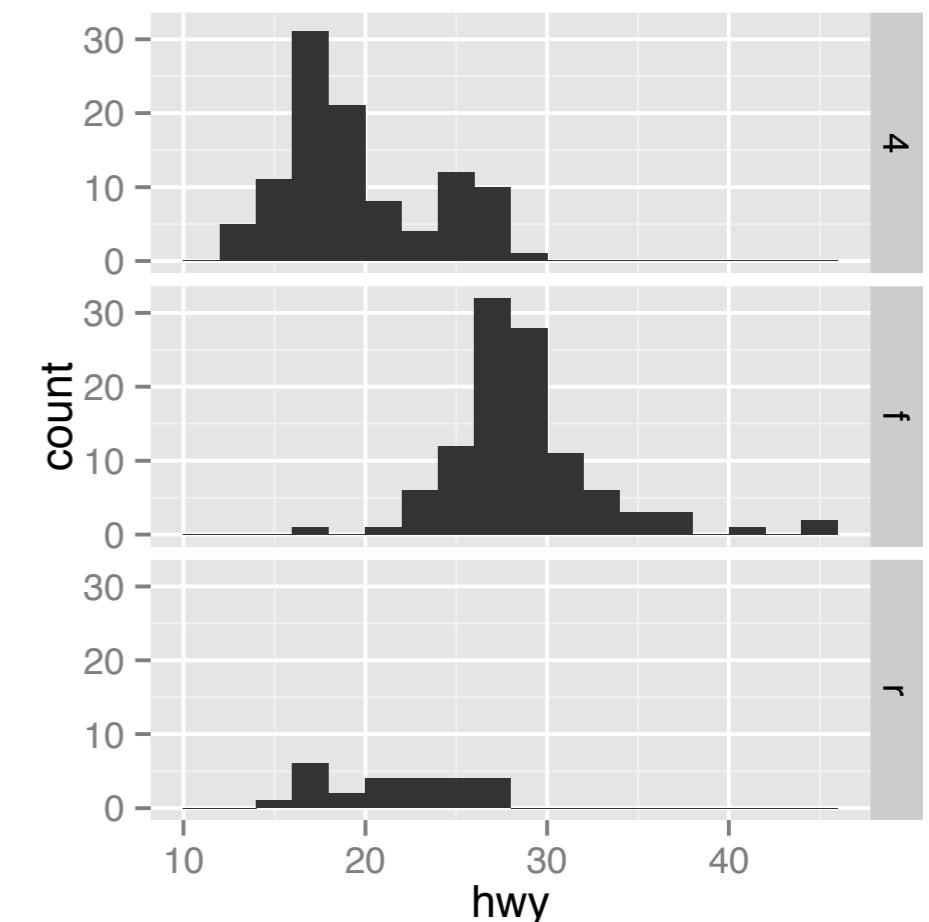
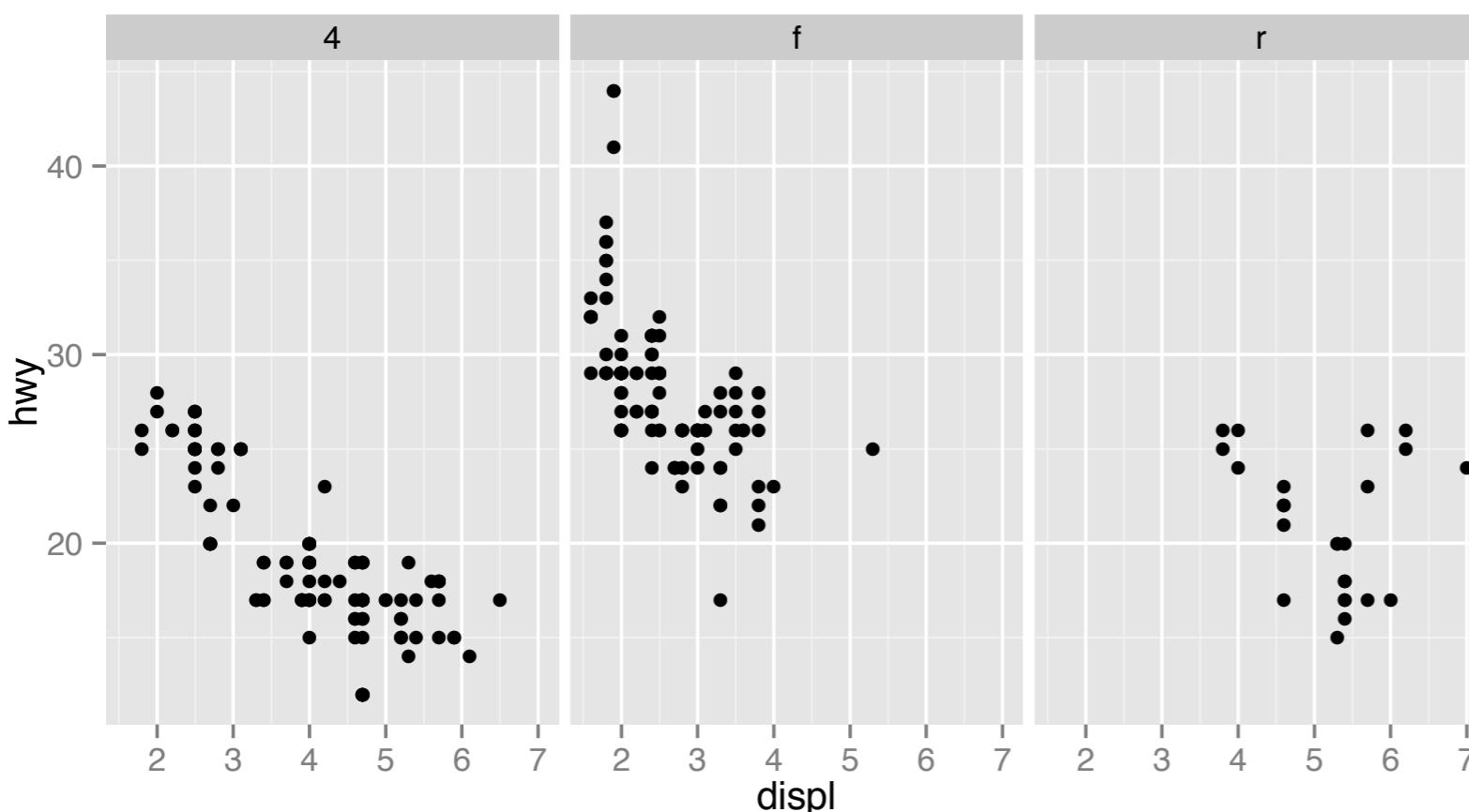
```
qplot(displ, hwy, data = mpg, geom = c("point", "smooth"))
```

# Histograms



# Facets

```
qplot(displ, hwy, data = mpg, facets = . ~ drv)
```



```
qplot(hwy, data = mpg, facets = drv ~ ., binwidth = 2)
```

# MAACS Cohort

- Mouse Allergen and Asthma Cohort Study
- Baltimore children (aged 5—17)
- Persistent asthma, exacerbation in past year
- Study indoor environment and its relationship with asthma morbidity
- Recent publication: <http://goo.gl/WqE9j8>

# Example: MAACS

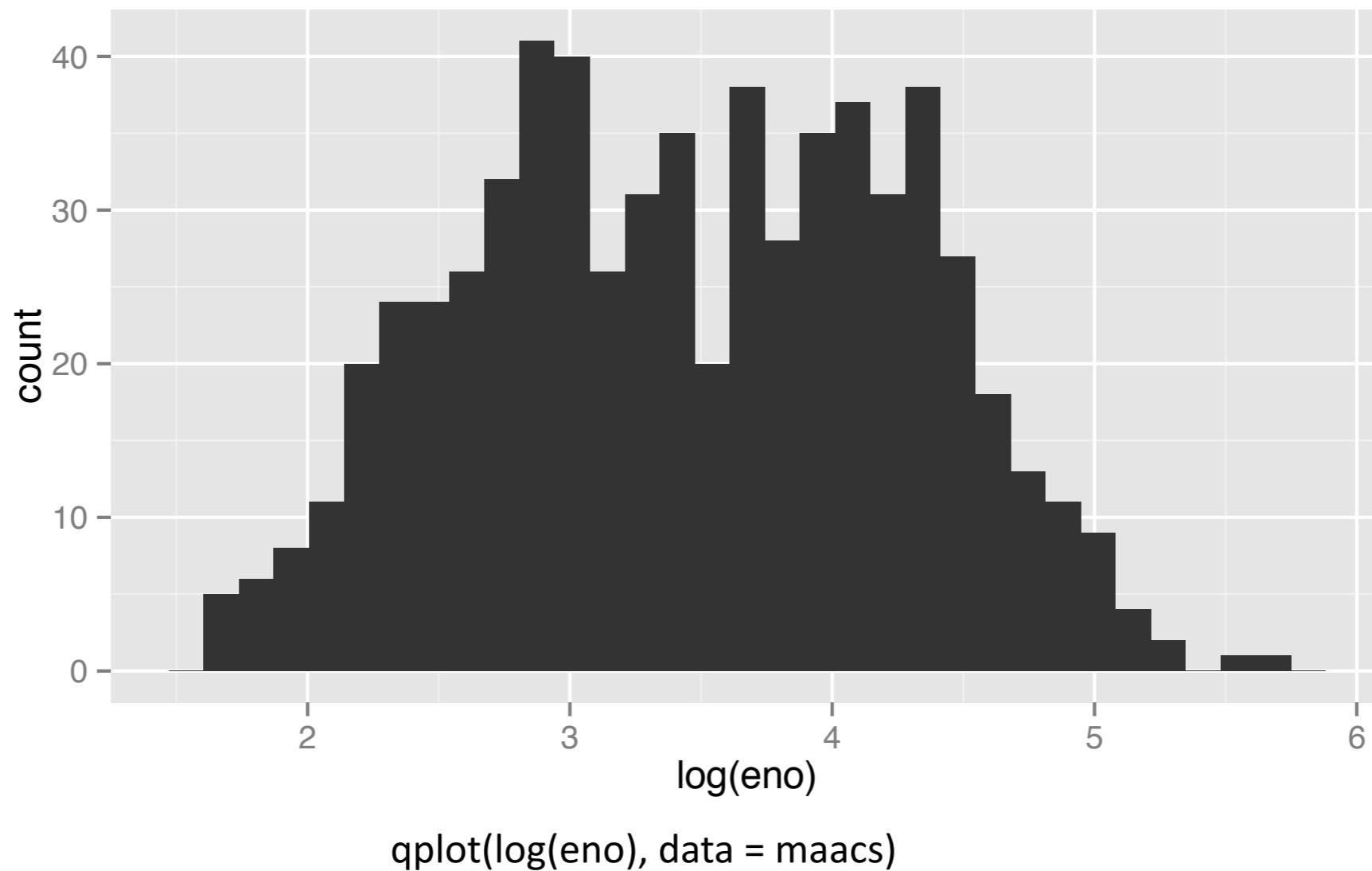
Exhaled nitric oxide

```
> str(maacs)
'data.frame': 750 obs. of 5 variables:
 $ id       : int 1 2 3 4 5 6 7 8 9 10 ...
 $ eno      : num 141 124 126 164 99 68 41 50 12 30 ...
 $ duBedMusM: num 2423 2793 3055 775 1634 ...
 $ pm25     : num 15.6 34.4 39 33.2 27.1 ...
 $ _mopos   : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
```

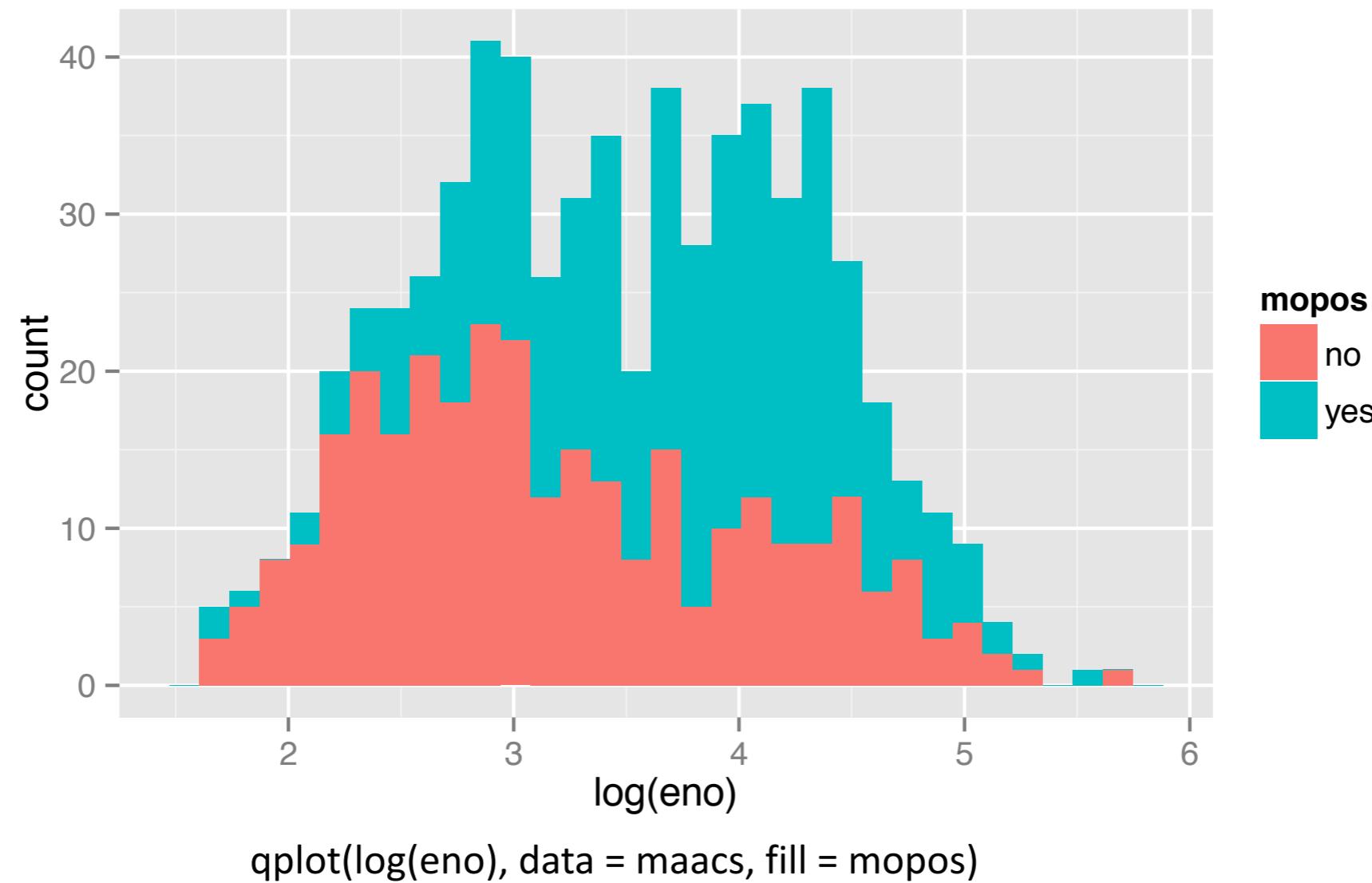
Fine particulate matter

Sensitized to mouse  
allergen

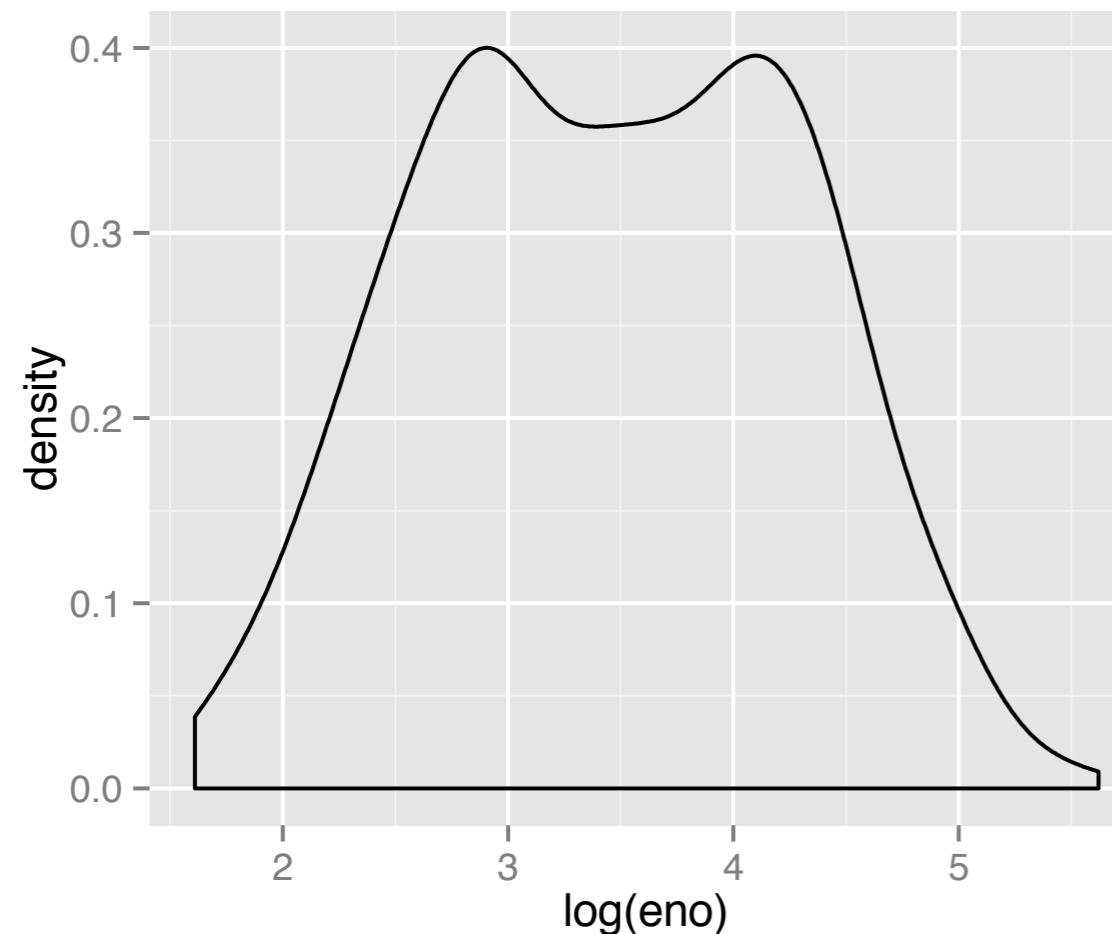
# Histogram of eNO



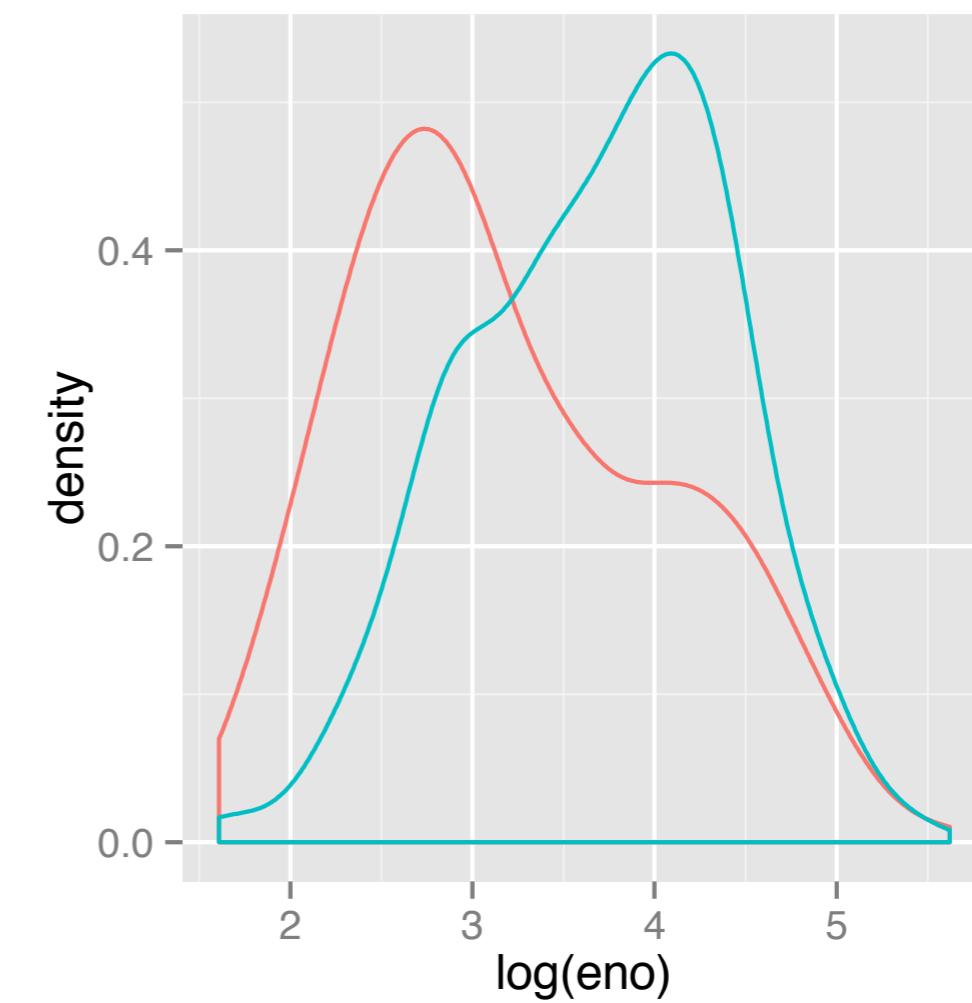
# Histogram by Group



# Density Smooth

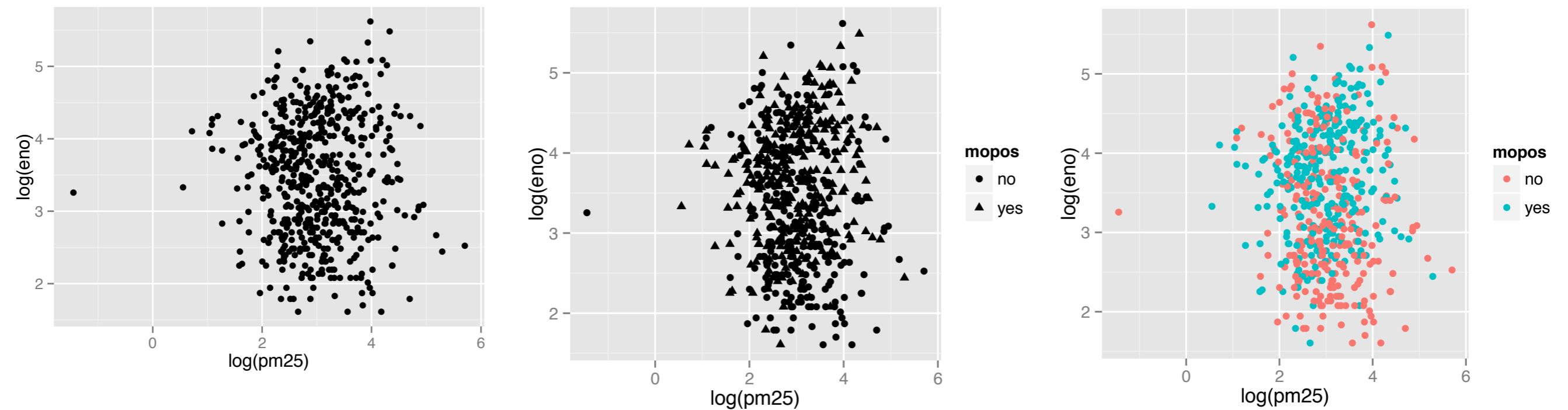


```
qplot(log(eno), data = maacs, geom = "density")
```



```
qplot(log(eno), data = maacs, geom = "density", color = mpos)
```

# Scatterplots: eNO vs. PM2.5

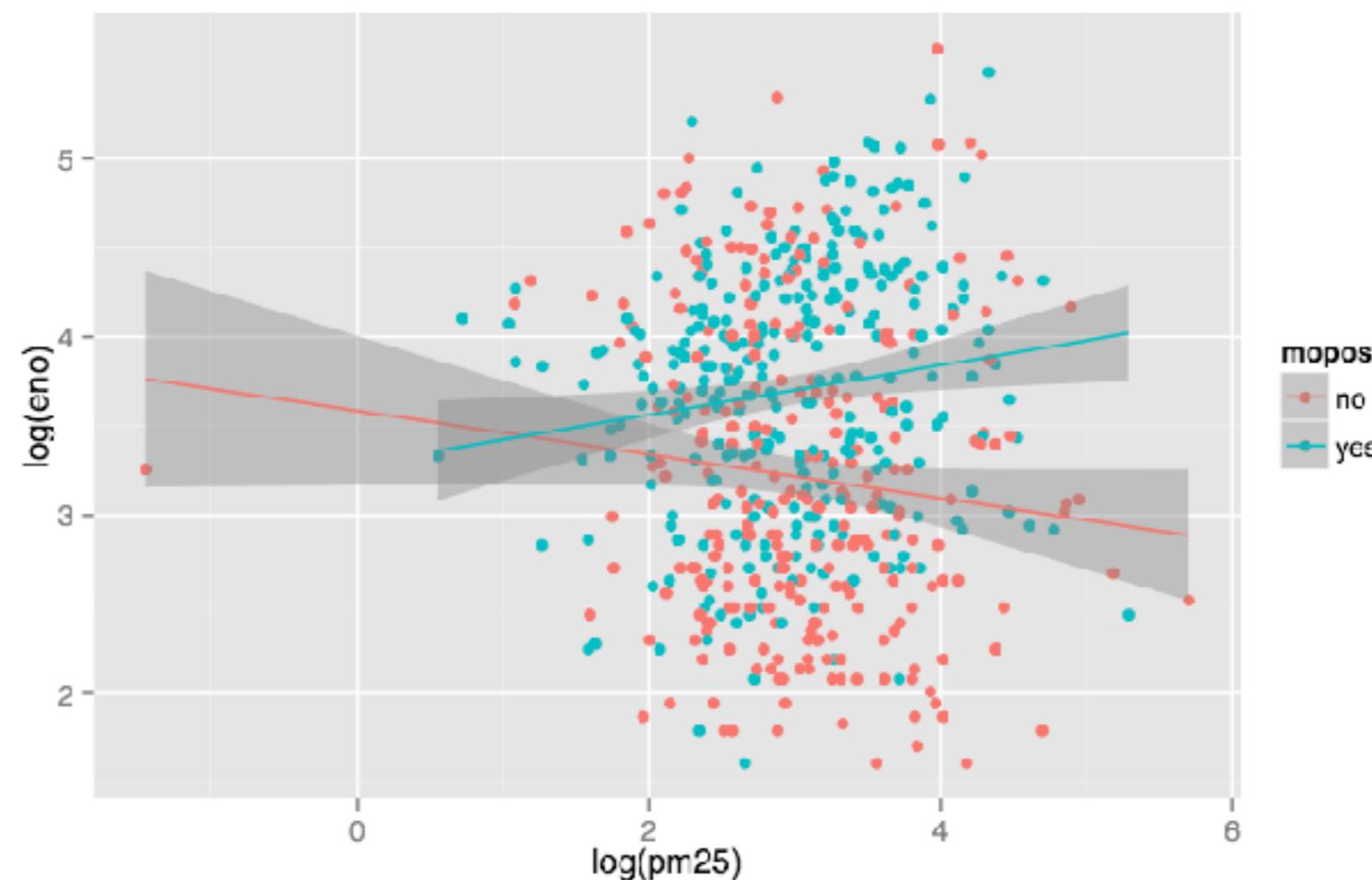


```
qplot(log(pm25), log(eno), data =  
maacs)
```

```
qplot(log(pm25), log(eno), data =  
maacs, shape = mopos)
```

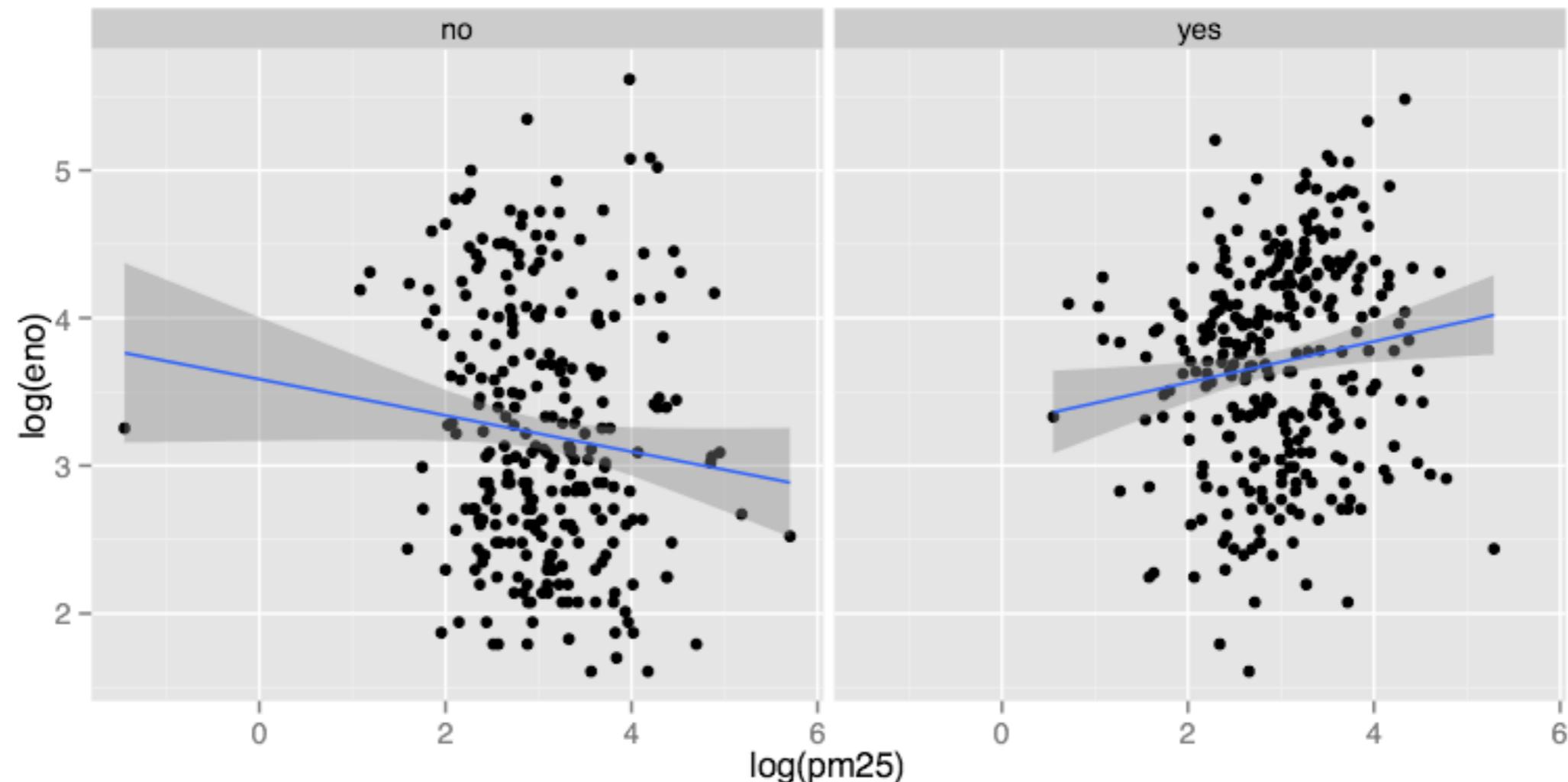
```
qplot(log(pm25), log(eno), data =  
maacs, color = mopos)
```

# Scatterplots: eNO vs. PM2.5



```
qplot(log(pm25), log(eno), data = maacs, color = mopos) + geom_smooth(method = "lm")
```

# Scatterplots: eNO vs. PM2.5



```
qplot(log(pm25), log(eno), data = maacs, facets = . ~ mopes) + geom_smooth(method = "lm")
```

# Summary of qplot()

- The qplot() function is the analog to plot() but with many built-in features
- Syntax somewhere in between base/lattice
- Produces very nice graphics, essentially publication ready (if you like the design)
- Difficult to go against the grain/customize (don't bother; use full ggplot2 power in that case)

# Resources

- The ggplot2 book by Hadley Wickham
- The R Graphics Cookbook by Winston Chang (examples in base plots and in ggplot2)
- ggplot2 web site (<http://ggplot2.org>)
- ggplot2 mailing list (<http://goo.gl/OdW3uB>), primarily for developers

# What is ggplot2?

- An implementation of the Grammar of Graphics by Leland Wilkinson
- Grammar of graphics represents and abstraction of graphics ideas/objects
- Think “verb”, “noun”, “adjective” for graphics
- Allows for a “theory” of graphics on which to build new graphics and graphics objects

# Basic Components of a ggplot2 Plot

- A data frame
- aesthetic mappings: how data are mapped to color, size
- geoms: geometric objects like points, lines, shapes.
- facets: for conditional plots.
- stats: statistical transformations like binning, quantiles, smoothing.
- scales: what scale an aesthetic map uses (example: male = red, female = blue).
- coordinate system

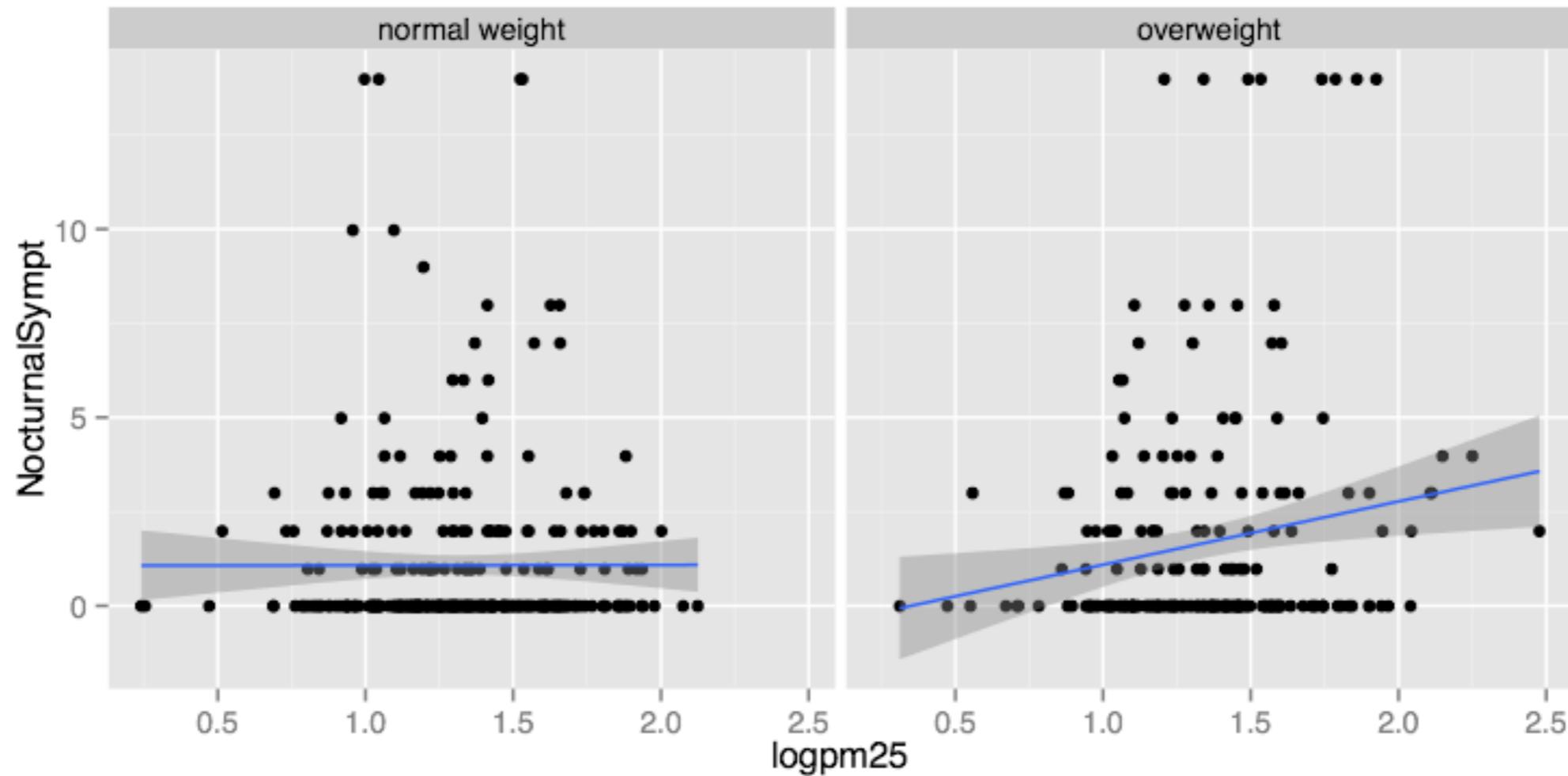
# Building Plots with ggplot2

- When building plots in ggplot2 (rather than using qplot) the “artist’s palette” model may be the closest analogy
- Plots are built up in layers
  - Plot the data
  - Overlay a summary
  - Metadata and annotation

# Example: BMI, PM2.5, Asthma

- Mouse Allergen and Asthma Cohort Study
- Baltimore children (age 5-17)
- Persistent asthma, exacerbation in past year
- Does BMI (normal vs. overweight) modify the relationship between PM2.5 and asthma symptoms?

# Basic Plot



```
qplot(logpm25, NocturnalSympt, data = maacs, facets = . ~ bmicat, geom = c("point", "smooth"), method = "lm")
```

# Building Up in Layers

```
> head(maac)  
logpm25    bmicat NocturnalSympt  
2 1.5361795 normal weight      1  
3 1.5905409 normal weight      0  
4 1.5217786 normal weight      0  
5 1.4323277 normal weight      0  
6 1.2762320  overweight        8  
8 0.7139103  overweight        0  
  
> g <- ggplot(maac, aes(logpm25, NocturnalSympt))  
  
> summary(g)  
data: logpm25, bmicat, NocturnalSympt [554x3]  
mapping: x = logpm25, y = NocturnalSympt  
faceting: facet_null()
```

Data Frame

Aesthetics

Initial call to ggplot

Summary of ggplot object

# No Plot Yet!

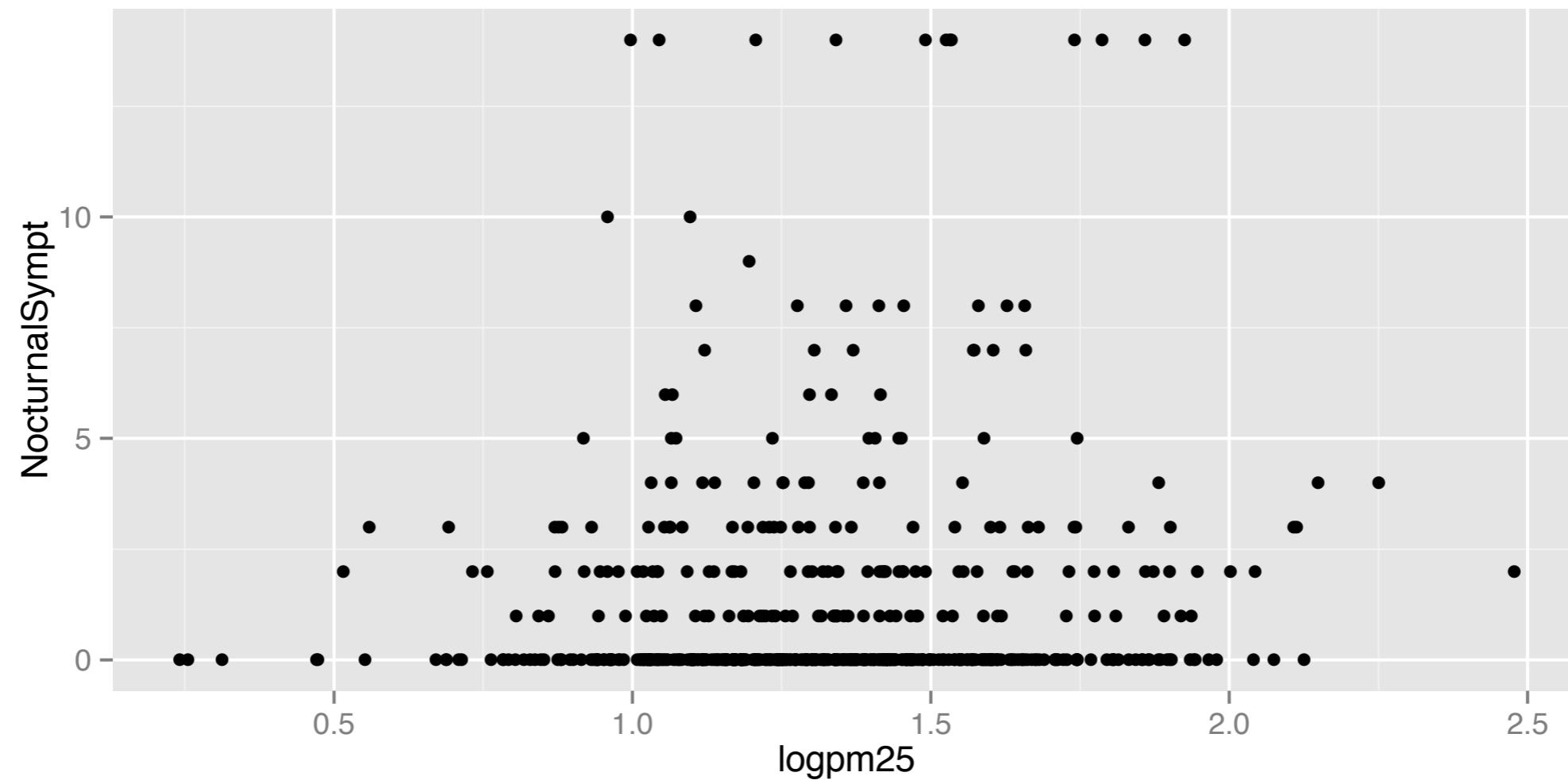
```
> g <- ggplot(maacs, aes(logpm25, NocturnalSympt))  
> print(g)  
Error: No layers in plot
```

```
> p <- g + geom_point()  
> print(p)  
  
> g + geom_point()
```

Explicitly save and print ggplot object

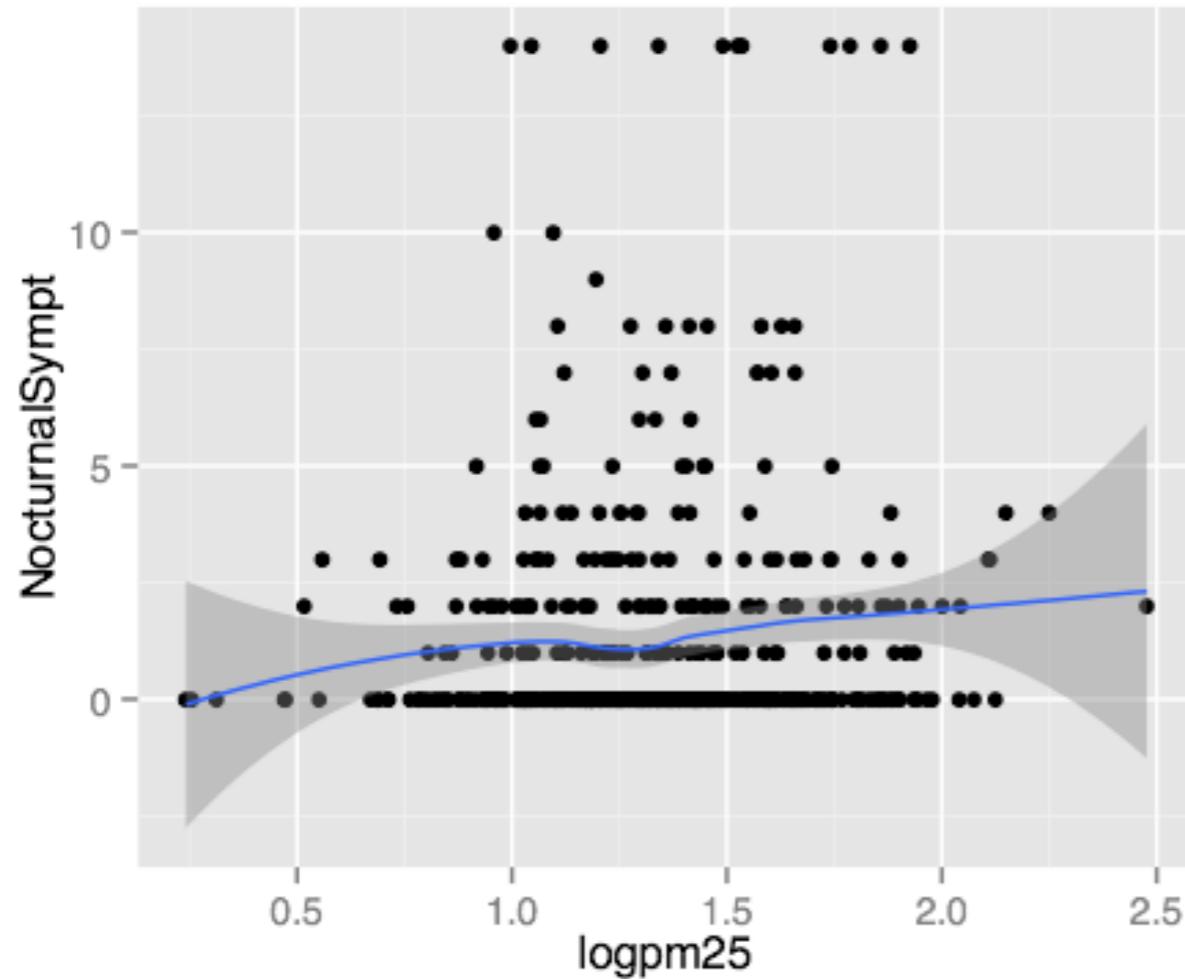
Auto-print plot object without saving

# First Plot with Point Layer

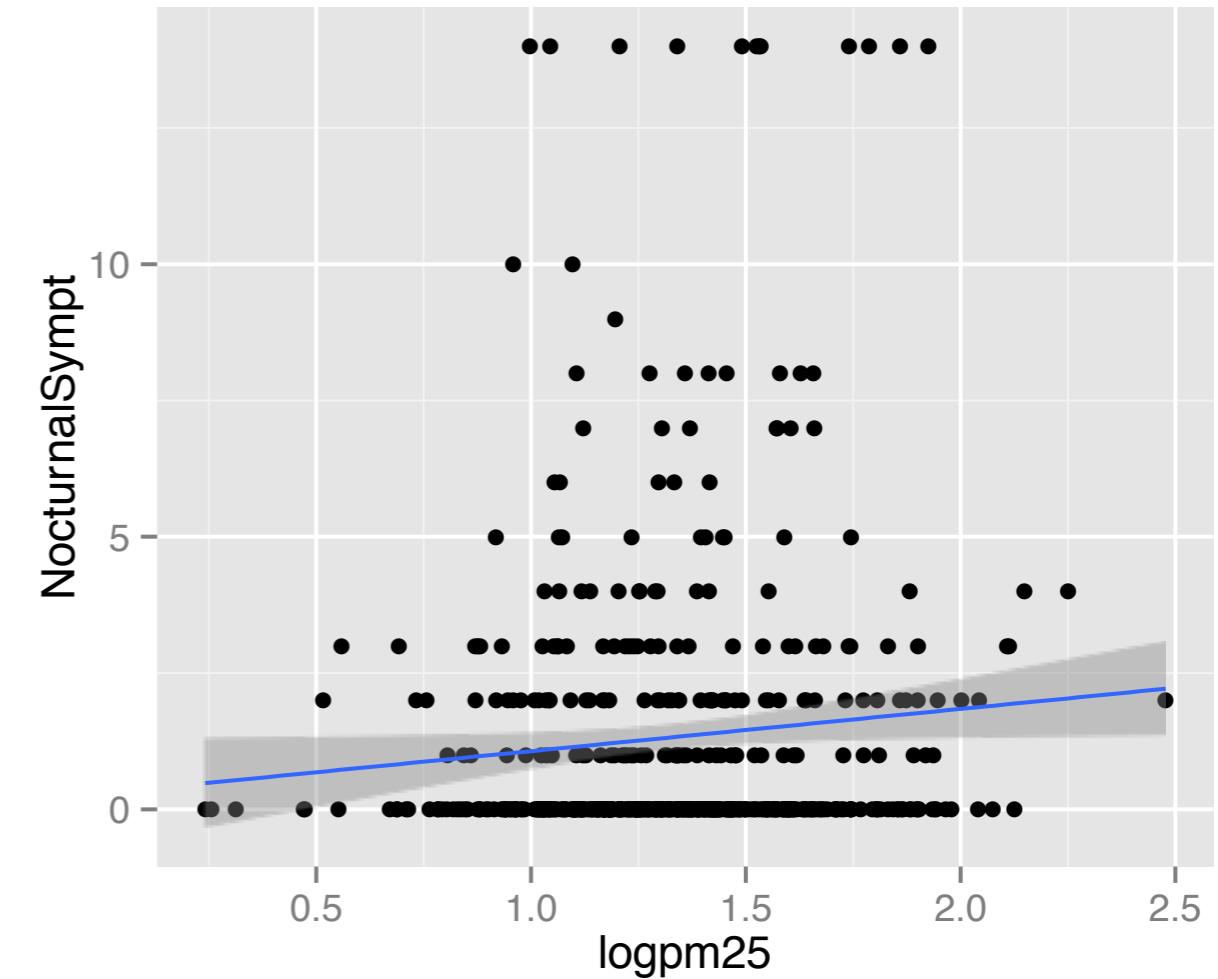


```
g <- ggplot(maacs, aes(logpm25, NocturnalSympt))  
g + geom_point()
```

# Adding More Layers: Smooth

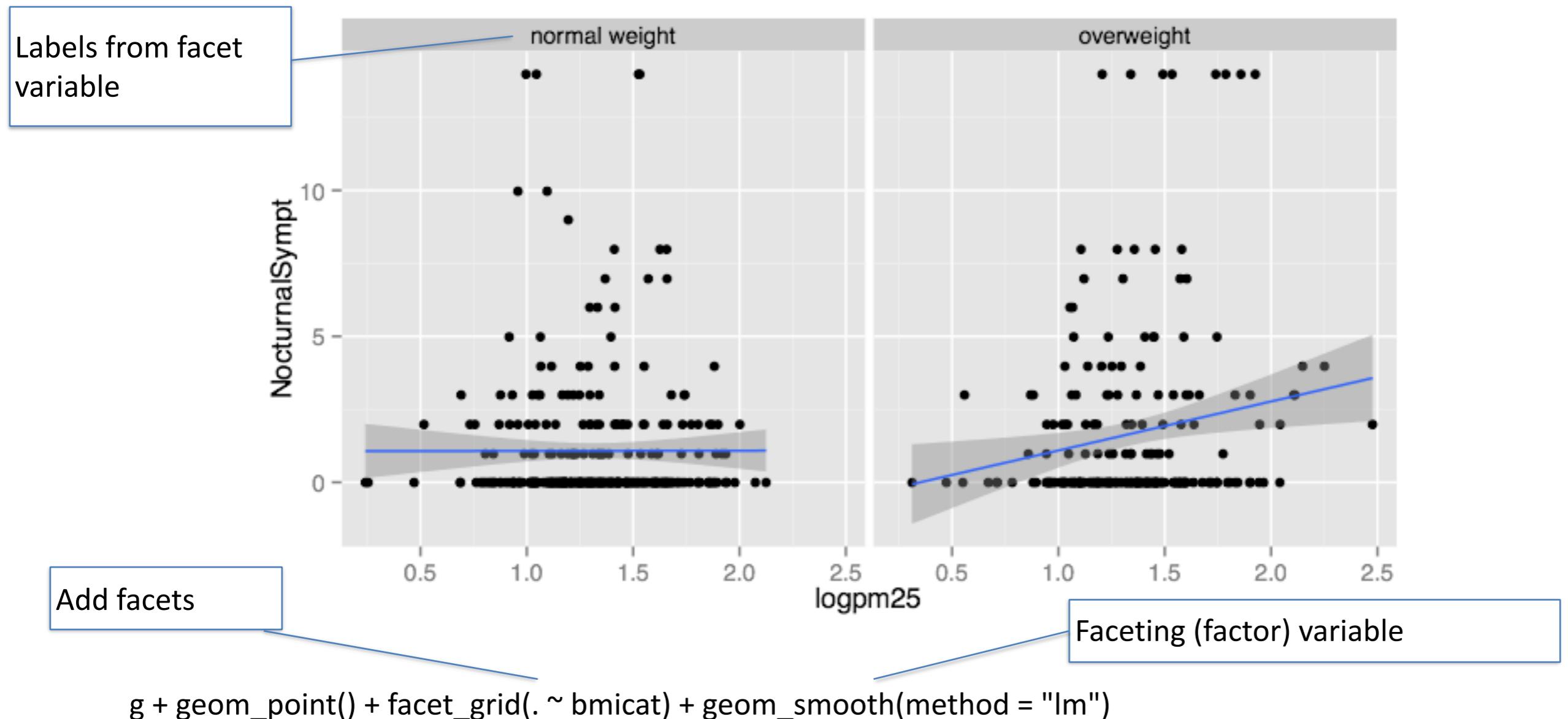


`g + geom_point() + geom_smooth()`



`g + geom_point() + geom_smooth(method = "lm")`

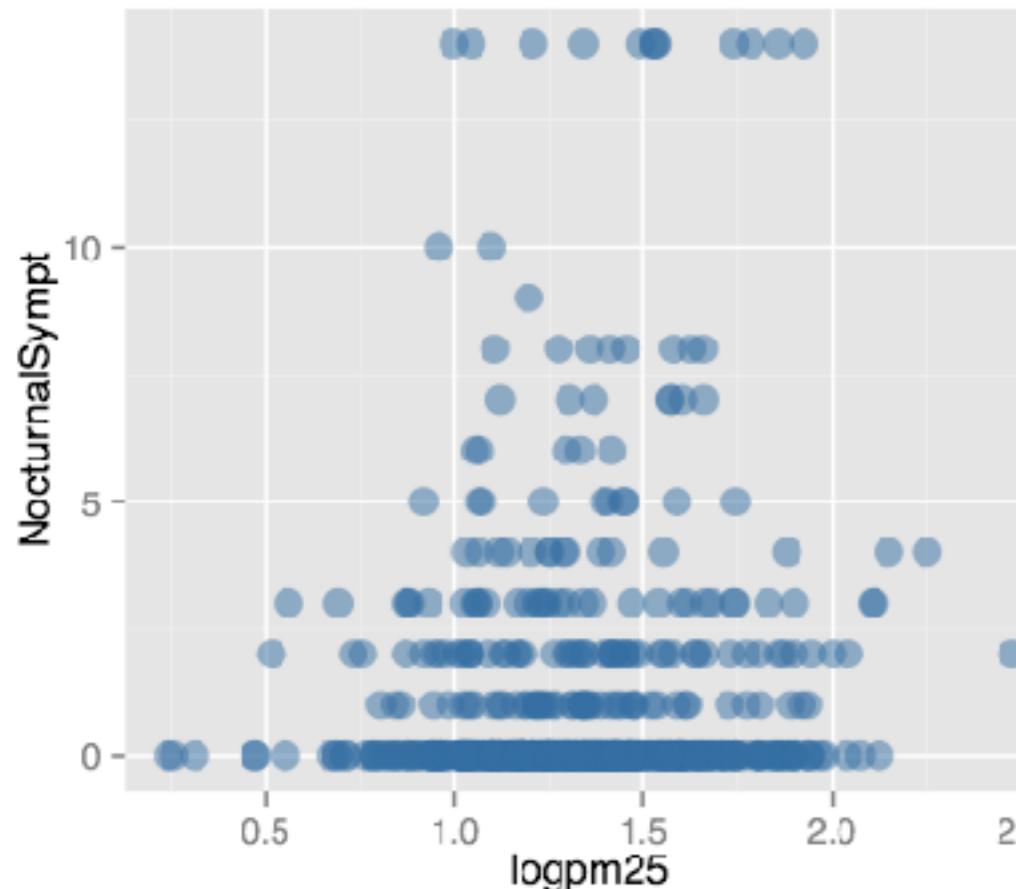
# Adding More Layers: Facets



# Annotation

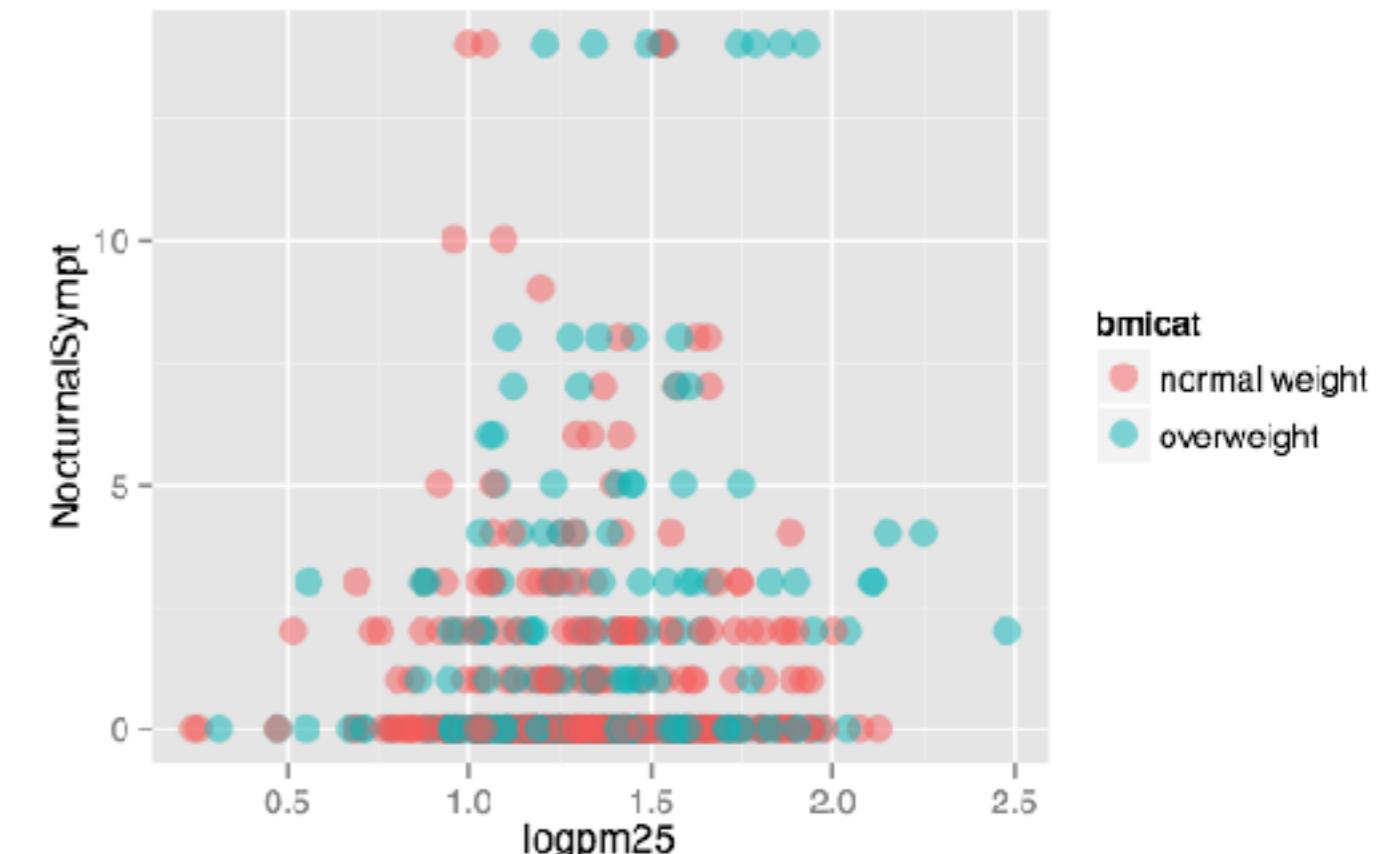
- Labels: `xlab()`, `ylab()`, `labs()`, `ggtitle()`
- Each of the “geom” functions has options to modify
- For things that only make sense globally, use `theme()`
  - Example: `theme(legend.position = "none")`
- Two standard appearance themes are included
  - `theme_gray()`: The default theme (gray background)
  - `theme_bw()`: More stark/plain

# Modifying Aesthetics



```
g + geom_point(color = "steelblue", size = 4, alpha = 1/2)
```

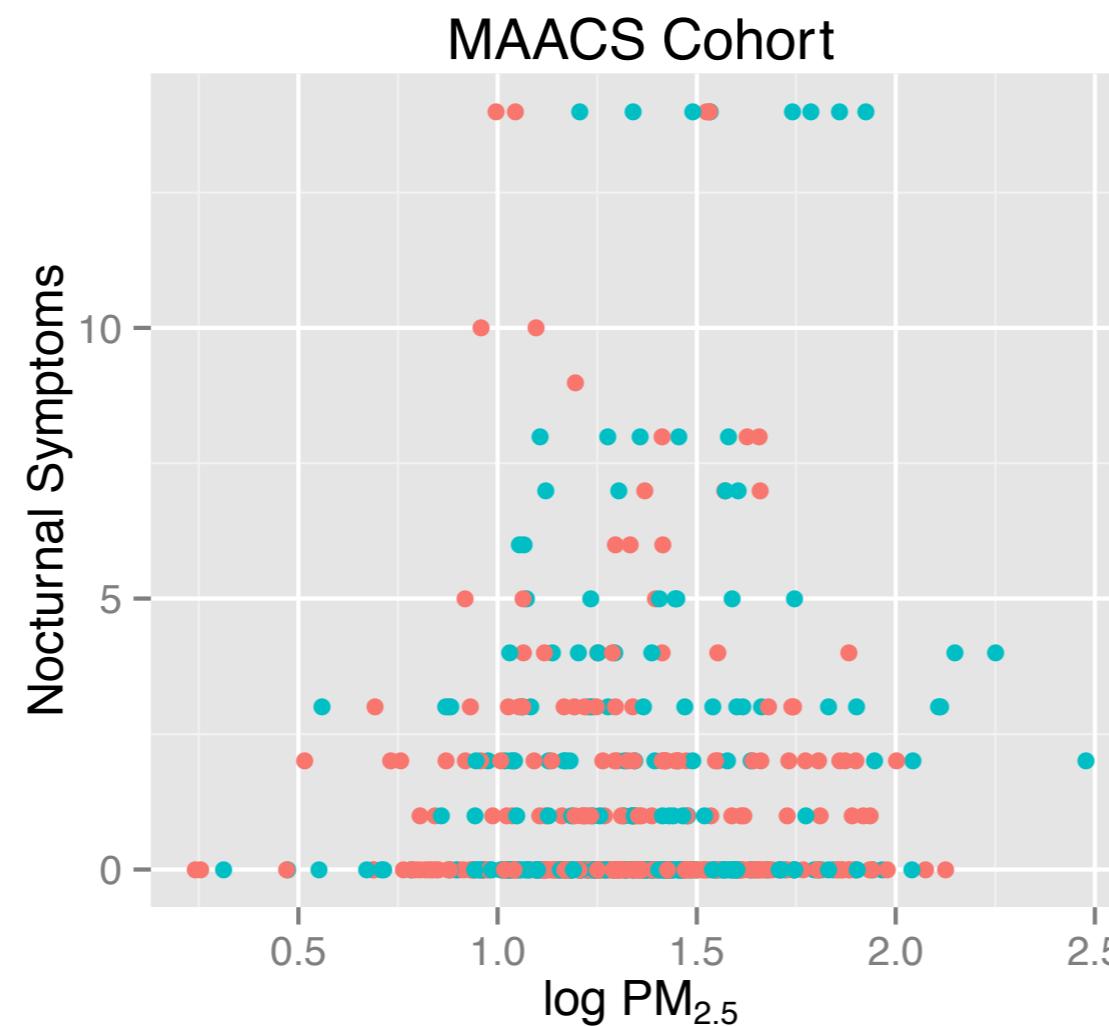
Constant values



```
g + geom_point(aes(color = bmicat), size = 4, alpha = 1/2)
```

Data variable

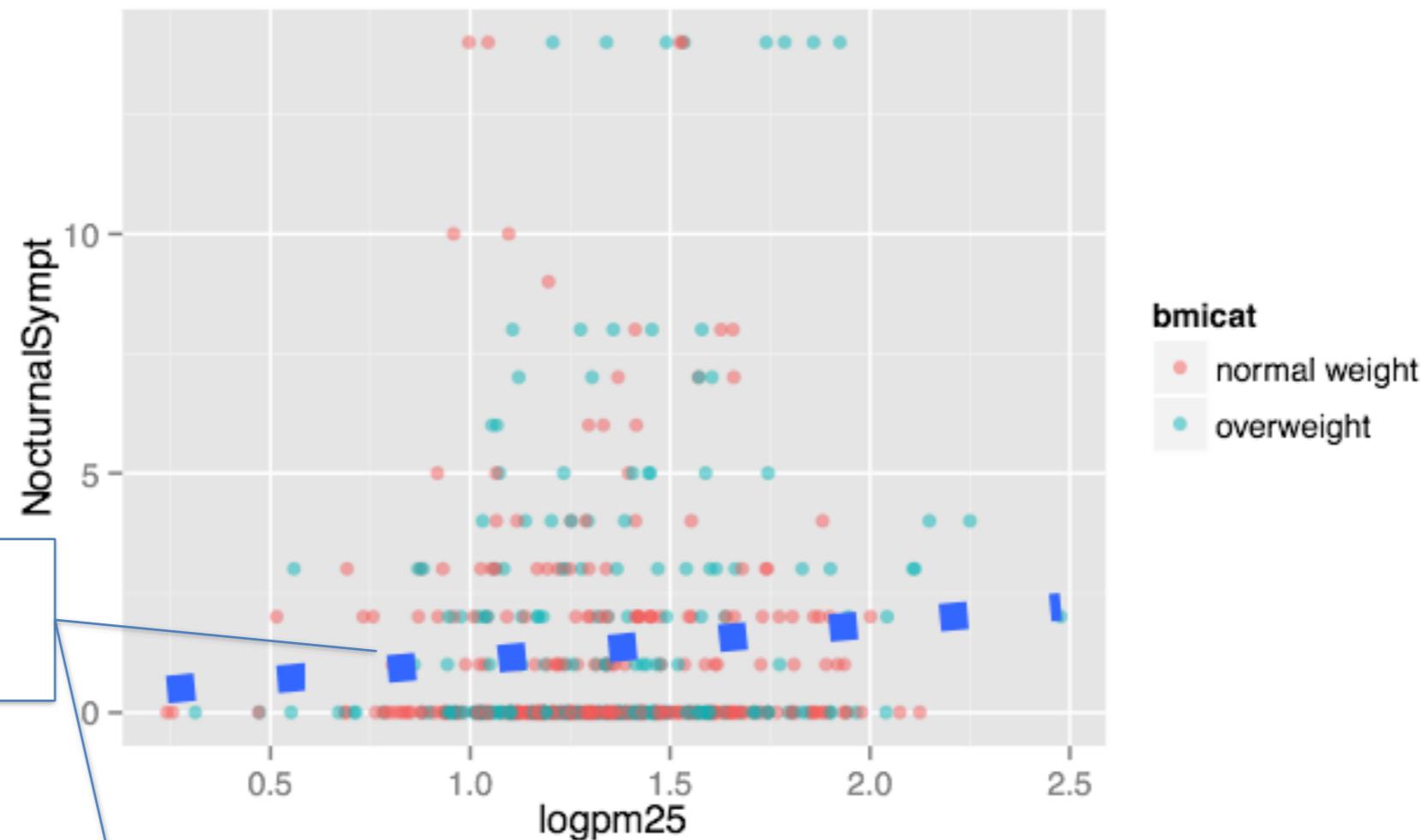
# Modifying Labels



labs() function for modifying titles and x-, y-axis labels

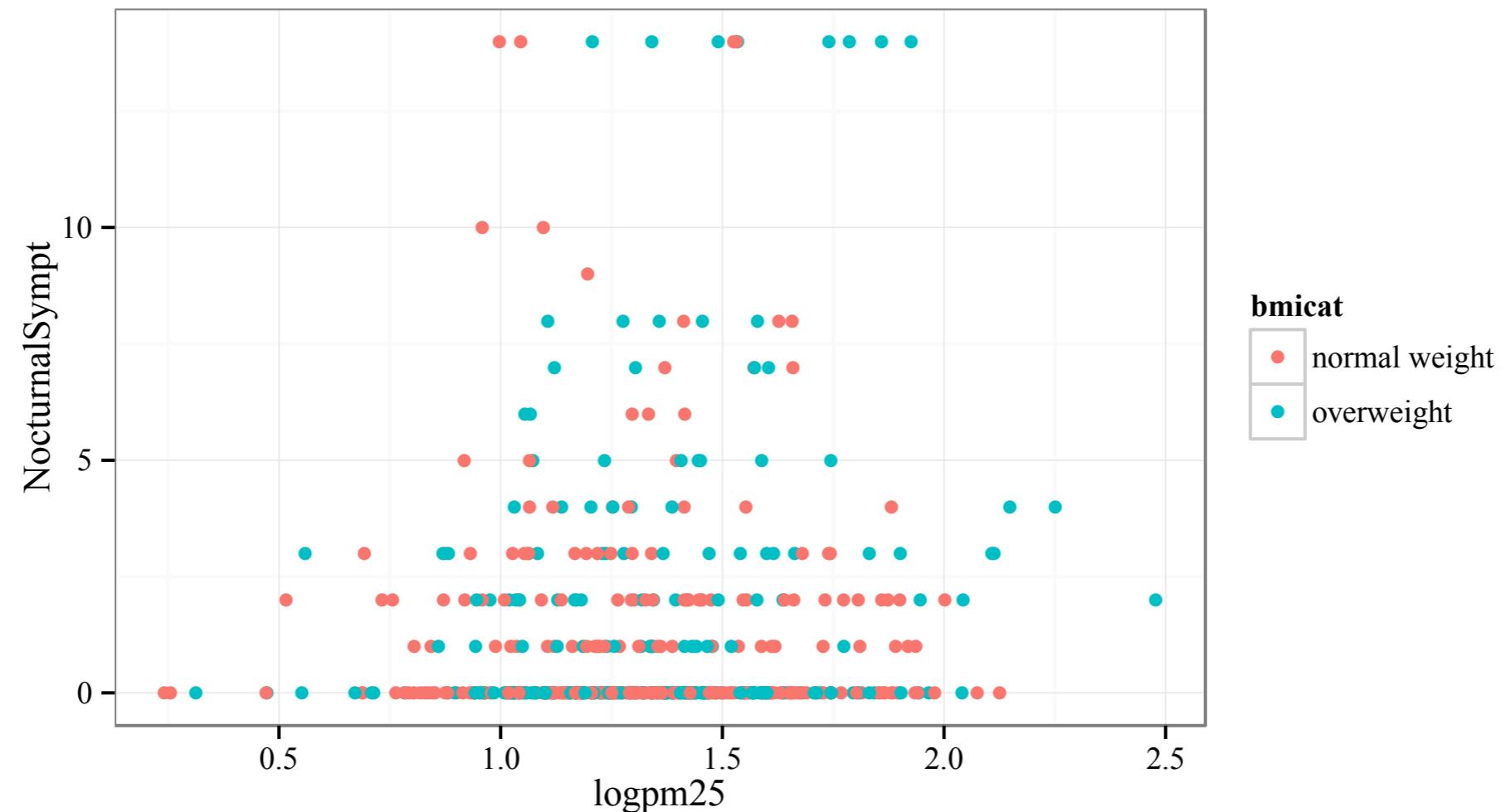
```
g + geom_point(aes(color = bmicat)) + labs(title = "MAACS Cohort") + labs(x = expression("log " * PM[2.5]), y = "Nocturnal Symptoms")
```

# Customizing the Smooth



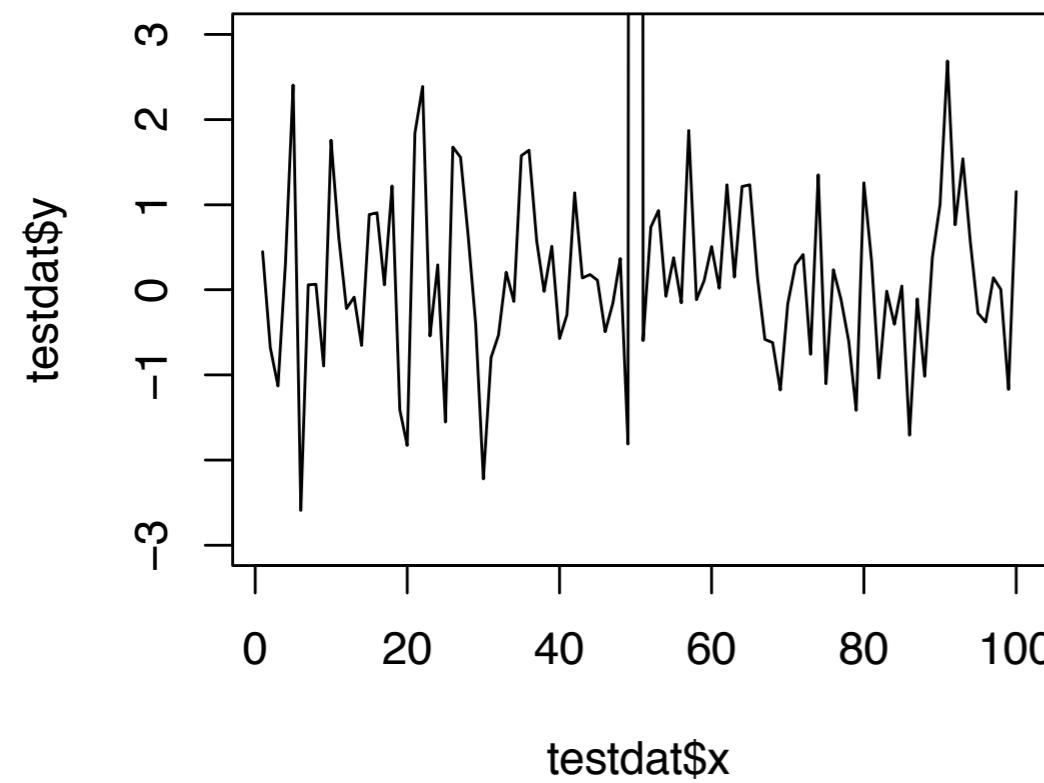
```
g + geom_point(aes(color = bmicat), size = 2, alpha = 1/2) + geom_smooth(size = 4, linetype = 3,  
method = "lm", se = FALSE)
```

# Changing the Theme

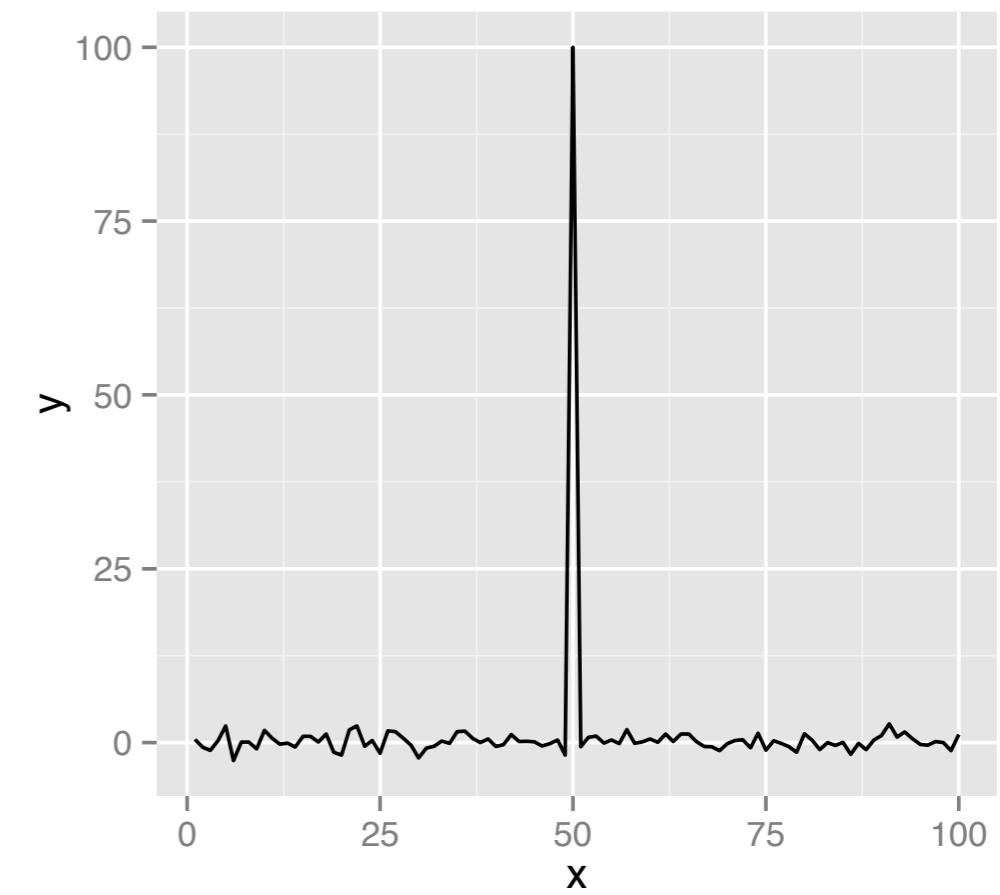


```
g + geom_point(aes(color = bmicat)) + theme_bw(base_family = "Times")
```

# A Notes about Axis Limits



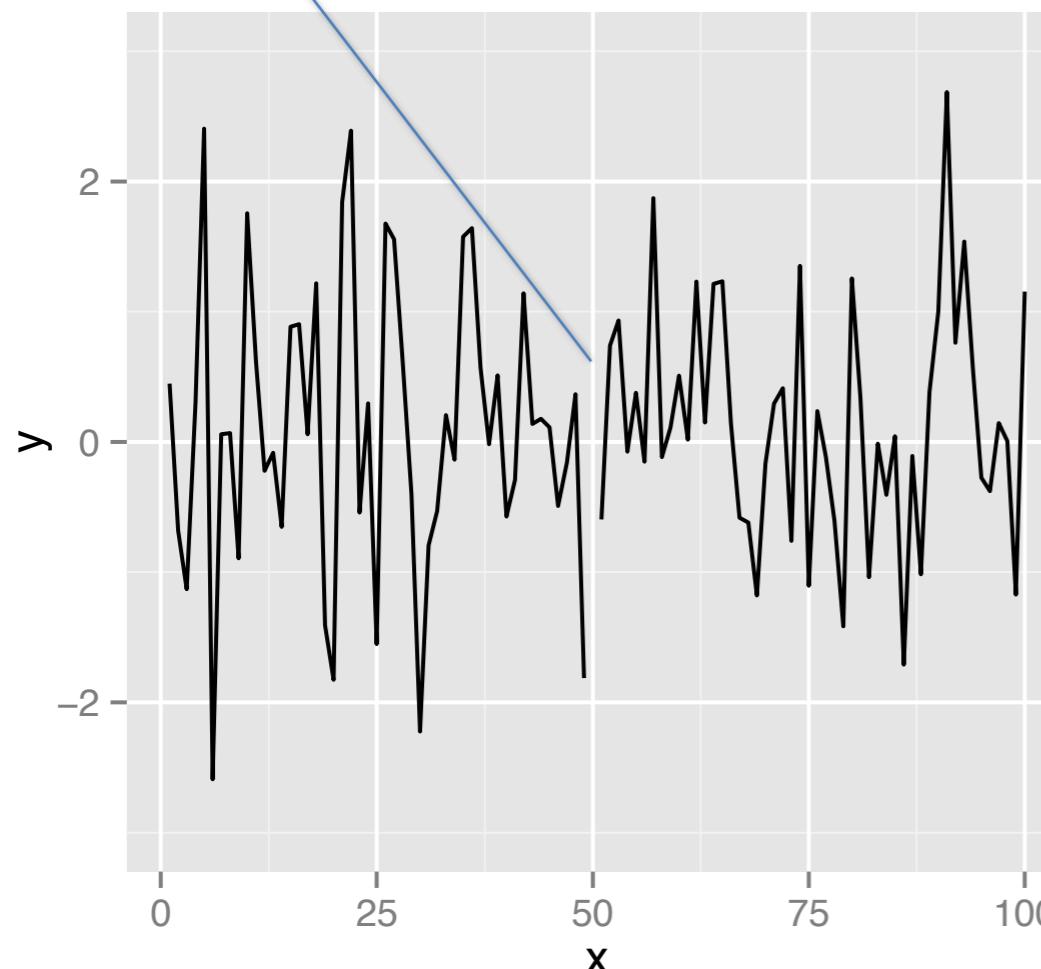
```
testdat <- data.frame(x = 1:100, y = rnorm(100))
testdat[50,2] <- 100 ## Outlier!
plot(testdat$x, testdat$y, type = "l", ylim = c(-3,3))
```



```
g <- ggplot(testdat, aes(x = x, y = y))
g + geom_line()
```

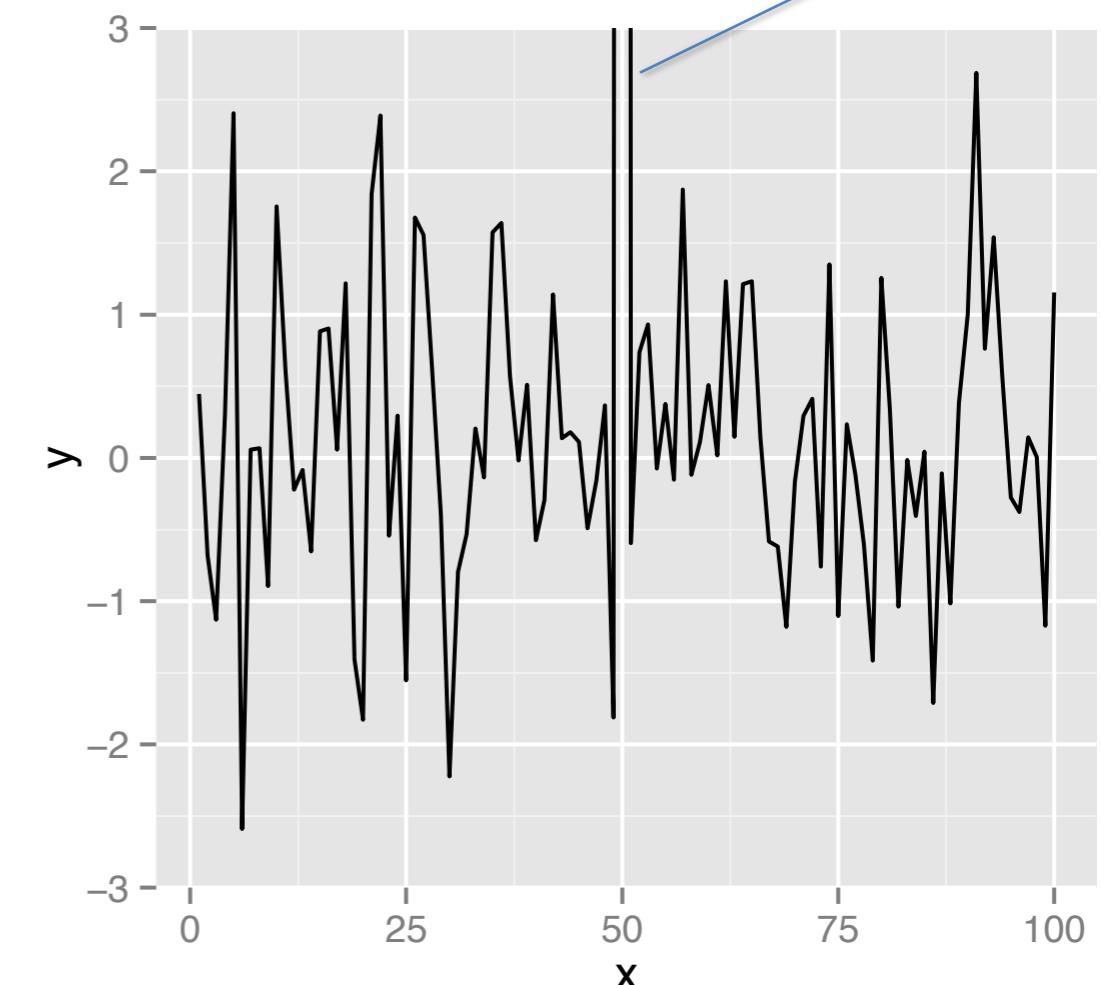
# Axis Limits

Outlier missing



```
g + geom_line() + ylim(-3, 3)
```

Outlier included



```
g + geom_line() + coord_cartesian(ylim = c(-3, 3))
```

# Workshop 6: Plot data

- Plotting the Cars93 data
- (a) Use qplot to create a scatterplot with Price on the y-axis and EngineSize on the x-axis.
- (b) Repeat part (a) using the ggplot function and geom\_point() layer.
- (c) Repeat part (b), but this time specifying that the color mapping should depend on Type and the shape mapping should depend on DriveTrain.

# Other Packages

# To load data

**RODBC, RMySQL, RPostgreSQL, RSQLite** - If you'd like to read in data from a database, these packages are a good place to start. Choose the package that fits your type of database.

**XLConnect, xlsx** - These packages help you read and write Microsoft Excel files from R. You can also just export your spreadsheets from Excel as .csv's.

**foreign** - Want to read a SAS data set into R? Or an SPSS data set? Foreign provides functions that help you load data files from other programs into R.

R can handle plain text files – no package required. Just use the functions `read.csv`, `read.table`, and `read.fwf..`

# xlsx

```
df <- read.xlsx("<name and extension of your file>",
                 sheetIndex = 1)
```

**Note** that it is necessary to add a sheet name or a sheet index to this function. In the example above, the first sheet of the Excel file was assigned. If you have a bigger data set, you might get better performance when using the `read.xlsx2()` function:

```
df <- read.xlsx2("<name and extension of your file>",
                  sheetIndex = 1,
                  startRow=2,
                  colIndex = 2)
```

```
write.xlsx(df,  
          "df.xlsx",  
          sheetName="Data Frame")
```

The function requires you first to specify what data frame you want to export. In the second argument, you specify the name of the file that you are outputting. If, however, you want to write the data frame to a file that already exists, you can execute the following command:

```
write.xlsx(df,  
          "<name and extension of your existing file>",  
          sheetName="Data Frame"  
          append=TRUE)
```

# To manipulate data

**dplyr** - Essential shortcuts for subsetting, summarizing, rearranging, and joining together data sets. dplyr is our go to package for fast data manipulation.

**tidyr** - Tools for changing the layout of your data sets. Use the gather and spread functions to convert your data into the tidy format, the layout R likes best.

**stringr** - Easy to learn tools for regular expressions and character strings.

**lubridate** - Tools that make working with dates and times easier.

# dplyr

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

## dplyr verbs

	Description
<code>select()</code>	select columns
<code>filter()</code>	filter rows
<code>arrange()</code>	re-order or arrange rows
<code>mutate()</code>	create new columns
<code>summarise()</code>	summarise values
<code>group_by()</code>	allows for group operations in the “split-apply-combine” concept

# select

Select a set of columns: the name and the sleep\_total columns.

```
sleepData <- select(msleep, name, sleep_total)  
head(sleepData)
```

```
##                                     name sleep_total  
## 1                               Cheetah      12.1  
## 2             Owl monkey        17.0  
## 3       Mountain beaver      14.4  
## 4 Greater short-tailed shrew    14.9  
## 5                           Cow        4.0  
## 6      Three-toed sloth      14.4
```

# Subtraction (-)

To select all the columns except a specific column, use the “-“ (subtraction) operator (also known as negative indexing)

```
head(select(msleep, -name))
```

```
##          genus vore      order conservation sleep_total sleep_rem
## 1    Acinonyx carni   Carnivora           lc       12.1        NA
## 2      Aotus omni    Primates         <NA>      17.0       1.8
## 3 Aplodontia herbi   Rodentia           nt      14.4       2.4
## 4     Blarina omni Soricomorpha         lc      14.9       2.3
## 5       Bos herbi Artiodactyla domesticated      4.0       0.7
## 6 Bradypus herbi     Pilosa         <NA>      14.4       2.2
##   sleep_cycle awake brainwt bodywt
## 1        NA   11.9      NA 50.000
## 2        NA    7.0 0.01550   0.480
## 3        NA    9.6      NA  1.350
## 4  0.1333333  9.1 0.00029   0.019
## 5  0.6666667 20.0 0.42300 600.000
## 6  0.7666667   9.6      NA   3.850
```

# colon (:)

To select a range of columns by name, use the ":" (colon) operator

```
head(select(msleep, name:order))
```

```
##          name   genus  vore      order
## 1     Cheetah Acinonyx carni  Carnivora
## 2 Owl monkey     Aotus  omni   Primates
## 3 Mountain beaver Aplodontia herbi Rodentia
## 4 Greater short-tailed shrew    Blarina  omni Soricomorpha
## 5             Cow       Bos herbi Artiodactyla
## 6 Three-toed sloth   Bradypus herbi Pilosa
```

To select all columns that start with the character string “sl”, use the function `starts_with()`

```
head(select(msleep, starts_with("sl")))
```

```
##   sleep_total sleep_rem sleep_cycle
## 1      12.1        NA         NA
## 2      17.0        1.8         NA
## 3      14.4        2.4         NA
## 4      14.9        2.3  0.1333333
## 5       4.0        0.7  0.6666667
## 6      14.4        2.2  0.7666667
```

Some additional options to select columns based on a specific criteria include

1. `ends_with()` = Select columns that end with a character string
2. `contains()` = Select columns that contain a character string
3. `matches()` = Select columns that match a regular expression
4. `one_of()` = Select columns names that are from a group of names

# filter

Filter the rows for mammals that sleep a total of more than 16 hours.

```
filter(msleep, sleep_total >= 16)
```

```
##          name      genus   vore      order conservation
## 1  Owl monkey     Aotus    omni  Primates       <NA>
## 2 Long-nosed armadillo  Dasypus   carni Cingulata        lc
## 3 North American Opossum  Didelphis   omni Didelphimorphia        lc
## 4    Big brown bat  Eptesicus insecti Chiroptera        lc
## 5 Thick-tailed opossum Lutreolina   carni Didelphimorphia        lc
## 6 Little brown bat     Myotis insecti Chiroptera       <NA>
## 7   Giant armadillo Priodontes insecti Cingulata        en
## 8 Arctic ground squirrel Spermophilus   herbi Rodentia        lc
##   sleep_total sleep_rem sleep_cycle awake brainwt bodywt
## 1      17.0       1.8           NA     7.0  0.01550   0.480
## 2      17.4       3.1  0.3833333     6.6  0.01080   3.500
## 3      18.0       4.9  0.3333333     6.0  0.00630   1.700
## 4      19.7       3.9  0.1166667     4.3  0.00030   0.023
## 5      19.4       6.6           NA     4.6       NA   0.370
## 6      19.9       2.0  0.2000000     4.1  0.00025   0.010
## 7      18.1       6.1           NA     5.9  0.08100  60.000
## 8      16.6       NA           NA     7.4  0.00570   0.920
```

# filter with and

Filter the rows for mammals that sleep a total of more than 16 hours *and* have a body weight of greater than 1 kilogram.

```
filter(msleep, sleep_total >= 16, bodywt >= 1)
```

```
##          name      genus     vore      order conservation
## 1 Long-nosed armadillo   Dasypus    carni  Cingulata        lc
## 2 North American Opossum Didelphis    omni Didelphimorphia    lc
## 3 Giant armadillo Priodontes insecti    carni  Cingulata        en
##   sleep_total sleep_rem sleep_cycle awake brainwt bodywt
## 1       17.4       3.1     0.3833333   6.6   0.0108     3.5
## 2       18.0       4.9     0.3333333   6.0   0.0063     1.7
## 3       18.1       6.1        NA     5.9   0.0810    60.0
```

# Pipe (%>%)

```
head(select(msleep, name, sleep_total))
```

```
##                                     name sleep_total
## 1                               Cheetah      12.1
## 2             Owl monkey       17.0
## 3     Mountain beaver       14.4
## 4 Greater short-tailed shrew   14.9
## 5                      Cow        4.0
## 6 Three-toed sloth       14.4
```

```
msleep %>%  
  select(name, sleep_total) %>%  
  head
```

```
##                                     name sleep_total  
## 1                          Cheetah      12.1  
## 2          Owl monkey        17.0  
## 3 Mountain beaver        14.4  
## 4 Greater short-tailed shrew    14.9  
## 5                      Cow        4.0  
## 6 Three-toed sloth        14.4
```

# sqldf

```
library(sqldf)
```

```
setwd()
crashes <- read.csv("crashes.csv")
roads <- read.csv("roads.csv")
head(crashes)
```

```
##   Year      Road N_Crashes Volume
## 1 1991 Interstate 65      25 40000
## 2 1992 Interstate 65      37 41000
## 3 1993 Interstate 65      45 45000
## 4 1994 Interstate 65      46 45600
## 5 1995 Interstate 65      46 49000
## 6 1996 Interstate 65      59 51000
```

```
tail(crashes)
```

```
##   Year      Road N_Crashes Volume
## 105 2007 Interstate 275      32 21900
## 106 2008 Interstate 275      21 21850
## 107 2009 Interstate 275      25 22100
## 108 2010 Interstate 275      24 21500
## 109 2011 Interstate 275      23 20300
## 110 2012 Interstate 275      22 21200
```

```
print(roads)
```

```
##          Road      District Length
## 1 Interstate 65      Greenfield     262
## 2 Interstate 70      Vincennes     156
## 3          US-36 Crawfordsville    139
## 4          US-40      Greenfield     150
## 5          US-52 Crawfordsville    172
```

# Outer join

```
join_string <- "select
  crashes.*
, roads.District
, roads.Length
from crashes
  left join roads
    on crashes.Road = roads.Road"
```

```
crashes_join_roads <- sqldf(join_string,stringsAsFactors = FALSE)
```

```
## Loading required package: tcltk
```

```
head(crashes_join_roads)
```

```
##   Year        Road N_Crashes Volume District Length
## 1 1991 Interstate 65       25 40000 Greenfield   262
## 2 1992 Interstate 65       37 41000 Greenfield   262
## 3 1993 Interstate 65       45 45000 Greenfield   262
## 4 1994 Interstate 65       46 45600 Greenfield   262
## 5 1995 Interstate 65       46 49000 Greenfield   262
## 6 1996 Interstate 65       59 51000 Greenfield   262
```

# Inner join

By using an inner join, only matching rows will be kept.

```
join_string2 <- "select
  crashes.*
, roads.District
, roads.Length
from crashes
inner join roads
on crashes.Road = roads.Road"
```

```
crashes_join_roads2 <- sqldf(join_string2, stringsAsFactors = FALSE)
head(crashes_join_roads2)
```

##	Year	Road	N_Crashes	Volume	District	Length
## 1	1991	Interstate 65	25	40000	Greenfield	262
## 2	1992	Interstate 65	37	41000	Greenfield	262
## 3	1993	Interstate 65	45	45000	Greenfield	262
## 4	1994	Interstate 65	46	45600	Greenfield	262
## 5	1995	Interstate 65	46	49000	Greenfield	262
## 6	1996	Interstate 65	59	51000	Greenfield	262

```
tail(crashes_join_roads2)
```

	##	Year	Road	N_Crashes	Volume	District	Length
##	83	2007	US-36	49	24000	Crawfordsville	139
##	84	2008	US-36	52	24500	Crawfordsville	139
##	85	2009	US-36	55	24700	Crawfordsville	139
##	86	2010	US-36	35	23000	Crawfordsville	139
##	87	2011	US-36	33	21000	Crawfordsville	139
##	88	2012	US-36	31	20500	Crawfordsville	139

# Where

```
join_string2 <- "select
  crashes.*
, roads.District
, roads.Length
  from crashes
    inner join roads
      on crashes.Road = roads.Road
    where crashes.Road = 'US-40'"
crashes_join_roads4 <- sqldf(join_string2,stringsAsFactors = FALSE)
head(crashes_join_roads4)
```

```
##   Year Road N_Crashes Volume District Length
## 1 1991 US-40        46 21000 Greenfield    150
## 2 1992 US-40       101 21500 Greenfield    150
## 3 1993 US-40        76 23000 Greenfield    150
## 4 1994 US-40        72 21000 Greenfield    150
## 5 1995 US-40        75 24000 Greenfield    150
## 6 1996 US-40       136 23500 Greenfield    150
```

```
tail(crashes_join_roads4)
```

```
##   Year Road N_Crashes Volume District Length
## 17 2007 US-40        45 59500 Greenfield    150
## 18 2008 US-40        23 61000 Greenfield    150
## 19 2009 US-40        67 65000 Greenfield    150
## 20 2010 US-40       102 67000 Greenfield    150
## 21 2011 US-40        87 67500 Greenfield    150
## 22 2012 US-40        32 67500 Greenfield    150
```

# To visualize data

**ggplot2** - R's famous package for making beautiful graphics. ggplot2 lets you use the grammar of graphics to build layered, customizable plots.

**ggvis** - Interactive, web based graphics built with the grammar of graphics.

**rgl** - Interactive 3D visualizations with R



**htmlwidgets** - A fast way to build interactive (javascript based) visualizations with R. Packages that implement htmlwidgets include:

**leaflet** (maps)

**dygraphs** (time series)

**DT** (tables)

**diagrammeR** (diagrams)

**network3D** (network graphs)

**threeJS** (3D scatterplots and globes).



**googleVis** - Let's you use Google Chart tools to visualize data in R.

Google Chart tools used to be called Gapminder, the graphing software

Hans Rosling made famous in his TED talk.

# To model data

**car** - car's Anova function is popular for making type II and type III Anova tables.

**mgcv** - Generalized Additive Models

**lme4/nlme** - Linear and Non-linear mixed effects models

**randomForest** - Random forest methods from machine learning

**multcomp** - Tools for multiple comparison testing

**vcd** - Visualization tools and tests for categorical data

**glmnet** - Lasso and elastic-net regression methods with cross validation

**survival** - Tools for survival analysis

# To report results

**shiny** - Easily make interactive, web apps with R. A perfect way to explore data and share findings with non-programmers.

**R Markdown** - The perfect workflow for reproducible reporting. Write R code in your markdown reports. When you run render, R Markdown will replace the code with its results and then export your report as an HTML, pdf, or MS Word document, or a HTML or pdf slideshow. The result? Automated reporting. R Markdown is integrated straight into RStudio.

**xtable** - The xtable function takes an R object (like a data frame) and

# Introduction to Shiny

- Open Sourced by RStudio November 2012
- Default widgets and settings make it easy to generate apps
- Don't need to know HTML, CSS and javascript to get started
- Twitter Bootstrap for default UI - looks good
- Web sockets for communication between client and server
- Reactive Programming model
- Works on Windows, Mac, Linux

# Introduction to Shiny

Ready to use shiny

- Install R from CRAN
- Useful to have Chrome, Firefox, Safari...
- Install Shiny using R command: `install.packages("shiny")`

# Simple Example

- `library(shiny)`
- `runExample("01_hello")`



# ui.R : : Controls the look of the App

```
library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

# server.R : Specifies what R is doing

```
library(shiny)

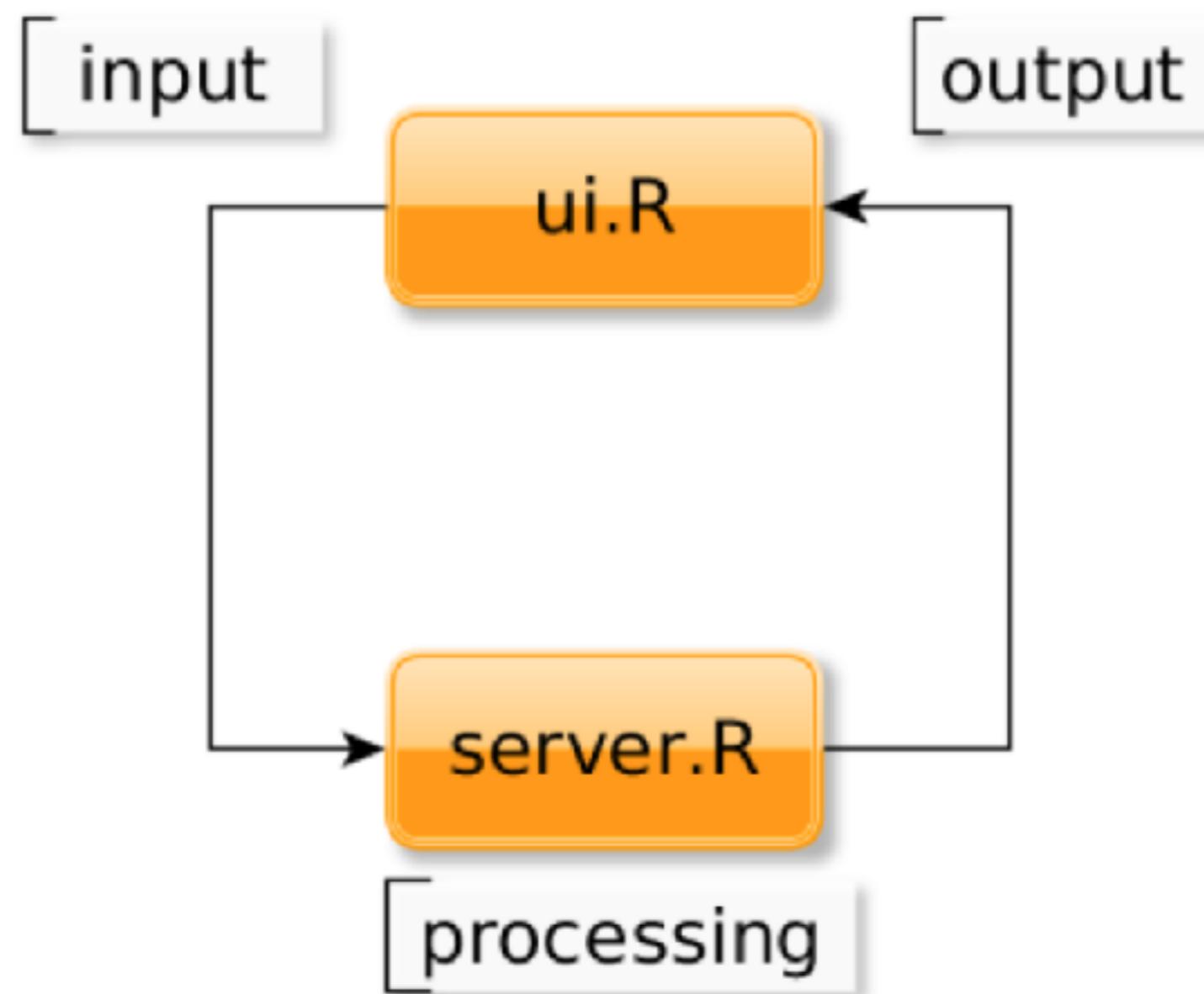
# Define server logic required to draw a histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  # 1) It is "reactive" and therefore should be automatically
  #    re-executed when inputs change
  # 2) Its output type is a plot

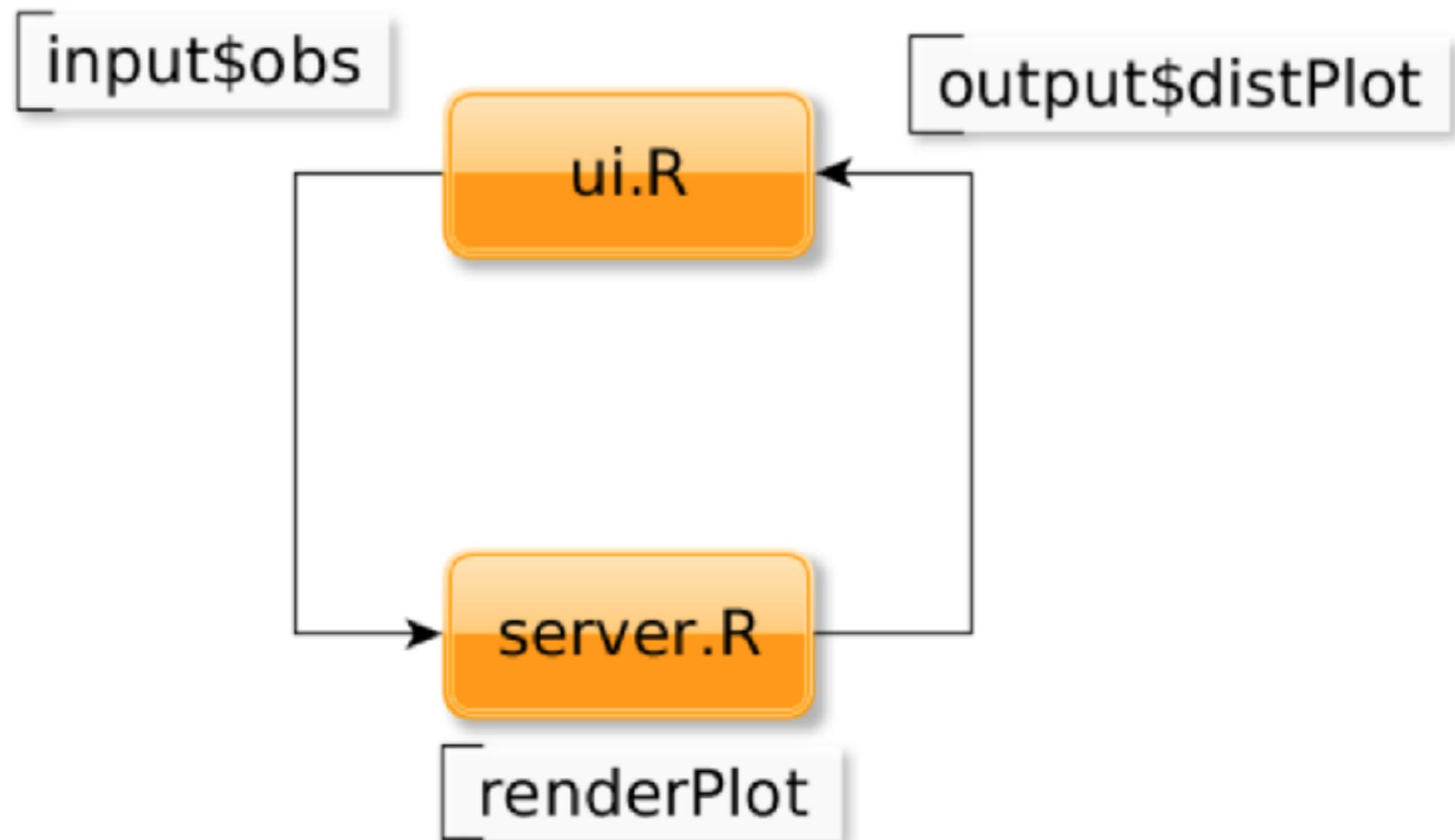
  output$distPlot <- renderPlot({
    x      <- faithful[, 2] # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
```

# Relationship of ui.R and server.R



# Relationship of ui.R and server.R



# For Spatial data

**sp, maptools** - Tools for loading and using spatial data including shapefiles.

**maps** - Easy to use map polygons for plots.

**ggmap** - Download street maps straight from Google maps and use them as a background in your ggplots.

# To write your own R packages

**devtools** - An essential suite of tools for turning your code into an R package.

**testthat** - testthat provides an easy way to write unit tests for your code projects.

**roxygen2** - A quick way to document your R packages. roxygen2 turns inline code comments into documentation pages and builds a package namespace.



# Build R Package

# Why write an R package?

- To keep track of the miscellaneous R functions that you write and reuse.
- To distribute the data and software that accompany a paper.

# Creating the package skeleton

1. New Project -> New Directory -> R Package
2. Pick the name for your package (only letters, numbers, and .), and where it'll live locally.
3. Check the “Create git repository” because why not.
4. Click on the DESCRIPTION file, and edit it.
5. Put your code files in the R/ subdirectory.
6. Load the package (“Build” tab, More -> Load all; or ⌘-⇧-L)

# DESCRIPTION file

Package: veepack

Version: 0.1

Date: 2016-11-27

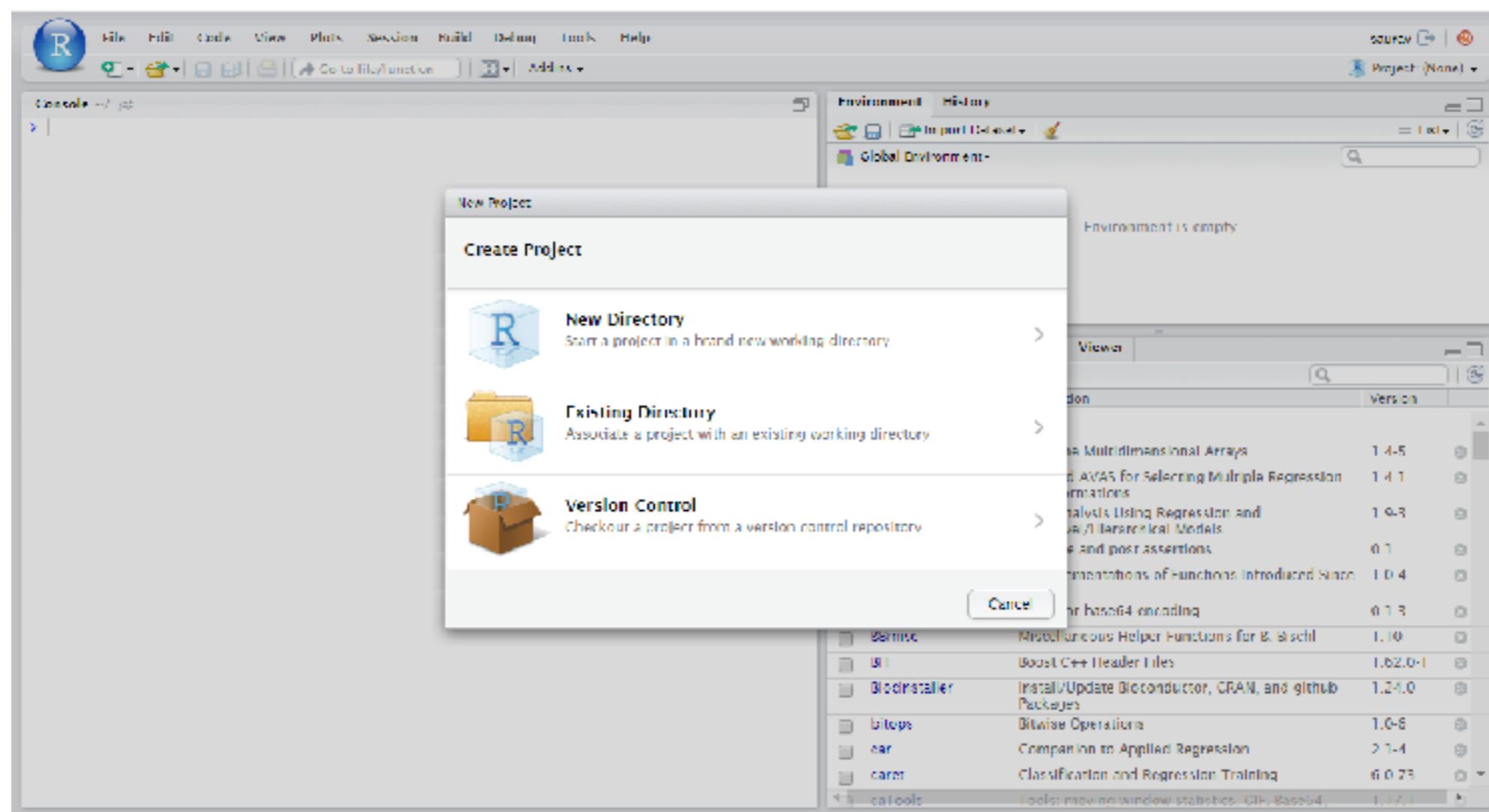
Title: Some functions

Description: Some description

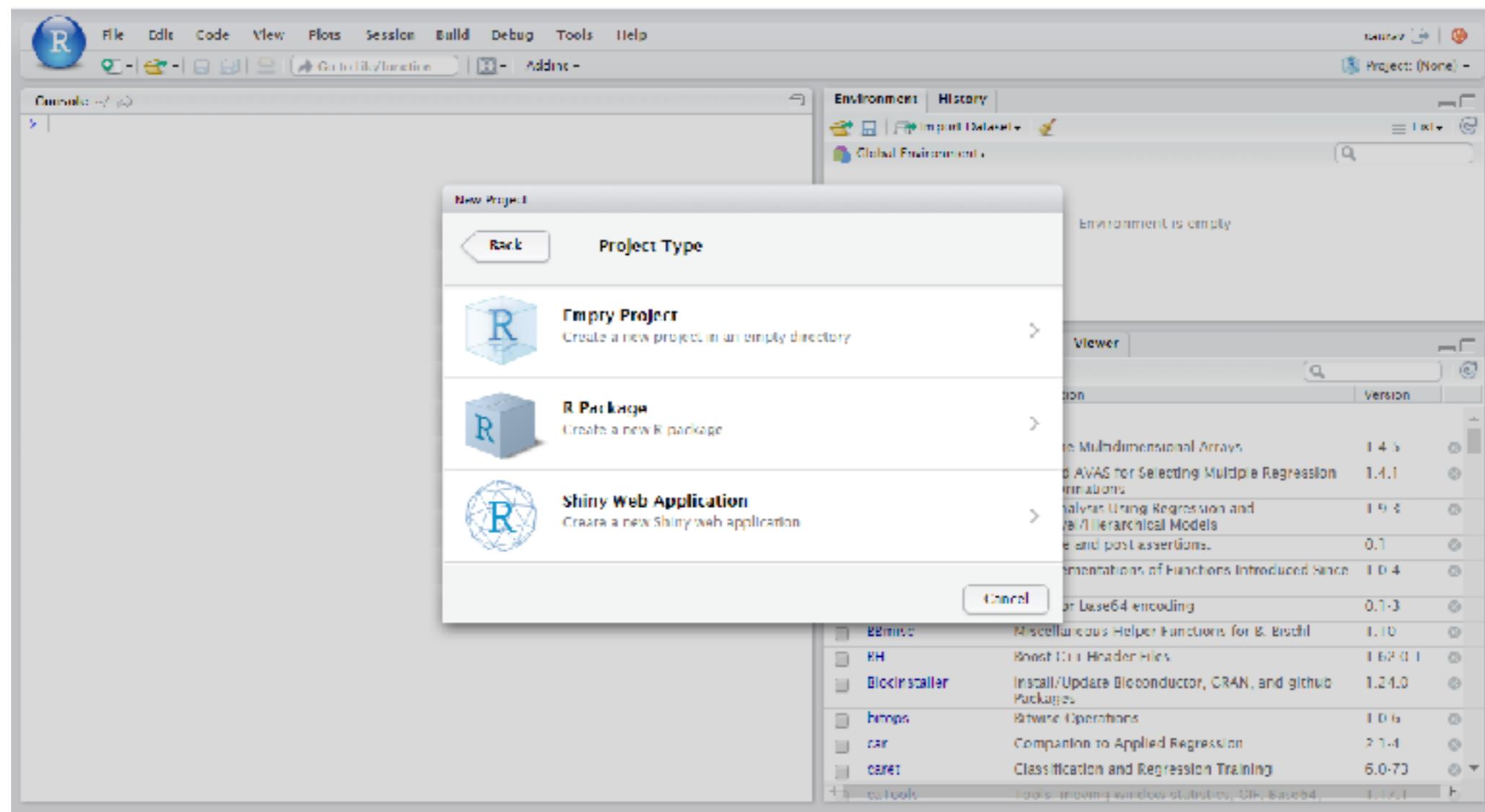
Author: Veerasak Kritsanaphan<veerasak.kritsanaphan@gmail.com>

Maintainer: Veerasak Kritsanaphan<veerasak.kritsanaphan@gmail.com>

1.Create a new project by going to File > New Project.

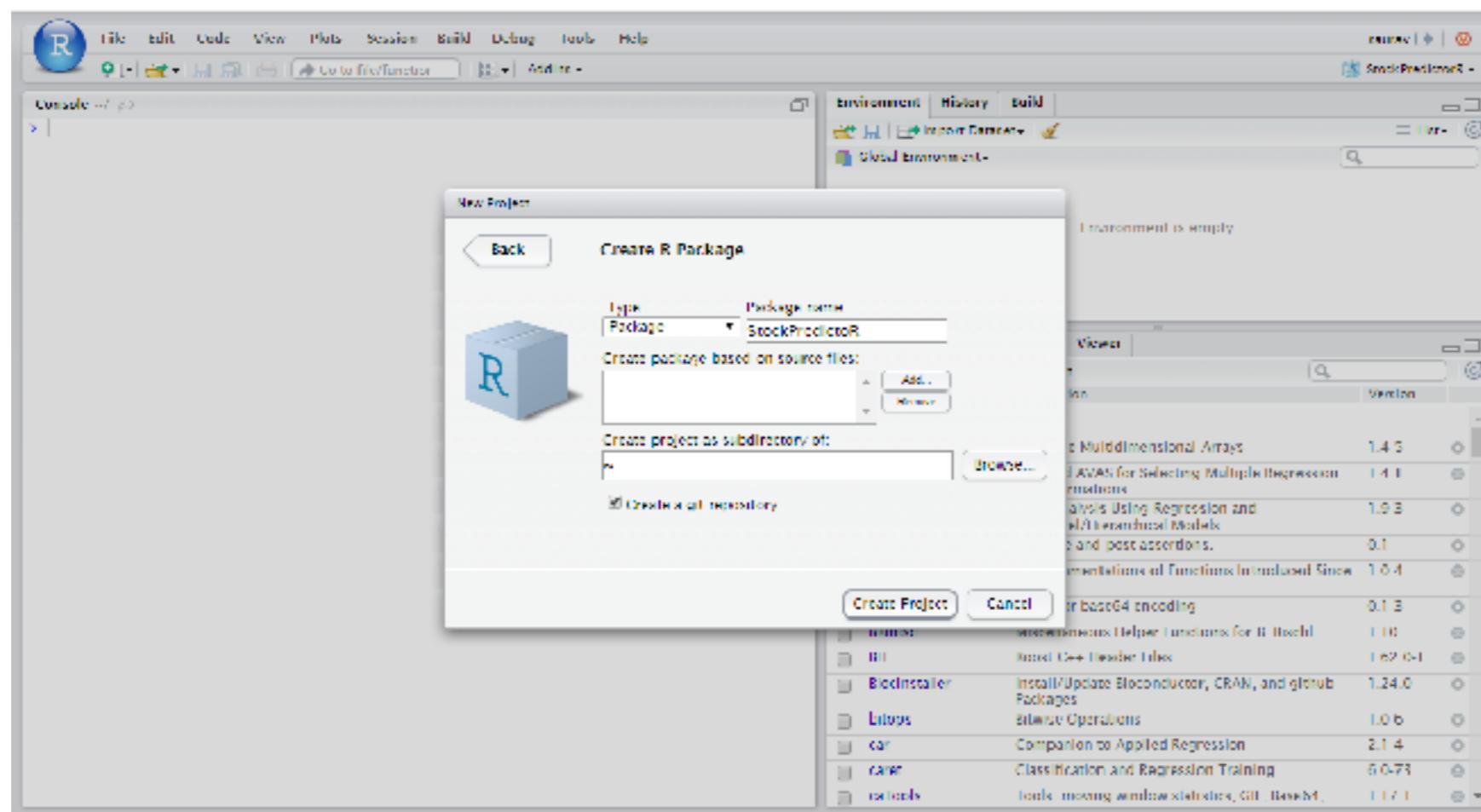


2. You can choose an existing directory or choose to create a new one. Then choose the project type as R package.

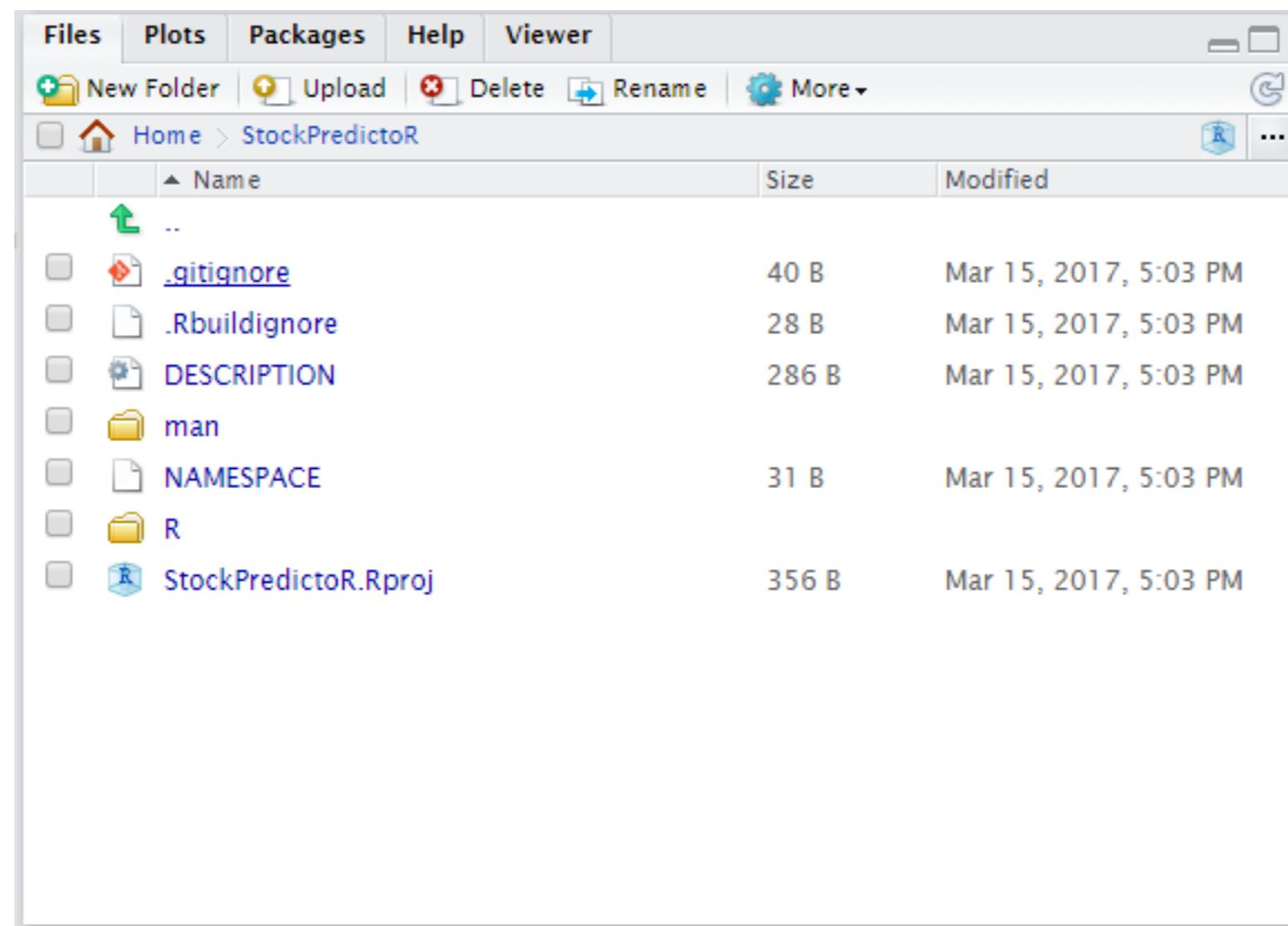


3. This is the time you choose an appropriate name for your package.

I'm going to name it StockPredictoR. For naming your packages, you can use periods (like predictoR ) or camel case as we are using here. I'll advice you to not use underscores while naming your packages. Also, choose the appropriate subdirectory that you want to store this project in.



4. This will create the following files in the directory. All the code will be stored in R folder while the manual and supporting documents will be stored in man folder.



## 5. Build package

The screenshot shows the RStudio IDE interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Tools, and Help. The Build menu is open, showing options: Build and Reload (Ctrl+Shift+B), Clean and Rebuild, Test Package (Ctrl+Alt+F7), Check Package (Ctrl+Shift+E), **Build Source Package** (highlighted in blue), Build Binary Package, Document (Ctrl+Shift+D), Stop Build, and Configure Build Tools... The main workspace shows an R script named DESCRIPTION.R with code for a package named StockPredictoR. The code includes a function stock\_predict that takes a symbol and returns a probability. The R console at the bottom shows the package being built and tested, with examples of the function being run.

```
R DESCRIPTION.R
#> #'
#> #' @title Predicts Stock Price Movement
#> #' @description This package predicts tomorrow's market
#> #' @param symbol
#> #' @return NULL
#> #' @examples stock_predict("AAPL")
#> #' @export
#>
#> stock_predict<-function(symbol)
#> {
#>   #Importing price data for the given symbol
#>   data<-data.frame(as.xts(getSymbols(symbol)))
#>   #
#> }
#>
#> (Top Level) : R Script
```

```
Console -/StockPredictoR/ ~
Restarting R session...

> library(StockPredictoR)
> example("stock_predict")

Stock p> stock_predict("AAPL")
[1] "Probability of Stock price going up tomorrow:"
[2] 0.448176
> stock_predict("GOOGL")
[1] "Probability of Stock price going up tomorrow:"
[1] 0.4718388

Restarting R session...

> library(StockPredictoR)
>
```

The right pane shows the package structure in the Environment tab:

```
3.2
* installing 'source' package 'StockPredictoR' ...
** R
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (StockPredictoR)
```

Files tab content:

Name	Size	Modified
.gitignore	10 B	Mar 15, 2017, 5:03 PM
Rbuildignore	28 B	Mar 15, 2017, 5:03 PM
Rhistory	8 B	Mar 15, 2017, 7:19 PM
DESCRIPTION	424 B	Mar 16, 2017, 3:16 PM
man		
NAMESPACE	31 B	Mar 15, 2017, 5:03 PM
R		
StockPredictoR.Rproj	433 B	Mar 16, 2017, 2:54 PM

# Workshop 7: Building Packages

Try build your own package

- A) Create your own Package name "addtwo"
- B) Create function addTwo
- C) Build and Install new package

# Hypothesis testing in R

# Fundamental of Hypothesis Testing

There two types of **statistical inferences**, **Estimation** and **Hypothesis Testing**

**Hypothesis Testing:** A hypothesis is a claim (assumption) about one or more population parameters.

- Average price of a six-pack in the U.S. is  $\mu = \$4.90$
- The population mean monthly cell phone bill of this city is:  
 $\mu = \$42$
- The average number of TV sets in U.S. Homes is equal to three;  $\mu = 3$



It Is always about a population parameter, not about a sample statistic

Sample evidence is used to assess the probability that the claim about the population parameter is true

**A. It starts with Null Hypothesis,  $H_0$**

$$H_0: \underline{\quad} = \underline{\quad}$$

1. We begin with the assumption that  $H_0$  is true and any difference between the sample statistic and true population parameter is due to chance and not a real (systematic) difference.
2. Similar to the notion of “innocent until proven guilty”
3. That is, “innocence” is a null hypothesis.

## Null Hypo, Continued

4. Refers to the status quo
5. Always contains “=”, “≤” or “≥” sign
6. May or may not be **rejected**

## B. Next we state the Alternative Hypothesis, $H_1$

1. Is the opposite of the null hypothesis
  1. e.g., The average number of TV sets in U.S. homes is not equal to 3  
 $( H_1: \mu \neq 3 )$
2. Challenges the status quo
3. Never contains the “=”, “≤” or “≥” sign
4. May or may not be **proven**
5. Is generally the hypothesis that the researcher is trying to prove.  
Evidence is always examined with respect to  $H_1$ , never with respect to  $H_0$ .
6. We never “accept”  $H_0$ , we either “reject” or “not reject” it



## Summary:

- In the process of hypothesis testing, the null hypothesis initially is assumed to be true
- Data are gathered and examined to determine whether the evidence is strong enough with respect to the alternative hypothesis to reject the assumption.
- In other words, the burden is placed on the researcher to show, using sample information, that the null hypothesis is false.
- If the sample information is sufficient enough in favor of the alternative hypothesis, then the null hypothesis is rejected. This is the same as saying if the persecutor has enough evidence of guilt, the “innocence is rejected.”
- Of course, erroneous conclusions are possible, type I and type II errors.

# Reason for Rejecting $H_0$

Illustration: Let say, we **assume** that average age in the US is 50 years ( $H_0=50$ ). If in fact this is the true (unknown) population mean, it is unlikely that we get a sample mean of 20. So, if we have a sample that produces an average of 20, then we **reject** that the null hypothesis that average age is 50. (note that we are rejecting our assumption or claim). (would we get 20 if the true population mean was 50? NO. That is why we reject 50)

## How Is the Test done?

We use the distribution of a Test Statistic, such as Z or t as the criteria.

## A. Rejection Region Method:

Divide the distribution into rejection and non-rejection regions

Defines the unlikely values of the sample statistic if the null hypothesis is true, the critical value(s)

Defines **rejection region** of the sampling distribution

Rejection region(s) is designated by  $\alpha$ , (level of significance)

Typical values are .01, .05, or .10

$\alpha$  is selected by the researcher at the beginning

$\alpha$  provides the critical value(s) of the test

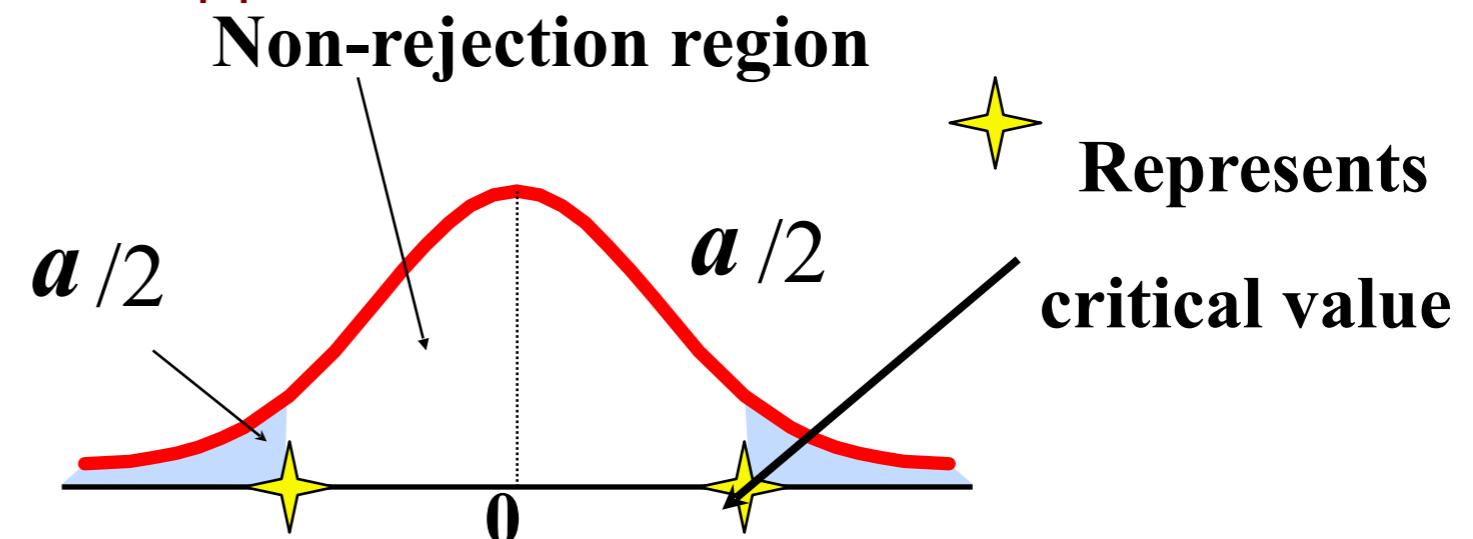
## Rejection Region or Critical Value Approach:

Level of significance =  $\alpha$

$$H_0: \mu = 12$$

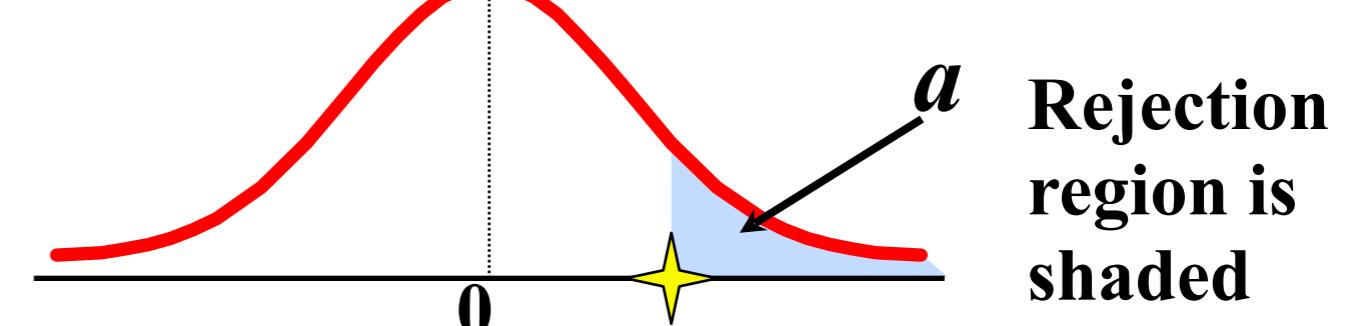
$$H_1: \mu \neq 12$$

Two-tail test



$$H_0: \mu \leq 12 \quad H_1: \mu > 12$$

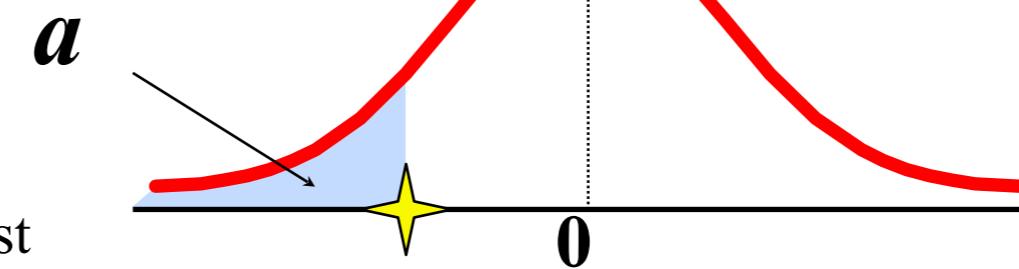
Upper-tail test



$$H_0: \mu \geq 12$$

$$H_1: \mu < 12$$

Lower-tail test



## P-Value Approach –

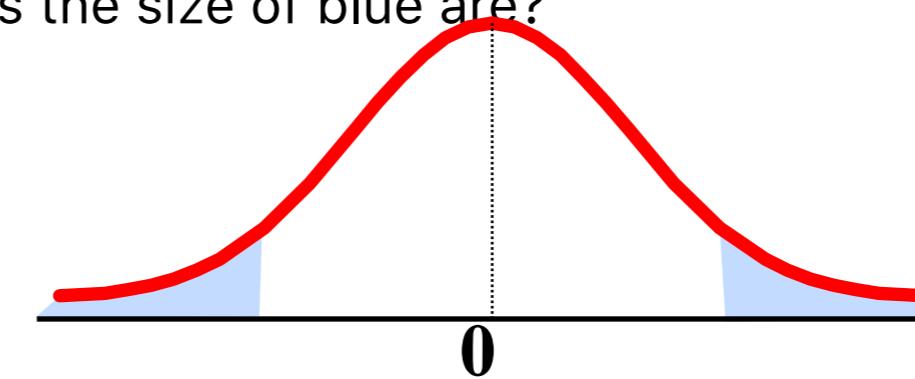
P-value=Max. Probability of (Type I Error), calculated from the sample.

Given the sample information what is the size of blue area?

$$H_0: \mu = 12$$

$$H_1: \mu \neq 12$$

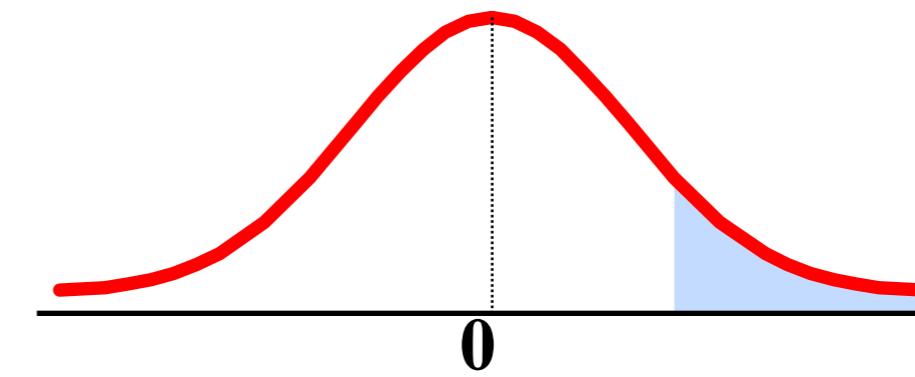
Two-tail test



$$H_0: \mu \leq 12 \quad H_1:$$

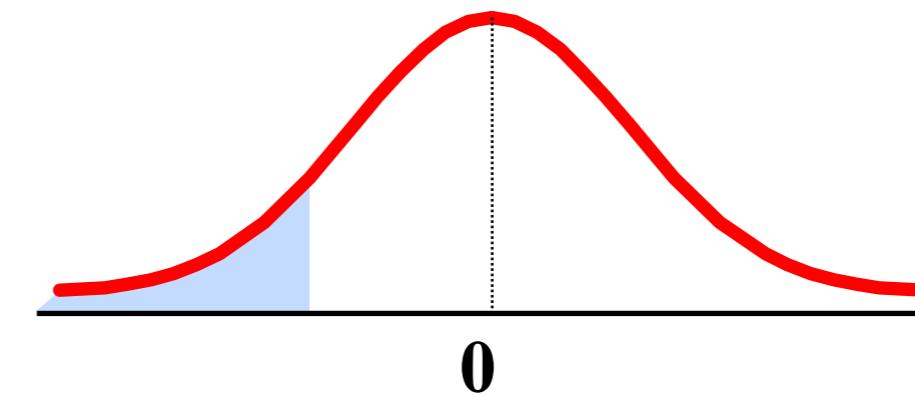
$$\mu > 12$$

Upper-tail test



$$H_0: \mu \geq 12$$

$$H_1: \mu < 12$$



# Type I and II Errors:

The size of  $\alpha$ , the rejection region, affects the risk of making different types of incorrect decisions.

## Type I Error

Rejecting a **true null hypothesis** when it should **NOT** be rejected

Considered a serious type of error

The probability of Type I Error is  $\alpha$

It is also called **level of significance** of the test

## Type II Error

Fail to reject a **false null hypothesis** that should have been rejected

The probability of Type II Error is  $\beta$

Decision		Actual Situation			
		Hypothesis Testing		Legal System	
		H0 True	H0 False	Innocence	Not innocence
Do Not Reject $H_0$	No Error $(1 - \alpha)$	Type II Error $(\beta)$		No Error (not guilty, found not guilty) $(1 - \alpha)$	Type II Error (guilty, found not guilty) $(\beta)$
	Type I Error $(\alpha)$	No Error $(1 - \beta)$		Type I Error (Not guilty, found guilty) $(\alpha)$	No Error (guilty, found guilty) $(1 - \beta)$



Type I and Type II errors cannot happen at the same time

1. Type I error can only occur if  $H_0$  is **true**
2. Type II error can only occur if  $H_0$  is **false**
3. There is a tradeoff between type I and II errors. If the probability of type I error (  $\alpha$  ) increased, then the probability of type II error (  $\beta$  ) declines.
4. When the difference between the hypothesized parameter and the actual true value is small, the probability of type two error (the non-rejection region) is larger.
5. Increasing the sample size,  $n$ , for a given level of  $\alpha$ , reduces  $\beta$

## B. P-Value approach to Hypothesis Testing:

1. The rejection region approach allows you to examine evidence but restrict you to not more than a certain probability (say  $\alpha = 5\%$ ) of rejecting a true  $H_0$  by mistake.
2. The P-value approach allows you to use the information from the sample and then calculate the **maximum probability of rejecting a true  $H_0$  by mistake.**
3. Another way of looking at P-value is the probability of observing a sample information of “A=11.5” when the true population parameter is “12=B”. The P-value is the **maximum probability** of such mistake taking place.

- 
4. That is to say that P-value is the smallest value of  $\alpha$  for which  $H_0$  can be rejected based on the sample information
  5. Convert Sample Statistic (e.g., sample mean) to Test Statistic (e.g., Z statistic)
  6. Obtain the **p-value** from a table or computer
  7. Compare the **p-value** with  $\alpha$

If  $p\text{-value} < \alpha$  , reject  $H_0$

If  $p\text{-value} \geq \alpha$  , do not reject  $H_0$

# Test of Hypothesis for the Mean

$\sigma$  known

$\sigma$  Unknown

The test statistic is:

$$z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

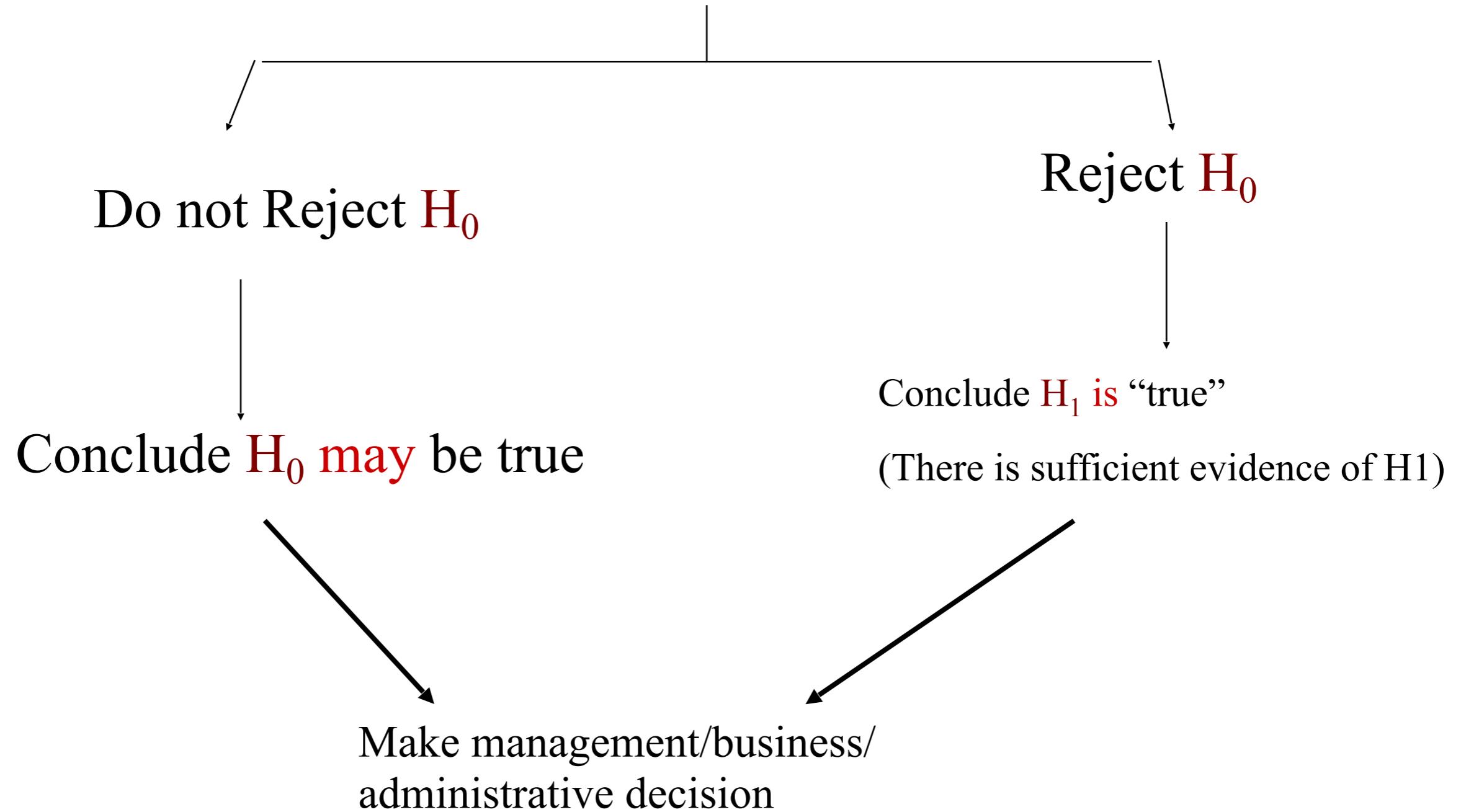
The test statistic is:

$$t_{n-1} = \frac{\bar{X} - \mu}{\frac{s}{\sqrt{n}}}$$

# Steps to Hypothesis Testing

1. State the  $H_0$  and  $H_1$  clearly
2. Identify the test statistic (two-tail, one-tail, and Z or t distribution)
3. Depending on the type of risk you are willing to take, specify the level of significance,
4. Find the decision rule, critical values, and rejection regions. If –  
 $CV < \text{actual value (sample statistic)} < +CV$ , then **do not reject the  $H_0$**

## Make statistical decision



# When do we use a two-tail test?

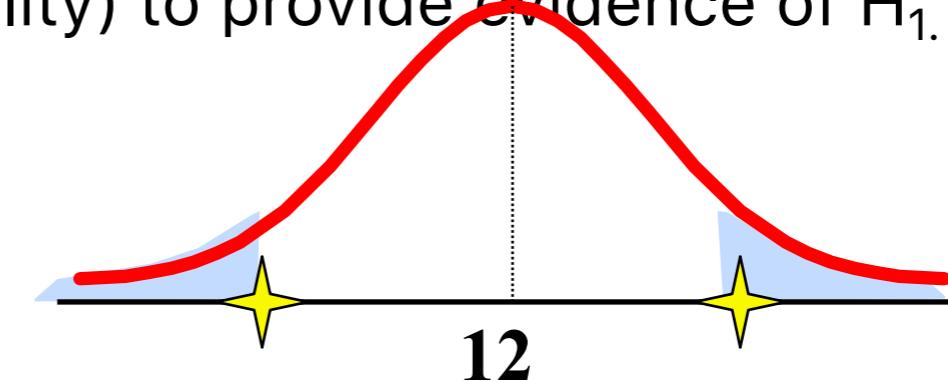
The answer depends on the question you are trying to answer.

A two-tail is used when the researcher has no idea which direction the study will go, interested in both direction. (example: testing a new technique, a new product, a new theory and we don't know the direction)

A new machine is producing 12 fluid once can of soft drink. The quality control manager is concern with cans containing too much or too little. Then, the test is a two-tailed test. That is the two rejection regions in tails is most likely (higher probability) to provide evidence of  $H_1$ .

$$H_0 : \mu = 12 \text{ oz}$$

$$H_1 : \mu \neq 12 \text{ oz}$$



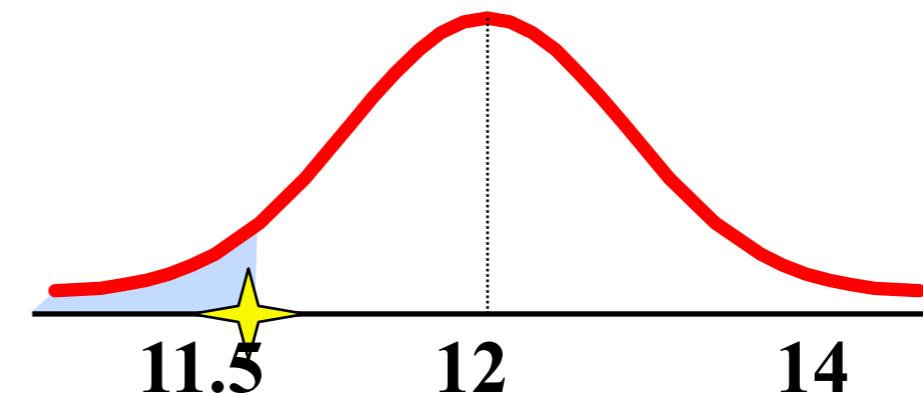
One-tail test is used when the researcher is interested in the direction.

Example: The soft-drink company puts a label on cans claiming they contain 12 oz. A consumer advocate desires to test this statement. She would assume that each can contains **at least** 12 oz and tries to find evidence to the contrary. That is, she examines the evidence for less than 12 oz.

What tail of the distribution is the most logical (higher probability) to find that evidence? The only way to reject the claim is to get evidence of less than 12 oz, left tail.

$$H_0 : \mu \geq 12 \text{ oz}$$

$$H_1 : \mu < 12 \text{ oz}$$



# Review of Hypo. Testing

What is HT?

Probability of making erroneous conclusions

Type I – only when Null Hypo is true

Type II – only when Null Hypo is false

Two Approaches

The Rejection or Critical Value Approach

The P-value Approach (we calculate the observed level of significance)

Test Statistics

Z- distribution if Population Std. Dev. is Known

t-distribution if the Population Std. Dev. is unknown

## Rejection Region or Critical Value Approach:

The given level of significance =  $\alpha$

$$H_0: \mu = 12$$

$$H_1: \mu \neq 12$$

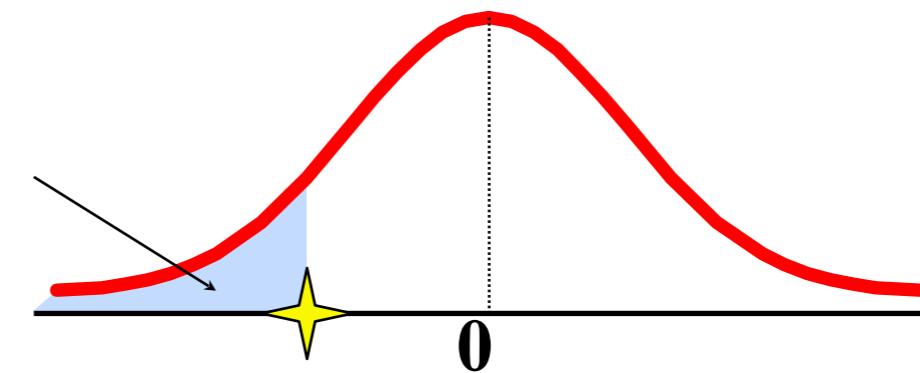
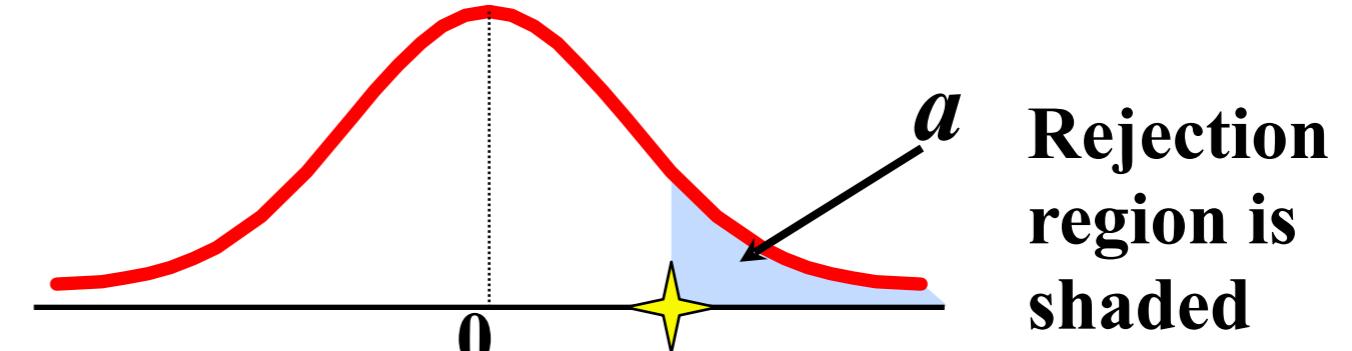
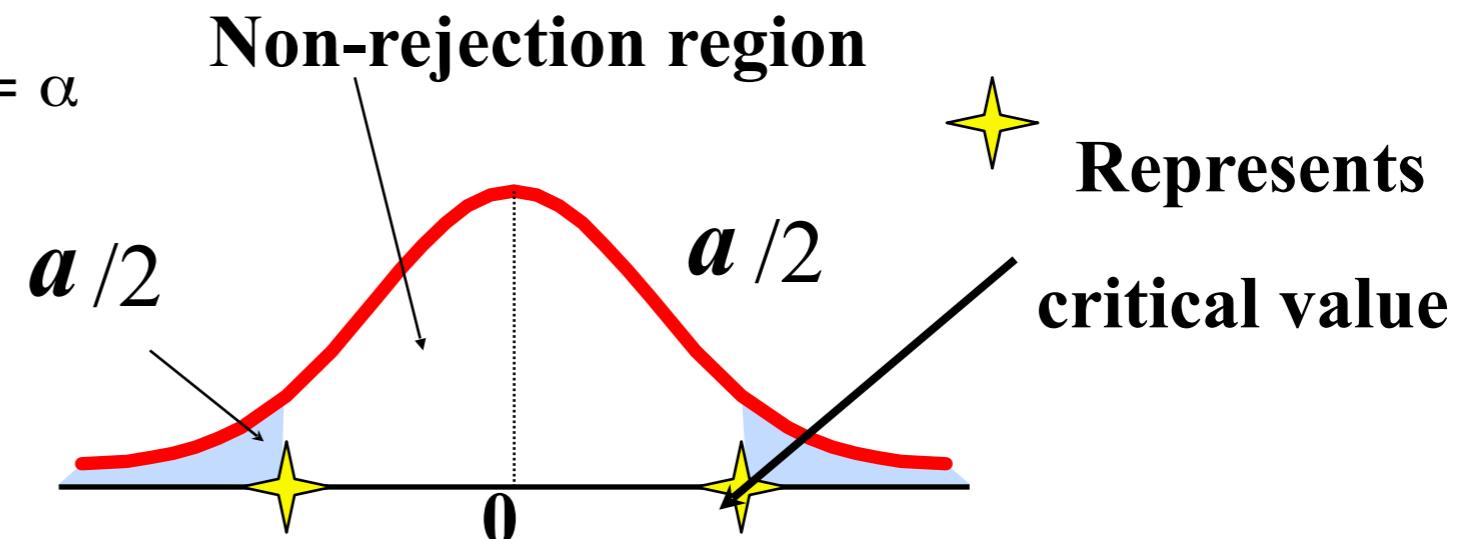
$$H_0: \mu \leq 12 \quad H_1: \mu > 12$$

$$H_0: \mu \geq 12 \quad H_1: \mu < 12$$

Two-tail test

Upper-tail test

Lower-tail test



## P-Value Approach –

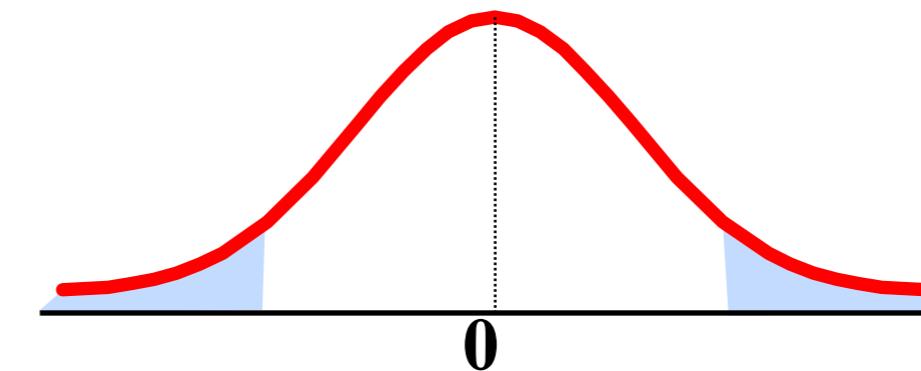
P-value=Max. Probability of (Type I Error), calculated from the sample.

Given the sample information what is the size of the blue areas? (The observed level of significance)

$$H_0: \mu = 12$$

$$H_1: \mu \neq 12$$

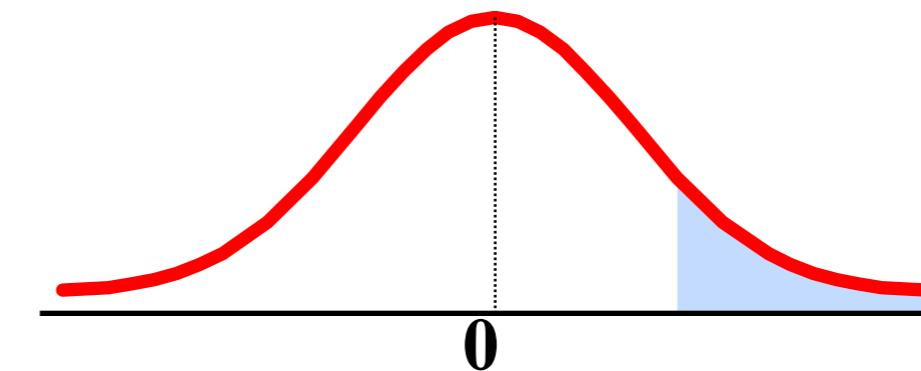
Two-tail test



$$H_0: \mu \leq 12 \quad H_1:$$

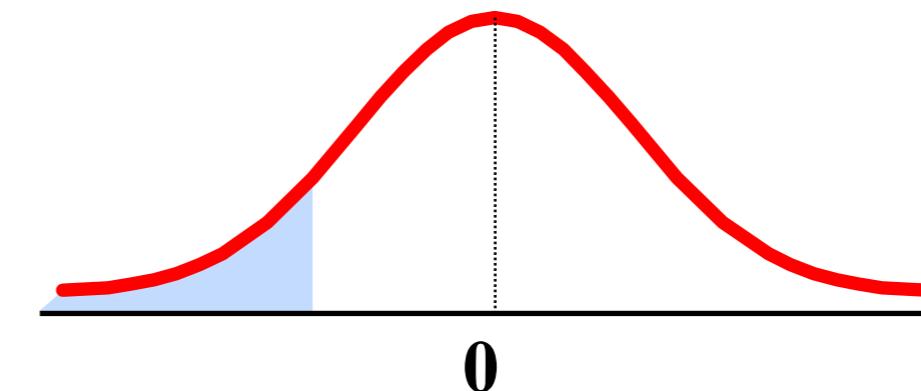
$$\mu > 12$$

Upper-tail test



$$H_0: \mu \geq 12$$

$$H_1: \mu < 12$$





Example 1:

Let's assume a sample of 25 cans produced a sample mean of 11.5 0z and the population std dev=1 0z.

Question 1:

At a 5% level of significance (that is allowing for a maximum of 5% prob. of rejecting a true null hypo), is there evidence that the population mean is different from 12 oz?

Null Hypo is:?

Alternative Hypo is?

Can both approaches be used to answer this question?

- 
- A: **Rejection region approach:** calculate the actual test statistics and compare it with the critical values
  - B: **P-value approach:** calculate the actual probability of type I error given the sample information. Then compare it with 1%, 5%, or 10% level of significance.

Interpretation of Critical Value/Rejection Region Approach:

Interpretation of P-value Approach:

## Question 2:

At a 5% level of significance (that is allowing for a maximum of 5% prob. of rejecting a true null hypo), is the evidence that the population mean is **less than 12 oz**?

Null Hypo is:?

Alternative Hypo is?

Can both approaches be used to answer this question?

Interpretation of Critical Value Approach:

Interpretation of P-value Approach:

### Question 3:

If in fact the pop. mean is 12 oz, what is the probability of obtaining a sample mean of 11.5 or less oz (sample size 25)? Null

Null Hypo is:?

Alternative Hypo is?

### Question 4:

If in fact the pop. mean is 12 oz, and the sample mean is 11.5 (or less), what is the probability of erroneously rejecting the null hypo that the pop. mean is 12 oz?

Null Hypo is:?

Alternative Hypo is?

Can both approaches be used to answer these question?

# Connection to Confidence Intervals

While the confidence interval estimation and hypothesis testing serve different purposes, they are based on same concept and conclusions reached by two methods are consistent for a two-tail test.

In CI method we estimate an interval for the population mean with a degree of confidence. If the estimated interval **contains** the hypothesized value under the hypothesis testing, then this is equivalent of **not rejecting** the null hypothesis. For example: for the beer sample with mean 5.20, the confidence interval is:

$$P(4.61 \leq \mu \leq 5.78) = 95\%$$

Since this interval contains the Hypothesized mean (\$4.90), we do not (did not) reject the null hypothesis at  $\alpha = .05$

Did not reject and within the interval, thus consistent results.



# t-test using R

# Testing differences in mean between two groups

Let's begin by loading the packages we'll need to get started

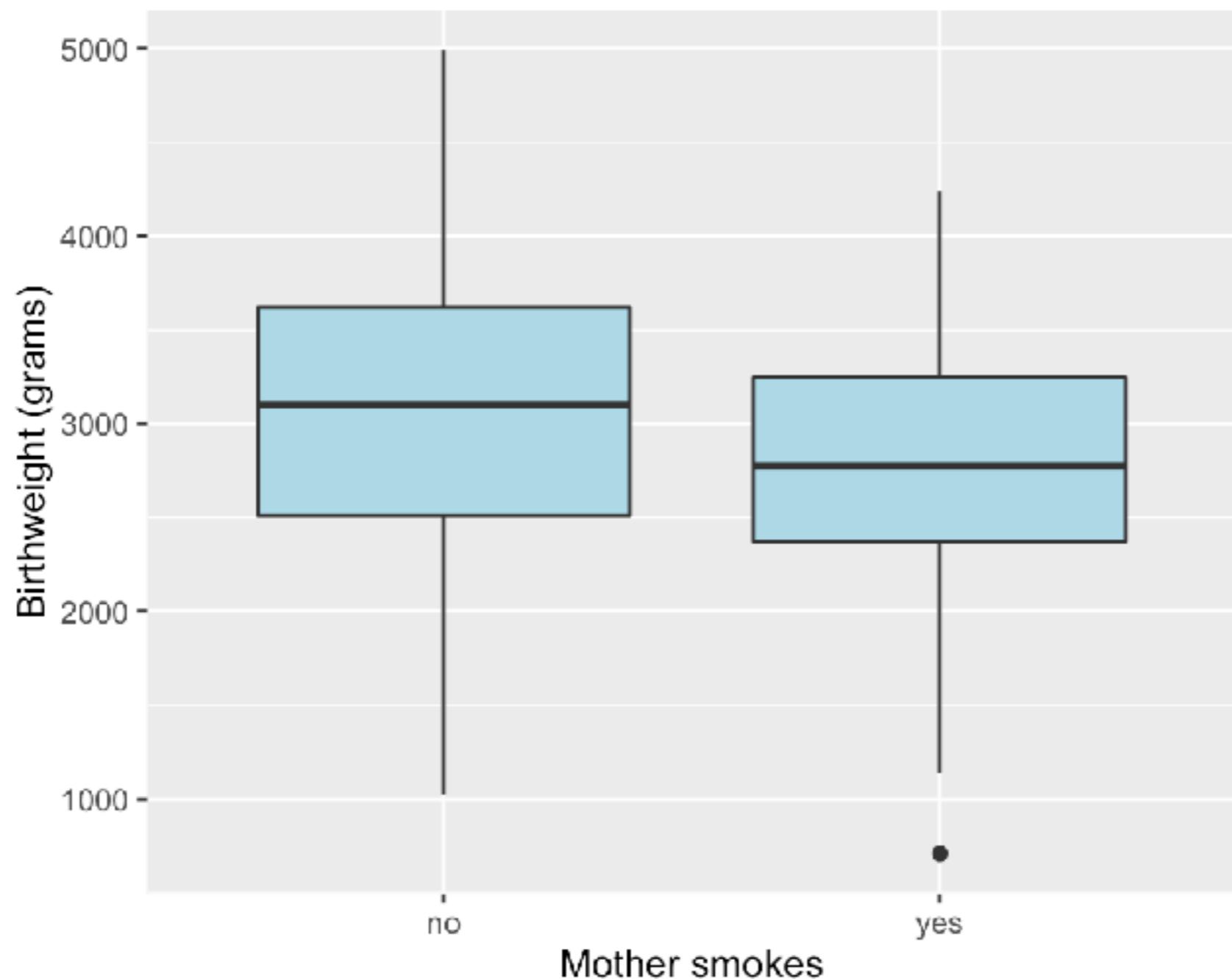
```
library(MASS)
library(plyr)
library(ggplot2)
```

```
# Rename the columns to have more descriptive names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
  "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
  "physician.visits", "birthwt.grams")

# Transform variables to factors with descriptive levels
birthwt <- transform(birthwt,
  race = as.factor(mapvalues(race, c(1, 2, 3),
    c("white", "black", "other"))),
  mother.smokes = as.factor(mapvalues(mother.smokes,
    c(0,1), c("no", "yes"))),
  hypertension = as.factor(mapvalues(hypertension,
    c(0,1), c("no", "yes"))),
  uterine.irr = as.factor(mapvalues(uterine.irr,
    c(0,1), c("no", "yes"))))
)
```

To start, it always helps to plot things

```
# Create boxplot showing how birthwt.grams varies between  
# smoking status  
qplot(x = mother.smokes, y = birthwt.grams,  
       geom = "boxplot", data = birthwt,  
       xlab = "Mother smokes",  
       ylab = "Birthweight (grams)",  
       fill = I("lightblue"))
```



This plot suggests that smoking is associated with lower birth weight.

# How can we assess whether this difference is statistically significant?

Let's compute a summary table

```
aggregate(birthwt.grams ~ mother.smokes, data = birthwt,  
          FUN = function(x) {c(mean = mean(x), sd = sd(x))})
```

```
##   mother.smokes birthwt.grams.mean birthwt.grams.sd  
## 1           no        3055.6957       752.6566  
## 2          yes       2771.9189       659.6349
```

The standard deviation is good to have, but to assess statistical significance we really want to have the standard error (which the standard deviation adjusted by the group size).

```
aggregate(birthwt.grams ~ mother.smokes, data = birthwt,  
          FUN = function(x) {c(mean = mean(x),  
                                se = sd(x) / sqrt(length(x))))})
```

```
##   mother.smokes birthwt.grams.mean birthwt.grams.se  
## 1           no        3055.69565     70.18559  
## 2          yes       2771.91892     76.68100
```

# t-test via t.test()

This difference is looking quite significant. To run a two-sample t-test, we can simple use the `t.test()` function.

```
birthwt.t.test <- t.test(birthwt.grams ~ mother.smokes, data = birthwt)  
birthwt.t.test
```

```
##  
## Welch Two Sample t-test  
##  
## data: birthwt.grams by mother.smokes  
## t = 2.7299, df = 170.1, p-value = 0.007003  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
##    78.57486 488.97860  
## sample estimates:  
## mean in group no mean in group yes  
##                3055.696                 2771.919
```

# p-value

```
birthwt.t.test$p.value # p-value
```

```
## [1] 0.007002548
```

```
birthwt.t.test$estimate # group means
```

```
## mean in group no mean in group yes  
## 3055.696 2771.919
```

```
birthwt.t.test$conf.int # confidence interval for difference
```

```
## [1] 78.57486 488.97860  
## attr(,"conf.level")  
## [1] 0.95
```

# Workshop 8: t-test

Testing means between two groups

(a) Using the Cars93 data and the `t.test()` function, run a t-test to see if average MPG.highway is different between US and non-US vehicles.

Try doing this both using the formula style input and the `x, y` style input.

(b) What is the confidence interval for the difference?

# Prepare data

```
library(MASS)
library(plyr)
library(ggplot2)
library(knitr)
```

```
# Rename the columns to have more descriptive names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
  "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
  "physician.visits", "birthwt.grams")

# Transform variables to factors with descriptive levels
birthwt <- transform(birthwt,
  race = as.factor(mapvalues(race, c(1, 2, 3),
    c("white", "black", "other"))),
  mother.smokes = as.factor(mapvalues(mother.smokes,
    c(0,1), c("no", "yes"))),
  hypertension = as.factor(mapvalues(hypertension,
    c(0,1), c("no", "yes"))),
  uterine.irr = as.factor(mapvalues(uterine.irr,
    c(0,1), c("no", "yes"))))
)
```

# One-way ANOVA example

Question: Is there a significant association between race and birthweight?

Here's a table showing the mean and standard error of birthweight by race.

```
aggregate(birthwt.grams ~ race, data = birthwt, FUN = mean)
```

```
##      race birthwt.grams
## 1 black     2719.692
## 2 other     2805.284
## 3 white     3102.719
```



**Terminology:** a  $k$ -way ANOVA is used to assess whether the mean of an outcome variable is constant across all combinations of  $k$  factors. The most common examples are 1-way ANOVA (looking at a single factor), and 2-way ANOVA (looking at two factors).

We'll use the `aov()` function. For convenience, `aov()` allows you to specify a formula.

```
summary(aov(birthwt.grams ~ race, data = birthwt))
```

```
##          Df  Sum Sq Mean Sq F value    Pr(>F)
## race       2 5015725 2507863   4.913 0.00834 ***
## Residuals 186 94953931  510505
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Workshop 9: ANOVA

Let's form our favourite birthwt data set.

```
# Rename the columns to have more descriptive names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
  "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
  "physician.visits", "birthwt.grams")

# Transform variables to factors with descriptive levels
birthwt <- transform(birthwt,
  race = as.factor(mapvalues(race, c(1, 2, 3),
    c("white", "black", "other"))),
  mother.smokes = as.factor(mapvalues(mother.smokes,
    c(0,1), c("no", "yes"))),
  hypertension = as.factor(mapvalues(hypertension,
    c(0,1), c("no", "yes"))),
  uterine.irr = as.factor(mapvalues(uterine.irr,
    c(0,1), c("no", "yes"))))
)
```

- 
- (a) Create a new factor that categorizes the number of physician visits into three levels: 0, 1, 2, 3 or more.
- Hint: One way of doing this is with `mapvalues`, by mapping all instances of 3, 4,... etc, to “3 or more”.
- (b) Run an ANOVA to determine whether the average birth weight varies across number of physician visits.

# Regression

# Introduction to Regression Analysis

Regression analysis is used to:

Predict values of a dependent variable,  $Y$ , based on its relationship with values of at least one independent variable,  $X$ .

Explain the impact of changes in an independent variable on the dependent variable by estimating the **numerical value** of the relationship

**Dependent variable:** the variable we wish to explain

**Independent variable:** the variable used to explain the dependent variable

# Simple Linear Regression Model

Only **one** independent variable (thus, simple), X

Relationship between X and Y is described by a linear function

Changes in Y are assumed to be caused by changes in X, that is,

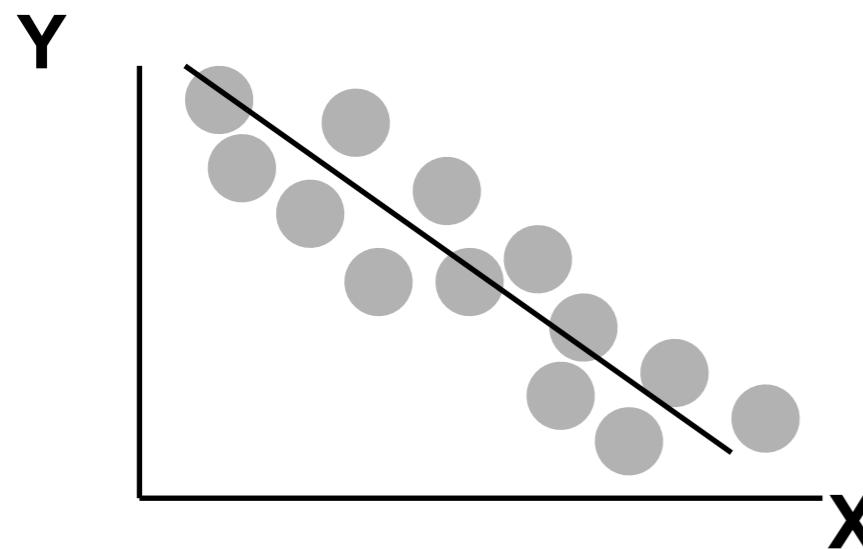
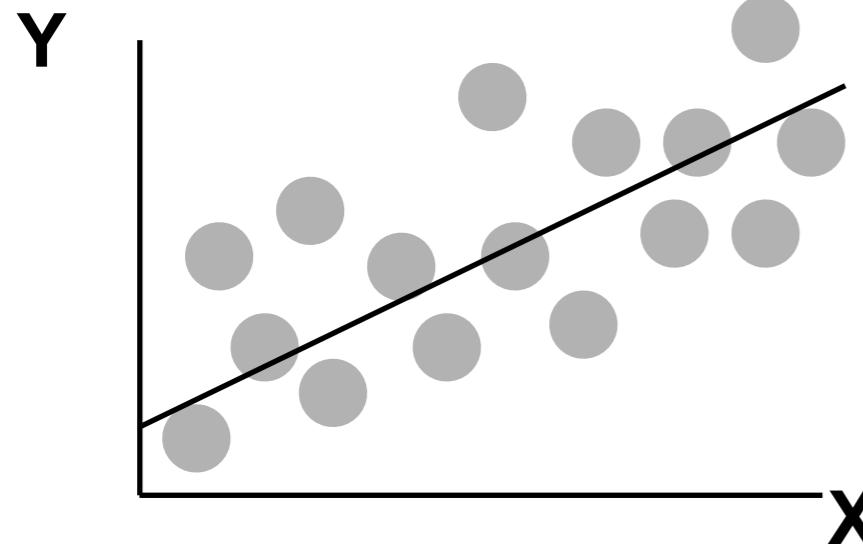
Change In X  $\xrightarrow{\text{Causes}}$  Change in Y

# Important points before we start a regression analysis:

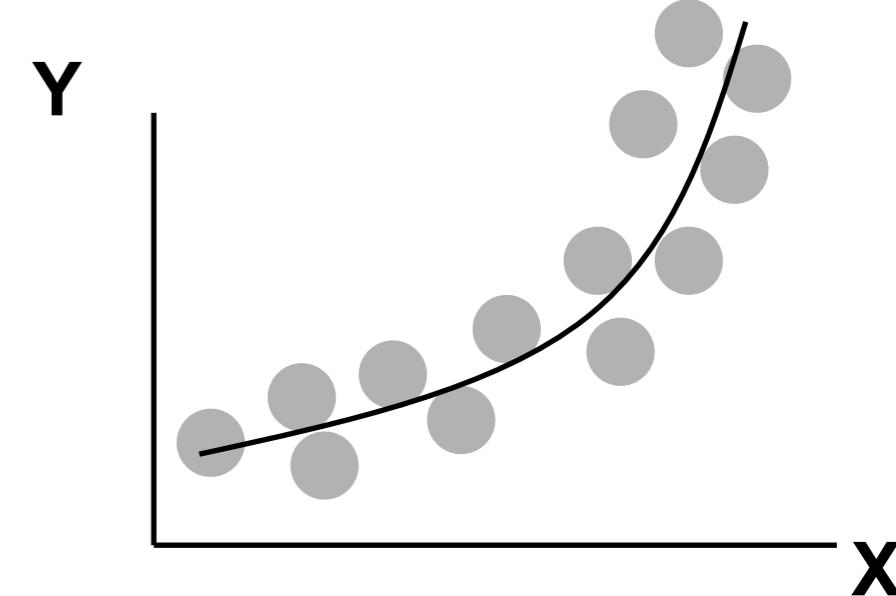
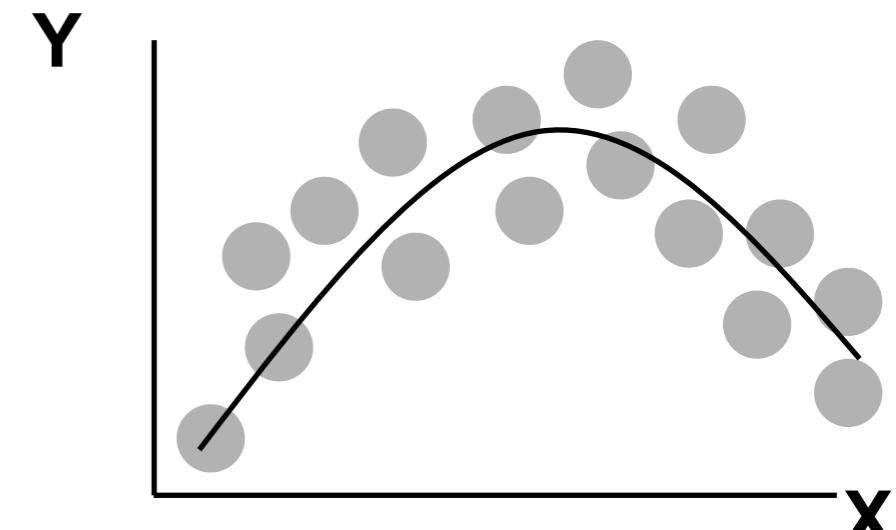
- The most important thing in deciding whether or not there is a relationship between X and Y is to have a systematic model that is based on logical reasons.
- Investigate the nature of the relationship between X and Y (use scatter diagram, covariance, correlation coefficient)
- Remember that regression is not an exact or deterministic mathematical equation. It is a **behavioral relationship** that is subject to randomness.
- Remember that X is not the only thing that explains the behavior of Y. There are other factor that you may not have information about.
- All you are trying to do is to have an estimate of the relationship using the **best linear fit** possible

# Types of Relationships

**Linear relationships**



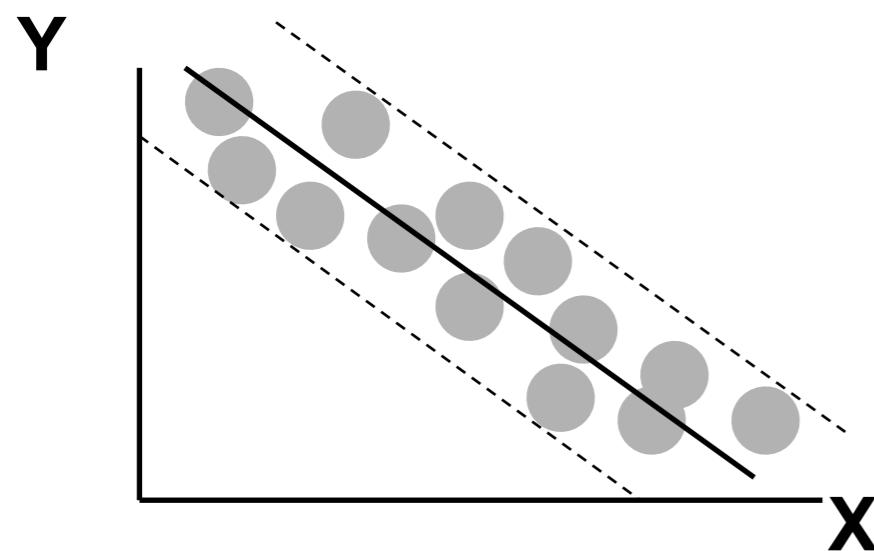
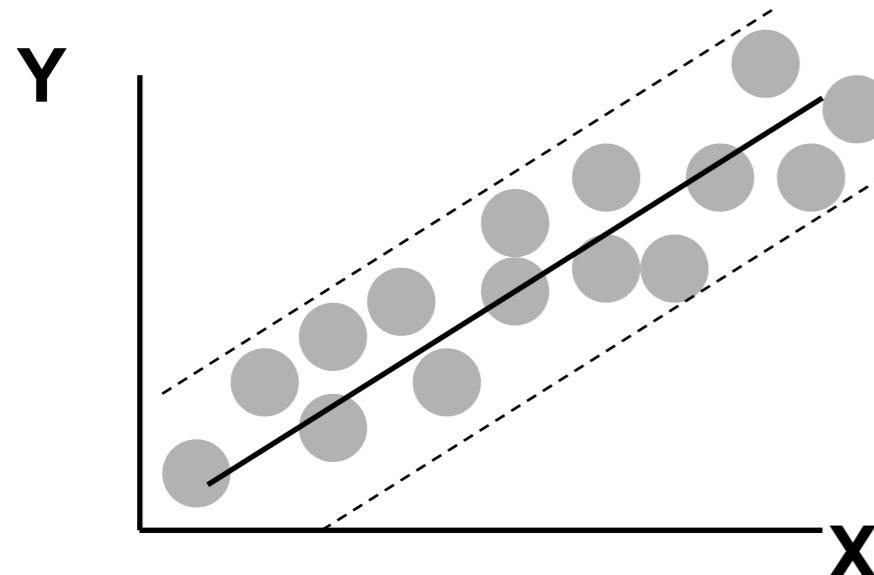
**Curvilinear relationships**



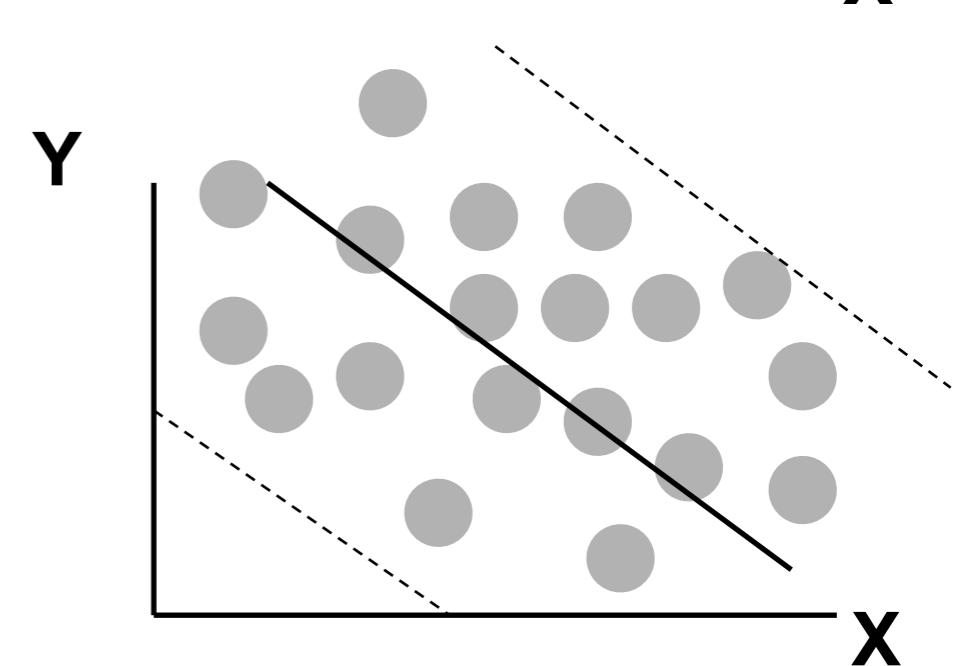
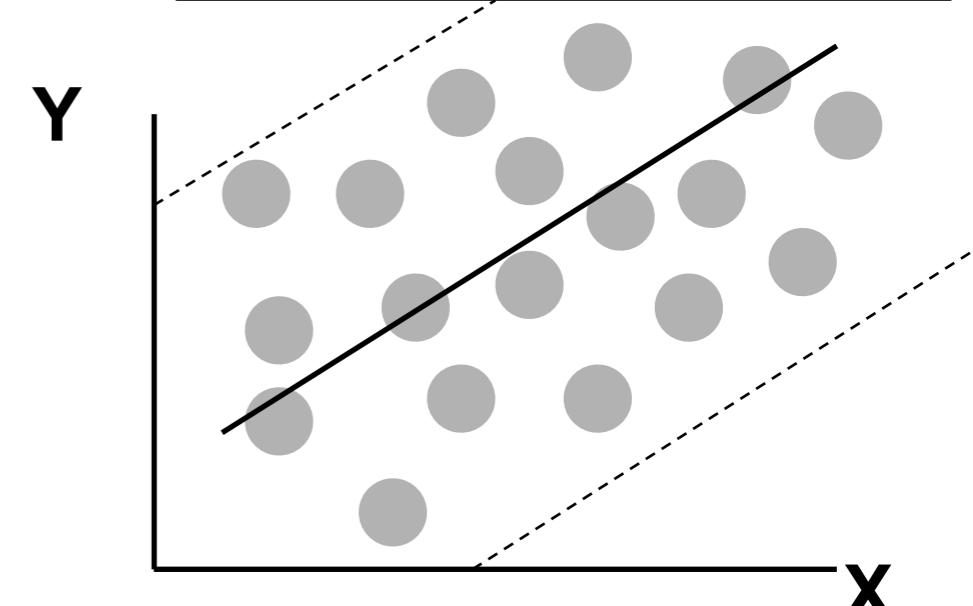
# Types of Relationships

(continued)

**Strong relationships**



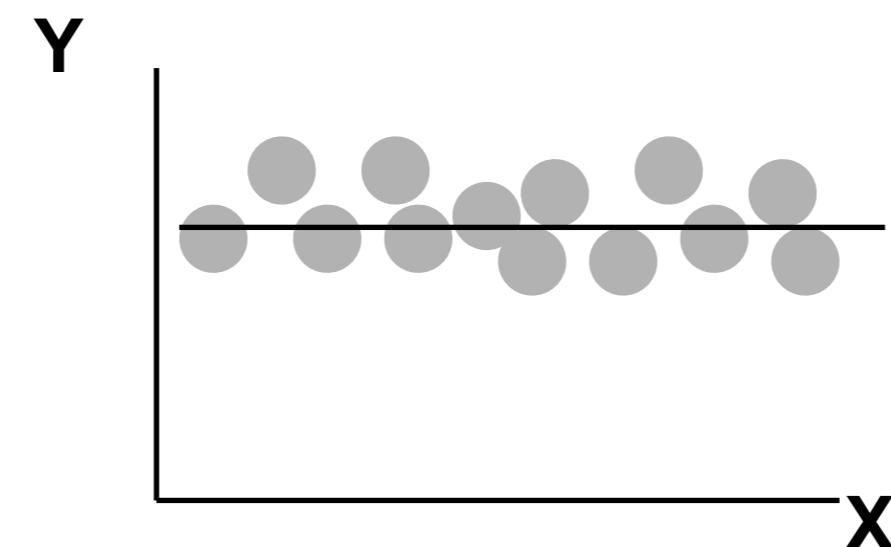
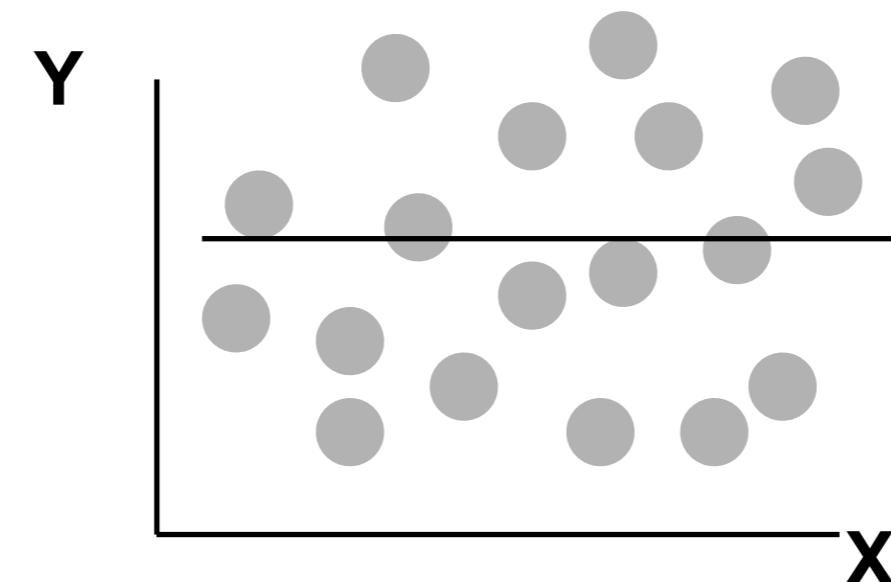
**Weak relationships**



# Types of Relationships

(continued)

**No relationship**



# Simple Linear Regression Conceptual Model

The population regression model: This is a conceptual model, a hypothesis, or a postulation

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

Diagram illustrating the components of the population regression model:

- Dependent Variable:  $Y_i$
- Population Y intercept:  $\beta_0$
- Population Slope Coefficient:  $\beta_1$
- Independent Variable:  $X_i$
- Random Error term:  $\epsilon_i$

The equation is divided into two main components:

- Linear component:**  $\beta_0 + \beta_1 X_i$
- Random Error component:**  $\epsilon_i$

- The model to be estimated from sample data is:

$$Y_i = b_0 + b_1 X_i + e_i$$

Residual  
(random  
error from  
the sample)

- The actual estimated from the sample

Estimated (or  
predicted) Y  
value for  
observation i

**Estimate** of the  
regression  
intercept

**Estimate** of the  
regression **slope**

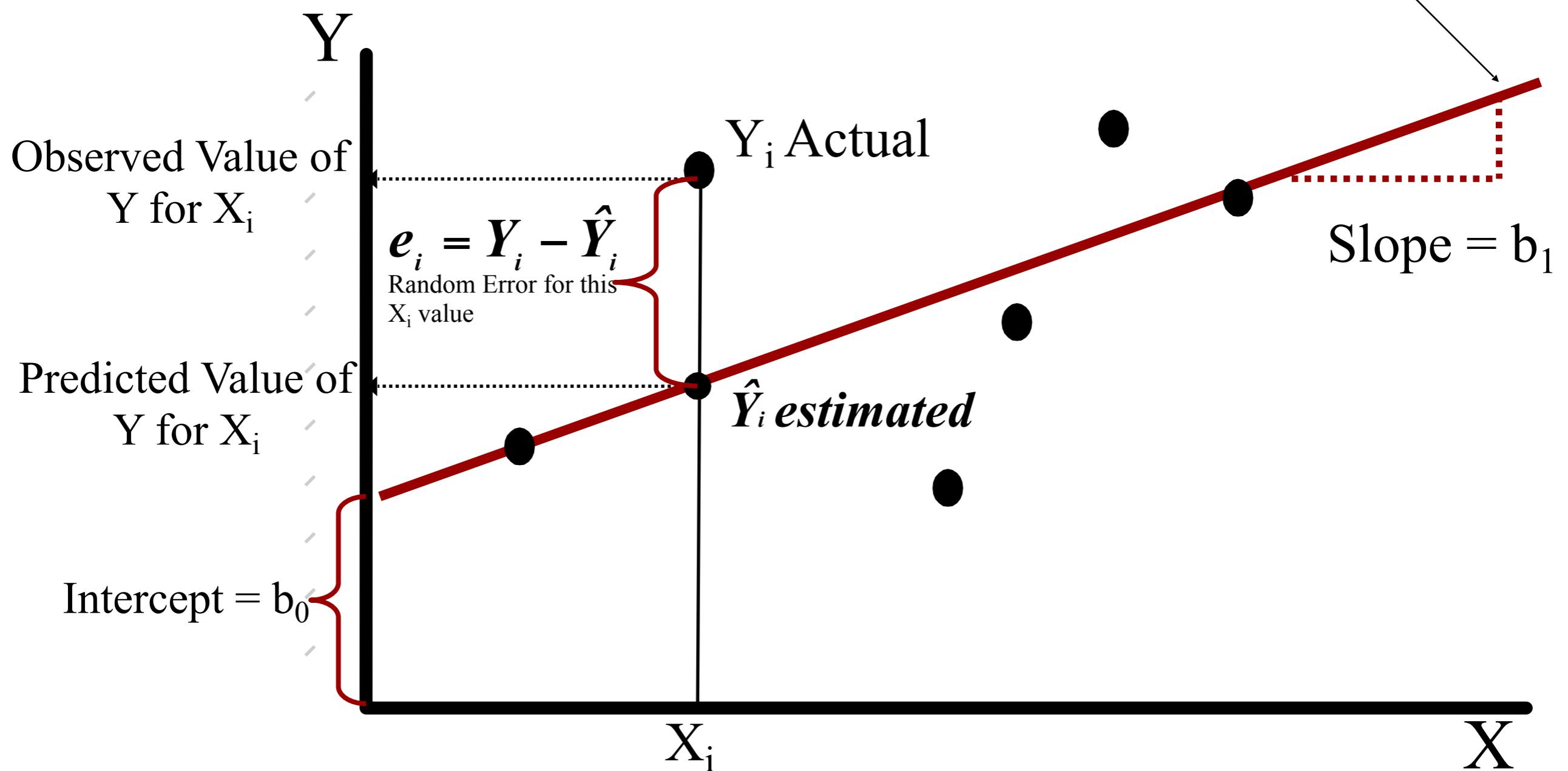
$$\hat{Y}_i = b_0 + b_1 X_i$$

Value of X for  
observation i

– Where  $e_i = Y_i - \hat{Y}_i$

# Simple Linear Regression Model

$$\hat{Y}_i = b_0 + b_1 X_i$$



# Interpretation of the slope and the intercept

$$\beta_0 = E(Y | X = 0) ; \quad \beta_1 = \Delta E(Y|X)/\Delta (X);$$

$b_0$  is the estimated average value of Y when the value of X is  $b_0$  zero

$b_1$  is the estimated change in the average value of Y as a result of a one-unit change in X

Units of measurement of X and Y are very important for the correct interpretation of the slope and the intercept

Example:  $\widehat{App\ Val} = 165.03 + 6.93 (Lot\ size)$

Predict the app. Value of a house with 10,000 s.f. lot size

$$\widehat{App\ Val} = 165.03 + 6.93 (10) = \$234,330$$

How Good is this prediction?

# How Good is the Model's prediction Power?

Total variation is made up of two parts:

$$\mathbf{SST} = \mathbf{SSR} + \mathbf{SSE}$$

Total Sum of  
Squares

Regression Sum  
of Squares

Error Sum of  
Squares

$$SST = \sum (Y_i - \bar{Y})^2 \quad SSR = \sum (\hat{Y}_i - \bar{Y})^2 \quad SSE = \sum (Y_i - \hat{Y}_i)^2$$

where:

$\bar{Y}$  = Average value of the dependent variable

$Y_i$  = Observed values of the dependent variable

$\hat{Y}_i$  = Predicted value of Y for the given  $X_i$  value



SST = total sum of squares

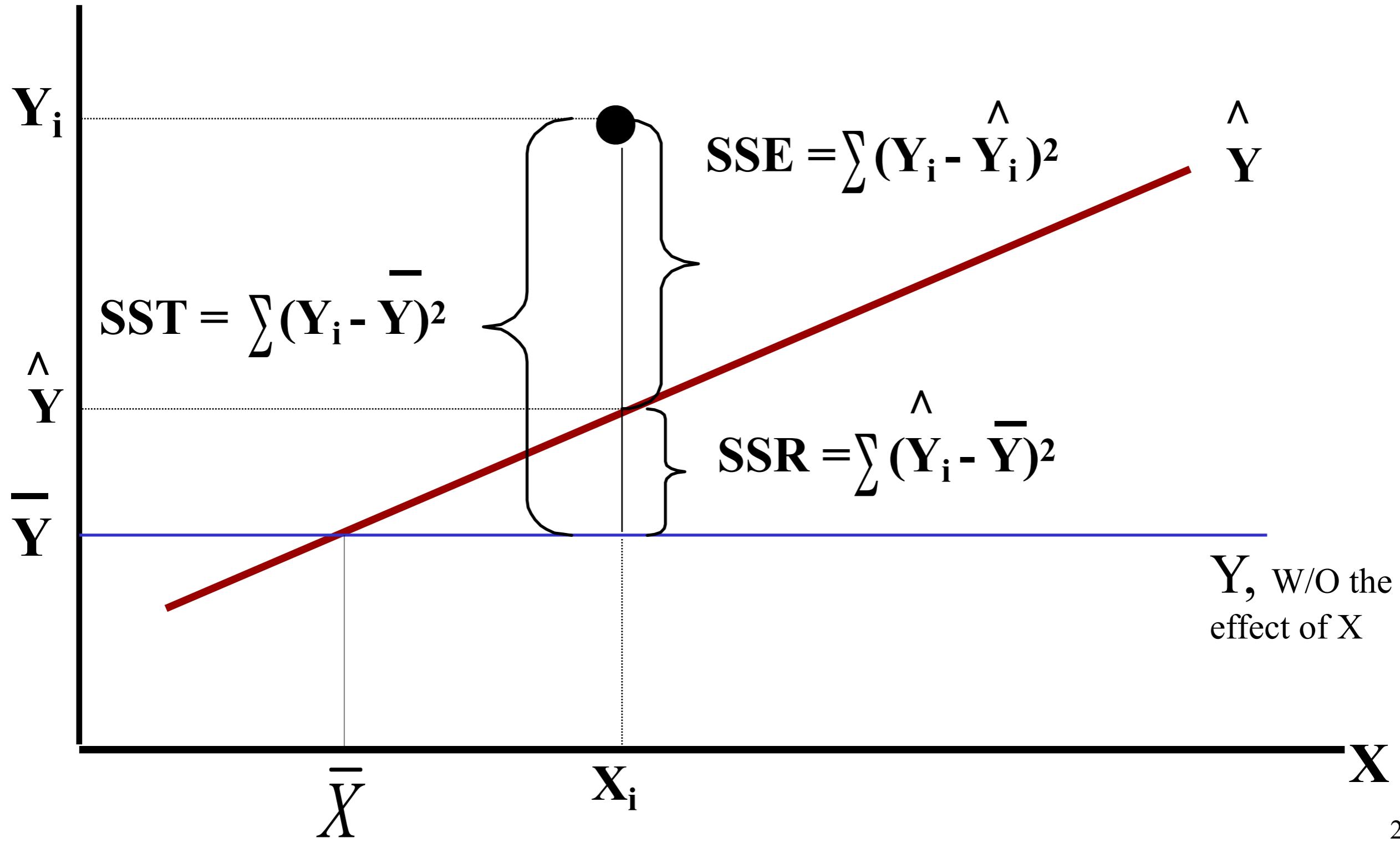
Measures total variation of the  $Y_i$  values around their mean

SSR = regression sum of squares (Explained)

Explained portion of total variation attributed to Y's relationship with X

SSE = error sum of squares (Unexplained)

Variation of Y values attributable to other factors than its relationship with X



# How Good is the Model's prediction Power?

The **coefficient of determination** is the portion of the total variation in the dependent variable, Y, that is explained by variation in the independent variable, X

The coefficient of determination is also called **r-squared** and is denoted as  **$r^2$**

$$r^2 = \frac{SSR}{SST} = \frac{\text{regression sum of squares}}{\text{total sum of squares}}$$

$$0 \leq r^2 \leq 1$$

# Standard Error of Estimate

The standard deviation of the variation of observations  
around the regression line is estimated by

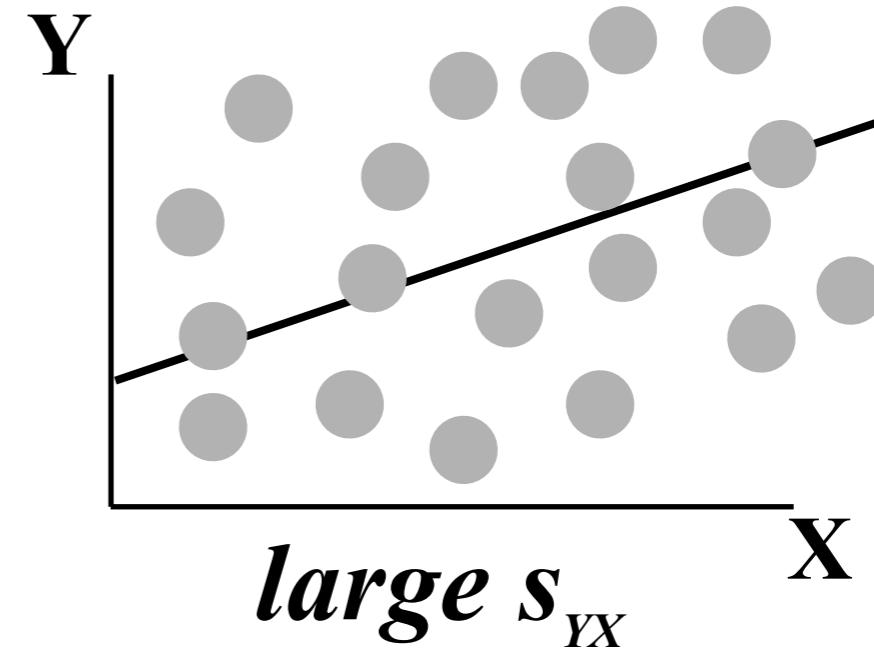
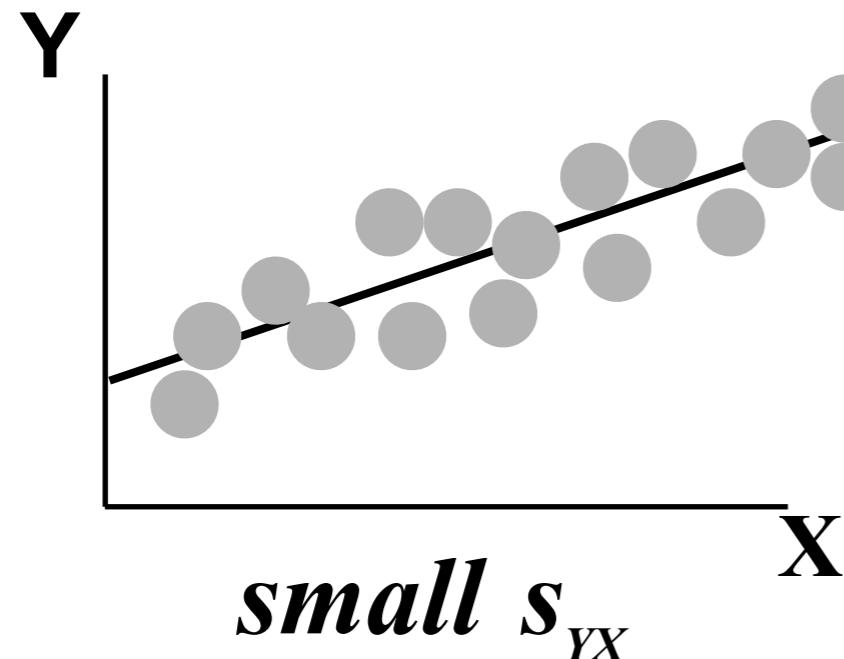
Where SSE = error sum of squares; n = sample size

$$S_{yx} = \sqrt{\frac{SSE}{n - 2}} = \sqrt{\frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{n - 2}} = \sqrt{MSE}$$

The concept is the same as the standard deviation  
(average difference) around the mean of a univariate

# Comparing Standard Errors

$S_{YX}$  is a measure of the variation of observed Y values from the regression line



The magnitude of  $S_{YX}$  should always be judged relative to the size of the Y values in the sample data

i.e.,  $S_{YX} = \$36.34K$  is moderately small relative to house prices in the \$200 - \$300K range (average 215K)

# Assumptions of Regression

## Normality of Error

Error values ( $\varepsilon$ ) are normally distributed for any given value of  $X$

## Homoscedasticity

The probability distribution of the errors has constant variance

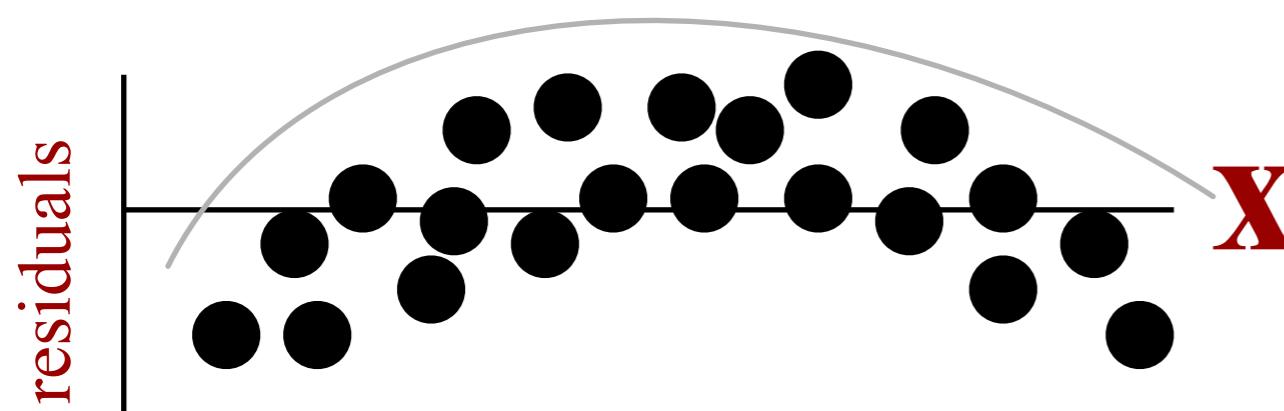
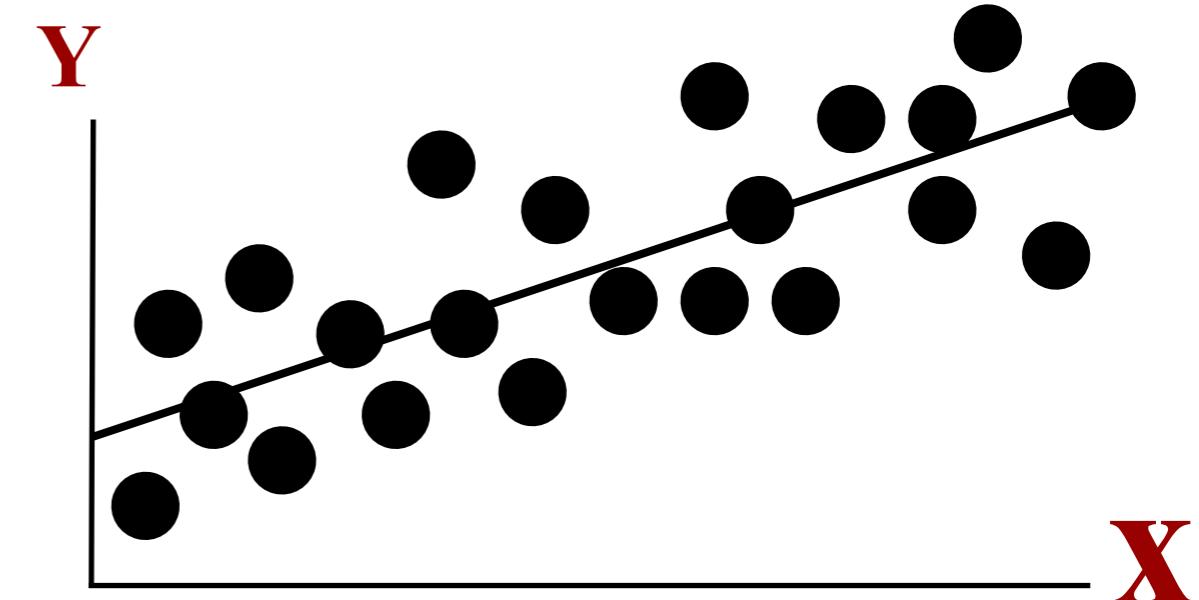
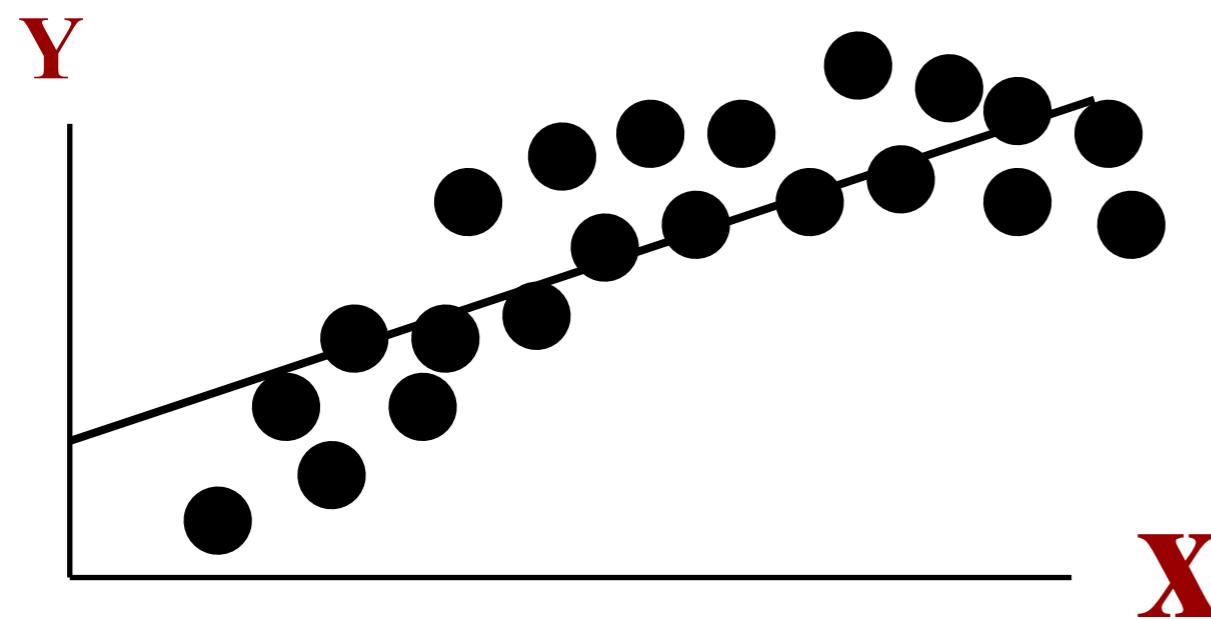
## Independence of Errors

Error values are statistically independent

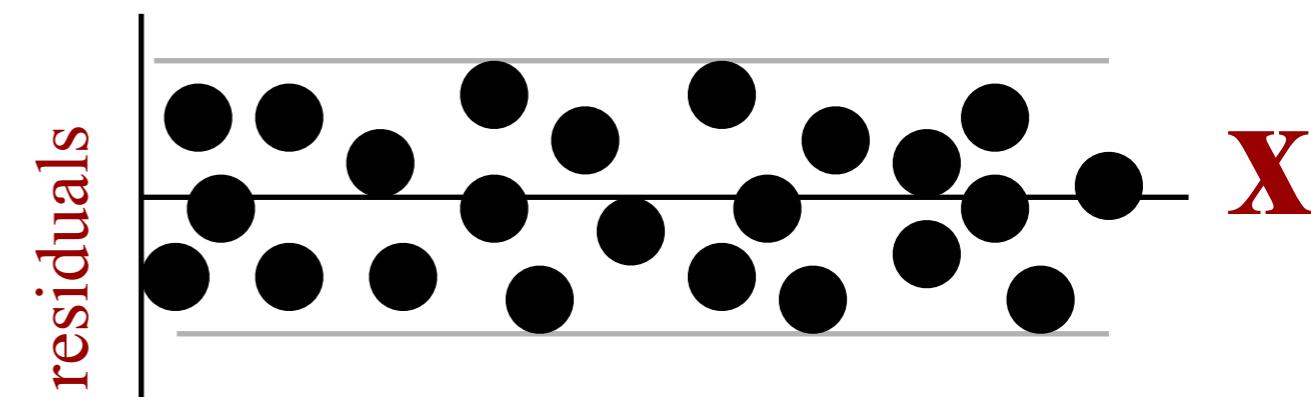
## How to investigate the appropriateness of the fitted model

- The residual for observation  $i$ ,  $e_i$ , is the difference between its observed and predicted value;
- Check the assumptions of regression by examining the residuals
$$e_i = Y_i - \hat{Y}_i$$
  - Examine for linearity assumption
  - Examine for constant variance for all levels of  $X$  (homoscedasticity)
  - Evaluate normal distribution assumption
  - Evaluate independence assumption
- Graphical Analysis of Residuals  
Can plot residuals vs.  $X$

# Residual Analysis for Linearity

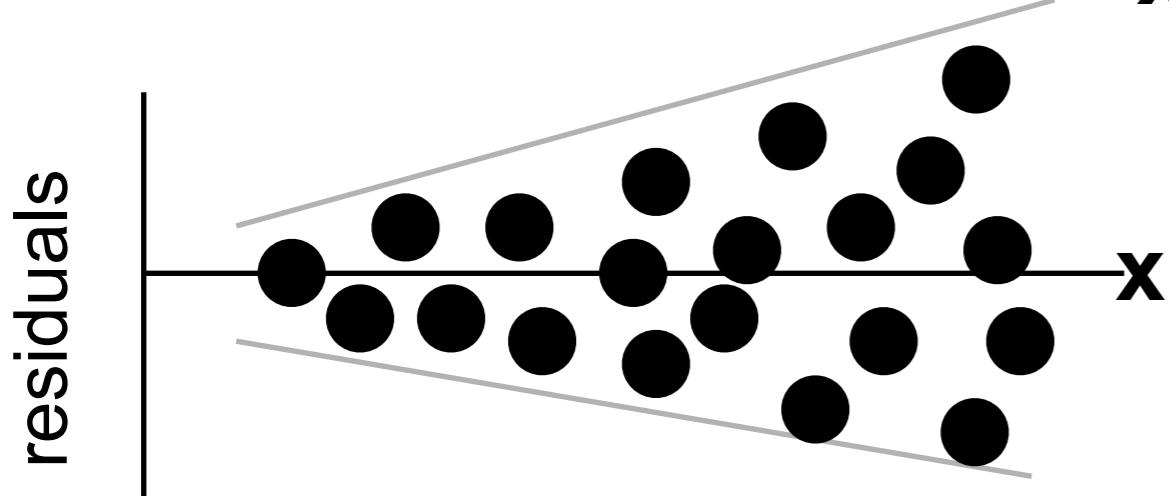
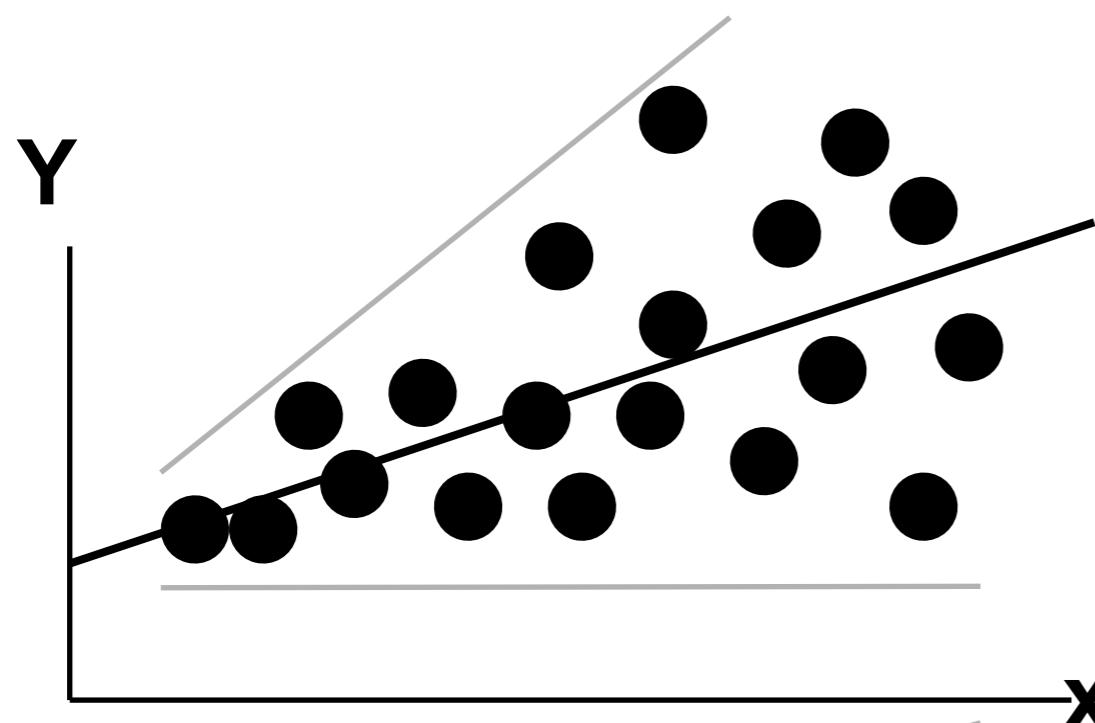


Not Linear

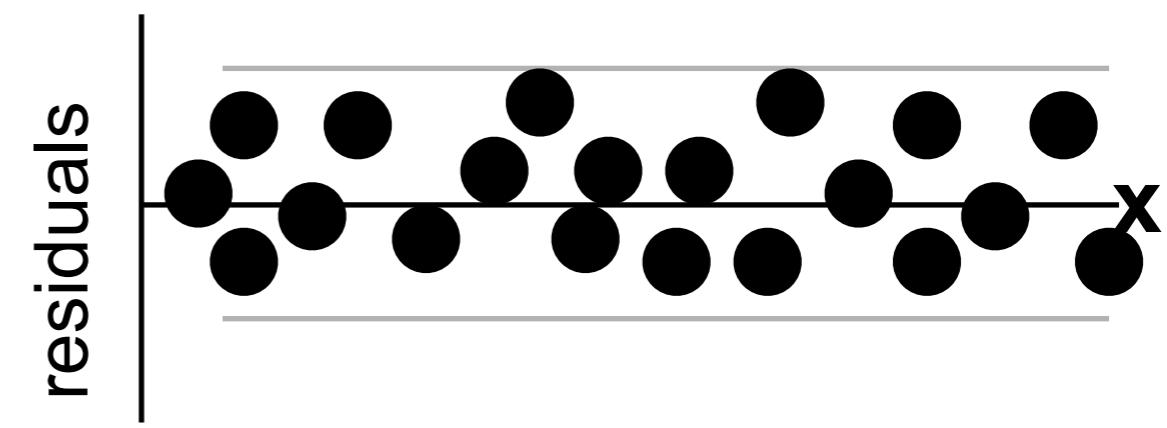
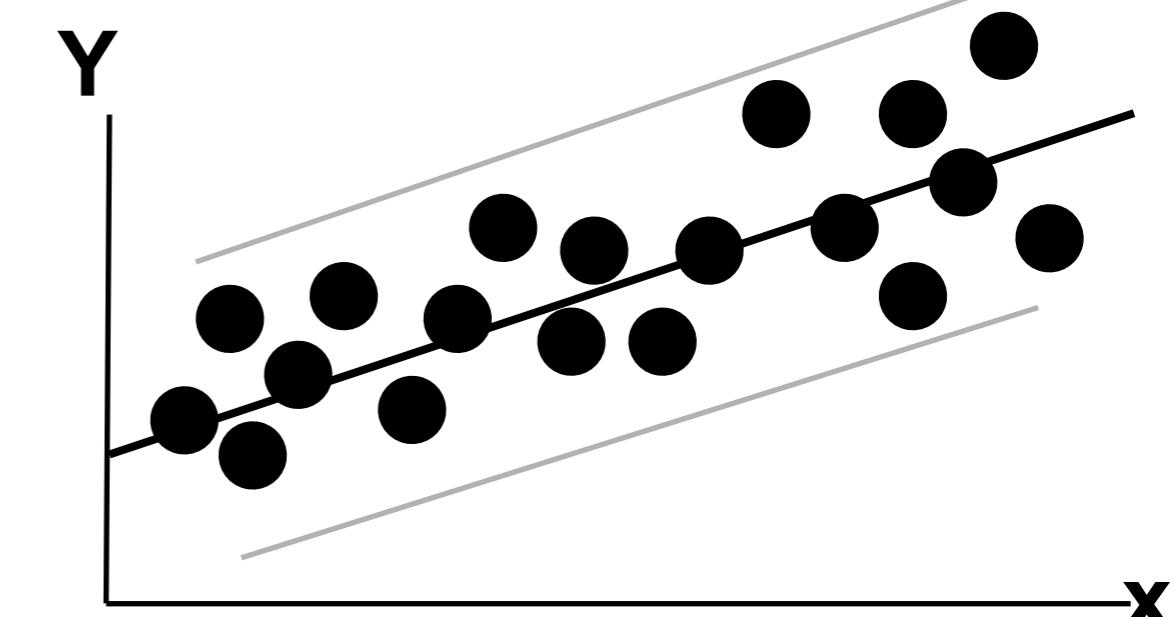


Linear

# Residual Analysis for Homoscedasticity

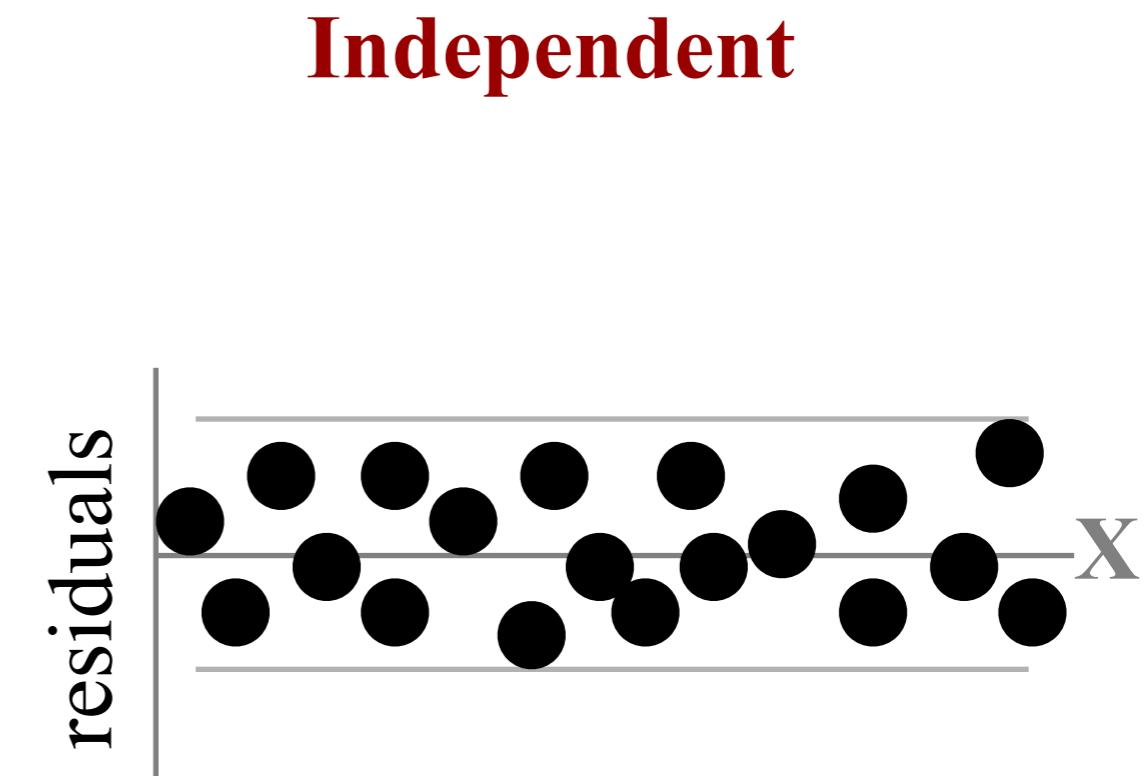
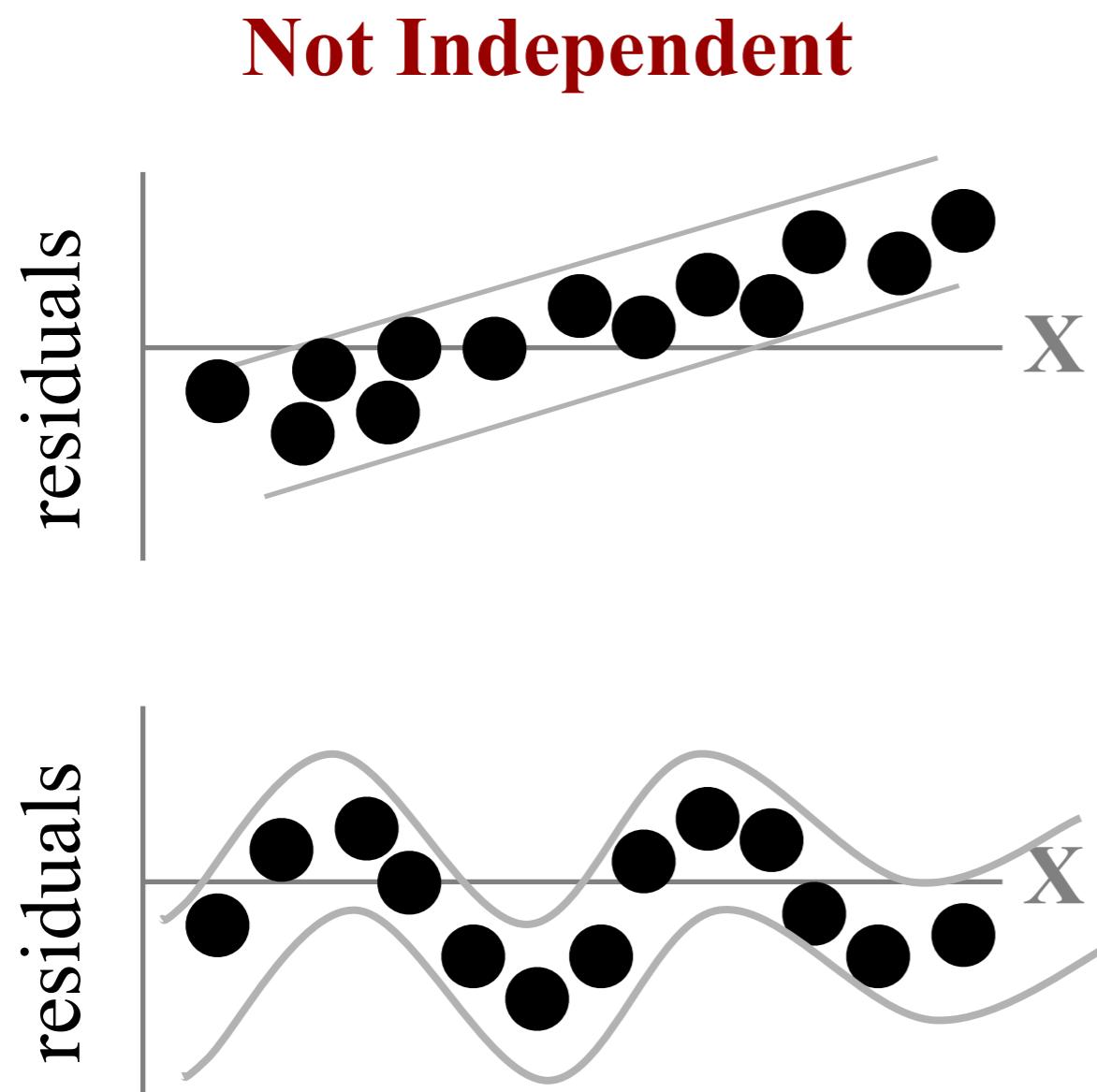


Non-constant variance



Constant variance

# Residual Analysis for Independence



# Regression

# Linear Regression

Let's first import the data set and see what we're working with.

```
# Import data set  
crime <- read.table("crime_simple.txt", sep = "\t", header = TRUE)
```

The variable names that this data set comes with are very confusing, and even misleading.

R: Crime rate: # of offenses reported to police per million population

Age: The number of males of age 14-24 per 1000 population

S: Indicator variable for Southern states (0 = No, 1 = Yes)

Ed: Mean # of years of schooling x 10 for persons of age 25 or older

Ex0: 1960 per capita expenditure on police by state and local government

Ex1: 1959 per capita expenditure on police by state and local government

LF: Labor force participation rate per 1000 civilian urban males age 14-24

M: The number of males per 1000 females

N: State population size in hundred thousands

NW: The number of non-whites per 1000 population

U1: Unemployment rate of urban males per 1000 of age 14-24

U2: Unemployment rate of urban males per 1000 of age 35-39

W: Median value of transferable goods and assets or family income in tens of \$

X: The number of families per 1000 earning below 1/2 the median income



We really need to give these variables better names

```
# Assign more meaningful variable names
colnames(crime) <- c("crime.per.million", "young.males", "is.south", "average.ed",
                      "exp.per.cap.1960", "exp.per.cap.1959", "labour.part",
                      "male.per.fem", "population", "nonwhite",
                      "unemp.youth", "unemp.adult", "median.assets", "num.low.salary")

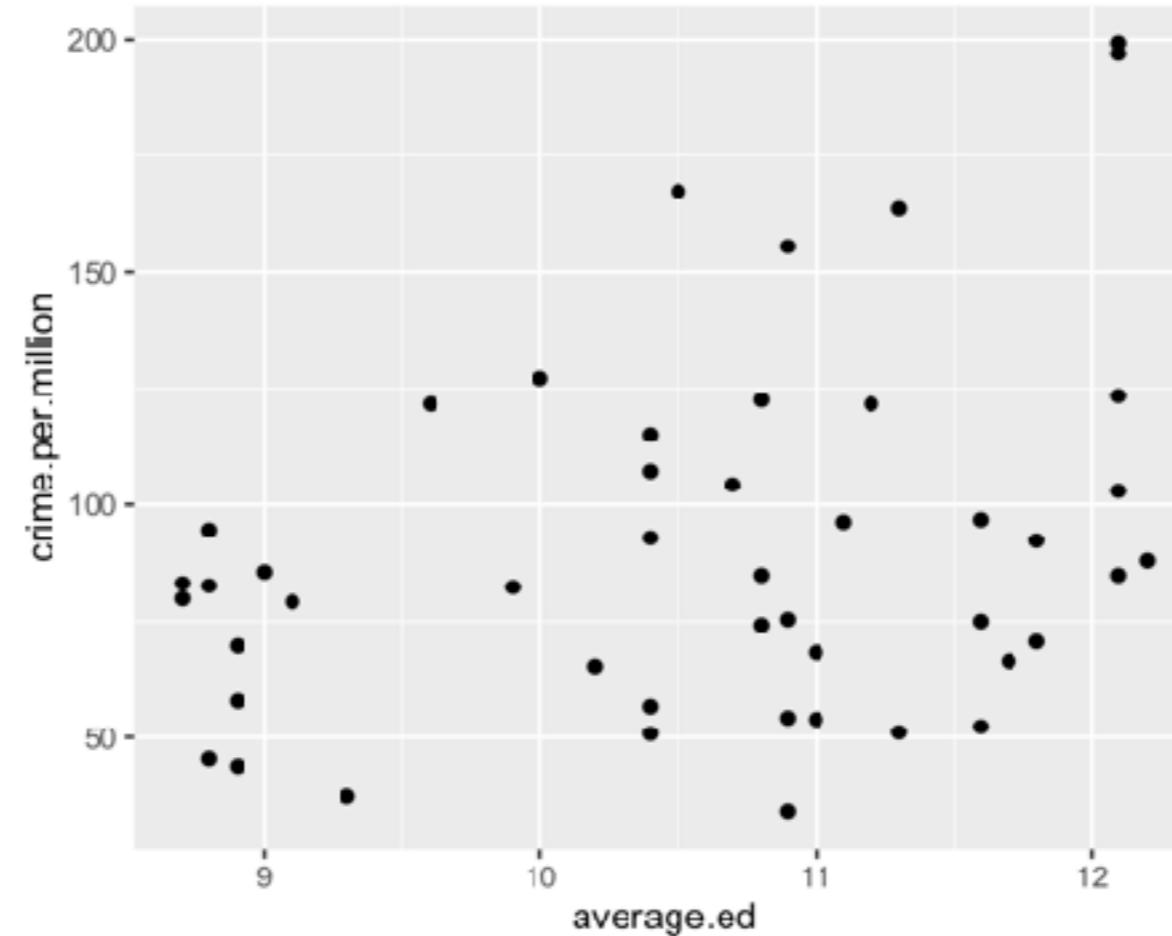
# Convert is.south to a factor
# Divide average.ed by 10 so that the variable is actually average education
# Convert median assets to 1000's of dollars instead of 10's
crime <- transform(crime, is.south = as.factor(is.south),
                     average.ed = average.ed / 10,
                     median.assets = median.assets / 100)

# print summary of the data
summary(crime)
```

```
## crime.per.million young.males is.south average.ed
## Min. : 34.20 Min. :119.0 0:31 Min. : 8.70
## 1st Qu.: 65.85 1st Qu.:130.0 1:16 1st Qu.: 9.75
## Median : 83.10 Median :136.0 Median :10.80
## Mean : 90.51 Mean :138.6 Mean :10.56
## 3rd Qu.:105.75 3rd Qu.:146.0 3rd Qu.:11.45
## Max. :199.30 Max. :177.0 Max. :12.20
## exp.per.cap.1960 exp.per.cap.1959 labour.part male.per.fem
## Min. : 45.0 Min. : 41.00 Min. :480.0 Min. : 934.0
## 1st Qu.: 62.5 1st Qu.: 58.50 1st Qu.:530.5 1st Qu.: 964.5
## Median : 78.0 Median : 73.00 Median :560.0 Median : 977.0
## Mean : 85.0 Mean : 80.23 Mean :561.2 Mean : 983.0
## 3rd Qu.:104.5 3rd Qu.: 97.00 3rd Qu.:593.0 3rd Qu.: 992.0
## Max. :166.0 Max. :157.00 Max. :641.0 Max. :1071.0
## population nonwhite unemp.youth unemp.adult
## Min. : 3.00 Min. : 2.0 Min. : 70.00 Min. : 20.00
## 1st Qu.: 10.00 1st Qu.: 24.0 1st Qu.: 80.50 1st Qu.: 27.50
## Median : 25.00 Median : 76.0 Median : 92.00 Median : 34.00
## Mean : 36.62 Mean :101.1 Mean : 95.47 Mean : 33.98
## 3rd Qu.: 41.50 3rd Qu.:132.5 3rd Qu.:104.00 3rd Qu.: 38.50
## Max. :168.00 Max. :423.0 Max. :142.00 Max. : 58.00
## median.assets num.low.salary
## Min. :2.880 Min. :126.0
## 1st Qu.:4.595 1st Qu.:165.5
## Median :5.370 Median :176.0
## Mean :5.254 Mean :194.0
## 3rd Qu.:5.915 3rd Qu.:227.5
## Max. :6.890 Max. :276.0
```

# First step: some plotting and summary statistics

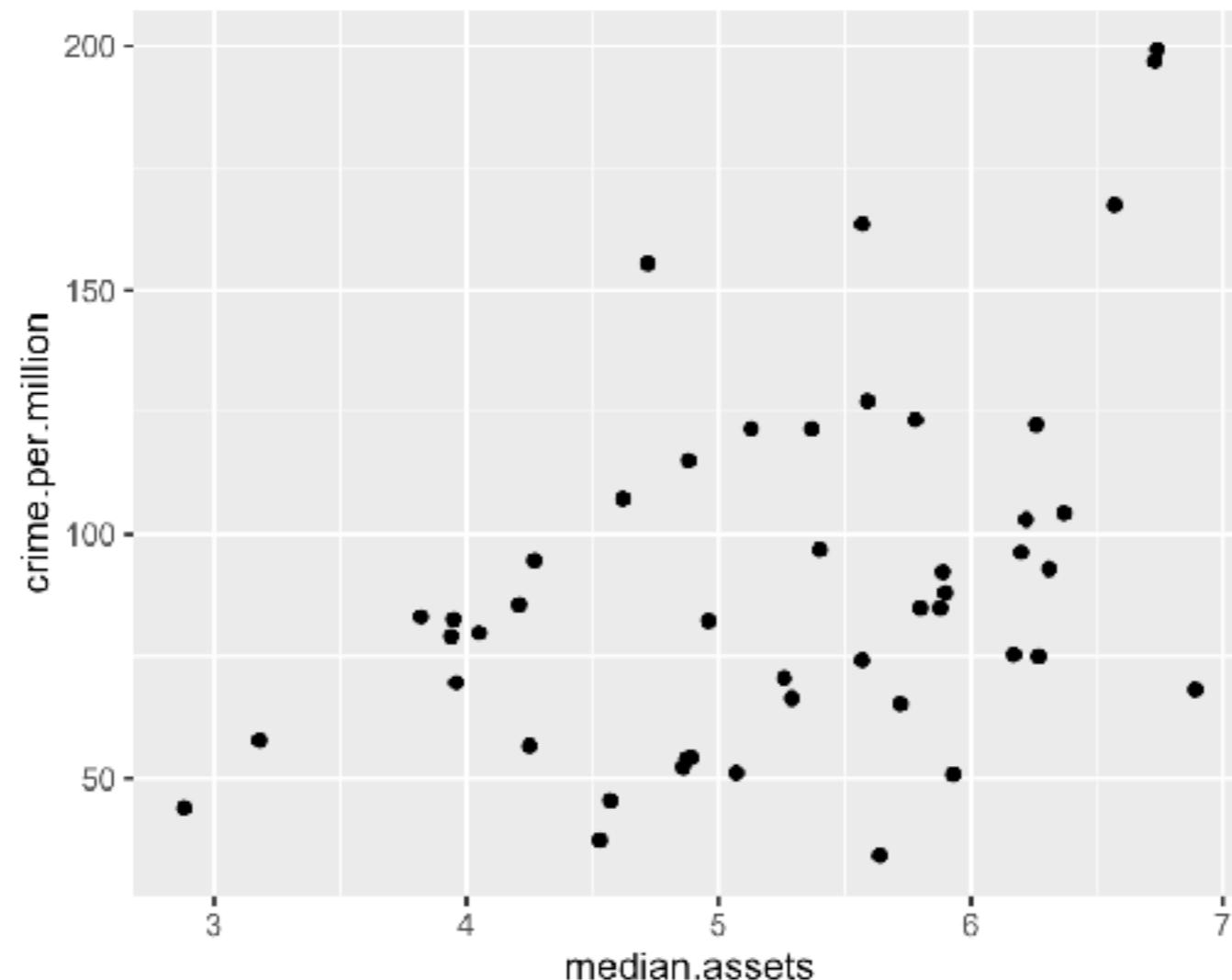
```
# Scatter plot of outcome (crime.per.million) against average.ed  
qplot(average.ed, crime.per.million, data = crime)
```



```
# correlation between education and crime  
with(crime, cor(average.ed, crime.per.million))
```

```
## [1] 0.3228349
```

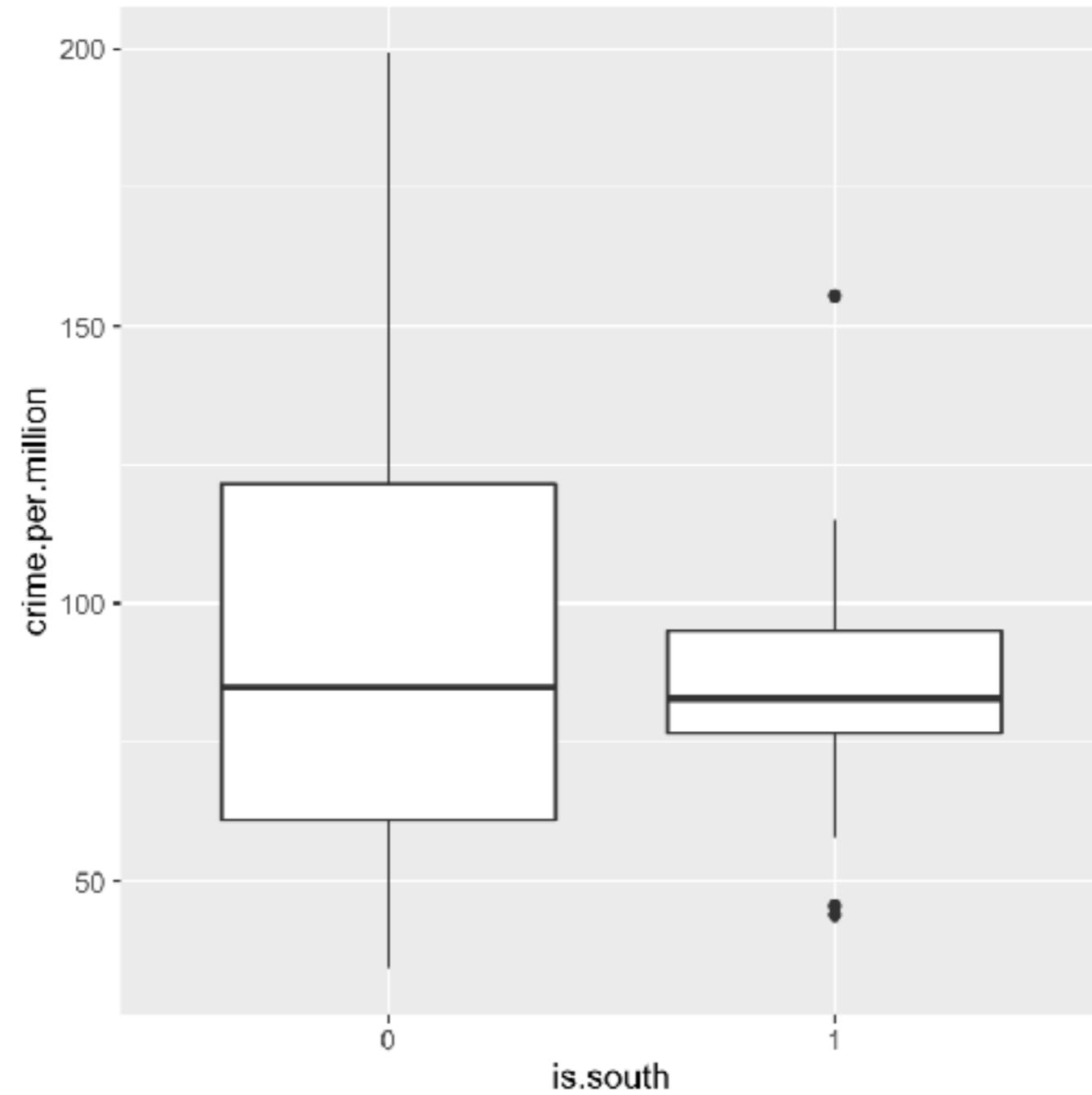
```
# Scatter plot of outcome (crime.per.million) against median.assets  
qplot(median.assets, crime.per.million, data = crime)
```



```
# correlation between education and crime  
with(crime, cor(median.assets, crime.per.million))
```

```
## [1] 0.4413199
```

```
# Boxplots showing crime rate broken down by southern vs non-southern state  
qplot(is.south, crime.per.million, geom = "boxplot", data = crime)
```



# Constructing a regression model

```
crime.lm <- lm(crime.per.million ~ ., data = crime)
# Summary of the linear regression model
crime.lm
```

```
##
## Call:
## lm(formula = crime.per.million ~ ., data = crime)
##
## Coefficients:
## (Intercept)      young.males      is.southl      average.ed
## -6.918e+02       1.040e+00      -8.308e+00      1.802e+01
## exp.per.cap.1960 exp.per.cap.1959      labour.part      male.per.fem
## 1.608e+00        -6.673e-01      -4.103e-02      1.648e-01
## population      nonwhite      unemp.youth      unemp.adult
## -4.128e-02       7.175e-03      -6.017e-01      1.792e+00
## median.assets   num.low.salary
## 1.374e+01        7.929e-01
```

```
summary(crime.lm)

##
## Call:
## lm(formula = crime.per.million ~ ., data = crime)
##
## Residuals:
##    Min      1Q  Median      3Q     Max 
## -34.884 -11.923 -1.135 13.495 50.560 
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -6.918e+02  1.559e+02 -4.438 9.56e-05 ***
## young.males  1.040e+00  4.227e-01  2.460  0.01931 *  
## is.south1    -8.308e+00  1.491e+01 -0.557  0.58117    
## average.ed   1.802e+01  6.497e+00  2.773  0.00906 ** 
## exp.per.cap.1960 1.608e+00  1.059e+00  1.519  0.13836    
## exp.per.cap.1959 -6.673e-01  1.149e+00 -0.581  0.56529    
## labour.part   -4.103e-02  1.535e-01 -0.267  0.79087    
## male.per.fem  1.648e-01  2.099e-01  0.785  0.43806    
## population   -4.128e-02  1.295e-01 -0.319  0.75196    
## nonwhite      7.175e-03  6.387e-02  0.112  0.91124    
## unemp.youth   -6.017e-01  4.372e-01 -1.376  0.17798    
## unemp.adult   1.792e+00  8.561e-01  2.093  0.04407 *  
## median.assets 1.374e+01  1.058e+01  1.298  0.20332    
## num.low.salary 7.929e-01  2.351e-01  3.373  0.00191 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.94 on 33 degrees of freedom
## Multiple R-squared:  0.7692, Adjusted R-squared:  0.6783 
## F-statistic: 8.462 on 13 and 33 DF,  p-value: 3.686e-07
```

```
options(scipen=4) # Set scipen = 0 to get back to default

summary(crime.lm)

## 
## Call:
## lm(formula = crime.per.million ~ ., data = crime)
## 
## Residuals:
##    Min      1Q  Median      3Q     Max 
## -34.884 -11.923 -1.135 13.495 50.560 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -691.837588 155.887918 -4.438 0.0000956 ***
## young.males   1.039810  0.422708  2.460  0.01931 *  
## is.south1     -8.308313  14.911588 -0.557  0.58117    
## average.ed    18.016011  6.496504  2.773  0.00906 ** 
## exp.per.cap.1960 1.607818  1.058667  1.519  0.13836    
## exp.per.cap.1959 -0.667258  1.148773 -0.581  0.56529    
## labour.part   -0.041031  0.153477 -0.267  0.79087    
## male.per.fem   0.164795  0.209932  0.785  0.43806    
## population    -0.041277  0.129516 -0.319  0.75196    
## nonwhite       0.007175  0.063867  0.112  0.91124    
## unemp.youth   -0.601675  0.437154 -1.376  0.17798    
## unemp.adult    1.792263  0.856111  2.093  0.04407 *  
## median.assets 13.735847 10.583028  1.298  0.20332    
## num.low.salary  0.792933  0.235085  3.373  0.00191 ** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 21.94 on 33 degrees of freedom
## Multiple R-squared:  0.7692, Adjusted R-squared:  0.6783 
## F-statistic: 8.462 on 13 and 33 DF,  p-value: 0.0000003686
```

# Exploring the lm object

What kind of output do we get when we run a linear model ( lm ) in R?

```
# List all attributes of the linear model  
attributes(crime.lm)
```

```
## $names  
## [1] "coefficients"   "residuals"          "effects"           "rank"  
## [5] "fitted.values"  "assign"            "qr"                "df.residual"  
## [9] "contrasts"       "xlevels"           "call"              "terms"  
## [13] "model"  
##  
## $class  
## [1] "lm"
```

```
# coefficients  
crime.lm$coef
```

```
##          (Intercept)      young.males      is.southl      average.ed  
## -691.837587905     1.039809653    -8.308312889    18.016010601  
## exp.per.cap.1960  exp.per.cap.1959      labour.part      male.per.fem  
##      1.607818377    -0.667258285    -0.041031047    0.164794968  
## population        nonwhite      unemp.youth      unemp.adult  
##      -0.041276887     0.007174688    -0.601675298    1.792262901  
## median.assets    num.low.salary  
##      13.735847285     0.792932786
```

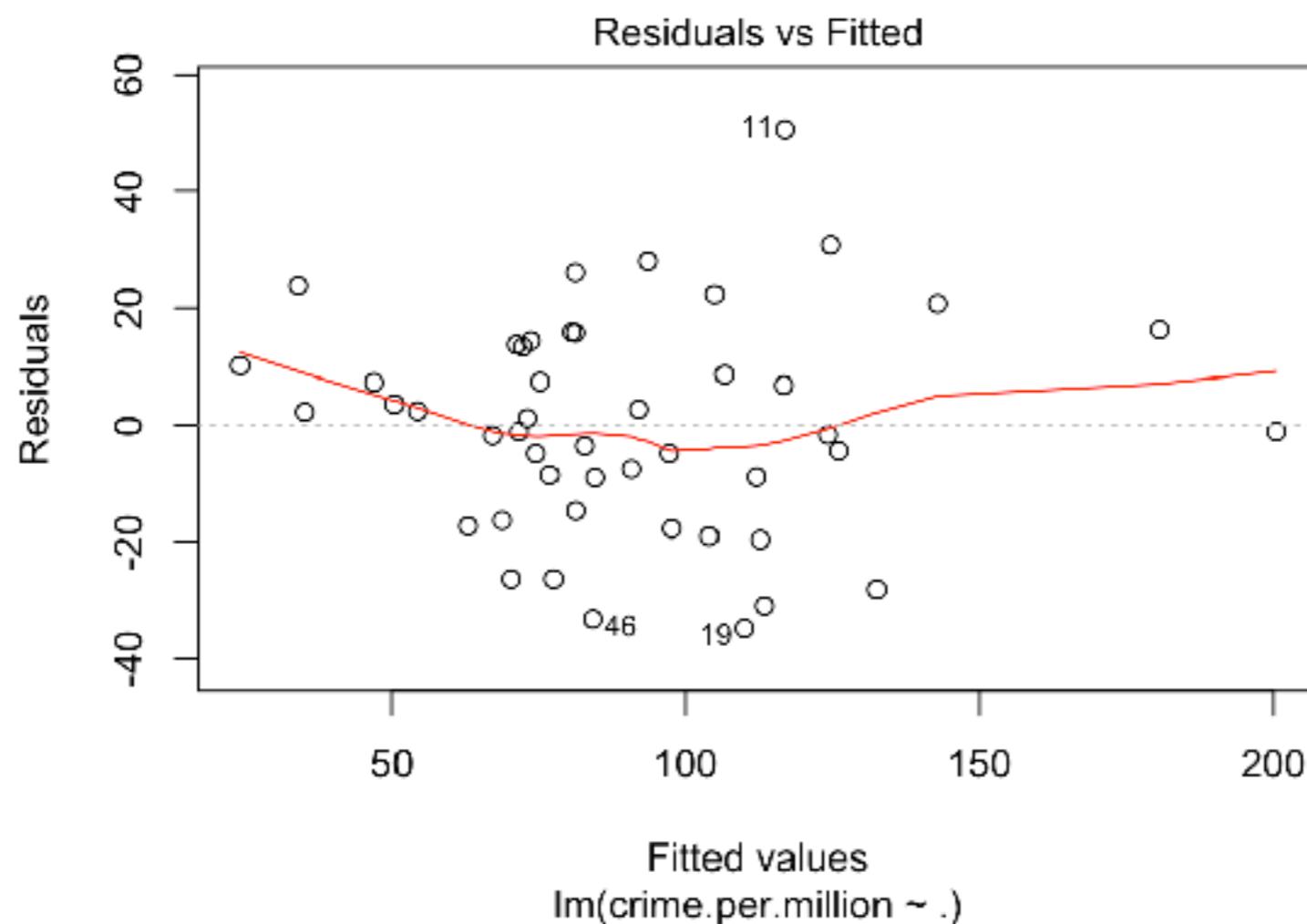
None of the attributes seem to give you p-values. Here's what you can do to get a table that allows you to extract p-values.

```
# Pull coefficients element from summary(lm) object  
round(summary(crime.lm)$coef, 3)
```

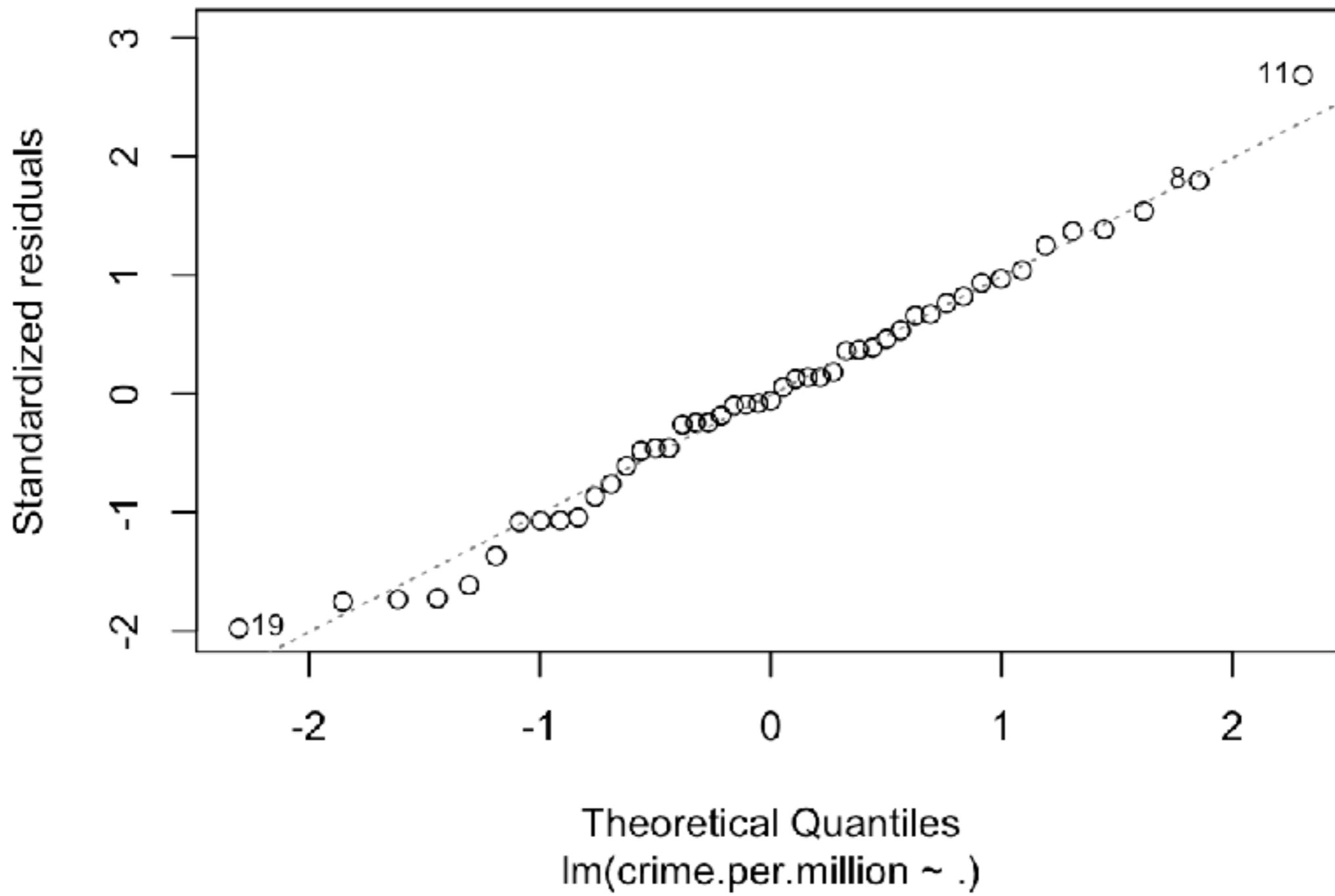
	Estimate	Std. Error	t value	Pr(> t )
## (Intercept)	-691.838	155.888	-4.438	0.000
## young.males	1.040	0.423	2.460	0.019
## is.south1	-8.308	14.912	-0.557	0.581
## average.ed	18.016	6.497	2.773	0.009
## exp.per.cap.1960	1.608	1.059	1.519	0.138
## exp.per.cap.1959	-0.667	1.149	-0.581	0.565
## labour.part	-0.041	0.153	-0.267	0.791
## male.per.fem	0.165	0.210	0.785	0.438
## population	-0.041	0.130	-0.319	0.752
## nonwhite	0.007	0.064	0.112	0.911
## unemp.youth	-0.602	0.437	-1.376	0.178
## unemp.adult	1.792	0.856	2.093	0.044
## median.assets	13.736	10.583	1.298	0.203
## num.low.salary	0.793	0.235	3.373	0.002

# Plotting the lm object

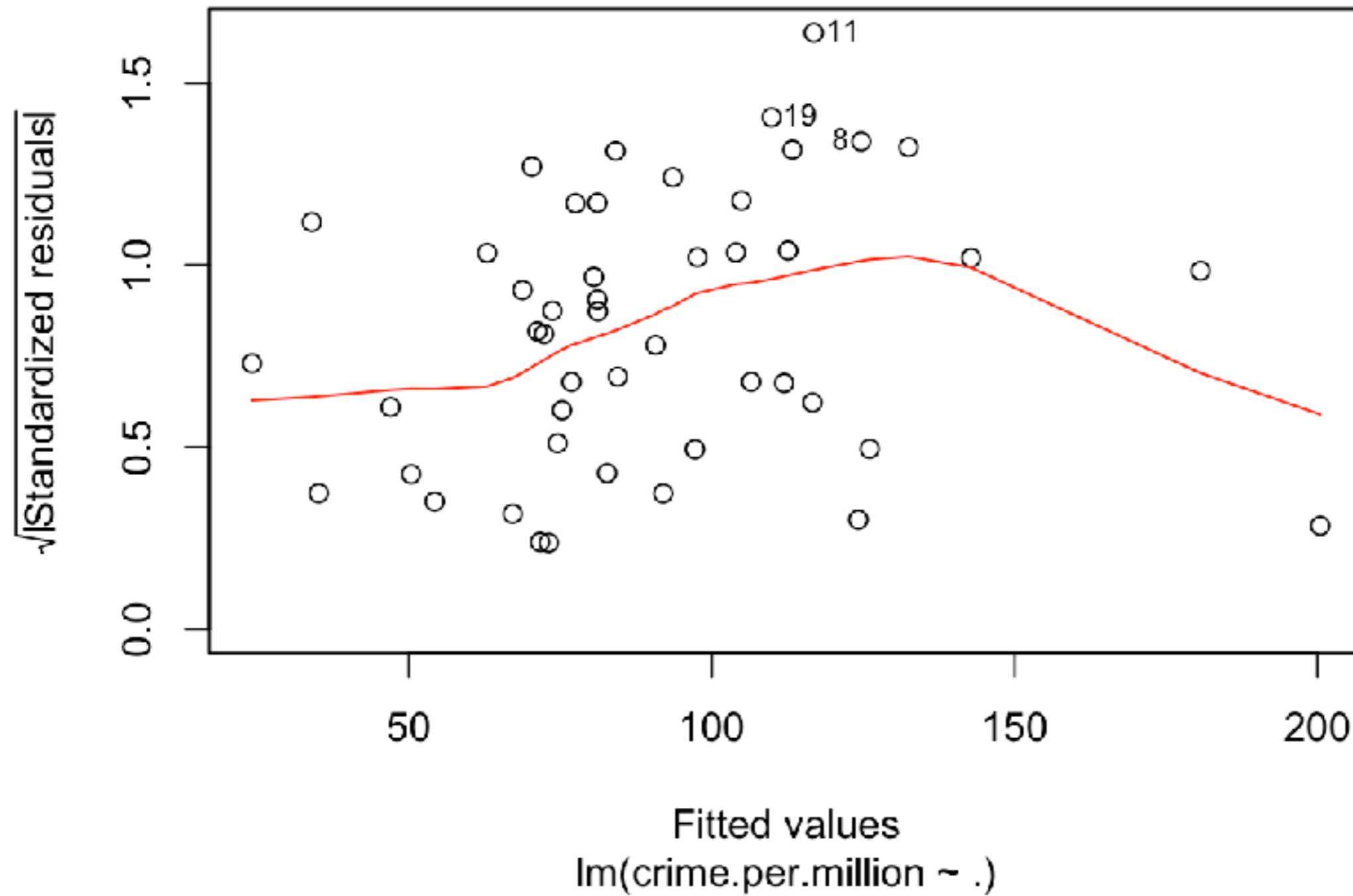
```
plot(crime.lm)
```



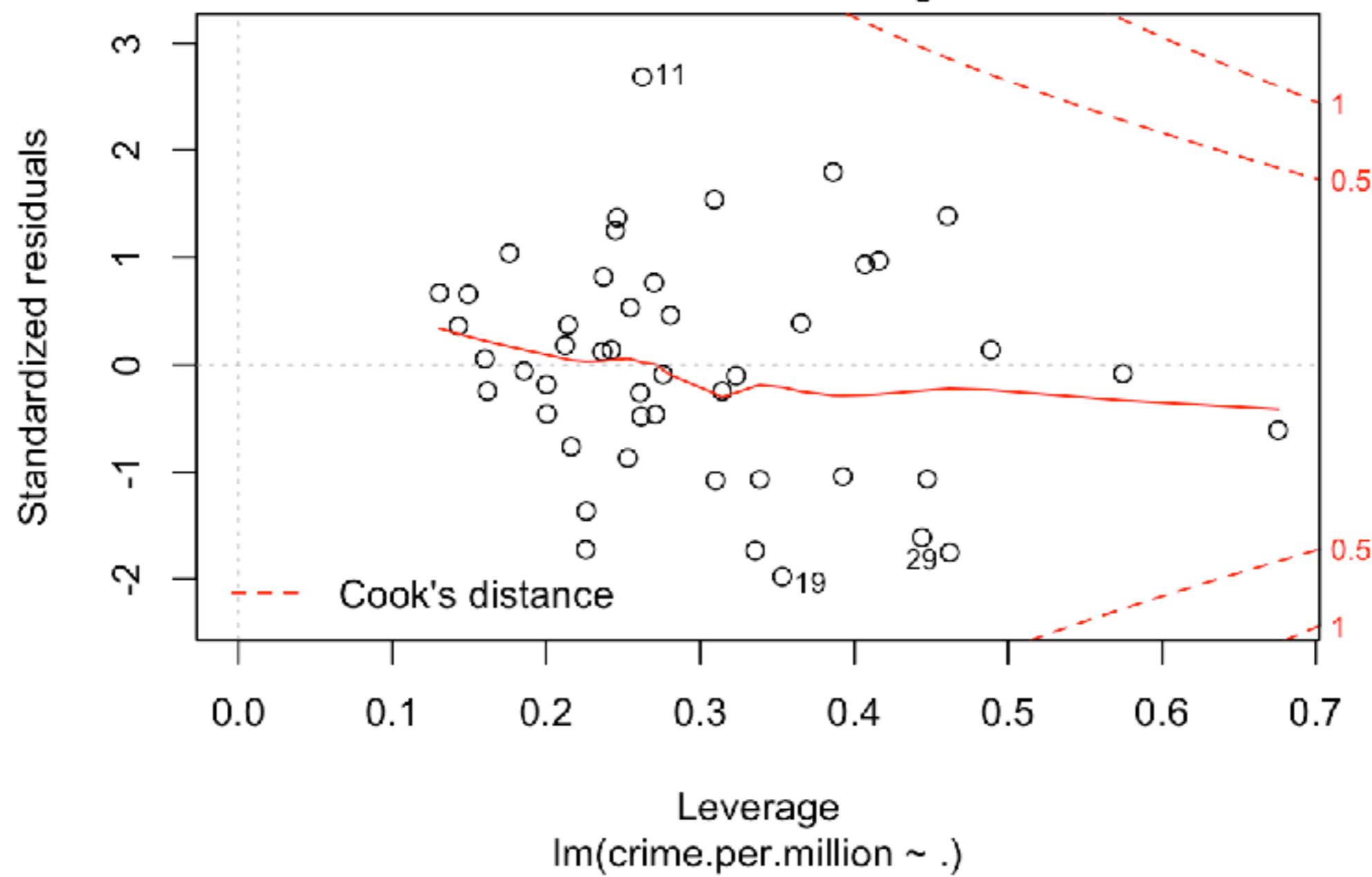
Normal Q-Q



### Scale-Location



Residuals vs Leverage

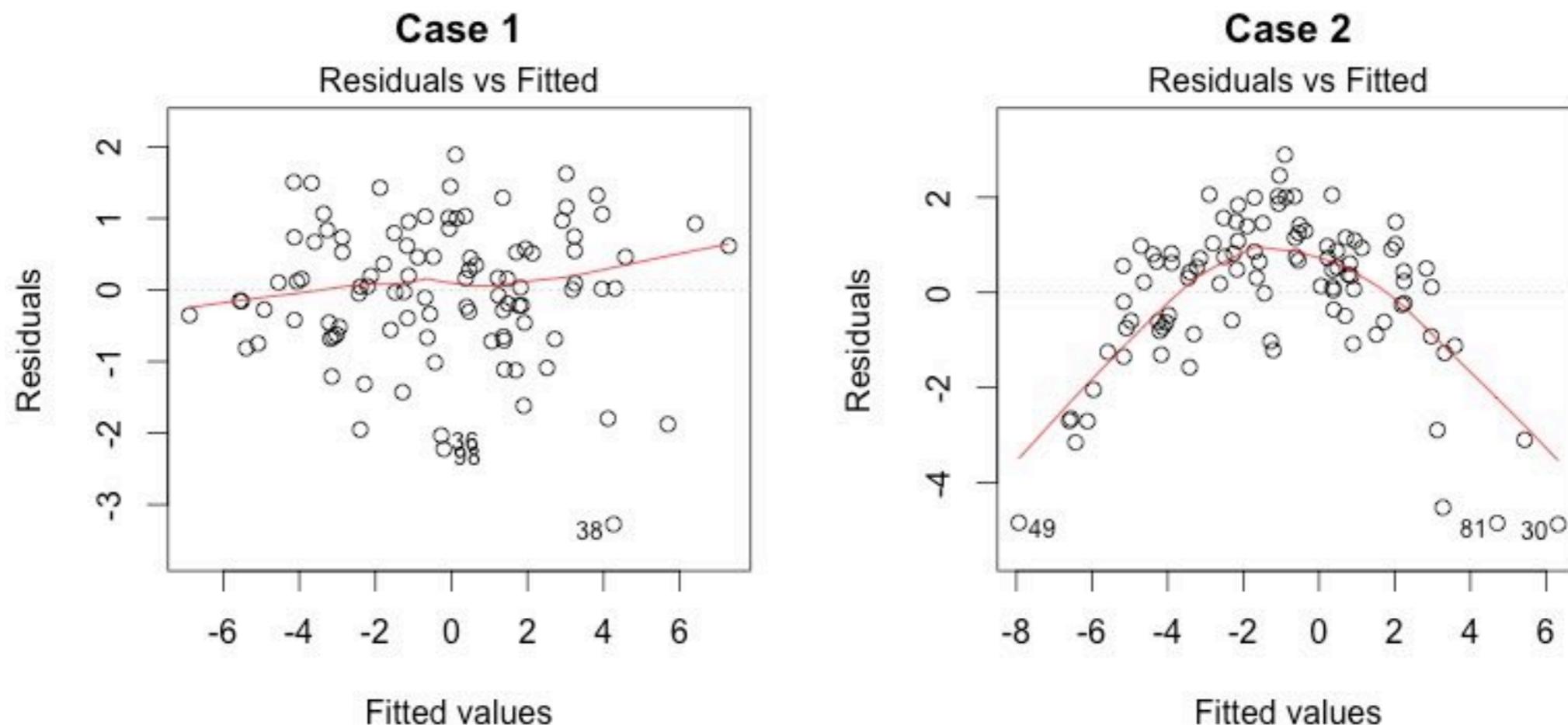


# Residuals vs Fitted Plot

Residuals vs. Fitted When a linear model is appropriate, we expect

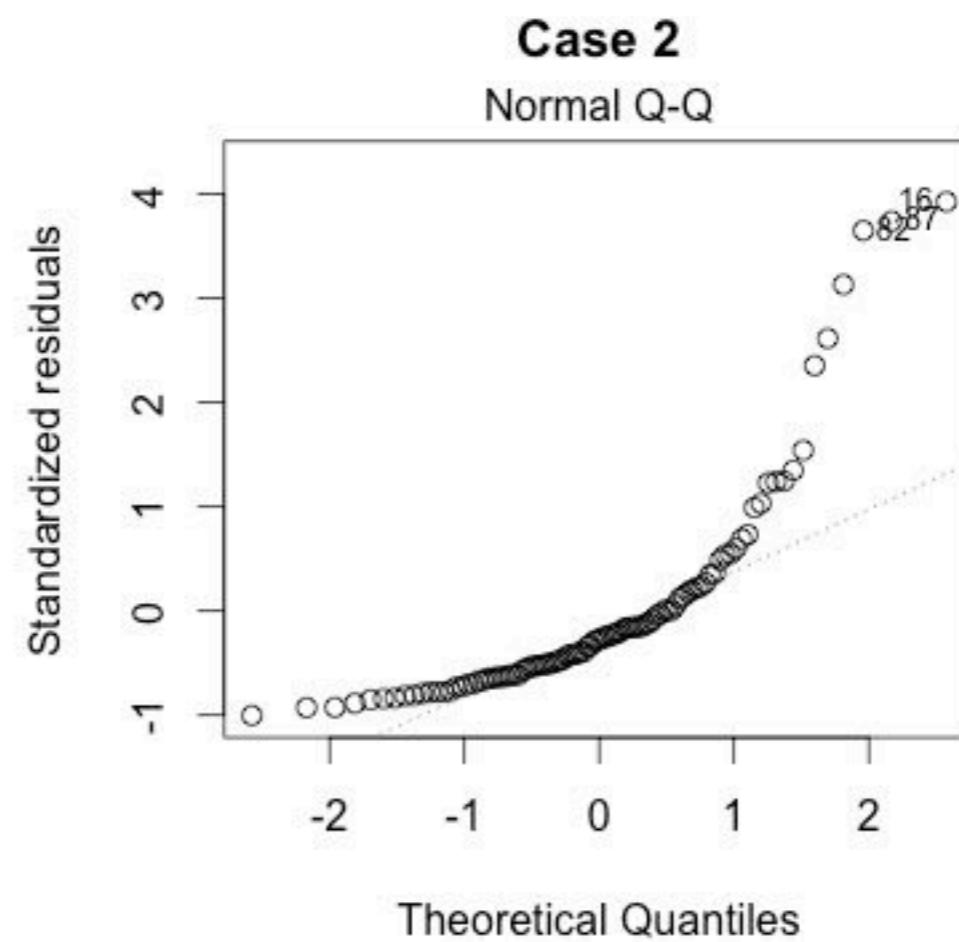
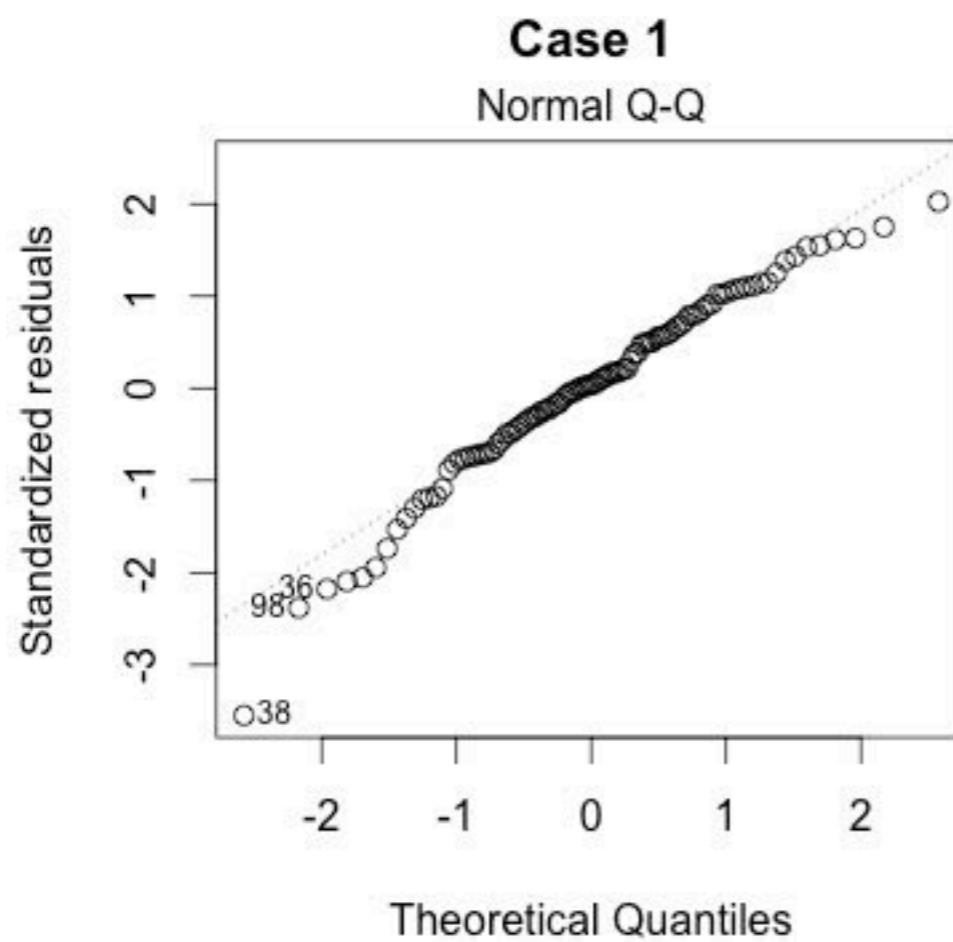
- 1 the residuals will have constant variance when plotted against fitted values; and
- 2 the residuals and fitted values will be uncorrelated.

If there are clear trends in the residual plot, or the plot looks like a funnel, these are clear indicators that the given linear model is inappropriate.



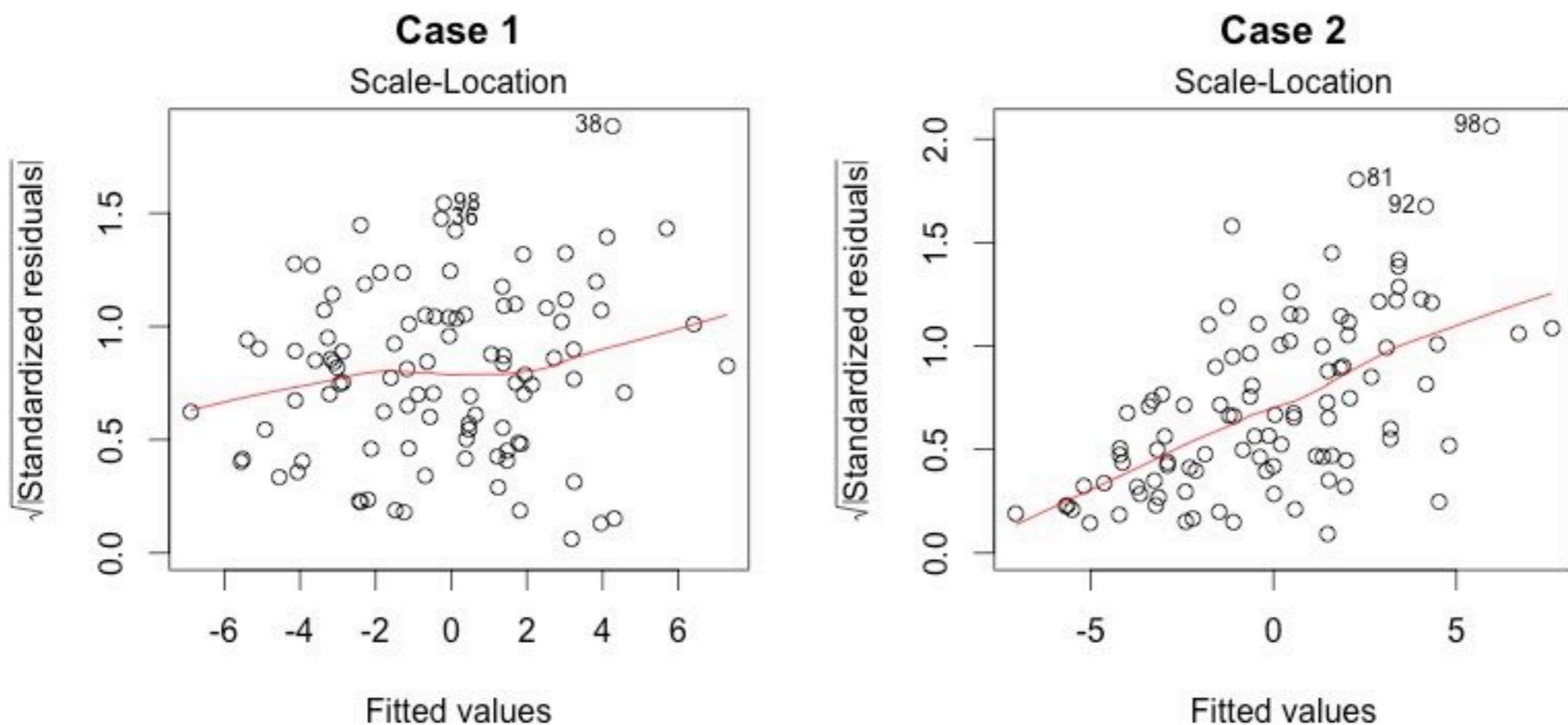
# Normal QQ Plot

Normal QQ plot You can use a linear model for prediction even if the underlying normality assumptions don't hold. However, in order for the p-values to be believable, the residuals from the regression must look approximately normally distributed.



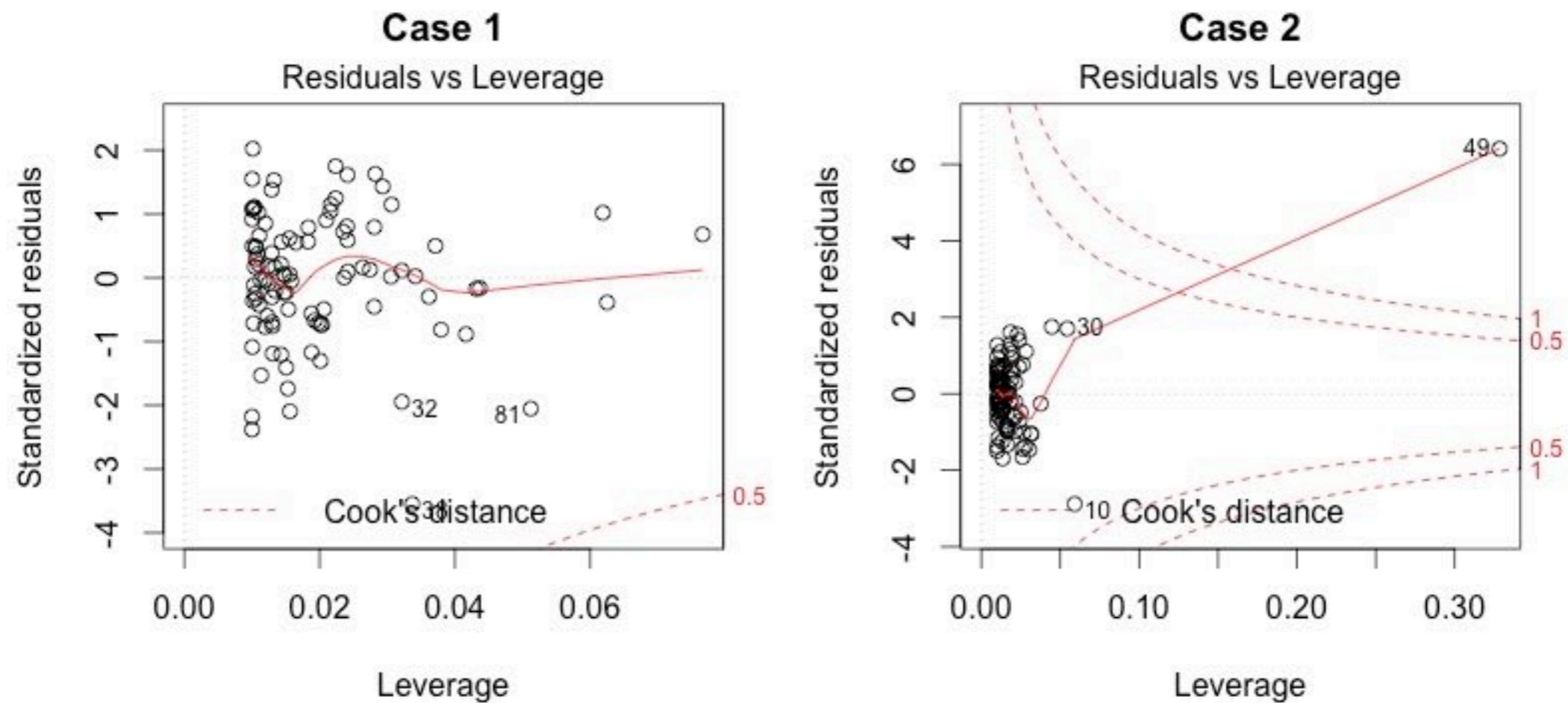
# Scale-Location Plot

Scale-location plot This is another version of the residuals vs fitted plot. There should be no discernible trends in this plot.



# Residuals vs Leverage

Residuals vs Leverage. Leverage is a measure of how much an observation influenced the model fit. It's a one-number summary of how different the model fit would be if the given observation was excluded, compared to the model fit where the observation is included. Points with high residual (poorly described by the model) and high leverage (high influence on model fit) are outliers. They're skewing the model fit away from the rest of the data, and don't really seem to fit with the rest of the data.



# Workshop 10: Regression

We'll begin by loading some packages.

```
library(MASS)  
library(plyr)
```

## Interaction terms in regression

```
# Building up the familiar birthwt data...  
  
# Rename the columns to have more descriptive names  
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",  
  "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",  
  "physician.visits", "birthwt.grams")  
  
# Transform variables to factors with descriptive levels  
birthwt <- transform(birthwt,  
  race = as.factor(mapvalues(race, c(1, 2, 3),  
    c("white", "black", "other"))),  
  mother.smokes = as.factor(mapvalues(mother.smokes,  
    c(0,1), c("no", "yes"))),  
  hypertension = as.factor(mapvalues(hypertension,  
    c(0,1), c("no", "yes"))),  
  uterine.irr = as.factor(mapvalues(uterine.irr,  
    c(0,1), c("no", "yes"))))  
)
```

# Workshop 10 : Regression

- (a) Run a linear regression to better understand how birthweight varies with the mother's age and smoking status (do not include interaction terms).
- (b) What is the coefficient of mother.age in your regression? How do you interpret this coefficient?
- (c) How many coefficients are estimated for the mother's smoking status variable? How do you interpret these coefficients?



[veerasak.kritsanaphan@gmail.com](mailto:veerasak.kritsanaphan@gmail.com)