Exploring data #1

Data from R packages

Data from R packages

So far you have gotten data to use in R three ways:

1. From flat files (either on your computer or online)
2. From files like SAS and Excel
3. From R objects (i.e., using `load()`)

Many R packages come with their own data, which is very easy to load and use.

Data from R packages

For example, the `faraway` package has a dataset called `worldcup` that you'll use today. To load it, use the `data()` function once you've loaded the package:

```
library(faraway)
data("worldcup")
```

Data from R packages

Unlike most data objects you'll work with, the data that comes with an R package will often have its own help file. You can access this using the ? operator:

```
?worldcup
```

Data from R packages

To find out all the datasets that are available in the packages you currently have loaded, run `data()` without an option inside the parentheses:

```
data()
```

As a note, you can similarly use `library()`, without the name of a package, to list all of the packages you have installed that you could call with `library()`:

```
library()
```

Plots to explore data

Plots to explore data

Plots can be invaluable in exploring your data.

This week, we will focus on **useful**, rather than **attractive** graphs, since we are focusing on exploring rather than presenting data.

Next week, we will talk more about making more attractive plots that would go into final reports.

Plots to explore data

We will look at the following types of plots to explore data today:

R function	Description
`hist()`	Histogram (one numeric variable)
`plot()`	Scatter plot (two [usually] numeric variables)
`pairs()`	Pairwise scatter plots (2+ [usually] numeric variables)
`boxplot()`	Boxplot (one numeric variable, possibly stratified by one factor variable)

Useful plot options

The following options will work for all or most of these plotting functions:

Option	Description
`main =`	The main title for the graph
`xlab =`, `ylab =`	Labels for the x- and y-axes
`xlim =`, `ylim =`	Limits for the x- and y-axes (a vector of length 2 giving the minimum and maximum values)
`col =`	Color of a plotting element
`cex =`	Size of points on plot (>1 for larger, <1 for smaller)

Example plots

For the example plots, I'll use a dataset in the `faraway` package called `nepali`. This gives data from a study of the health of a group of Nepalese children.

```
library(faraway)
data(nepali)
```

I'll be using functions from `dplyr` and `ggplot2`:

```
library(dplyr)
library(ggplot2)
```

Example plots

Each observation is a single measurement for a child; there can be multiple observations per child. I'll subset out child id, sex, weight, height, and age, and I'll limit to each child's first measurement.

```
nepali <- nepali %>%
  # Subset to certain columns
  select(id, sex, wt, ht, age) %>%
  # Convert id and sex to factors
  mutate(id = factor(id),
         sex = factor(sex)) %>%
  # Limit to first obs. per child
  filter(!duplicated(id))
```

nepali EXAMPLE DATA

The data now looks like:

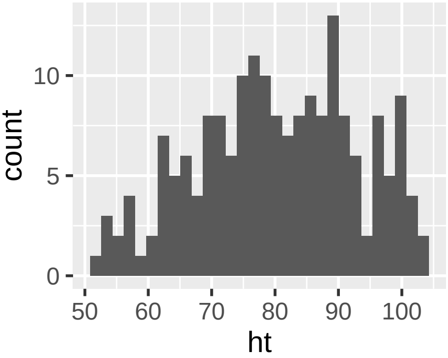
```
head(nepali)
```

```
##        id sex   wt    ht age
## 1 120011   1 12.8  91.2  41
## 2 120012   2 14.9 103.9  57
## 3 120021   2  7.7  70.1   8
## 4 120022   2 12.1  86.4  35
## 5 120023   1 14.2  99.4  49
## 6 120031   1 13.9  96.4  46
```

hist() example

For `hist()`, the main argument is the (numeric) vector for which you want to create a histogram:

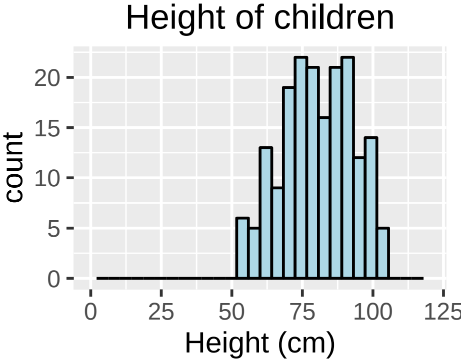
```
ggplot(nepali, aes(x = ht)) +
  geom_histogram()
```



hist() example

You can try out some of the options on the histogram, like `main`, `xlab`, `xlim`, and `col`:

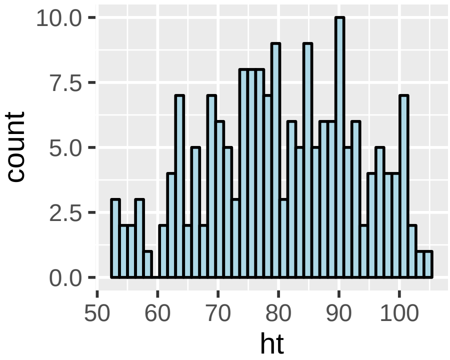
```
ggplot(nepali, aes(x = ht)) +
  geom_histogram(fill = "lightblue", color = "black") +
  ggtitle("Height of children") +
  xlab("Height (cm)") + xlim(c(0, 120))
```



hist() example

`geom_histogram` also has its own special argument, `bins`. You can use this to change the number of bins that are used to make the histogram:

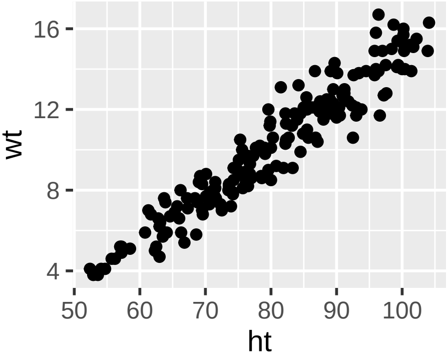
```
ggplot(nepali, aes(x = ht)) +
  geom_histogram(fill = "lightblue", color = "black",
                 bins = 40)
```



plot() example

The main arguments for the `plot()` function are the vectors to plot on the x- and y-axes. You can use `x=` and `y=` to specify these, or you can just put them in the correct order into the function (x value first, y value second):

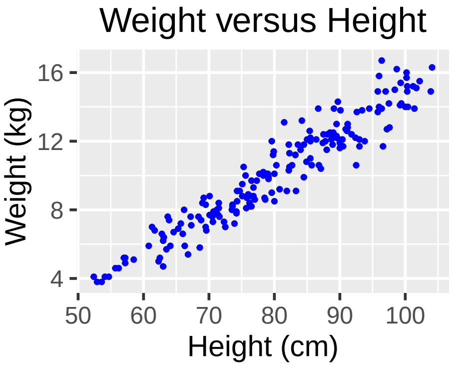
```
ggplot(nepali, aes(x = ht, y = wt)) +
  geom_point()
```



plot() example

Again, you can use some of the options to change the plot appearance:

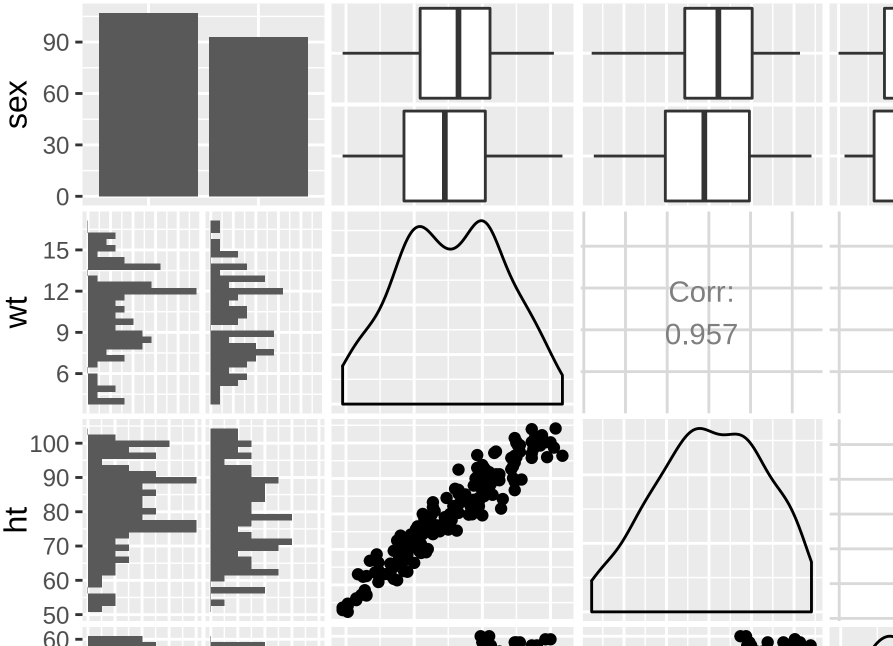
```
ggplot(nepali, aes(x = ht, y = wt)) +
  geom_point(color = "blue", size = 0.5) +
  ggtitle("Weight versus Height") +
  xlab("Height (cm)") + ylab("Weight (kg)")
```



ggpairs() example

You can use the `ggpairs` function from the `GGally` package to plot all pairs of scatterplots for several variables.
Usually, you just do this for numeric variables. However, you could also include factors or logical variables, since R saves them, underneath the form you see, as a numeric value (e.g `TRUE` = 1). The argument is a matrix with the columns to plot.
The next slide shows the output for:

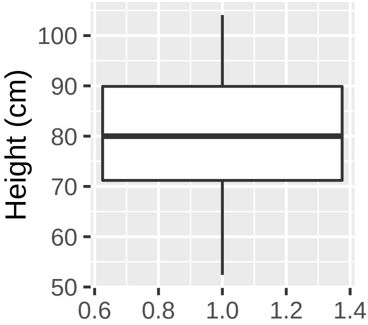
```
library(GGally)
ggpairs(nepali[, c("sex", "wt", "ht", "age")])
```


sex
wt
ht
90
60
30
0
15
12
9
6
100
90
80
70
60
50
60
Corr:
0.957

boxplot() example

The main argument of `boxplot()` is the (numeric) vector you want to plot:

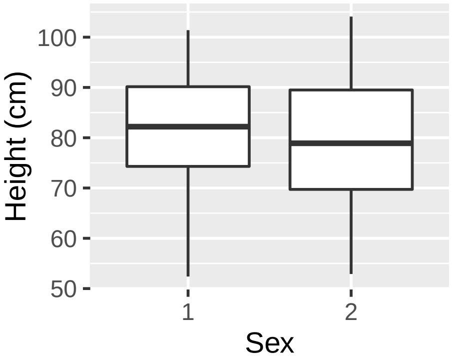
```
ggplot(nepali, aes(x = 1, y = ht)) +
  geom_boxplot() +
  xlab("")+ ylab("Height (cm)")
```



boxplot() example

You can also do separate boxplots by a factor. This call uses the formula notation we'll talk about in a few minutes, when we talk about regression models:

```
ggplot(nepali, aes(x = sex, y = ht)) +
  geom_boxplot() +
  xlab("Sex")+ ylab("Height (cm)")
```



with() digression

with() digression

You'll notice that, in some of these functions, you have to type out the dataframe's object name more than once. You can use the `with()` function to save a bit of time.
`with()` wraps around another function, and it tells the function it's wrapping to look inside a certain dataframe first when it's looking for objects. For example:

```
summary(nepali$wt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    3.80    7.90   10.10   10.18   12.40   16.70 
##    NA's 
##      15
```

```
with(nepali, summary(wt))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    3.80    7.90   10.10   10.18   12.40   16.70 
##    NA's 
##      15
```

`with()` digression

Some of you may have heard of `attach()` and `detach()` as an alternative to this. You can use these to attach a dataframe, so R looks there first for objects for any code until you detach the dataframe. Most heavy-duty R users don't use `attach()` and `detach()`. They can create confusion or errors in your code. I would recommend using `with` in cases when it will save you time, but not `attach` or `detach`.

with() digression

From Google's style page for R:

> *"The possibilities for creating errors when using attach are numerous. Avoid it."*

Simple statistics to explore data

Simple statistics functions

Here are some simple statistics functions you will likely use often:

Function	Description
`range()`	Range (minimum and maximum) of vector
`min()`, `max()`	Minimum or maximum of vector
`mean()`, `median()`	Mean or median of vector
`table()`	Number of observations per level for a factor vector
`cor()`	Determine correlation(s) between two or more vectors
`summary()`	Summary statistics, depends on class

Simple statistic examples

All of these take, as the main argument, the vector(s) for which you want the statistic. If there are missing values in the vector, you'll need to add an option to say what to do when them (e.g., `na.rm` or `use="complete.obs"`).

```
mean(nepali$wt, na.rm = TRUE)
```

```
## [1] 10.18432
```

```
range(nepali$ht, na.rm = TRUE)
```

```
## [1]  52.4 104.1
```

```
table(nepali$sex)
```

```
##
##   1   2
## 107  93
```

Simple statistic examples

The cor function can take two or more vectors. If you give it multiple values, it will give the correlation matrix for all the vectors.

```r
cor(nepali$wt, nepali$ht, use = "complete.obs")
```

```
## [1] 0.9571535
```

```r
cor(nepali[ , c("wt", "ht", "age")], use = "complete.obs")
```

```
##            wt        ht       age
## wt  1.0000000 0.9571535 0.8931195
## ht  0.9571535 1.0000000 0.9287129
## age 0.8931195 0.9287129 1.0000000
```

summary(): A bit of OOP

R supports object-oriented programming. This shows up with summary(). R looks to see what type of object it's dealing with, and then uses a method specific to that object type.

```
summary(nepali$wt)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     3.80    7.90   10.10   10.18   12.40   16.70
##     NA's
##       15
```

```
summary(nepali$sex)
```

```
##   1   2
## 107  93
```

We'll see more of this when we do regression models.

A bit more on logical statements

Logical statements

Last week, you learned a lot about logical statements and how to use them with the `subset()` function.
You can also use logical vectors, created with these statements, for a lot of other things. For example, you can use them directly in the square bracket indexing (`[..., ...]`).

Logical vectors

A logical statement run on a vector will create a logical vector the same length as the original vector:

```
is_male <- nepali$sex == "male"
length(nepali$sex)
```

```
## [1] 200
```

```
length(is_male)
```

```
## [1] 200
```

Logical vectors

The logical vector will have the value `TRUE` at any position where the original vector met the logical condition you tested, and `FALSE` anywhere else:

```
head(nepali$sex)
```

```
## [1] 1 2 2 2 1 1
## Levels: 1 2
```

```
head(is_male)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

Logical vectors

You can "flip" this logical vector (i.e., change every TRUE to FALSE and vice-versa) using !:

```
head(is_male)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
head(!is_male)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

Logical vectors

You can do a few cool things now with this vector. For example, you can use it with indexing to pull out just the rows where `is_male` is `TRUE`:

```
head(nepali[is_male, ])
```

```
## [1] id  sex wt  ht  age
## <0 rows> (or 0-length row.names)
```

Logical vectors

Or, with `!`, just the rows where `is_male` is `FALSE`:

```
head(nepali[!is_male, ])
```

```
##       id sex   wt    ht age
## 1 120011   1 12.8  91.2  41
## 2 120012   2 14.9 103.9  57
## 3 120021   2  7.7  70.1   8
## 4 120022   2 12.1  86.4  35
## 5 120023   1 14.2  99.4  49
## 6 120031   1 13.9  96.4  46
```

Logical vectors

You can also use `sum()` and `table()` to find out how many males and females are in the dataset:

```
sum(is_male); sum(!is_male)
```

```
## [1] 0
```

```
## [1] 200
```

```
table(is_male)
```

```
## is_male
## FALSE
##   200
```

Using regression models to explore data

Formula structure

In R, formulas take this structure:

```
[response variable] ~ [indep. var. 1] +  [indep. var. 2] + ...
```

Notice that ~ used to separate the independent and dependent variables and the + used to join independent variables.
You will use this type of structure in a lot of different function calls, including those for linear models (`lm`) and generalized linear models (`glm`).

Linear models

To fit a linear model, you can use the function `lm()`. Use the `data` option to specify the dataframe from which to get the vectors. You can save the model as an object.

```
mod_a <- lm(wt ~ ht, data = nepali)
```

This call fits the model:

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \epsilon_i$$

where:

- Y_i : weight of child i
- $X_{1,i}$: height of child i

Using model objects

Some functions you can use on model objects:

Function	Description
`summary()`	Get a variety of information on the model, including coefficients and p-values for the coefficients
`coef()`	Pull out just the coefficients for a model
`residuals()`	Get the model residuals
`fitted()`	Get the fitted values from the model (for the data used to fit

Examples of using a model object

For example, you can get the coefficients from the model we just fit:

```
coef(mod_a)
```

```
## (Intercept)          ht
##   -8.694768    0.235050
```

You can also pull out the residuals:

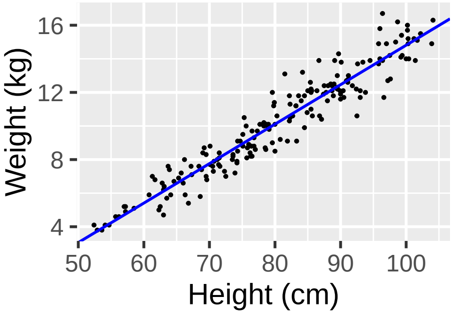
```
head(residuals(mod_a))
```

```
##           1           2           3           4
##  0.05820415 -0.82693141 -0.08223993  0.48644436
##           5           6
## -0.46920621 -0.06405608
```

Examples of using a model object

You can also plot the data you used to fit the model and add a regression based on the model fit, using abline():

```
mod_coef <- coef(mod_a)
ggplot(nepali, aes(x = ht, y = wt)) +
  geom_point(size = 0.2) +
  xlab("Height (cm)") + ylab("Weight (kg)") +
  geom_abline(aes(intercept = mod_coef[1],
                  slope = mod_coef[2]), col = "blue")
```



Examples of using a model object

The `summary()` function gives you a lot of information about the model:

```
summary(mod_a)
```

(see next slide)

```
##
## Call:
## lm(formula = wt ~ ht, data = nepali)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.44736 -0.55708  0.01925  0.49941  2.73594
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.694768   0.427398  -20.34   <2e-16
## ht           0.235050   0.005257   44.71   <2e-16
##
## (Intercept) ***
## ht          ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9017 on 183 degrees of freedom
##   (15 observations deleted due to missingness)
```

Summary for lm objects

The object created when you use the `summary()` function on an `lm` object has several different parts you can pull out using the `$` operator:

```
names(summary(mod_a))
```

```
##  [1] "call"          "terms"
##  [3] "residuals"     "coefficients"
##  [5] "aliased"       "sigma"
##  [7] "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"
## [11] "cov.unscaled"  "na.action"
```

```
summary(mod_a)$coefficients
```

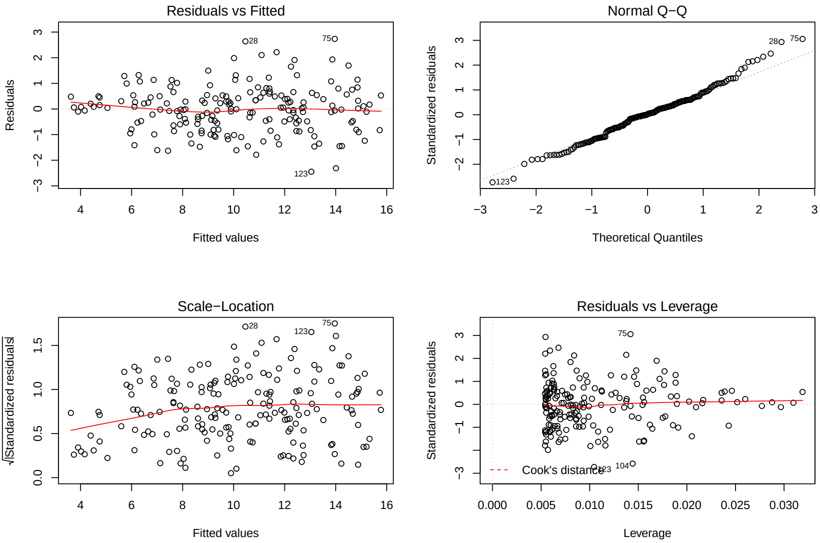
```
##              Estimate  Std. Error   t value
## (Intercept) -8.694768 0.427397843 -20.34350
## ht           0.235050 0.005256822  44.71334
##                 Pr(>|t|)
## (Intercept)  7.424640e-49
## ht          1.962647e-100
```

Using plot() with lm objects

You can use `plot` with an `lm` object to get a number of useful diagnostic plots to check regression assumptions:

```
plot(mod_a)
```

(See next slide)


Residuals vs Fitted
Residuals
Fitted values
28
75
123
Normal Q−Q
Standardized residuals
Theoretical Quantiles
Scale−Location
√|Standardized residuals|
Fitted values
Residuals vs Leverage
Standardized residuals
Leverage
Cook's distance
104

Fitting a model with a factor

You can also use binary variables or factors as independent variables in regression models:

```
mod_b <- lm(wt ~ sex, data = nepali)
summary(mod_b)$coefficients
```

```
##               Estimate Std. Error   t value
## (Intercept) 10.497980  0.3110957 33.745177
## sex2        -0.674724  0.4562792 -1.478752
##                 Pr(>|t|)
## (Intercept) 1.704550e-80
## sex2        1.409257e-01
```

This call fits the model:

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \epsilon_i$$

where $X_{1,i}$: sex of child i, where $0 =$ male; $1 =$ female

Linear models versus GLMs

You can fit a variety of models, including linear models, logistic models, and Poisson models, using generalized linear models (GLMs).

For linear models, the only difference between `lm` and `glm` is how they're fitting the model (least squares versus maximum likelihood). You should get the same results regardless of which you pick.

Linear models versus GLMs

For example:

```
mod_c <- glm(wt ~ ht, data = nepali)
summary(mod_c)$coef
```

```
##              Estimate  Std. Error   t value
## (Intercept) -8.694768 0.427397843 -20.34350
## ht           0.235050 0.005256822  44.71334
##                  Pr(>|t|)
## (Intercept)  7.424640e-49
## ht          1.962647e-100
```

```
summary(mod_a)$coef
```

```
##              Estimate  Std. Error   t value
## (Intercept) -8.694768 0.427397843 -20.34350
## ht           0.235050 0.005256822  44.71334
##                  Pr(>|t|)
## (Intercept)  7.424640e-49
## ht          1.962647e-100
```

GLMs

You can fit other model types with `glm()` using the `family` option:

Model type	`family` option
Linear	`family = gaussian(link = "identity")`
Logistic	`family = binomial(link = "logit")`
Poisson	`family = poisson(link = "log")`

Logistic example

For example, say we wanted to fit a logistic regression for the `nepali` data of whether the probability that a child weighs more than 13 kg is associated with the child's height.

First, create a binary variable for `wt_over_13`:

```
nepali <- nepali %>%
  mutate(wt_over_13 = wt > 13)
head(nepali)
```

```
##        id sex   wt    ht age wt_over_13
## 1 120011   1 12.8  91.2  41      FALSE
## 2 120012   2 14.9 103.9  57       TRUE
## 3 120021   2  7.7  70.1   8      FALSE
## 4 120022   2 12.1  86.4  35      FALSE
## 5 120023   1 14.2  99.4  49       TRUE
## 6 120031   1 13.9  96.4  46       TRUE
```

Logistic example

Now you can fit a logistic regression:

```
mod_d <- glm(wt_over_13 ~ ht, data = nepali,
             family = binomial(link = "logit"))
summary(mod_d)$coef
```

```
##                Estimate Std. Error   z value
## (Intercept) -32.7016520 5.85196755 -5.588147
## ht            0.3495227 0.06331892  5.520036
##                 Pr(>|z|)
## (Intercept) 2.295060e-08
## ht          3.389307e-08
```

Here, the model coefficient gives the **log odds** of having a weight higher than 13 kg associated with a unit increase in height.

Formula structure

Here are some conventions used in R formulas:

Convention	Meaning
`I()`	calculate the value inside before fitting (e.g., `I(x1 + x2)`)
`:`	fit the interaction between two variables (e.g., `x1:x2`)
`*`	fit the main effects and interaction for both variables (e.g., `x1*x2` equals `x1 + x2 + x1:x2`)
`.`	fit all variables other than the response (e.g., `y ~ .`)
`-`	do not include a variable (e.g., `y ~ . - x1`)
`1`	intercept (e.g., `y ~ 1`)

To find out more

A great (and free-to-you) resource to find out more about using R for basic statistics:

- Introductory Statistics with R

If you want all the details about fitting linear models and GLMs in R, Faraway's books are fantastic:

- Linear Models with R
- Extending the Linear Model with R