R and Statistics

Matt Owen
mowen@iq.harvard.edu

Outline

1. R Basics
   a. **Review**: Variables, Functions and Data
   b. **New Concepts**: Packages and Formulas
2. Programming Descriptive Statistics
   a. **Numeric**: Measuring and Plotting
   b. **Factor**: Measuring and Plotting
   c. **Goodness of Fit**: QQ-plots

Outline

3. Programming Inferential Statistics
   3. The Basic Linear Model (`lm`)
   4. Logistic Regression with Zelig (`logit`)
   5. Multiple Imputation with Amelia (`Amelia`)

Format for Slides

Regular text will be presented just like this.

```
# Comments will be grayed-out and preceded by a hash

# Code will be in preceded by “> ”
> sin(1.3)
```

Results from code example will be in a grayed out box with a dashed border.

```
[1] 0.9635582
```

R Basics

Review: Variables, Functions and Data

R Basics

Types of Variables in R

1. **Old Concepts (Review)**
   1. Numerics, Strings and Lists
   2. Functions
   3. Data

2. **New Concepts**
   1. Packages
   2. Formulas

Basic Variables

- Three basic types of variables
  - Numeric (integers and real numbers)
  - Character (letters, words and sentences)
  - List (a collection of other variables)
- Remember:
  - How to assign values to a variable
  - How to print the values of a variable

Basic Variables

Quick demonstration on data-types and assigning variables

```
# Numerical
> Number <- 4
> Numbers <- c(1, 1, 2, 3)
```

```
# Character-strings (words and sentences)
> String <- "Hello, World."
```

```
# Lists are their own variable but
# can be composed of numerical and character information
> List <- list(number = 4, string = "Some text")
```

Functions

- Functions are composed of two parts:
  - A function name, e.g. "median", "plot" or "sin"
  - A parameter list
- Functions typically return values
  - E.g. the "sin" function returns a numeric value
  - "plot" and “message" functions typically do not
- **We won’t be writing new functions today**

Functions

Many mathematical functions are already in R.

```
# Computing the sine of 1.3
> sin(1.3)
```

```
# Computing the square-root of 3
> sqrt(3)
```

```
# Most statistical functions are available
> numbers <- c(1, 3, 4, 4, 5)
> mean(numbers)
> median(numbers)
```

```
# Arithmetic operators can be used along with functions
> 4 + 3
> 2 * numbers
> 2 + sin(1.3)
```

Functions Continued

Additionally, non-mathematical can be functions.

```
# Printing a message to the screen
> message("This is a sample message!")
```

```
# Getting the help-file for media
> help("median")
> ?median
```

Functions are immensely important!
And we will be using them constantly.

Important Note: The "`help`" function (featured above) prints information about other functions and data sets.

Data

- Data sets are referred to as "**`data frames`**“
- “**`Data frames`**” are a type of variable
- They are **usually** table-shaped (**rows** and **columns**)
- **Rows** represent individual data points
- **Columns** represent variables and specific information about the data point.

Data

- The "ChickWeight" data frame comes packaged with R.
- Load it using the "`data`" function

```
> data(ChickWeight)
```

If it seems like nothing happened, the data frame is now loaded.

```
# Help will give us information about this data frame
> help("ChickWeight")
```

Data

“summary” is an important function for most types of variables.

```
# Data sets can be summarized
> summary(ChickWeight)
```

“nrow” is useful for counting the number of data points in a set

```
# Display the number of rows.
> nrow(ChickWeight)
```

“colnames” gives important information about variables within a data frame

```
# List the names of the columns of "ChickWeight"
> colnames(ChickWeight)
```

Data

```
# Columns are retrieved using the following syntax:
> ChickWeight[ , “weight”]
```

```
# Rows are retrieved using the following syntax:
> ChickWeight[1, ]
```

```
# Specific variable information is retrieved:
> ChickWeight[1, “weight”]
```

Data (Conclusion)

- "`data()`" loads data that comes with R
- Other ways to load data:
  - `read.table()`
  - `read.csv()`
  - `load()`

Exercise: Variables, Functions and Data Frames

1. Load the “Seatbelts” Data (using “`data`”)
2. Use “help” to list information about this data frame
3. Count the number of data points (`nrow`)
4. List the names of all the variables (`colnames`)
5. Compute the **median** and **mean** of the “`DriversKilled`” column

Questions?

New Concepts: Packages and Formula

R Basics

Packages

- Are collections of **data sets** and **functions**.
- Add specific mathematical techniques to complement R's built-in features.
- Can be downloaded from an interactive R session or directly from CRAN:
  http://cran.r-project.org/

Loading Packages

Packages, that are already installed, can be loaded by using the "**library**" function.

```
# Load the "MASS" Package
> library(MASS)

# Or load "Zelig", a statistcal package we'll be using later
> library(Zelig)
```

- Most advanced statistical techniques require the use of packages.
- For a complete list, use "`library()`" without parameters.

```
# List all available R packages
> library()
```

Packages: A Quick Example

- The **MASS** package contains a data set titled "**`Animals`**”.
- You can’t load “`Animals`” until you load the **MASS** package!

```
# Display the data frame "Animals"
> data(Animals)

Warning message:
In data(Animals) : data set 'Animals' not found

# Load the "MASS" package
> library(MASS)

# Display the data frame "Animals"
> data(Animals)
> Animals
```

Your R session should now display the data frame "Animals"

```
> library(MASS)
Warning message:
package 'MASS' was built under R version 2.13.1
> Animals
                      body  brain
Mountain beaver      1.350    8.1
Cow                465.000  423.0
Grey wolf           36.330  119.5
Goat                27.660  115.0
Guinea pig           1.040    5.5
Dipliodocus      11700.000   50.0
Asian elephant    2547.000 4603.0
Donkey             187.100  419.0
Horse              521.000  655.0
Potar monkey        10.000  115.0
Cat                  3.300   25.6
Giraffe            529.000  680.0
Gorilla            207.000  406.0
Human               62.000 1320.0
African elephant  6654.000 5712.0
Triceratops       9400.000   70.0
Rhesus monkey        6.800  179.0
Kangaroo            35.000   56.0
Golden hamster       0.120    1.0
Mouse                0.023    0.4
Rabbit               2.500   12.1
Sheep               55.500  175.0
Jaguar             100.000  157.0
Chimpanzee          52.160  440.0
Rat                  0.280    1.9
Brachiosaurus    87000.000  154.5
Mole                 0.122    3.0
Pig                192.000  180.0
```

Packages (Conclusion)

- Most statistical techniques are in packages
- We will be uses packages extensively
- Remember: “`library()`" will list all available packages that can be used.

Reminder: Data Sets

The following script

```
# Load the "ChickWeight" data frame
> data(ChickWeight)
> colnames(ChickWeight)
```

Should produce the result:

```
[1] "weight" "Time"    "Chick"   "Diet"
```

The **column names** are all viable components to a formula relating to the **ChickWeight** data frame.

What is a Formula?

- A type of variable
- A language construct
- Used with the tilde (~) operator
- Has the form:

```
> X ~ Y1 + Y2 + Y3
```

Read:
“The response variable X is dependent upon Y1, Y2 and Y3.”

Formula Fact Sheet

- Formulae relate **columns** of **data sets** to one another.
- Formulae **only** make sense in reference to a **data frame**.
- The tilde "**~**" is used to separate the left and right hand sides of formulae.
- The *left-hand side* specifies **dependent variables**.
- The *right-hand side* specifies **independent variables**.

Example: Formula

Formula can be useful to draw scatter plots.

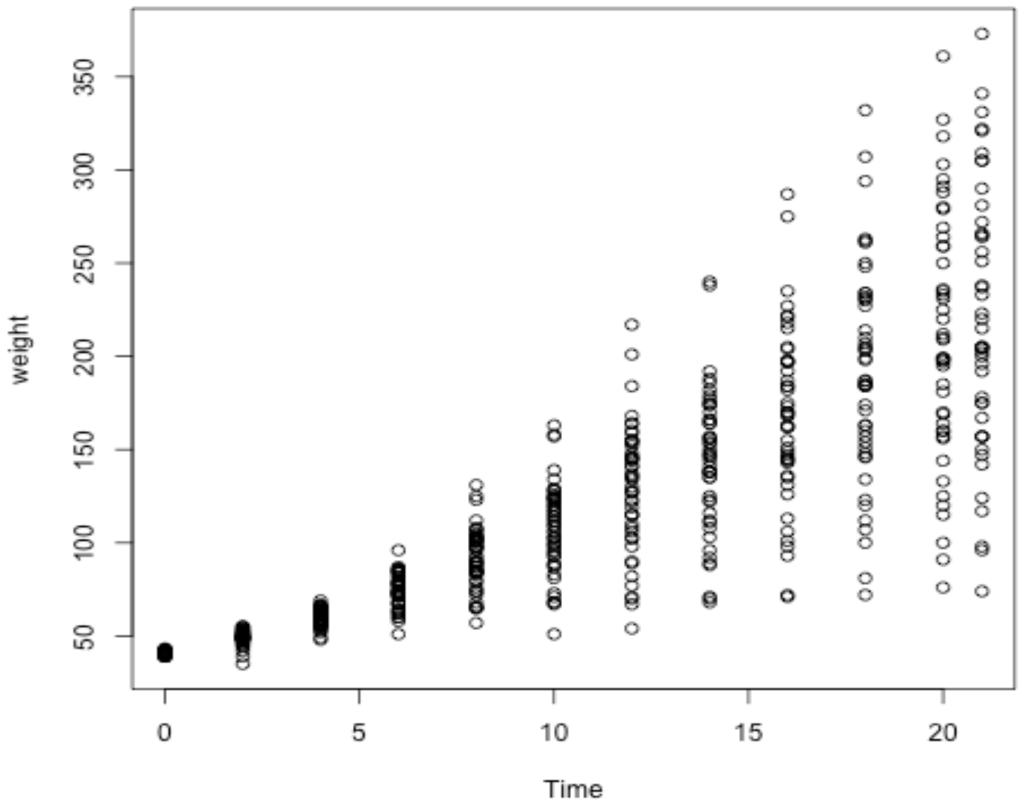
In this example we will plot weight as dependent on time

```
# Read: Weight is dependent upon Time
> chick.formula <- weight ~ Time
```

```
# Draw the scatter plot
> plot(weight ~ Time, data=ChickWeight)

# Or equivalently
> plot(chick.formula, data=ChickWeight)
```

Formula Example (Plot)



- displays the growth rate with respect to time (N=50)
- growth rate may vary differently between chicks

We can do some simple graphical analysis to see how growth rate varies.

Recap: Formula

- The **formula** (`weight ~ time`) only specifies that weight and time are related
- Does not specify parameters
- We use a regression to determine this.
- "Fitting" the statistical model determines these **unknown parameters**

Example: Formula

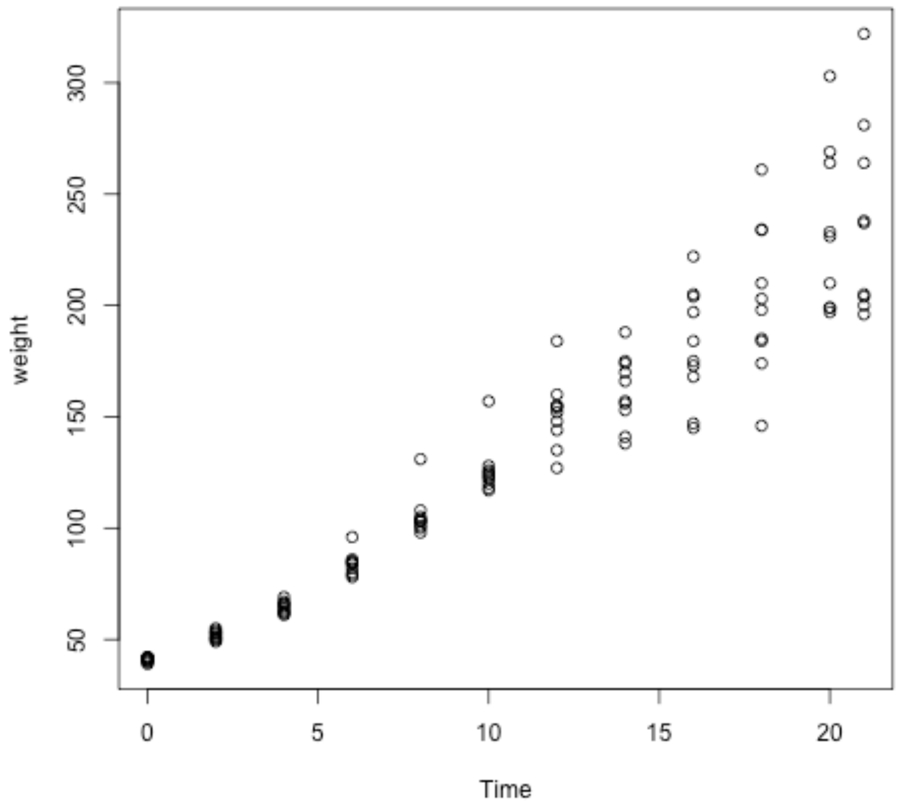
- Chick Weight is clearly not **entirely** based on Time
- So let's explore the individual diet as well
- Use "`subset`" to create a smaller data frame
- In particular, we will examine all chicks fed with the 4th diet

```
# Subset the "ChickWeight" data frame, and plot results
> ChickDiet4 <- subset(ChickWeight, Diet == 4)
> plot(weight ~ Time, ChickDiet4)
```

All values of the "Diet" column of "ChickDiet4" should read "4"

```
# Read the help documentation about the"subset" function
> help("subset")
```

Chick Diet #4 Plot



This plot displays the growth rate with respect to time of **only chicks that were on the fourth diet**.

•Results look more linear
•We can use **numerical** tools in addition to **graphical** tools

Recap: Formula

- "Formulae" (in R) describe the relationship between **independent** and **dependent** variables.
- To create a formula, you must use the tilde ("**~**").
- Formulae only make sense in the context of a data set.

Looking Ahead

- Formulae will be further exploring the relationship between variables, expressed within a formula, and data sets.
- Formulae, data sets and statistical methods will be **crucial** to everything we do for the remainder of the training session.

Questions?

Exercise: Data and Formulae

1. Load the “`MASS`” package
2. Using “`data`”, load the “`mammals`” data set
3. Create a formula linking brain-mass and body weight
4. Draw a scatter plot using this formula along with the mammals data set
5. Subset the data frame, so that it only contains animals with body weight less than **10kg**
6. Draw a scatter plot using the same formula with this **new data set**

Programming Descriptive Statistics

Outline: Descriptive Statistics

1. **Measuring and Plotting**
   a. Numeric Data
   b. Factor Data
2. **Statistical Distributions**
   a. Simulating Distributions
   b. Goodness of Fit (QQ-Plot)

Measuring and Plotting

Programming Descriptive Statistics

Programming Descriptive Statistics

- We know how to use:
  - Formulae and Data Frames
  - Functions
  - Plots
- So we can now work with them all together to do actual useful statistics
- Measuring and Plotting statistics is basic
- After this, **statistical inference**

Example: Measuring Numeric Data

```
# Load this example's data set "survey"
> data(survey)

# Read the documentation, and list the column names
> help("survey")
> colnames(survey)
```

```
 [1] "Sex"    "Wr.Hnd" "NW.Hnd" "W.Hnd"  "Fold"   "Pulse"  "Clap"   "Exer"
 [9] "Smoke"  "Height" "M.I"    "Age"
```

Now, we will:

1. Compute the **`mean, median`** and **`standard deviation`** of "**`Height`**"
2. Measure the frequency of smokers in the population
3. Measure the gender distribution of the population

Example: Measuring Numeric Data

"`mean`", "`median`" and "`sd`" respectively correspond to computing the statistical mean, median and standard deviation of a sample.

```
# Compute the mean, median and standard deviation of
# students` heights
> mean(survey$Height, na.rm=TRUE)
```

```
[1] 172.3809
```

```
> median(survey$Height, na.rm=TRUE)
```

```
[1] 171
```

```
> sd(survey$Height, na.rm=TRUE)
```

```
[1] 9.847528
```

Recap: Measuring Numeric Data

- "`mean`", "`median`" and "`sd`" work **best with vectors** not **data.frames.**
- "`na.rm=TRUE`" removes "**Not Available**" data before computing these measures.
- These functions **only** work with numerical input. **NO FACTORS, etc.**
- These functions return numerical values.

Example: Plotting Numeric Data

- Describing data visually
- All of the tools from basic statistics are available
- Tools that we will need:
  - histogram (`hist`)
  - density plot (`density, plot`)
  - boxplot (`boxplot`)

Example: Plotting Numeric Data

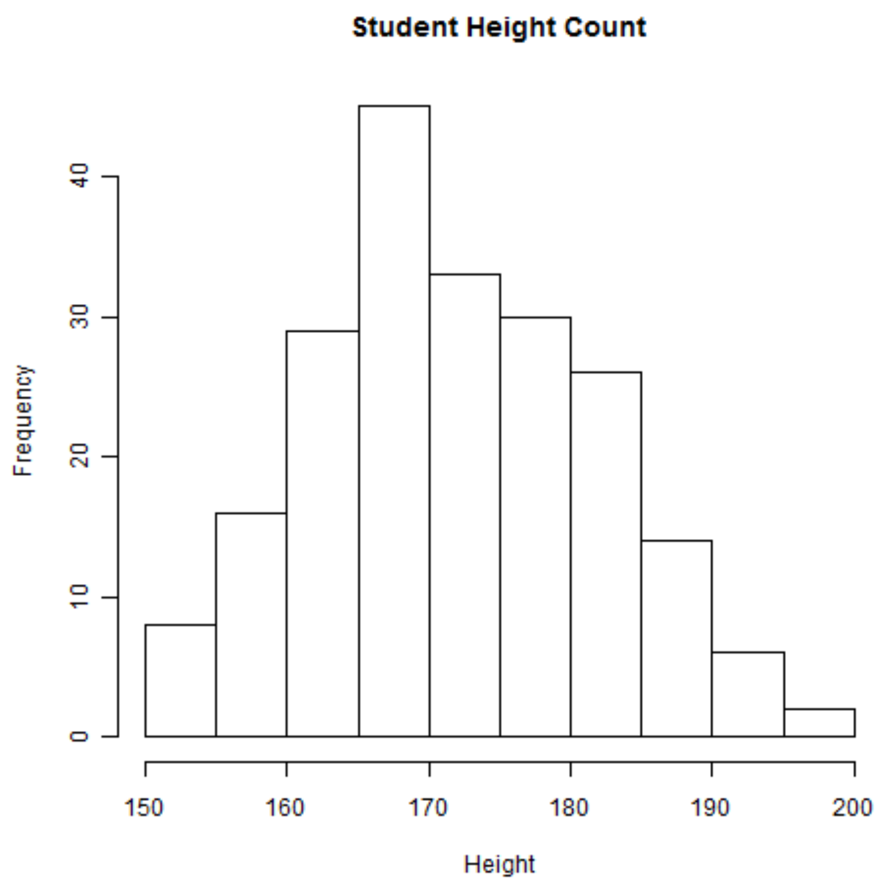
We will be working with the "survey" data set again.

```
# Load "MASS", "survey" and store "student.heights"
> library(MASS)
> data(survey)
```

Let's visualize the frequency of student heights.

```
# "hist" plots histograms
> hist(survey$Height, main = “Heights”, ylab = “Density”)
```

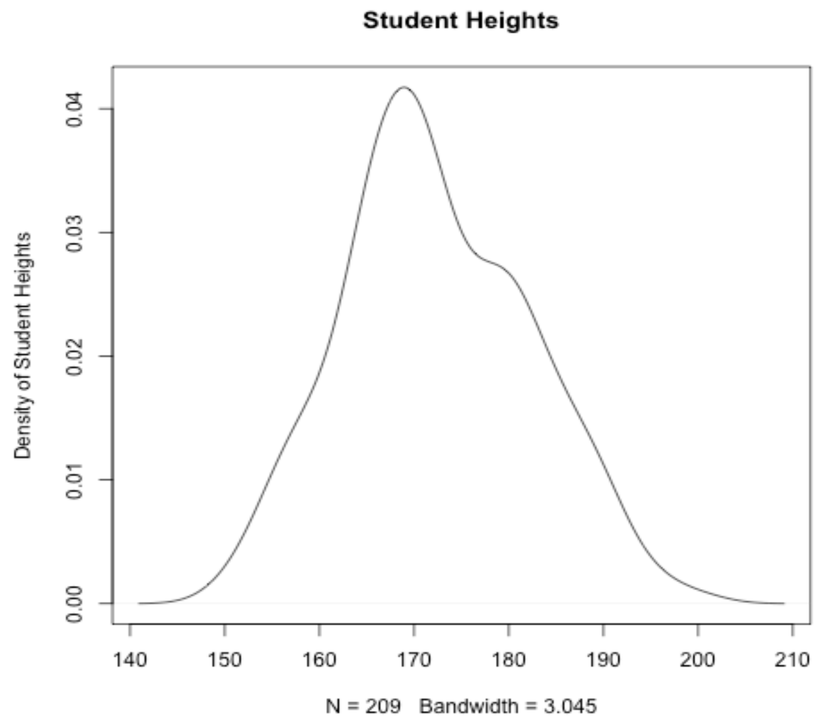
Example: Plotting Numeric Data



- Histograms are useful for showing the density of results
- We can change the number of breaks by setting the "`breaks`" parameter
- Similar data can be displayed by using the "`plot`" and "`density`" functions together

Example: Plotting Numeric Data

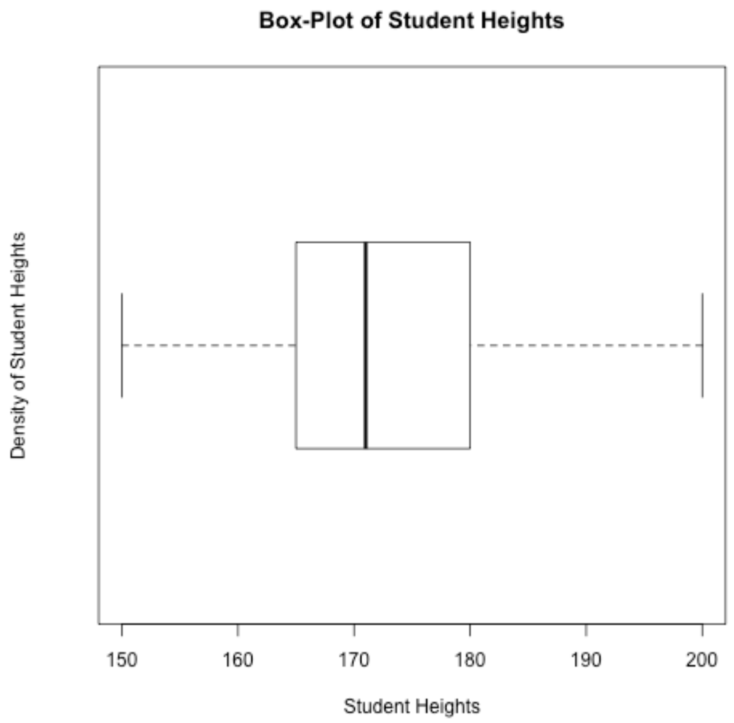
```
# Construct a density-plot (like a histogram but smoother)
> student.density <- density(survey$Height, na.rm=TRUE)
> plot(student.density, main="Student Heights", ylab=mylabel)
```



- Density-plots are essentially identical to histograms
- Require the use of both the "**`density`**" and "**`plot`**" functions
- "**`na.rm=TRUE`**" is requires to ensure that missing data is ignored

Example: Plotting Numeric Data

```
# Construct a box-plot, note that you do not use "density"
> library(MASS)
> boxplot(survey$Height, horizontal = TRUE)
```



- Box-Plots are useful for displaying quantile information
- "`horizontal=TRUE`" specifies that we want to display our plot horizontally

Recap: Plotting Numeric Data

- "`hist`" plot histograms
- "`plot`" and "`density`" can be used <u>together</u> to draw density plots
- “`boxplot`" draws box-plots

Review: Factor Data

- **Categorical and Nominal Data**
- "`levels`" returns all the different categories
- "`table`" returns the quantity of each category that is available
- "**barplots**" can display this type of categorical data
- Not all plots can!

Example: Describing Factor Data

We will be working with the "survey" data set again.

```
# Load "MASS", "survey" and store "student.smokes"
> library(MASS)
> data(survey)
```

To display all the categories of smokers, use the "levels" function

```
# Display the all categories
> levels(survey[ , “Smoke”])
```

```
[1] "Heavy" "Never" "Occas" "Regul"
```

Students' smoking habits can **only** fall into the above **4 categories**.

Example: Describing Factor Data

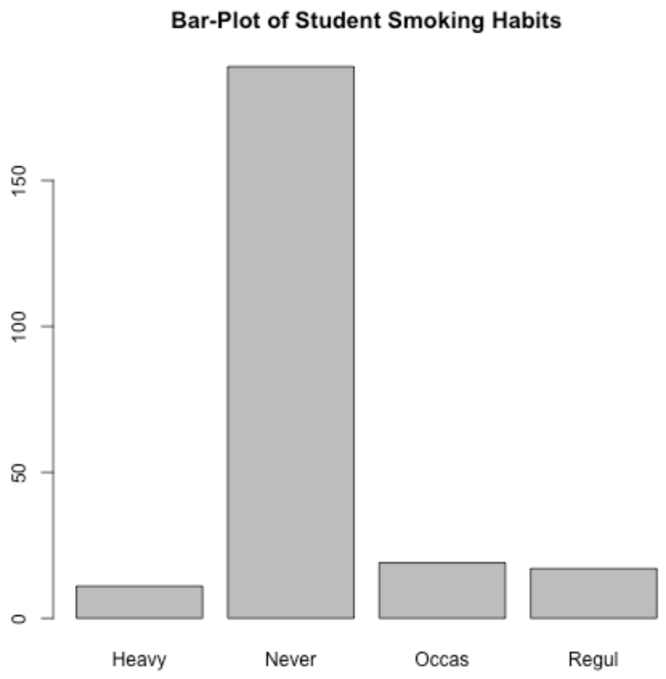
"table" will numerically display how much of each category is within the sample

```
# How many students belong to each smoker category?
> table(survey$Smoke)
```

```
Heavy Never Occas Regul
   11   189    19    17
```

Example 1: Plotting Factor Data

```
# Simply use the "plot" function. R can figure out what
# is meant by context.
> plot(survey[, "Smoke"])
```



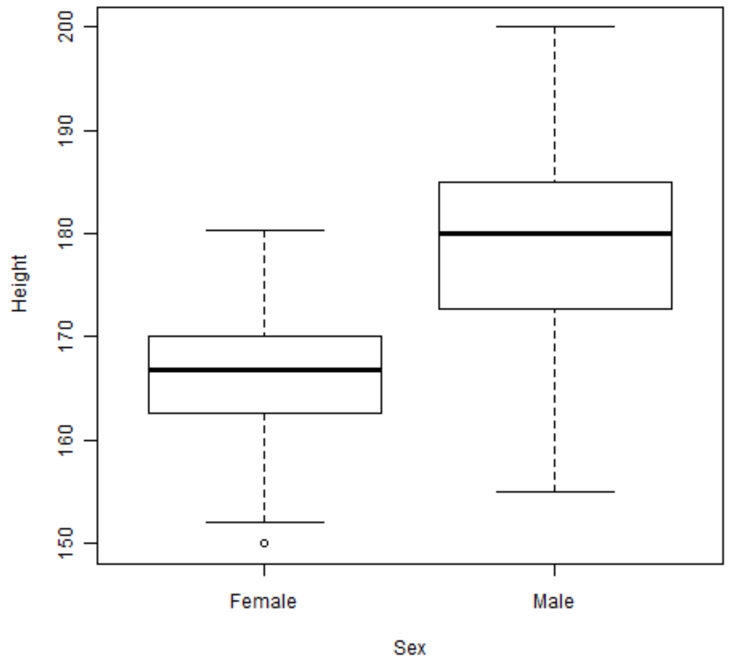
- Bar plots are drawn by the "`plot`" function

Example 2: Plotting Factor Data

- Formulae can also be used with factor data
- Typically, this is used to produce bar plots of **numeric** data dependent upon **factor** data

Example 2: Plotting Factor Data

```
# Simply use the "plot" function. R can figure out what
# is meant by context.
> plot(Height ~ Sex, data=survey)
```



- Two bar-plots are drawn side by side (one for each factor-level)
- Stacked barplots can be generated by have factor data as the **outcome** variable
- Try this on your own

Exercise: Measuring and Plotting

1. Load the “cabbages” data set from the “MASS” package
2. Compute the **mean** and **standard deviation (sd)** of cabbage head weight (“HeadWt”)
3. Plot head weight (HeadWt) relative to:
    1. Cultivar (“Cult”)
    2. Day Planted (“Date”)
4. Plot “ascorbic acid content” (VitC) relative to head weight
5. (Optional) Plot “Date” (date planted) relative to “Cult” (cultivar)

Statistical Distributions

(Descriptive Statistics)

Important Concepts

The following are basic components of any statistical distribution.

- **Probability Density Function** (`dnorm`)
- **Cumulative Density Function** (`pnorm`)
- **Sampling Function** (`rnorm`)
- **Quantile Function; Inverse of Cumulative Density Function** (`qnorm`)

Briefly: "seq"

The "seq" function creates a **sequence of points** from a starting value to and ending value.

```
# From = 0.0
# To   = 2.0
# By   = 0.5
> seq(from = 0, to = 2, by = 0.5)
```

```
[1] 0.0 0.5 1.0 1.5 2.0
```

This technique will be used in the upcoming example to create values for the X-axis.

Example: The Normal Distribution

Construct the values for the X-axis, using the "seq" function

```
> Values <- seq(from = -3, to = 3, by = .05)
```

Construct the corresponding density curve (using the Gaussian pdf) for a Normal distribution with mean = 0 and standard deviation = 1

```
> Density <- dnorm(Values, mean = 0, sd = 1)
```

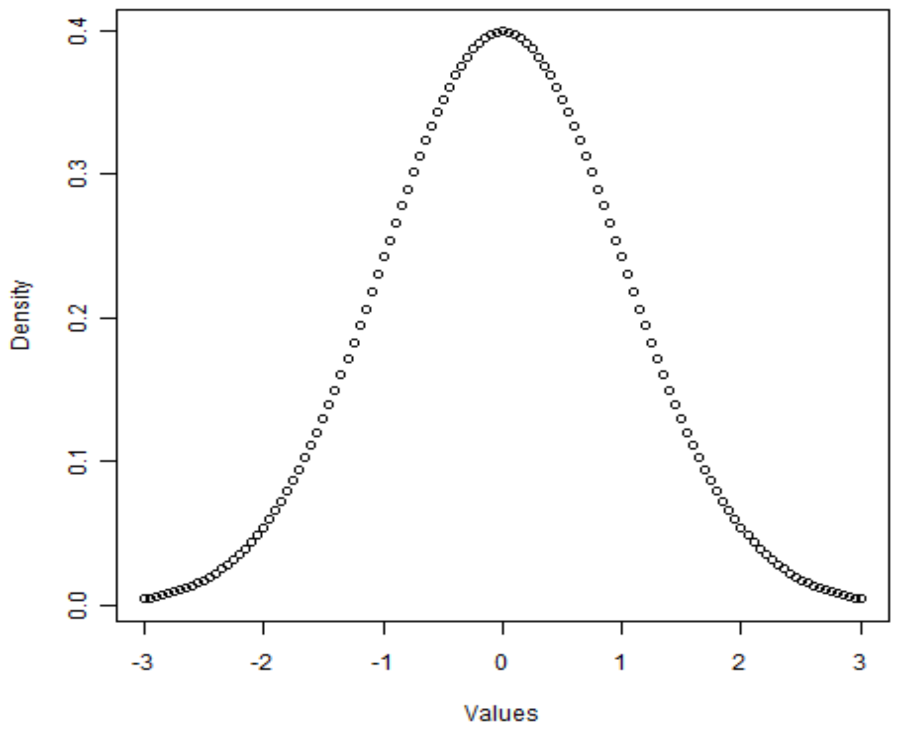
Plot the results, using:

- "`Values`" as the X-axis and
- "`Density`", the probability density function, as the Y-axis

```
> plot(x = Values, y = Density)
```

Example: The Normal Distribution

`dnorm`



- "**`dnorm`**" is the normal function's probability density function.
- Specifies **relative likelihood**
- This defines the "bell curve" shape of the normal distribution.
- This is descriptive but not so useful.

Example: The Normal Distribution

Construct the values for the X-axis, using the "seq" function

```
> Values <- seq(from = -3, to = 3, by = .05)
```

Construct the cumulative probability curve
for a Normal distribution with mean = 0 and standard deviation = 1

```
> Cumulative <- pnorm(Values, mean = 0, sd = 1)
```

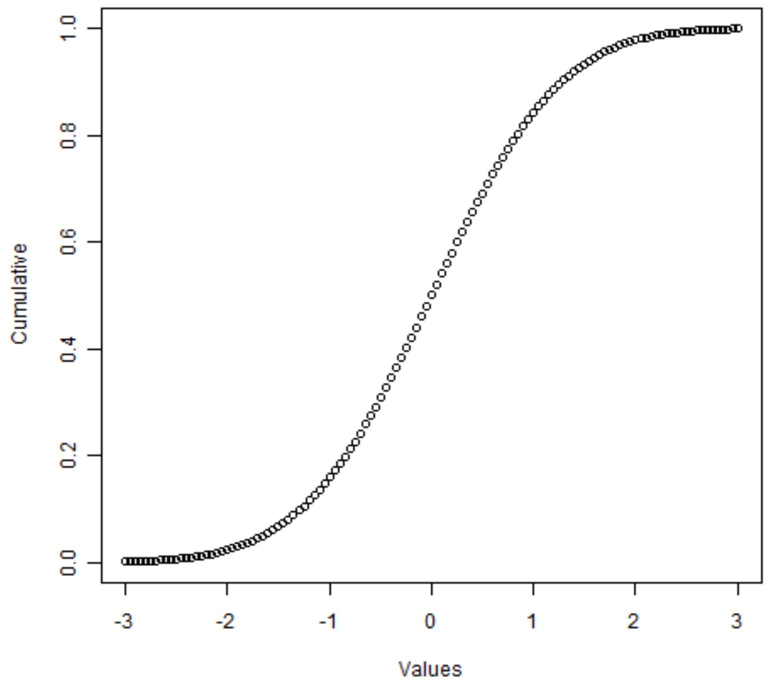
Plot the results, using:

- "`Values`" as the X-axis and
- "`Density`", the probability density function, as the Y-axis

```
> plot(x = Values, y = Cumulative)
```

Example: The Normal Distribution

`qnorm`



- "**`qnorm`**" is the normal function's cumulative density function.
- Specifies **probability of a value belonging to a region less-than x**
- This is useful for considering how reasonable a result is

Once Again: Normal Distribution

- "**`rnorm`**" generates random samples from a normal distribution
- "**`dnorm`**" is the density function
- "**`pnorm`**" is the cumulative distribution function
- Corresponding functions are available for most statistical distributions (Student, Binomial, Poisson, etc.)

Questions?

Exercise: Descriptive Statistics

1. Load the "`survey`" data set
2. Using Height column, compute the mean (`mean`) and standard deviation (`sd`)
3. Generate a simulated population (N = 1000) such that `Y ~ Normal(`μ`, `σ^2`)` (`Use rnorm`)
4. Plot the original population and the newly modeled population

Quantile-Quantile Plots

- Assess "goodness of fit" graphically
- Answers are two data sets similar?
- Three Functions:
  - `qqnorm` (produce a Normal QQ-Plot)
  - `qqline` (produce the QQ-Line for reference)
  - `qqplot` (compare two **different** data samples)

Example: QQ-Plots

Once again, the "`survey`" data

```
> library(MASS)
> data(survey)
```

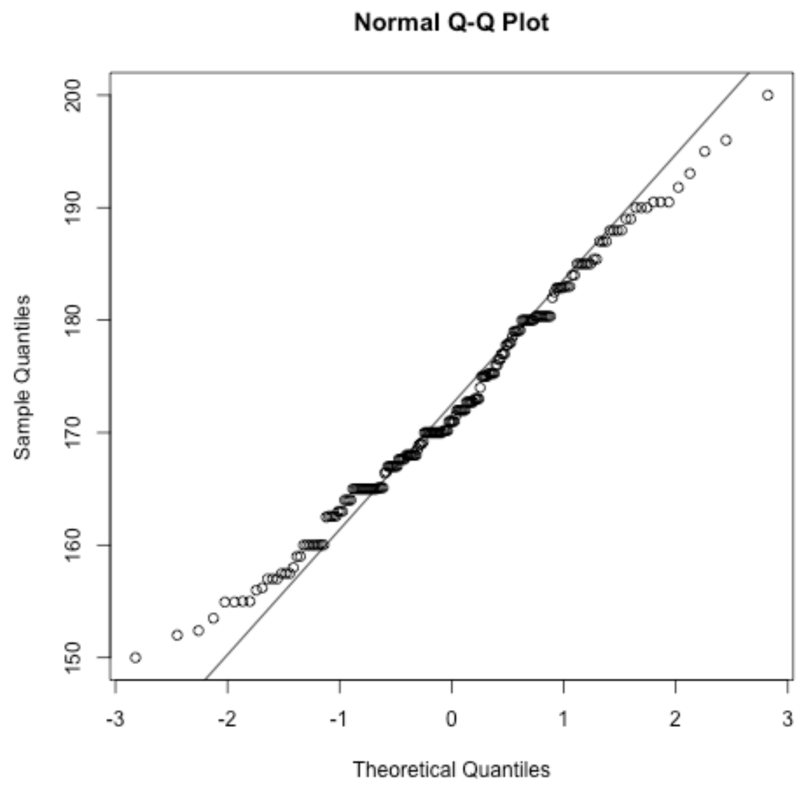
Draw the normal QQ-Plot of the heights of the students

```
# Create the quantile-quantile plot
> qqnorm(survey$Height)

# Draw the reference line
> qqline(survey$Height)
```

Example: QQ-Plots

`qqnorm`



- "**`qqnorm`**" is draws a QQ-plot of the sample data against a normal distribution
- The "linear-ness" of the dots describes the data's normality
- This can be useful in the absence of numeric tools

Example: QQ-Plots

Let's see an example of two things not being similar

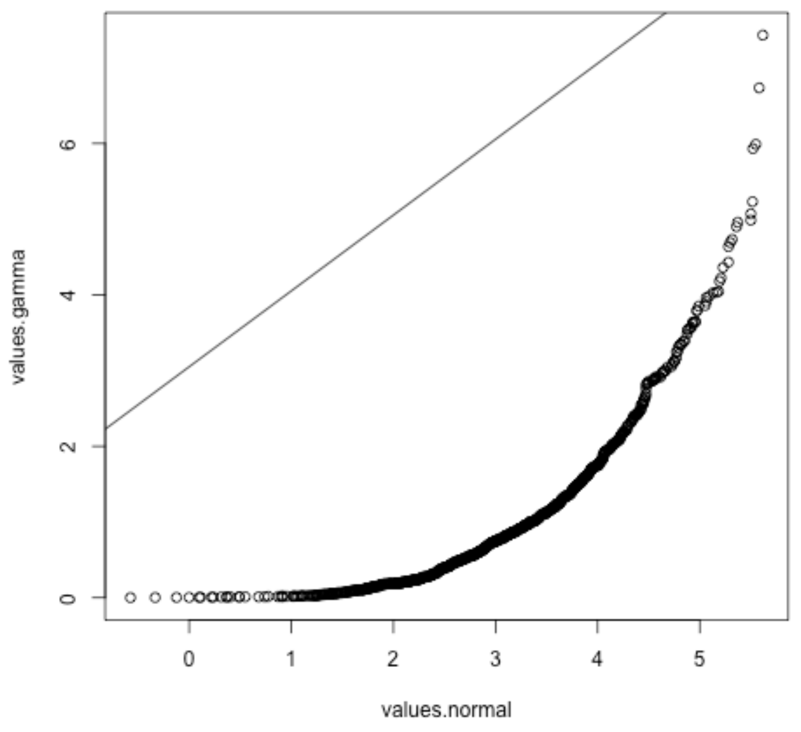
```
> values.normal <- rnorm(1000, mean=3, sd=1)
> values.gamma <- rgamma(1000, shape=1, scale=1)
```

Compare the two using a QQ-plot

```
> qqplot(values.normal, values.gamma)
> qqline(values.normal)
```

Example: QQ-Plots

`qqnorm`



- "**`qqplot`**" is draws a QQ-plot of the sample data against a normal distribution
- The "linear-ness" **similarity** between the two samples
- The gamma and normal distribution are not so similar.

Example: QQ-Plots

Let's demonstrate something we already know. The T-distribution becomes normal with large degrees of freedom

```
> par(mfrow=c(1, 2))
> t.values <- rt(n=1000, df=3)
> normal.values <- rnorm(n=1000)
```

Compare the two using a QQ-plot

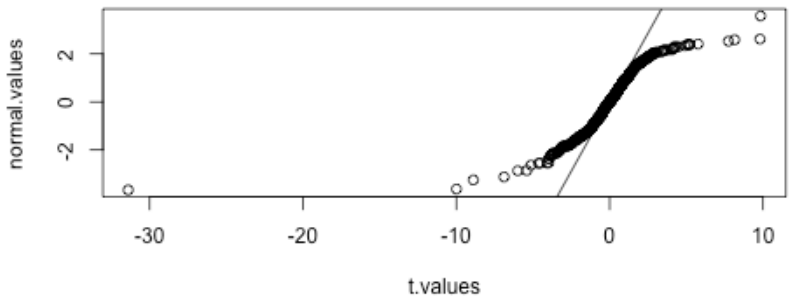
```
> qqplot(t.values, normal.values)
> qqline(t.values)
```

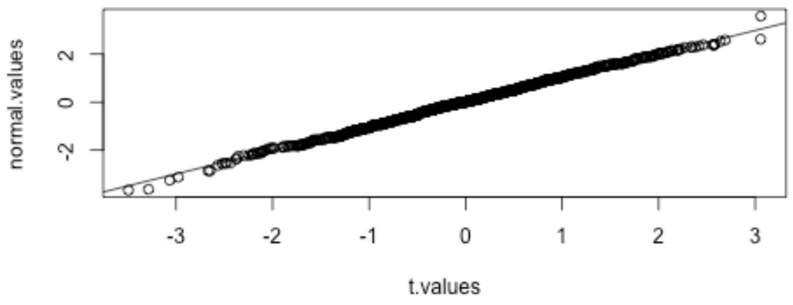
Then

```
> t.values <- rt(n=1000, df=30)
> qqplot(t.values, normal.values)
> qqline(t.values)
```

Example: QQ-Plots

`qqnorm`





- The top represents T-distribution with "`df=3`"
- The bottom is "`df=30`"
- "`par(mfrow=c(2, 1))`" specifies that that plots should be stacked on top of anothert

Questions?

Inferential Statistics

Inferential Statistics

- Not strictly desctiptive
- Interpolates and Extrapolates Data
- Much more interesting than descriptive stats

The Basic Linear Model

- Outcome is considered **linear** with respect to dependent variables
- "lm" is the builtin method used by R
- Some functions can be applied to its return-value:
  - vcov
  - coef
  - plot

Example: The Basic Linear Model

Load ChickWeight and fit the statistical model

```
> data(ChickWeight)
> fitted <- lm(weight ~ Diet + Time, data=ChickWeight)
```

Compute some interesting measures

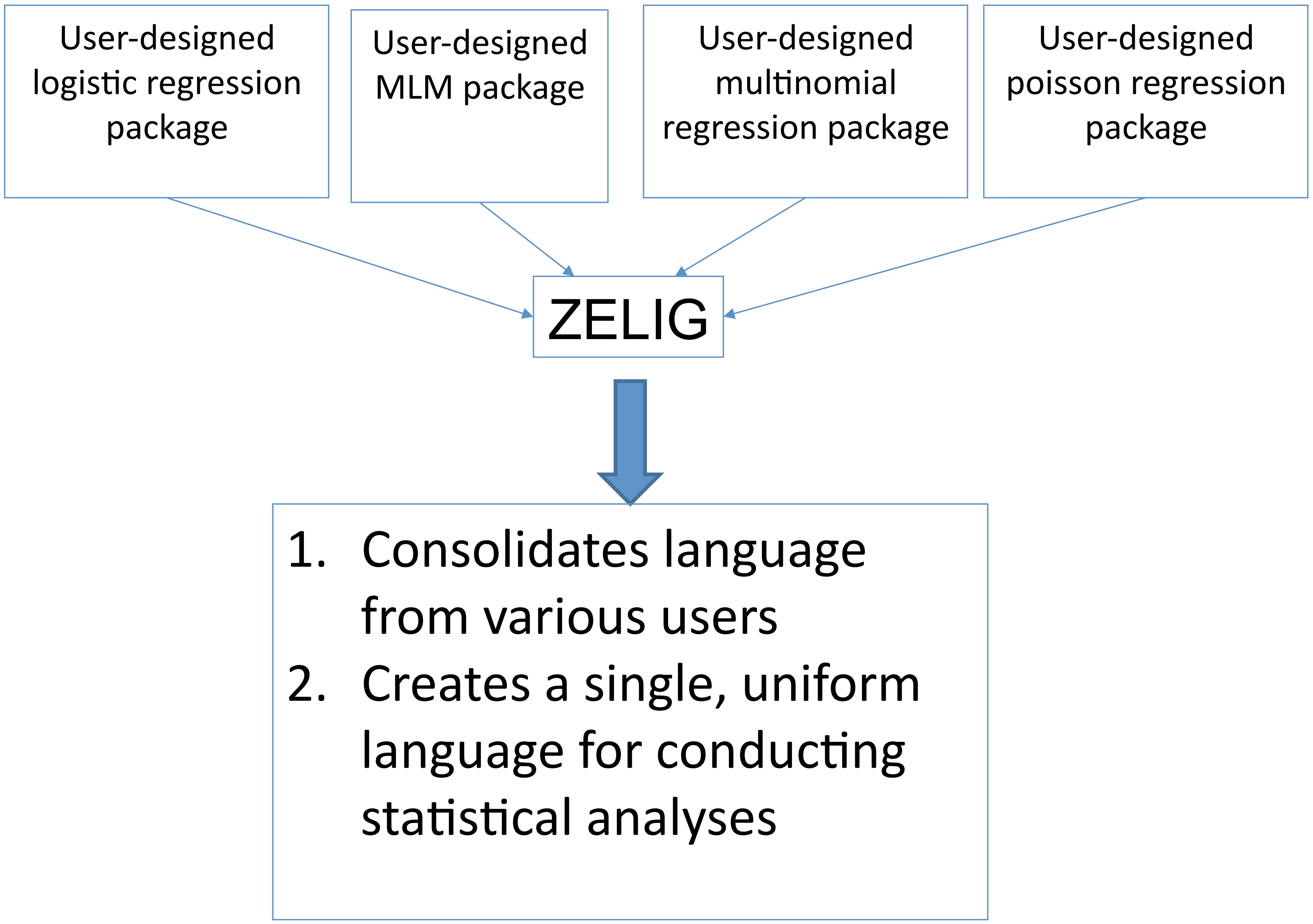
```
> vcov(fitted)
> coef(fitted)
```

And Plot

```
> plot(fitted)
```

Zelig

- Incorporates R code from many different researchers
- Provides single format for entering code from a variety of sources
- Great documentation
- http://gking.harvard.edu/zelig


User-designed logistic regression package
User-designed MLM package
User-designed multinomial regression package
User-designed poisson regression package
ZELIG
1. Consolidates language from various users
2. Creates a single, uniform language for conducting statistical analyses

Installation: Zelig

Install Zelig

```
> install.packages("Zelig")
```

Load the library

```
> library(Zelig)
```

Regression Models in Zelig

- Zelig offers a wide variety of regression models
  - Multinomial, logistic, poisson, continuous dependent variable models
  - Mixed models
  - Survey data models
  - See Zelig manual for a complete list

  S:\R and Statistics\RandStatistics\Zelig Manual.pdf

Regression Models in Zelig

- All models are specified using the same basic format
  1. Specify your regression model
  2. Request the regression procedure you’d like to use
  3. Tell Zelig which dataset to use

```
# 1) "y ~ x1 + x2" will be the formula we need to fit
# 2) "ls" specifies that we are using "least squares"
# 3) "dataname" is the name of the data frame that where we
#    can find the variables - y, x1, x2
> zelig( y ~ x1 + x2, model="ls", data=dataname )
```

Linear Regression (Zelig)

- Predict body mass index (bmi) based on
  1. Number of cigarettes smoked per day (cigsday)
  2. Duration of moderate exercise (modmin)
  3. Sleep (sleep)
- Load the National Health Interview Survey

```
# Load the NatHealth2008 data frame
> load("S:\\NatHealth2008.rdata")

# Load the Zelig library
> library(Zelig)
```

Linear Regression (Zelig)

Use the zelig funciton to use a least squares regression to fit the data.

Note that "least squares" is an appropriate model, because "body mass index" is a continuous variable

```
# Linearly fit the model, and summarize the results
> z.out <- zelig( bmi ~ cigsday + modmin + sleep,
                  model = "ls",
                  data = NatHealth2008
                 )
> summary(z.out)
```

```
Call:
zelig(formula = bmi ~ cigsday + modmin + sleep, model = "ls",
    data = NatHealth2008)

Residuals:
     Min       1Q   Median       3Q      Max
-1295.79  -399.36   -79.89   291.20  3588.20

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 2903.2941    72.1368  40.247  < 2e-16 ***
cigsday       -0.5936     1.3984  -0.425  0.67124
modmin        -0.3008     0.2019  -1.490  0.13634
sleep        -26.5090     9.5673  -2.771  0.00565 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 567.6 on 1952 degrees of freedom
Multiple R-squared: 0.005009,   Adjusted R-squared: 0.00348
F-statistic: 3.276 on 3 and 1952 DF,  p-value: 0.0203
```

Logistic Regression (Zelig)

- Diagnosing hypertension based on:
  - age (`AGE_P`)
  - sex (`sex`)
  - sleep (`sleep`)
  - body mass index (`bmi`)

```
# Comments will be grayed-out and preceded by a hash
> z.out <- zelig( hypev ~ AGE_P + sleep + bmi,
                  model = "logit",
                  data  = NatHealth2008
                )
> summary(z.out)
```

Logistic Regression

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4823  -0.7491  -0.4302   0.8471   2.7953

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.763e+00  1.412e-01 -47.908   <2e-16 ***
AGE_P        6.548e-02  1.109e-03  59.031   <2e-16 ***
sex          6.238e-03  3.485e-02   0.179    0.858
sleep       -1.711e-02  1.223e-02  -1.399    0.162
bmi          9.863e-04  2.899e-05  34.023   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 25927  on 20717  degrees of freedom
Residual deviance: 20265  on 20713  degrees of freedom
AIC: 20275
```

Logistic Regression

- Zelig also allows meaningful interpretations of equations
- For example, computing likelihood of hypertension based on age:

```
# Make predictions for likelihood of hypertension if
# the interviewee is age 33
> x.young <- setx( z.out, AGE_P = 33 )

# And another if the interviewee is age 66
> x.old <- setx( z.out, AGE_P = 66 )
```

Logistic Regression

The values have been stored in:

- x.young
- x.old

The quantities of interest for these values are computed:

```
# Simulate the quantities of interest (predicted value, etc.)
> s.out <- sim( z.out, x = x.young, x1 = x.old )

# Display summary information and plots
> summary(s.out)
> plot(s.out)
```

Logistic Regression

```
Values of X
  (Intercept) AGE_P       sex    sleep      bmi
1           1    33 0.5549281 7.179747 2763.163

Values of X1
  (Intercept) AGE_P       sex    sleep      bmi
1           1    66 0.5549281 7.179747 2763.163

Expected Values: E(Y|X)
       mean          sd      2.5%     97.5%
1 0.1198186 0.003121433 0.1140429 0.1262853

Predicted Values: Y|X
      0     1
1 0.874 0.126

First Differences in Expected Values: E(Y|X1)-E(Y|X)
       mean          sd      2.5%     97.5%
1 0.4215098 0.006301644 0.4094499 0.4336803

Risk Ratios: P(Y=1|X1)/P(Y=1|X)
      mean        sd     2.5%    97.5%
1 4.520915 0.1250251 4.273213 4.763982
```

Expected value of hypertension in a 33 year old (y|x) given current sampling distribution

Predicted values of y|x given Binomial distribution

Difference in expected value of a 66 vs. 33 year old

Relative risk ratio (similar idea to odds ratio, but different calculation)

There is a 3.8 times greater probability of hypertension for a 66 year old vs. a 33 year old

Questions?

Exercise: Zelig Regression Model

1. Load "Zelig" and the "turnout" data set
2. Plot the fitted model (using the plot function)
3. Predict **voter turnout** based on 2 or more explanatory variables
4. Answer "Would 10 year olds show up to the voter polls?“
5. And “Would someone show up if they only graduated 5th grade?”
6. Plot the results of the simulations

7. (As a class) Compute first differences

Multiple Imputation

- Majority of datasets contain missing data
  - Produces a variety of problems and limitations to data analysis
- Multiple imputation (MI) generates multiple, complete datasets that contain estimations of missing datapoints

Multiple Imputation

- MI typically thought of as involving three steps:
    1. Selection of imputation model
    2. Generation of imputed datasets
    3. Combining results across imputed datasets
- We're focusing on 2 & 3
- **Please make sure you have a solid understanding of all steps before performing MI with your own data**

Multiple Imputation

- Amelia package in R is powerful, fast, and easy to use
- Relies on Expectation maximization baysian (EMB) algorithm (Honaker & King, 2010)
- See Amelia documentation for more information about its imputation procedures

S:\DataClass\RandStatistics\Amelia Documentation.pdf

Multiple Imputation

- First, we'll need to install Amelia:

```
install.packages("Amelia",repos="http://r.iq.harvard.edu")

library(Amelia)
```

Multiple Imputation

- We're going to create several datasets to look at a model predicting the number of days of work missed/year (wkdayr)
- Let's say we want to predict wkdayr using:
  - Cigarettes smoked/day (cigsday)
  - Amount of moderate exercise (modmin)
  - Sleep (sleep)

Multiple Imputation

- Before running Amelia you'll need a single dataset completely prepped for MI
  - Data should be completely cleaned
  - Categorical variables should be dummied (with one omitted)*
  - Data should contain only variables in your imputation model + any ID variables you'll need to merge your datasets

*Amelia can dummy your variables for you – but you won't be able to control your omitted category

Multiple Imputation

- Now, load your data – it’s already been prepped for MI

```
load("N:\\R and Statistics\\NatHealth2008MI.Rdata")
attach(NatHealth2008MI)
```

- Make one final review of your data

```
Summary(NatHealth2008MI)
```

Multiple Imputation

- Amelia Imputation Options:
  - "idvars" – specifies identification variables that you want to keep in your dataset, but are not part of the imputation model
  - "noms" tells R which variables are nominal
  - "ords" tells R which variables are ordinal
  - "ts" used to signify time series variables
  - "cs" used to signify cross-sectional variables

Multiple Imputation

- Now, let’s create our imputed datasets

```
NatHealth.MI <- amelia(NatHealth2008MI, m=5,
   idvars=c("id“))
```

- How many datasets did we create?
- What is idvars option doing here?

Multiple Imputation

- Now let's review the results our MI
- Compare observed to imputed values

  `plot(NatHealth.MI, which.vars=9:12)`

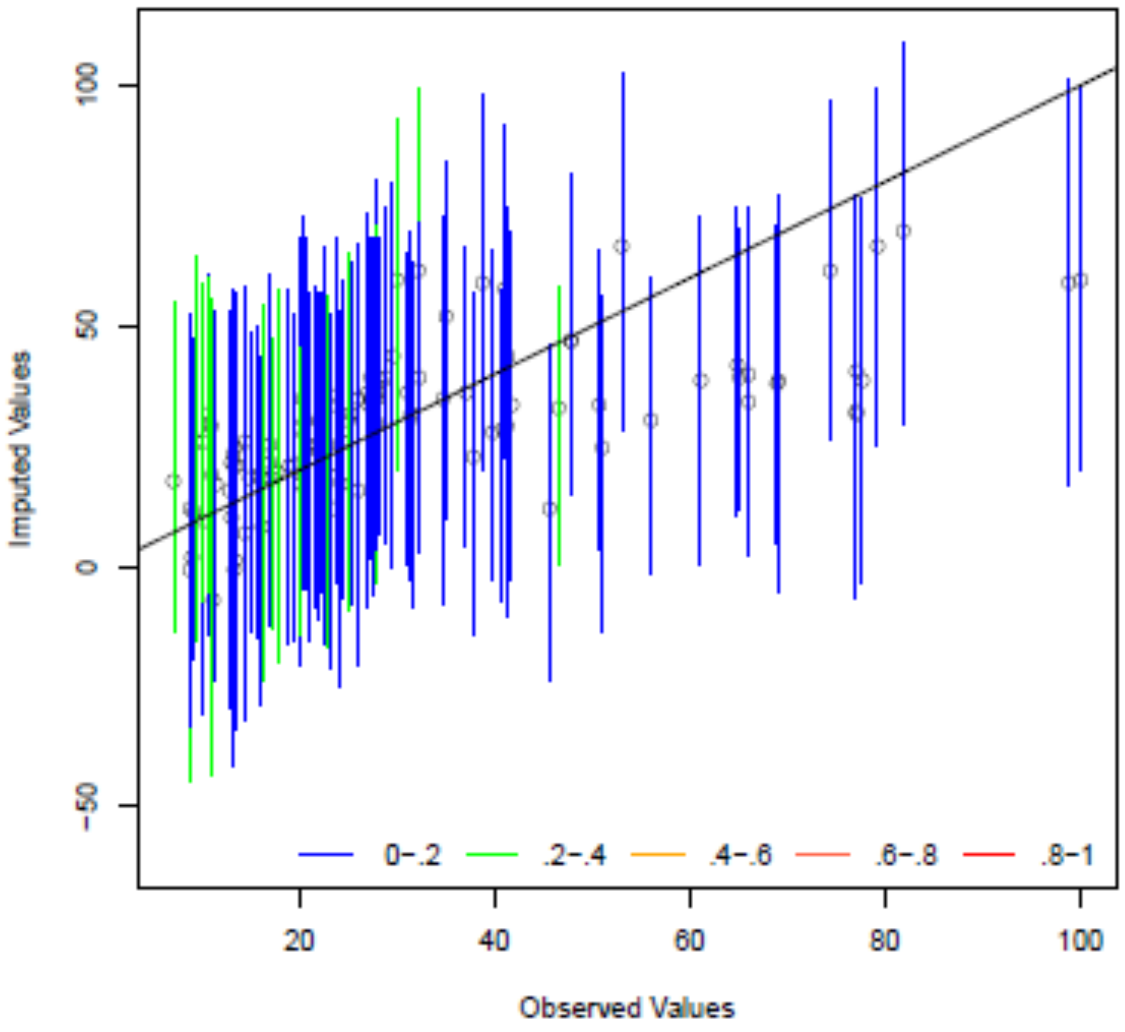
  - We're asking for plots of variables 9-12
  - Red line = imputed values
  - Black line = observed values
  - If variable is fully observed, it is plotted in blue

Multiple Imputation

- We can also overimpute variables to gauge quality of our imputation
  - Treats every OBSERVED value as if it was missing
  - Then, impute many values for that observed value and view confidence intervals of these estimates
  - Examine whether confidence intervals overlap the "true" value on the x=y line
  - Circles represent the mean imputed value

```
overimpute(NatHealth.MI, var="beddayr")
```

Multiple Imputation



Multiple Imputation

- Saving imputed data as one dataset

  save(NatHealth.MI, file=" S:\\DataClass\\RandStatistics \\NatHealthImputed.Rdata")

- Save datasets separately

  write.amelia(obj=NatHealth.MI, file.stem=" S:\\DataClass\\RandStatistics \\NatHealthImputed")

Multiple Imputation

- Combining MI results

```
miWKDAYR <- zelig(wkdayr~cigsday+modmin+sleep,
    model="ls",
data = mi(NatHealthImputed1, NatHealthImputed2,
NatHealthImputed3, NatHealthImputed4,
    NatHealthImputed5))

summary(miWKDAYR)
```

Combining MI Results

```
zelig(formula = wkdayr ~ cigsday + modmin + sleep, model = "ls",
    data = mi(NatHealthImputed1, NatHealthImputed2, NatHealthImputed3,
        NatHealthImputed4, NatHealthImputed5))

Coefficients:
                   Value Std. Error      t-stat   p-value
(Intercept) -10.967779007 7.79393642 -1.40721946 0.1888879
cigsday       0.005148175 0.15353988  0.03352989 0.9732959
modmin       -0.034469722 0.02450667 -1.40654426 0.1597445
sleep         2.509761238 1.15469415  2.17352902 0.0602578

For combined results from datasets i to j, use summary(x, subset = i:j).
For separate results, use print(summary(x), subset = i:j).
```

Questions?

Programming Statistical Models in Zelig

When: Jan 17th – 20th, 2012

Where: Knafel Building, CGIS (This building)

TOPICS INCLUDE:

- Developing Statistical Packages in the R programming language
- Leveraging Zelig to Simplify Statistical Software Distribution and Development
- Programming Statistical Simulation
- Publishing Statistical Packages

Register Here:

hvrd.me/Zelig_Jan_2012

Thanks!

- More questions? Email: mowen@iq.harvard.edu
- More Information about R: http://cran.r-project.org/
- For more information about R and Zelig: http://gking.harvard.edu/zelig/
- More lecture series are available!
- R Developers course starting this Winter. Register here: hvrd.me/Zelig_Jan_2012