Introduction to R

Jefferson Davis
Research Analytics

Plan of Attack

- We’re going to assume no knowledge of R.
- I do encourage people to have an active R session running (six slides from now.)
- The ur-text for the talk
  Zuur, A. F, Ieno, E. N, & Meesters, E. H.W.G. (2009). *A beginner's guide to R.* Dordrecht: Springer.

R: background

- First created in the early 1990s by Ross Ihaka and Robert Gentleman as an implementation of S.
- Development soon shifted to a larger core group.
- Distributed under the GNU General Public License.

R: Why should I use it?

- It’s open source and free. (R vs. SAS)
- It has a large body of academic users. Most statistical tests will have some implementation in R.
- It is a full-blown programming language. (R vs. SPSS)
- It's free.

R: some caveats

- Development by non-computer language people can lead to some gotchas.

- Many functions are in packages outside base R.

- Scaling to run intensive programs can require a bit of work. (See Hui Zhang's talk next week.)

- (I resisted the temptation to call this slide "R: some dRawbacks.)

R: Installation and Start

- R is installed on all the computers here in the SSRC
- Also installed in IUAnyware
- and the STCs
- and Big Red II
- and Karst

R: Installation and Start

- Since R is free you should install it on your home computer.

- Download link

  https://www.r-project.org/

  http://ftp.ussg.iu.edu/CRAN/

R: Installation and Start

- Most R users also use an integrated development environment (IDE) for coding in R. Rstudio is the most popular one.

- Rstudio is installed on the machines here in the SSRC.

- A server version of Rstudio is available at
  https://rstudio.iu.edu
  (Requires an account on Karst.)

- If you haven't already please start Rstudio on your local machine.

R: some useful tidbits

- The question mark will display a function's help text. This is a shortcut for the help() function
  `? sin`
- The up arrow key will go back to previous commands
- The hash tag is used for comments
  `#This is an R comment`
- The command `R CMD BATCH --no-save R_input.R` runs in batch
- The command `system()` is used for shell commands
  `system("rm core")`
- The rm() command clears variables
  `rm(list=ls(all=TRUE))  #Clear all variables`

R: Packages

- An R package refers to a collection of previously programmed functions, often including functions for specific tasks.
- R currently has thousands of packages.
- Most packages are easy to install and are often well documented.
- Decent packages include sample data, supporting statistical theory, and a list of examples using each of the functions they contain.
- Good packages will maintain some sort of mailing list or message board for feedback.

R: Installing Packages

- From the command line
  `install.packages("coin")`

  This goes to CRAN, looks for a package called “coin”, and downloads it to your computer.

  This only saves the files to your machine. To use the package in an R session
  `library(coin)`

R: Installing Packages

- Some useful packages are not on CRAN. Installing these can be a hodgepodge

```
source("http://bioconductor.org/biocLite.R")
biocLite

install.packages("devtools")
library(devtools)
install_url("https://github.com/hadley/string
r/archive/master.zip")
```

R: Creating variables

We will spend the next few slides building up the ornithological data set below

Wingcrd	Tarsus	Head	Wt
59	22.3	31.2	9.5
55	19.7	30.4	13.8
53.5	20.8	30.6	14.8
55	20.3	30.3	15.2
52.5	20.8	30.3	15.5
57.5	21.5	30.8	15.6
53	20.6	32.5	15.6
55	21.5	NA	15.7

Morphometric measurements of eight birds. The symbol NA stands for a missing value. The measured variables are lengths of the wing (measured as the wing chord), leg (a standard measure of the tarsus), head (from the bill tip to the back of the skull), and weight.

R: Creating variables

- The left arrow operator assigns values to variables.

```
a <- 59
b <- 55
c <- 53.5
```

- The equal sign can also be used for assignment.

```
a = 59
```

- After doing this in R, a has the value of 59

```
a
[1] 59
a == 59
[1] TRUE
```

R: Creating variables

- Generally, you will have a more joyful life if you variables that have some meaning

```
Wing1 <-  59
Wing2 <-  55
Wing3 <-  53.5
Wing4 <-  55
Wing5 <-  52.5
```

R: Creating variables

- We can, of course, do arithmetic operations on these variable names

```
sqrt(Wing1)
[1] 7.681146
(Wing1 + Wing2 + Wing3 + Wing4 + Wing5)/5
[1] 55
```

- Or store the results in other variables

```
SQ.Wing1 <- sqrt(Wing1)
Mean.Wing <-
   (Wing1 + Wing2 + Wing3 + Wing4 + Wing5)/5
```

R: Creating variables

- It's kind of weird to store the data about Wing lengths in five separate variables.
- It would be more natural to store the date in a 5-tuple or vector of length five.
- We can use the c() funtion to combine numbers (or vectors) into a vector of data

```
Wingcrd <- c(59, 55, 53.5, 55, 52.5, 57.5,
53, 55)
```

R: Creating variables

- We can access values in the vector by indexing with brackets.

```
Wingcrd[1]
 [1] 59
Wingcrd[3]
 [1] 53.5
Wingcrd[1:3]
 [1] 59.0 55.0 53.5
Wingcrd[-2]
 [1] 59.0 53.5 55.0 52.5 57.5 53.0 55.0
sum(Wingcrd)
 [1] 440.5
```

R: Creating variables

- We finish off the other columns

```
Tarsus <- c(22.3, 19.7, 20.8, 20.3, 20.8,
21.5, 20.6, 21.5)

Head <- c(31.2, 30.4, 30.6, 30.3, 30.3, 30.8,
32.5, NA)

Wt <- c(9.5, 13.8, 14.8, 15.2, 15.5, 15.6,
15.6, 15.7)
```

R: Creating variables

- Note the missing value in Head cascades through other calculations

  ```
  sum(Head)
  [1] NA
  ```

- We can work around this if we want to

  ```
  sum(Head, na.rm = TRUE)
  [1] 216.1
  ```

R: Creating variables

- To assemble the data together we can bind our vectors coulmn-wise with cbind()

```
BirdData <- cbind(Wingcrd, Tarsus, Head, Wt)
BirdData
     Wingcrd Tarsus Head   Wt
[1,]    59.0   22.3 31.2  9.5
[2,]    55.0   19.7 30.4 13.8
[3,]    53.5   20.8 30.6 14.8
[4,]    55.0   20.3 30.3 15.2
[5,]    52.5   20.8 30.3 15.5
[6,]    57.5   21.5 30.8 15.6
[7,]    53.0   20.6 32.5 15.6
[8,]    55.0   21.5   NA 15.7
```

R: Creating variables

- We can pick out values by specifying rows and columns

```
BirdData[1, 2]
 Tarsus
  22.3
BirdData[1, 2:4]
 Tarsus   Head     Wt
   22.3   31.2    9.5
BirdData[, 3]
 [1] 31.2 30.4 30.6 30.3 30.3 30.8 32.5   NA
BirdData[1, c(2, 3)]
Tarsus   Head
   22.3   31.2
```

R: Importing Data

- Most data sets will be created outside of R and imported. We create a csv file to import by writing the data to the disk

```
getwd()
[1] "/Users/ranalytics"

write.table(BirdData, "BirdData.csv",
sep=",")
```

R: Importing Data

- Now to import.
```
BirdDF <- read.csv("BirdData.csv",
header=TRUE)
```
- While the data we wrote to the disk was a matrix when R reads the csv file it defaults a dataframe—R's preferred datatype.
- Now the column names give you access to the data.
```
BirdDF$Wingcrd
[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0
```

R: Importing Data

- Acessing entries still works the same

```
BirdDF[1,2]
[1] 22.3
BirdDF[1,2:4]
Tarsus Head  Wt1
   22.3 31.2 9.5
BirdDF[, 3]
[1] 31.2 30.4 30.6 30.3 30.3 30.8 32.5   NA
BirdDF[1, c(2, 3)]
Tarsus Head1
   22.3 31.2
```

R: Importing Data

- However, matrix operations won't work on the dataframe data

```
BirdDF %*% t(BirdDF)
Error in BirdDF %*% t(BirdDF) :
   requires numeric/complex matrix/vector
arguments
```

- Casting from from datatype to the other is easy

```
as.matrix(BirdDF)
as.data.frame(BirdData)
```

R: Plotting Data

- The most basic plotting command in R is plot()

- As a high-level function it will create axes, tick marks, etc
.
- Many user-written classes will have default plot() functions that act reasonably

R: Plotting Data

- We'll explore this with a dataset of car speeds and stopping distances from the 1920s in R as `cars` (Ezekiel, M. (1930) *Methods of Correlation Analysis*. Wiley.)

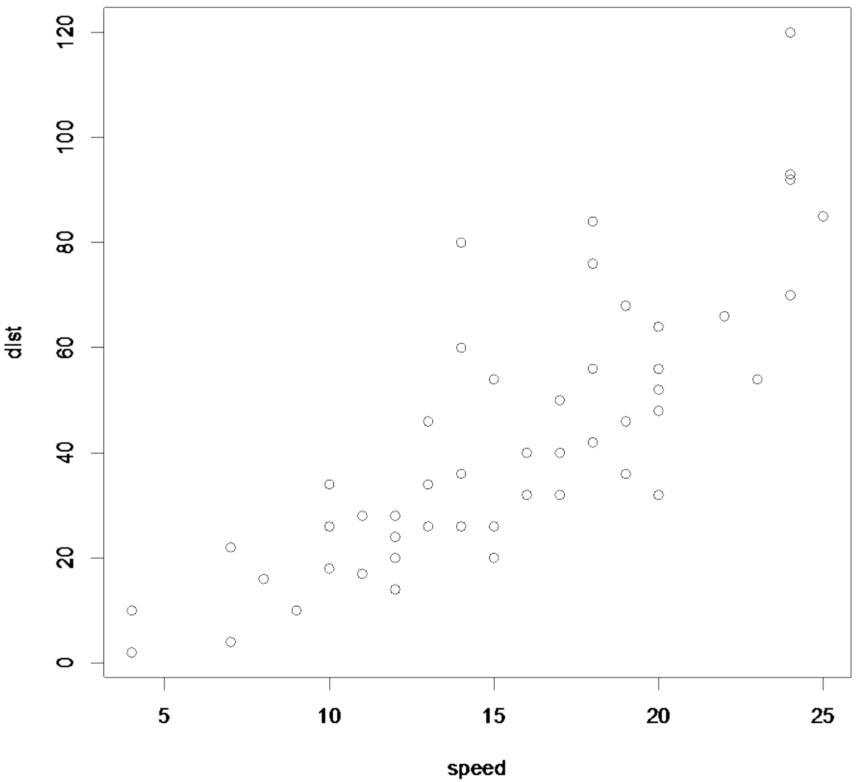
```
head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

R: Plotting Data

- By default plot() produces a scatterplot
  `plot(cars)`

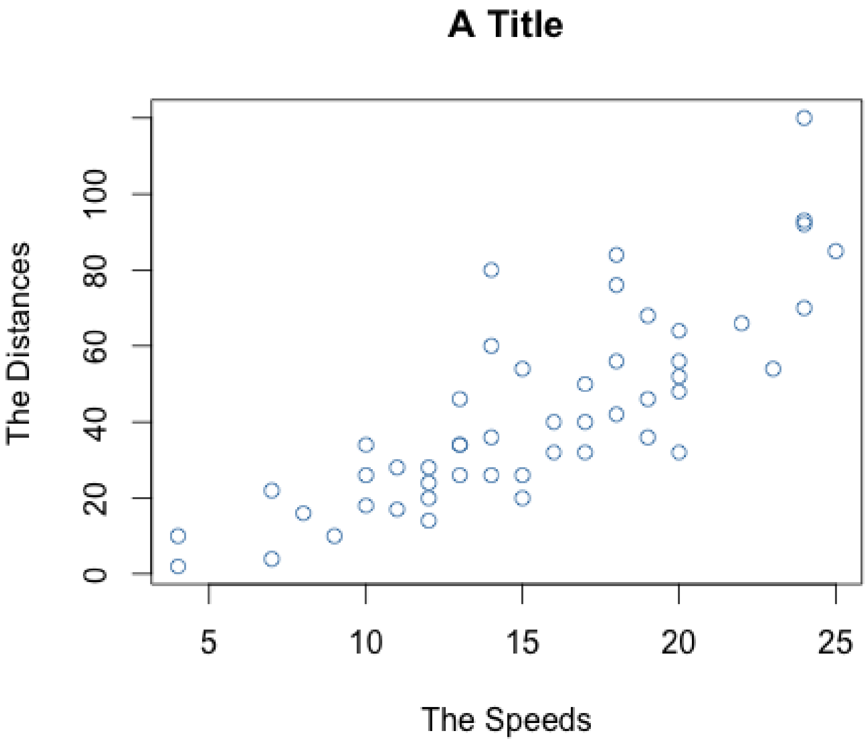
- Axis labels are from the names in the data frame

- Axis scale is from the range of the data



R: Plotting Data

```
plot(cars$speed,
cars$dist,
main = "A Title",
xlab = "The Speeds",
ylab = "The Distances",
col="steel blue")
```

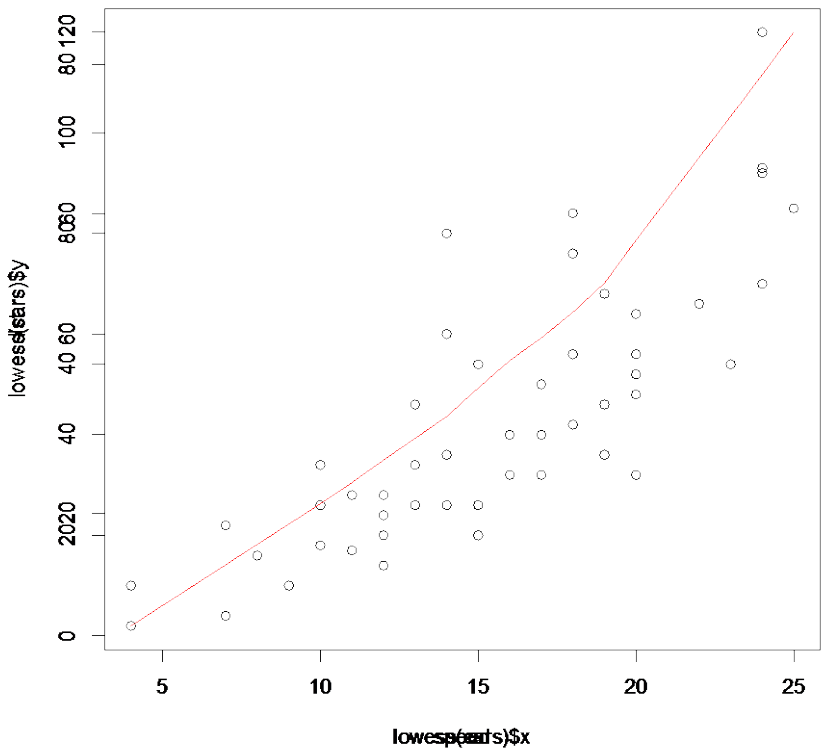


R: Plotting Data

- Each plot() call creates its own axes, labels, etc
- Combining plot() calls can be messy

```
plot(cars)
par(new=TRUE)
plot((lowess(cars),
type="l",
col="red"))
```

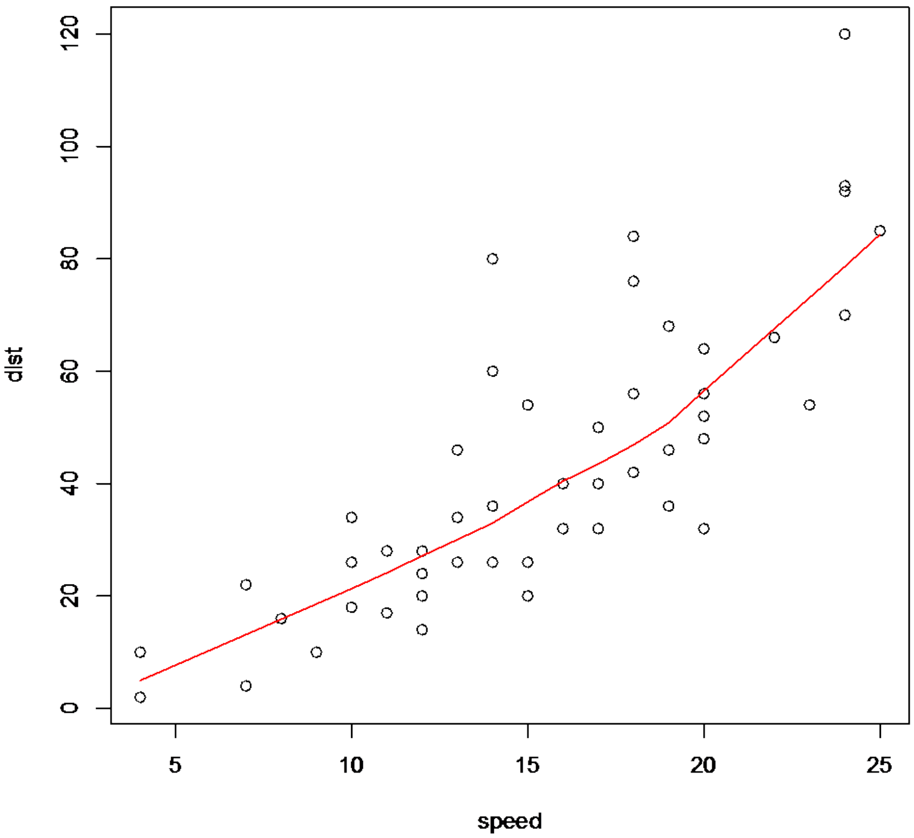
- The graphs labels are clearly bad. While not obvious, the scales don't match either.



R: Plotting Data

- To add details it’s better to use so-called low-level functions.

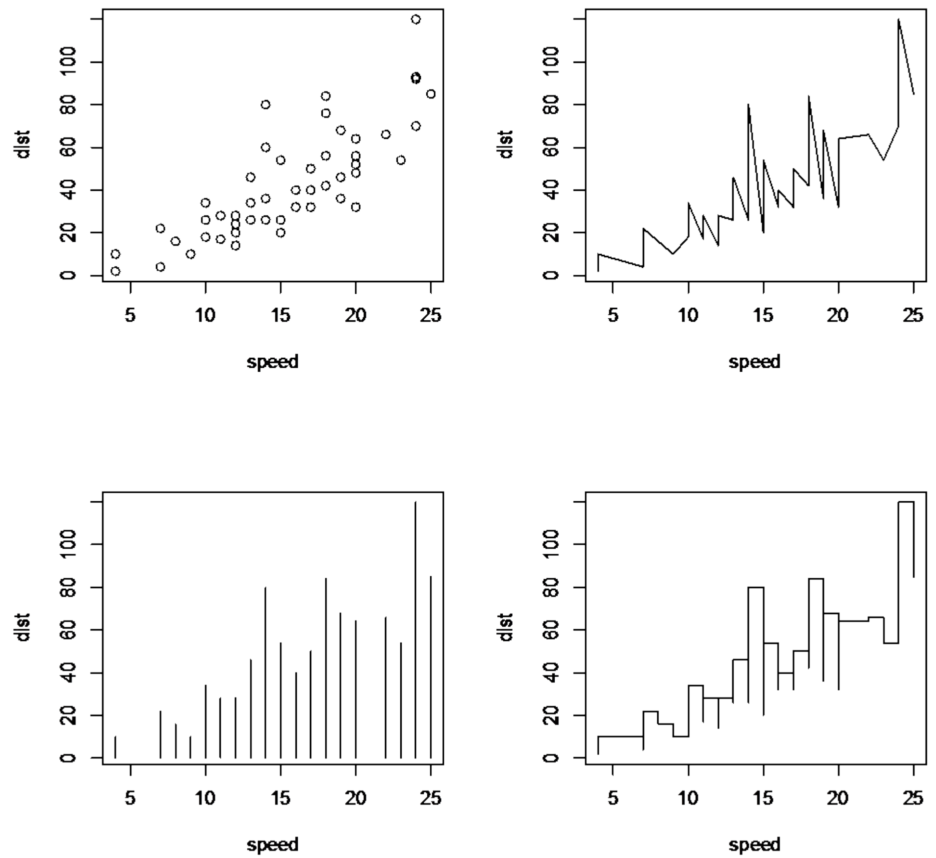
```
plot(cars)
line(lowess(cars),
col="red")
```



R: Plotting Data

- To show multiple plots in one graphics window use the par() command with the mfrow parameter

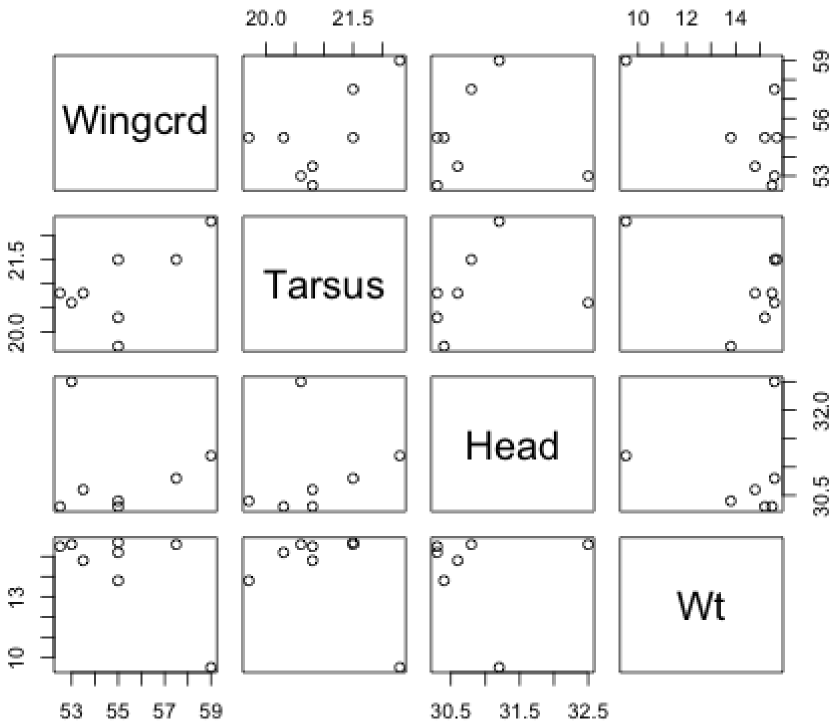
```
par(mfrow=c(2, 2))
plot(cars, type="p")
plot(cars, type="l")
plot(cars, type="h")
plot(cars, type="s")
```



R: Plotting Data

- The default plot type for a dataframe is a scatterplot

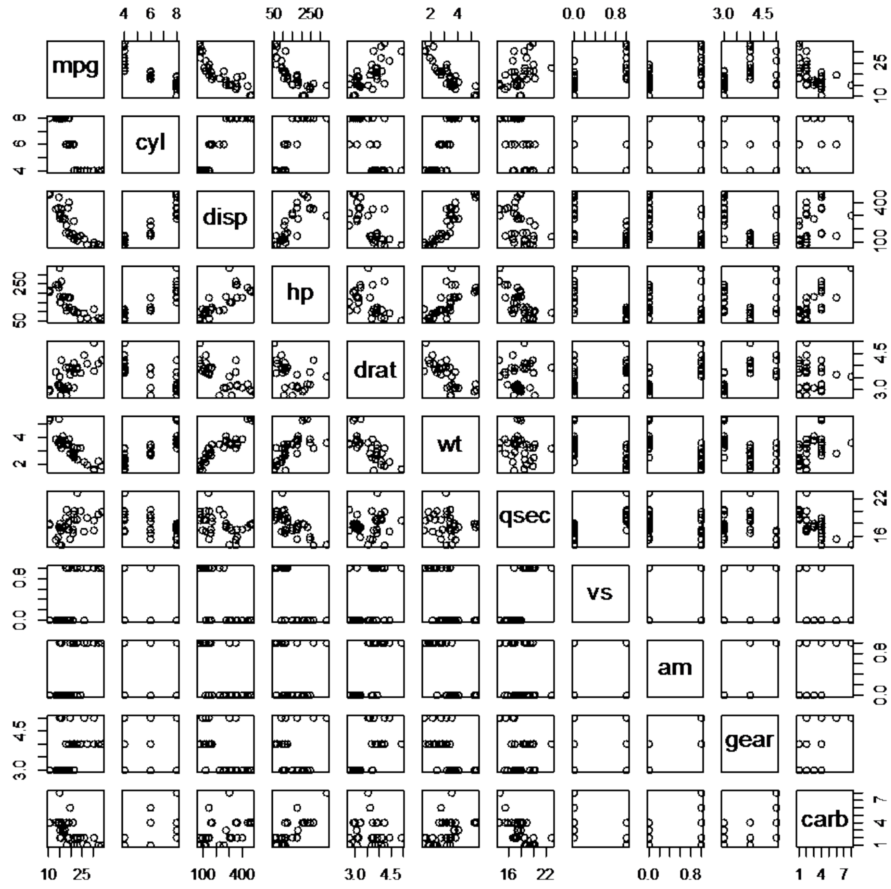
```
plot(BirdDF)
```



R: Plotting Data

- For dataframes with lots of data scatterplots can be a bit much

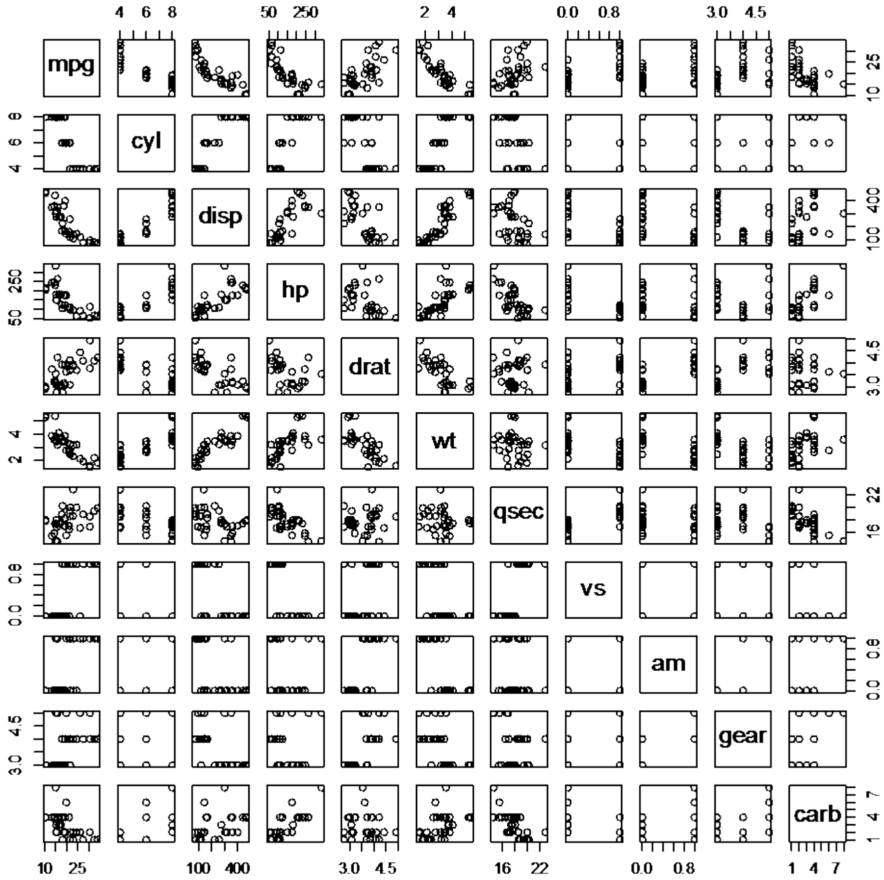
  `plot(mtcars)`



R: Plotting Data

- If you are working in batch, you can have plot write to a file.

```
png(filename =
"mess.png")
plot(mtcars)
dev.off()
```



R: Plotting Data

- For more fine-tuned control of graphics consider the lattice ggplot2 packages.
- They probably deserve a talk on their own. (Or talk to me later.)

R: Analyzing data

- Common statistical tests are very straightforward in R. How about a t-test that the mean of the speeds in `cars` is not 12?
  `t.test(cars$speed, mu=12)`

```
    One Sample t-test
data:  cars$speed
t = 4.5468, df = 49, p-value = 3.588e-05
alternative hypothesis: true mean is not
equal to 12
95 percent confidence interval:
 13.89727 16.90273
sample estimates:
mean of x
     15.4
```

R: Analyzing data

- We can change the parameters of t-test.

```
t.test(cars$speed, mu=12, alternative="less",
conf.level=.99)

	One Sample t-test
data:  cars$speed
t = 4.5468, df = 49, p-value = 1
alternative hypothesis: true mean is less
than 12
99 percent confidence interval:
     -Inf 17.19834
sample estimates:
mean of x
      15.4
```

R: Analyzing data

- Regression analysis is one of the most popular and important tools in statistics. If R goofed here, it would be worthless.
- R uses the function lm() for linear models.
- Generic syntax
  ```
  lm(DV ~ IV1, NAME_OF_DATAFRAME)
  ```
- The above tells R that to regress the dependent variable (DV) onto independent variable IV1. We can include other variables and interaction effects.
  ```
  lm(DV ~ IV1 + IV2 + IV1*IV2,
  NAME_OF_DATAFRAME)
  ```

R: Analyzing data

- Let's do an example using the 1920s cars data set. How about regressing stopping distance on speed.

```
lm(dist ~ speed, cars)

Call:lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)        speed
    -17.579        3.932
```

- To work more let's store this in a variable

```
car.fit <- lm(dist ~ speed, cars)
```

R: Analyzing data

```
summary(car.fit)
Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
    Min       1Q  Median       3Q      Max
 -29.069  -9.525  -2.272   9.215  43.201
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
speed         3.9324     0.4155   9.464 1.49e-12 ***
```

R: Analyzing data

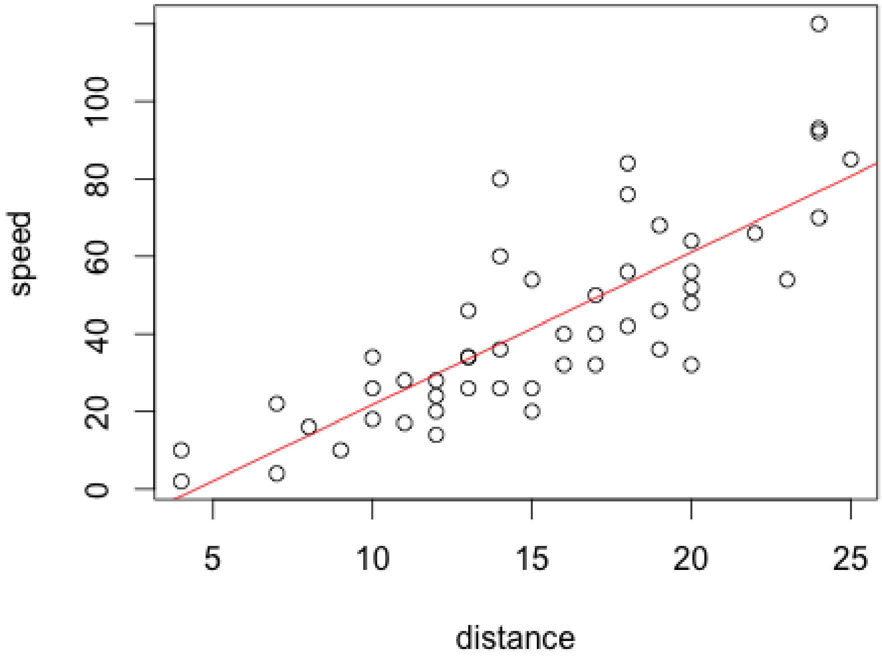
- We can also look at individual fields of the lm object.

```
car.fit$coefficients
 (Intercept)       speed
  -17.579095    3.932409
car.fit$residuals[1:3]
        1         2         3
 3.849460 11.849460 -5.947766
car.fit$fitted.values[1:3]
         1         2         3
 -1.849460 -1.849460  9.947766
```

R: Analyzing data

- Plot the fit

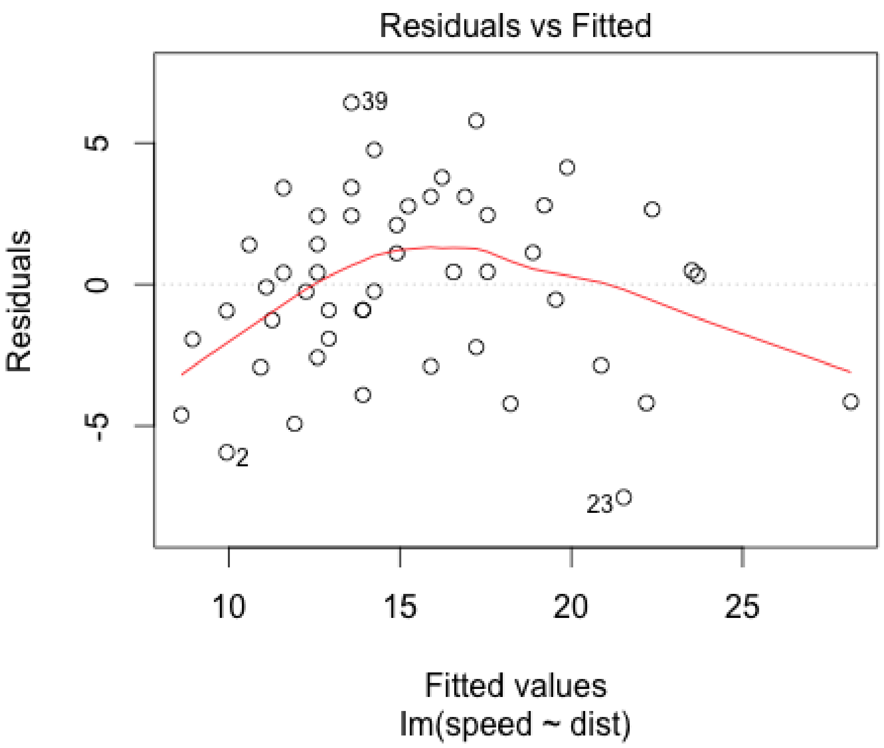
```
plot(cars$speed,
cars$dist,
xlab="distance",
ylab="speed")

abline(car.fit,
col="red")
```



R: Analyzing data

- Class lm object have their own overloaded plot() function

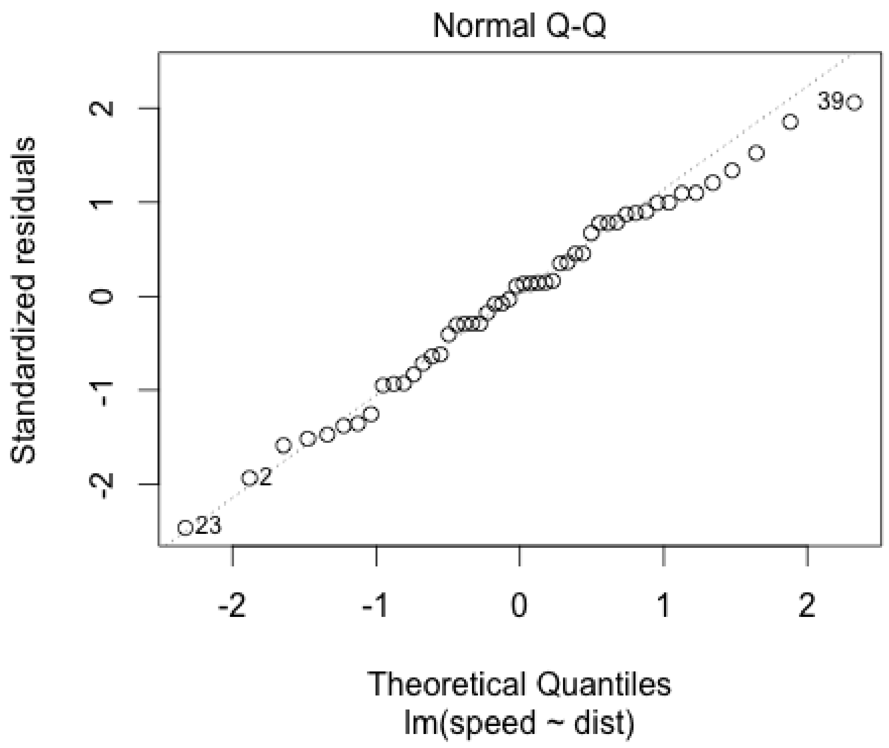
```
plot(car.fit)
```



R: Analyzing data

- Class lm object have their own overloaded plot() function

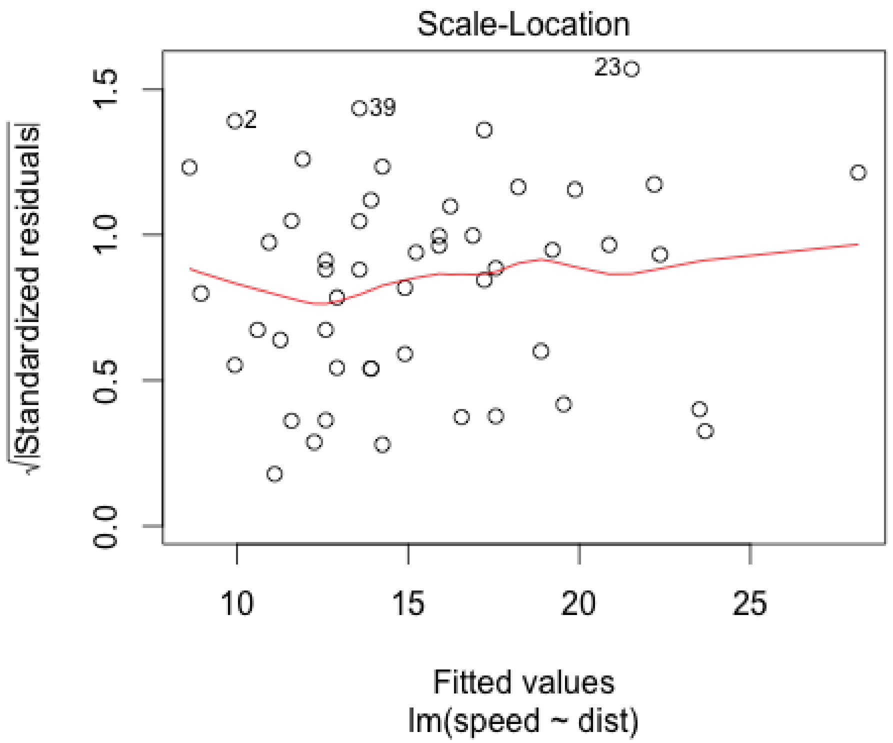
```
plot(car.fit)
```



R: Analyzing data

- Class lm object have their own overloaded plot() function

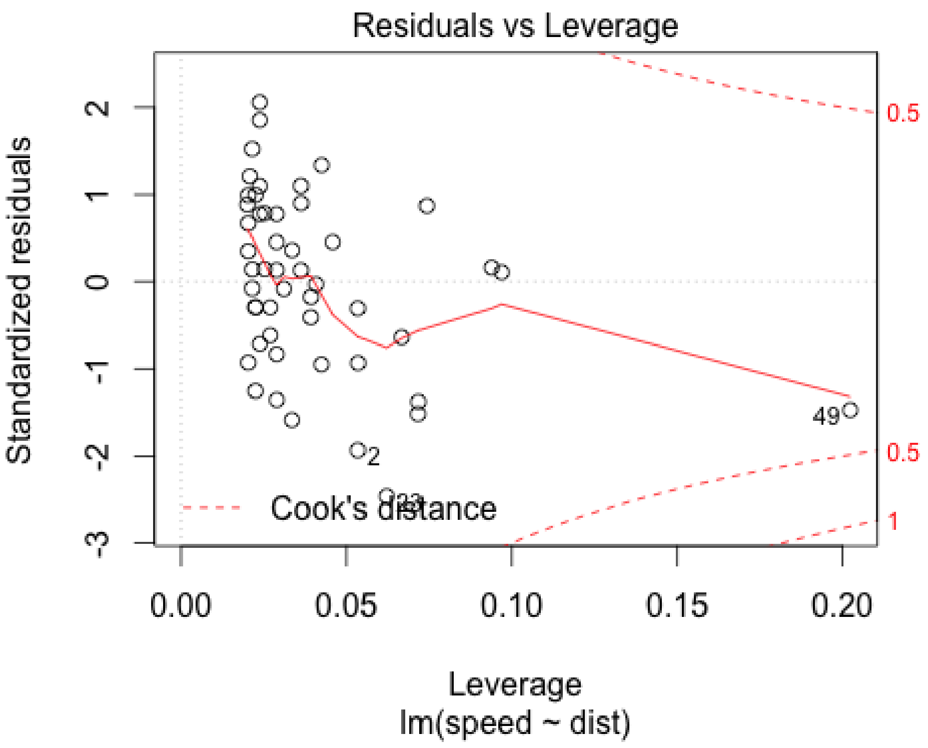
```
plot(car.fit)
```



R: Analyzing data

- Class lm object have their own overloaded plot() function

```
plot(car.fit)
```



R: Some exercises

- Install R on your own computer. (This might be homework.)

- Install the ggplot2 package and make it available to your R session. Test by running the command
  `qplot(y = mpg, data = mtcars)`

- The R dataset `faithful` is a dataset with waiting time between eruptions and the durations of the eruption for the Old Faithful in Yellowstone. With this dataset plot eruptions vs waiting time.

- With the `faithful` dataset fit a linear model. Is it a good fit?