# Getting Started with R

*Thomas E. Love*

*2017-09-05*

## Contents



### 0.1 Link to Markdown file

The link to the plain text R Markdown version of this document is at our data and code page

# 1 Purpose of this Document

This is meant to walk you through the steps of completing some elementary analyses using R, R Studio, and R Markdown. Working through this document will definitely help you get rolling on Assignment 1.

## 1.1 Preliminary Steps

1. Follow the instructions to install R, R Studio and the R packages and data and code that we'll use in 431. Once you have these things installed, you're ready for the steps below.
2. Select or create a subdirectory on your computer for your project. We'll call that your **project folder**.
    - Don't use the same subdirectory/folder for multiple projects. You'll have many projects this semester.
    - A good directory path might be something like `pqhs431/2017-08-29_my-first-R-project`
    - You won't believe how important it is for you to understand where your files are and a well-designed naming scheme is an enormous time-saver. "A place for everything, and everything in its place" and all that...
3. Put any data you plan to import into R, and a copy of the `431-r-template.Rmd` file into your project folder.
4. Start R Studio and begin by opening up a **project**.
    - Use **File ... New Project** to create a new project in your **project folder**.
        - Create the R project in the main **project folder**, regardless of whether you've placed the data in that same folder, or in a data subdirectory of that folder.
        - One and only one R Project per project folder is the way to happiness.
    - When not doing this for the first time, use **File ... Open Project** to open an R Studio Project you've already created.

## 1.2 Use the template

A R Markdown file is just a plain text document, with interspersed R code that lets you produce reports that combine narration with results, and that can be easily exported as an HTML, Word or PDF file. It's a great tool. Dr. Love builds virtually everything you'll see in this class with R Markdown. R Markdown files use the `.Rmd` extension.

Open the template file `431-r-template.Rmd` by clicking on it in the **Files** tab on the lower right of your R Studio setup, or selecting **File ... Open** from the main menu. + The start of the template file is a top-line set of instructions to R Markdown about how to process the rest of the document. It is referred to as the YAML material, and looks like this:

```
---
title: "R Markdown Template"
author: "Your Name"
date: "2017-09-05"
output:
  html_document:
    toc: yes
    code_folding: show
---
```

Now, edit the file to include a meaningful Title for this work, and place your actual name in the author section. Then use **File ... Save as** to save the Markdown file under a new project-specific name, rather than the generic `431-r-template.Rmd`. Your result should look something like this:

```
---
title: "My extremely exciting first data analysis"
```

```
author: "Chris Traeger"
date: "2017-09-05"
output:
  html_document:
    toc: yes
    code_folding: show
---
```

In most cases, changing only the **title**, **author** and **date**, but otherwise leaving this as is, will work well for our purposes.

To learn more about using R Markdown, we recommend working through some of the tutorials at http://rmarkdown.rstudio.com/lesson-1.html

## 2 Loading the Packages

To begin, we'll load the packages (libraries) and set up options that we will use in our analyses.

```
knitr::opts_chunk$set(comment=NA)

library(magrittr); library(tidyverse)
```

## 3 The chickwts study

The `chickwts` data, available as part of the base installation of R (in the `datasets` package) describe an experiment conducted to measure and compare the effectiveness of various feed supplements on the growth rate of chickens. For more on the `chickwts` data, type `?(chickwts)` into the R console.

We'll begin by placing the data in a tibble called `chick`.

```
chick <- tbl_df(chickwts)
chick
```

```
# A tibble: 71 x 2
   weight      feed
    <dbl>    <fctr>
 1    179 horsebean
 2    160 horsebean
 3    136 horsebean
 4    227 horsebean
 5    217 horsebean
 6    168 horsebean
 7    108 horsebean
 8    124 horsebean
 9    143 horsebean
10    140 horsebean
# ... with 61 more rows
```

- The `weight` variable is numeric (double-precision) and gives the chick's weight.
- The `feed` variable is categorical (a factor in R) and gives the feed type.

## 3.1 Summarizing the distribution of a categorical variable, `feed`

The regular `summary` function can provide some useful results.

```
chick %>%
    select(feed) %>%
    summary()
```

```
        feed
 casein   :12
 horsebean:10
 linseed  :12
 meatmeal :11
 soybean  :14
 sunflower:12
```

There are lots of ways to generate a table for a factor (categorical variable) like this.

```
chick %>%
  select(feed) %>%
    table() %>%
    addmargins()
```

```
.
   casein horsebean   linseed  meatmeal   soybean sunflower       Sum
       12        10        12        11        14        12        71
```

## 3.2 Describing the distribution of a continuous variable, `weight`, numerically

The regular `summary` function provides a five-number summary, plus the mean. We can do this with ...

```
chick %>%
    select(weight) %>%
    summary()
```

```
     weight
 Min.   :108.0
 1st Qu.:204.5
 Median :258.0
 Mean   :261.3
 3rd Qu.:323.5
 Max.   :423.0
```

Or, we can use a different pipe than the usual `%>%` - here `%$%` exposes the pieces of the `chick` tibble (the variables) to the function `summary()`.

```
chick %$% summary(weight)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  108.0   204.5   258.0   261.3   323.5   423.0
```

The `favstats` function from the `mosaic` package produces a more extensive set of numerical summaries. Here, we are forced to use the new pipe `%$%` to identify the variables for the function `favstats` in `mosaic`.

```
chick %$%
    mosaic::favstats(weight)
```

```
 min    Q1 median    Q3 max     mean       sd  n missing
```

```
 108 204.5    258 323.5 423 261.3099 78.0737 71       0
```

Another way to accomplish the same end is

```
mosaic::favstats(chick$weight)
```

```
 min    Q1 median    Q3 max     mean      sd  n missing
 108 204.5    258 323.5 423 261.3099 78.0737 71       0
```

Here is a smaller numerical summary of the weights broken down by feed category.

```
chick %>%
  group_by(feed) %>%
  summarize(mean(weight), sd(weight), median(weight))
```

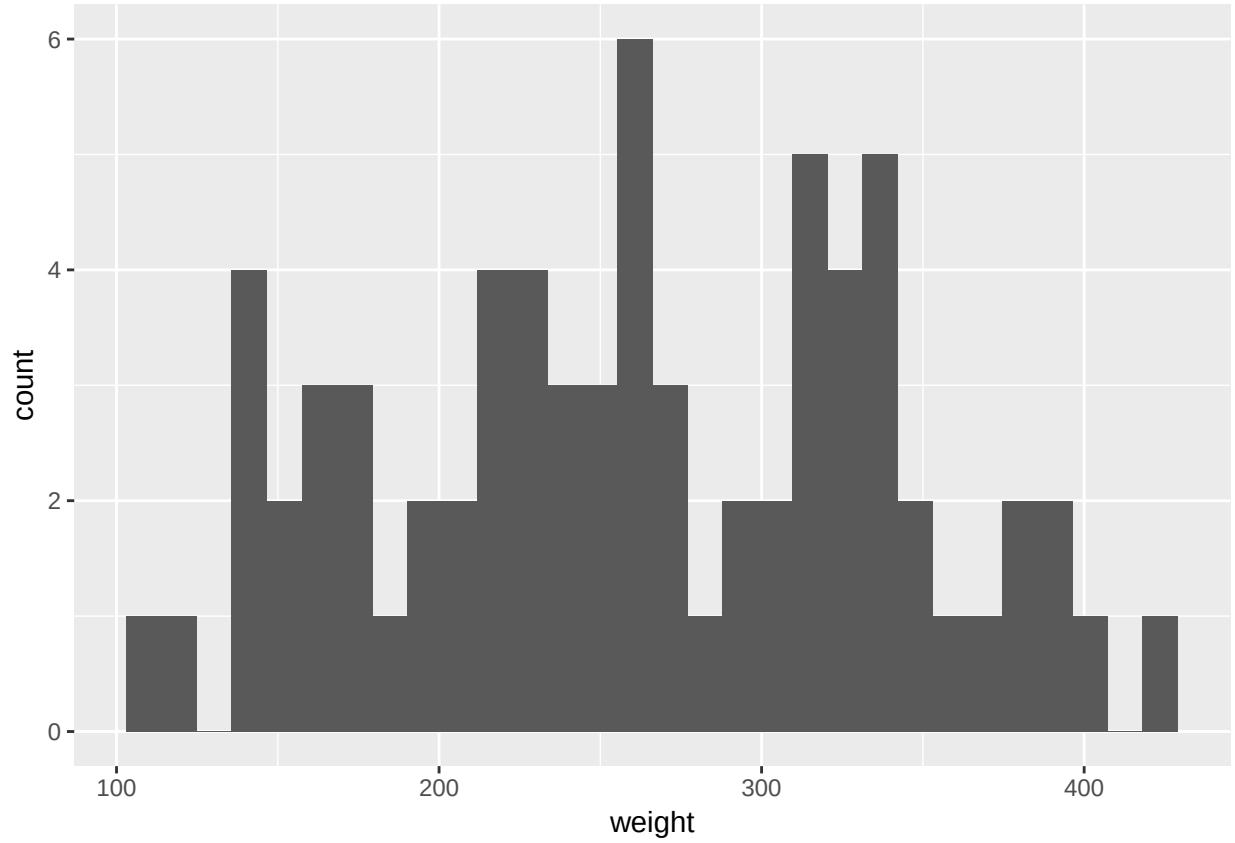
```
# A tibble: 6 x 4
       feed `mean(weight)` `sd(weight)` `median(weight)`
     <fctr>          <dbl>        <dbl>            <dbl>
1    casein       323.5833     64.43384            342.0
2 horsebean       160.2000     38.62584            151.5
3   linseed       218.7500     52.23570            221.0
4  meatmeal       276.9091     64.90062            263.0
5   soybean       246.4286     54.12907            248.0
6 sunflower       328.9167     48.83638            328.0
```

### 3.3 Drawing an attractive histogram of the `weight` data

Here is the default approach.

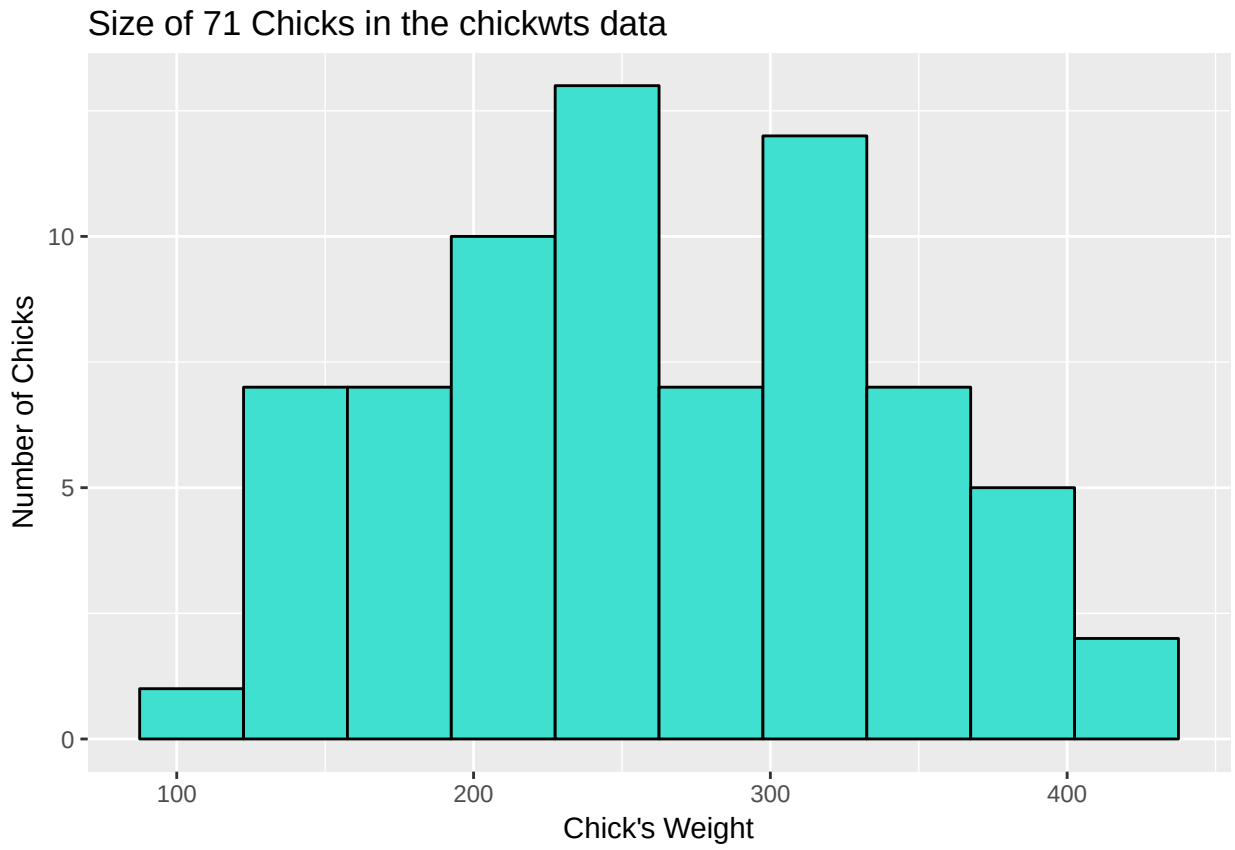
```
ggplot(chick, aes(x = weight)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



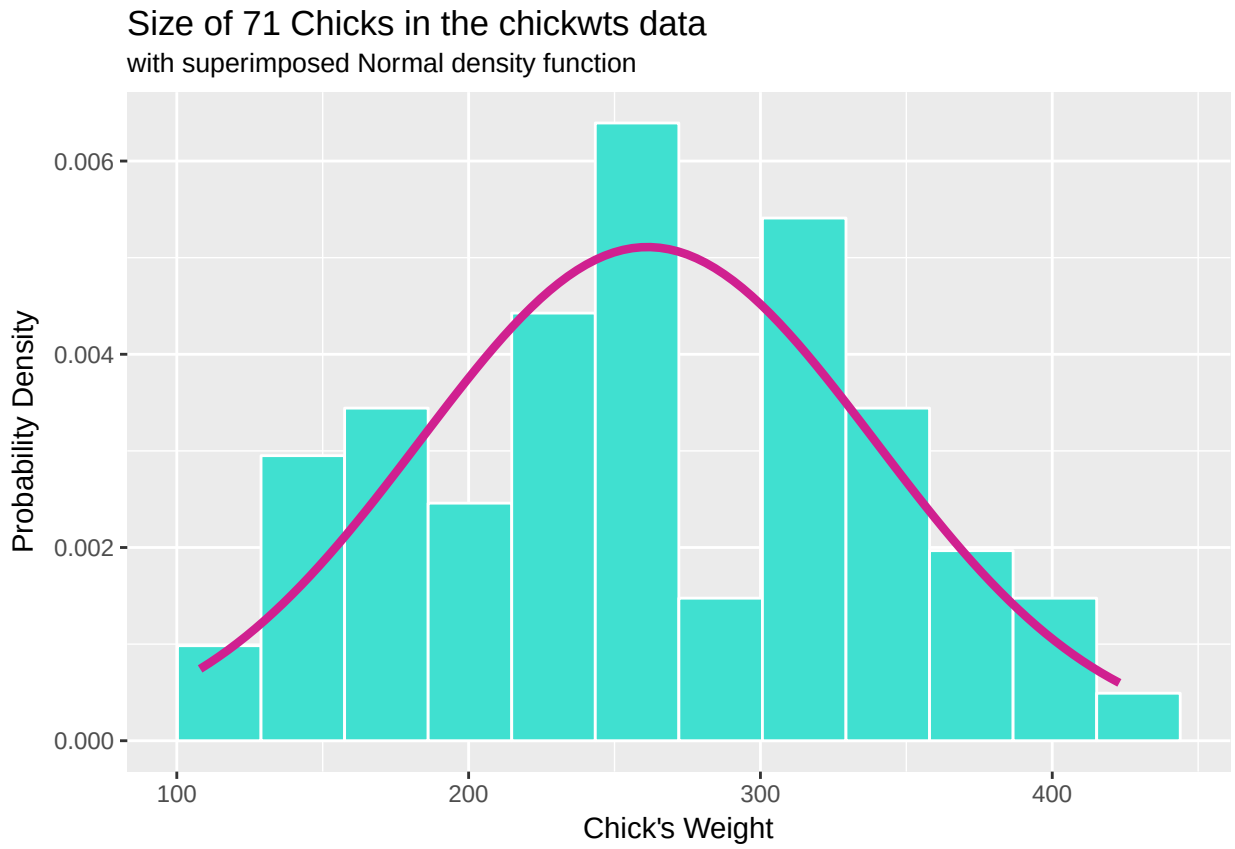
Let's make that slightly more attractive, revise the labels, and place a title.

```
ggplot(chick, aes(x = weight)) +
  geom_histogram(bins = 10, color = "black", fill = "turquoise") +
  labs(x = "Chick's Weight", y = "Number of Chicks",
       title = "Size of 71 Chicks in the chickwts data")
```



Another option would be to plot the density function, rather than the raw counts, and compare it directly to what we would expect from a Normal model with the same mean and standard deviation as the weights in the `chick` data.

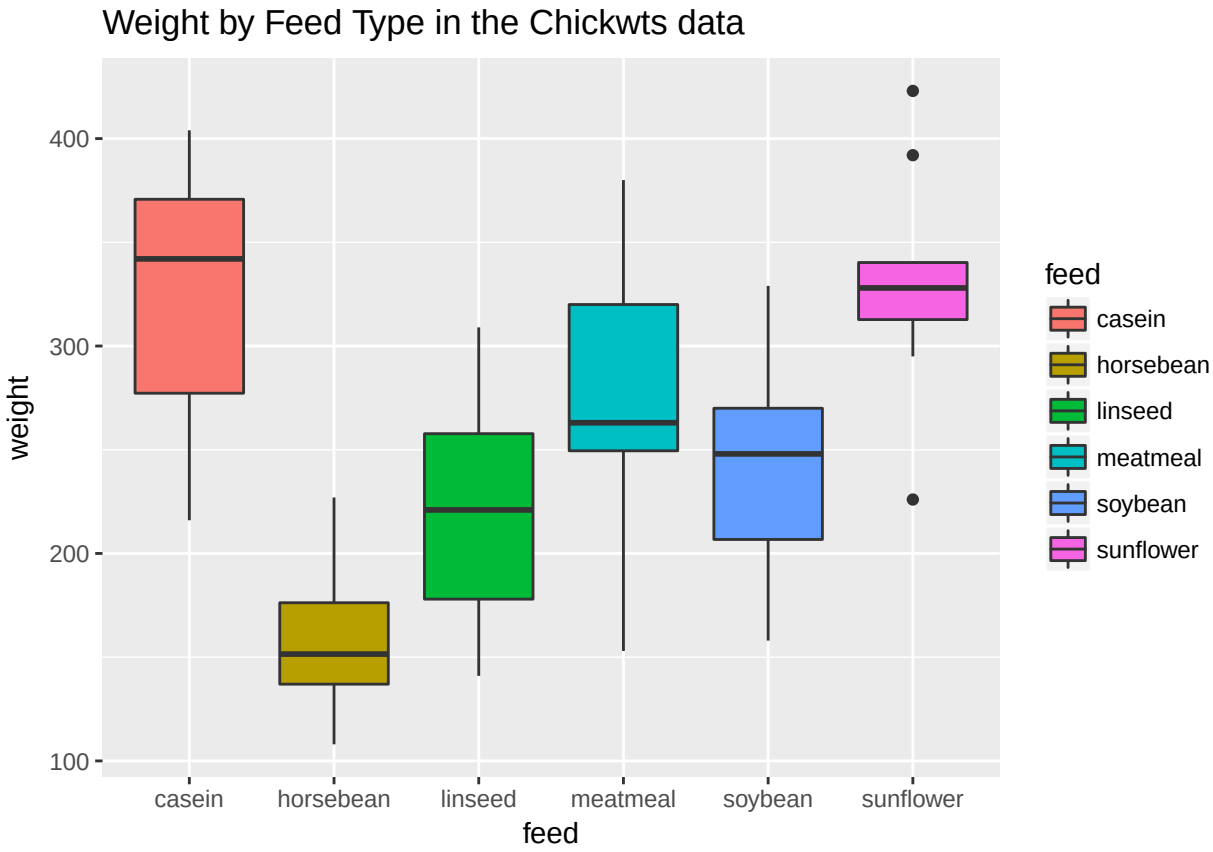
```
ggplot(chick, aes(x = weight)) +
  geom_histogram(aes(y = ..density..), bins = 12, color = "white", fill = "turquoise") +
  stat_function(fun = dnorm,
                args = list(mean = mean(chick$weight), sd = sd(chick$weight)),
                lwd = 1.5, col = "violetred") +
  labs(x = "Chick's Weight", y = "Probability Density",
       title = "Size of 71 Chicks in the chickwts data",
       subtitle = "with superimposed Normal density function")
```



### 3.4 Drawing a Boxplot of the Weights by Feed Type

A boxplot might, for instance, compare the weight distributions for each of the various types of feed.

```
ggplot(chick, aes(x = feed, y = weight, fill = feed)) +
    geom_boxplot() +
    labs(title = "Weight by Feed Type in the Chickwts data")
```




We could drop the labels on the right hand side by adding `guides(fill = FALSE) +` somewhere in our `ggplot` call.

### 3.5 Drawing a Normal Q-Q plot of the Weights

A Normal Q-Q plot of the weights is probably most easily obtained using base graphics, rather than ggplot. For example,

```
qqnorm(chick$weight, main = "Normal Q-Q plot of Chick Weights")
qqline(chick$weight, col = "red")
```

# 4 The Orange Study

The Orange data frame has 35 rows and 3 columns of records of the growth of orange trees. Let's get the data into a tibble.

```
orange <- tbl_df(Orange)
orange
```

```
# A tibble: 35 x 3
    Tree   age circumference
 * <ord> <dbl>         <dbl>
 1     1   118            30
 2     1   484            58
 3     1   664            87
 4     1  1004           115
 5     1  1231           120
 6     1  1372           142
 7     1  1582           145
 8     2   118            33
 9     2   484            69
10     2   664           111
# ... with 25 more rows
```

- `tree` is an ordinal factor, which indicates the tree on which the measurement was made. The ordering is by increaing maximum diameter of the five trees.
- `age` is a numerical variable, containing the age of the tree as measured in days since 1968-12-31.
- `circumference` is a numerical variable, containing the trunk circumference (probably at "breast height") in mm.

## 4.1 Numerical Summary

And here's the standard numerical summary for the full data set.

```
summary(orange)
```

```
 Tree       age         circumference
 3:7   Min.   : 118.0   Min.   : 30.0
 1:7   1st Qu.: 484.0   1st Qu.: 65.5
 5:7   Median :1004.0   Median :115.0
 2:7   Mean   : 922.1   Mean   :115.9
 4:7   3rd Qu.:1372.0   3rd Qu.:161.5
       Max.   :1582.0   Max.   :214.0
```

Next, we'll look at the mean age and circumference, within each of the seven measurements per tree.

```
orange %>%
  group_by(Tree) %>%
  summarize(mean(age), mean(circumference))
```

```
# A tibble: 5 x 3
   Tree `mean(age)` `mean(circumference)`
  <ord>       <dbl>                 <dbl>
1     3    922.1429              94.00000
2     1    922.1429              99.57143
3     5    922.1429             111.14286
4     2    922.1429             135.28571
```

```
5     4    922.1429             139.28571
```

It looks like each of the trees was measured at exactly the same time (age).

```
table(orange$age, orange$Tree)
```

```
      3 1 5 2 4
  118  1 1 1 1 1
  484  1 1 1 1 1
  664  1 1 1 1 1
  1004 1 1 1 1 1
  1231 1 1 1 1 1
  1372 1 1 1 1 1
  1582 1 1 1 1 1
```

Yes, each tree was measured at precisely the same five times.

## 4.2 Correlation between Age and Circumference

Here's another case that calls for the %$% pipe.

```
orange %$% cor(age, circumference)
```

```
[1] 0.9135189
```

Or, obtain the identical result with. . .

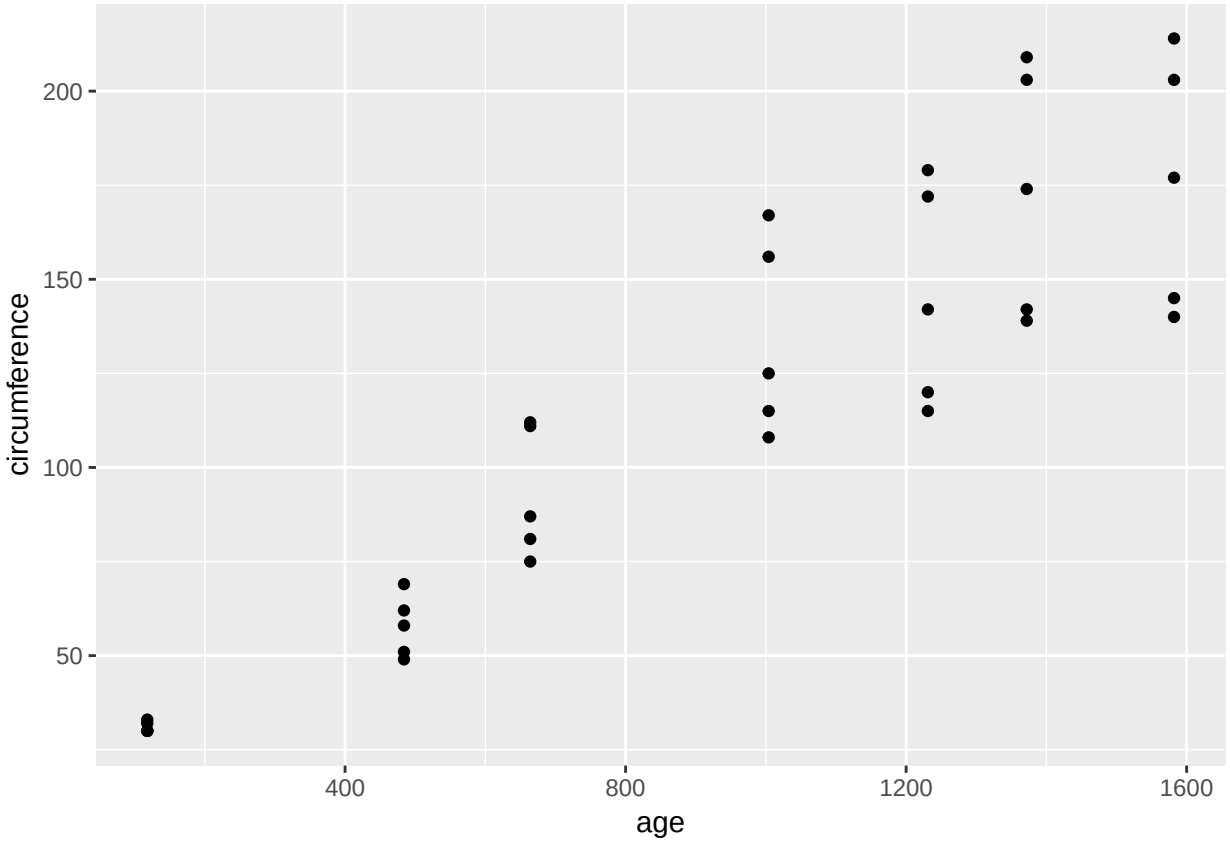
```
cor(orange$age, orange$circumference)
```

```
[1] 0.9135189
```

The Pearson correlation of age and circumference is 0.91 which is pretty strong, indicating that we'd expect to see a fairly positive and mostly linear association in a scatterplot. So, let's see if that's what we get.

### 4.3 Scatterplot predicting Circumference using Age across all Trees

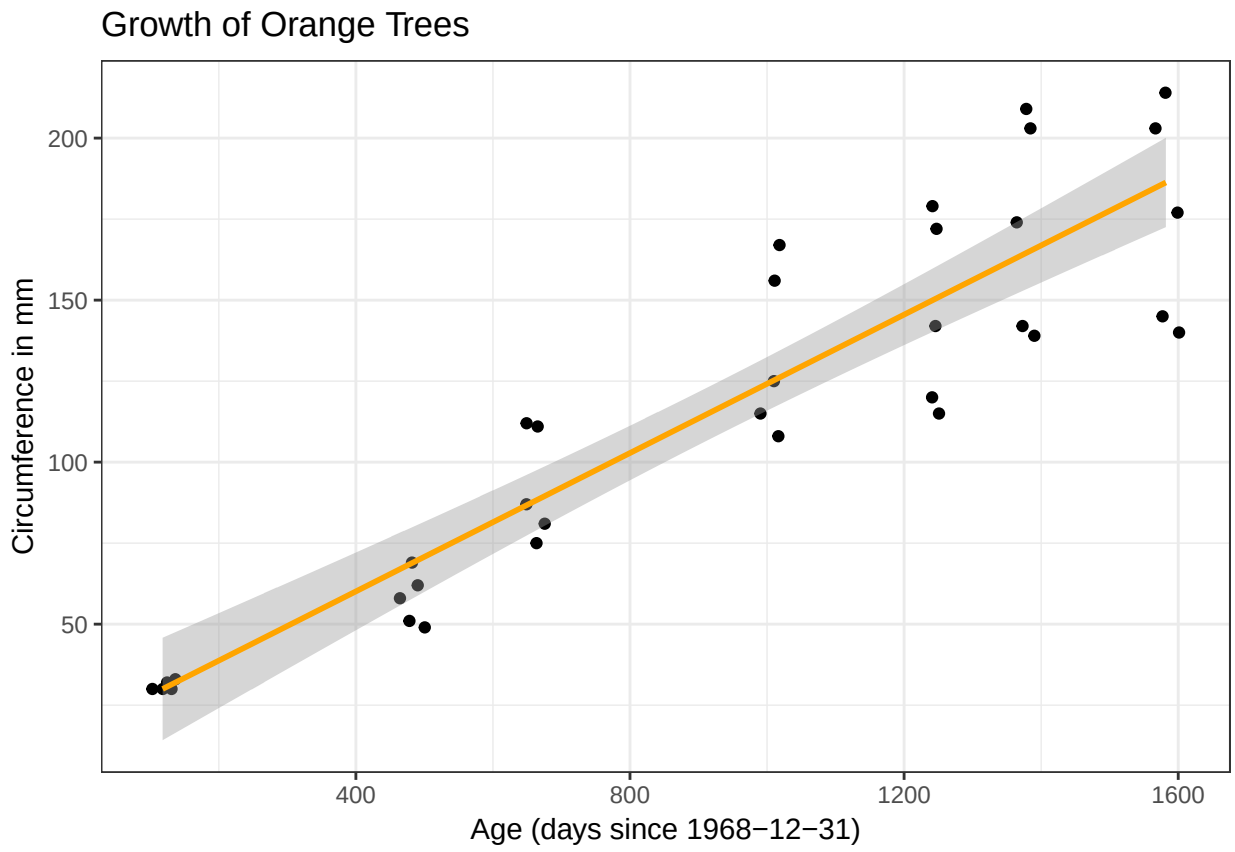
```
ggplot(orange, aes(x = age, y = circumference)) +
  geom_point()
```




OK. Let's add a linear model to this plot, as well as some better labels, and we'll change from mapping the points as observed to using `geom_jitter` to add a little horizontal (x-axis) jitter to the points, so that we don't have so much overlap.

### 4.4 Scatterplot with Linear Fit predicting Circumference using Age across all Trees

```
ggplot(orange, aes(x = age, y = circumference)) +
    geom_jitter(width = 20, height = 0) +
    geom_smooth(method = "lm", col = "orange") +
    labs(title = "Growth of Orange Trees",
         x = "Age (days since 1968-12-31)",
         y = "Circumference in mm") +
    theme_bw()
```



## 4.5 The Linear Model

The linear model fitted here is summarized below:

```
model1 <- lm(circumference ~ age, data = orange)
summary(model1)
```

```
Call:
lm(formula = circumference ~ age, data = orange)

Residuals:
    Min      1Q  Median      3Q     Max
-46.310 -14.946  -0.076  19.697  45.111

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 17.399650   8.622660   2.018   0.0518 .
age          0.106770   0.008277  12.900 1.93e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.74 on 33 degrees of freedom
Multiple R-squared:  0.8345,    Adjusted R-squared:  0.8295
F-statistic: 166.4 on 1 and 33 DF,  p-value: 1.931e-14
```

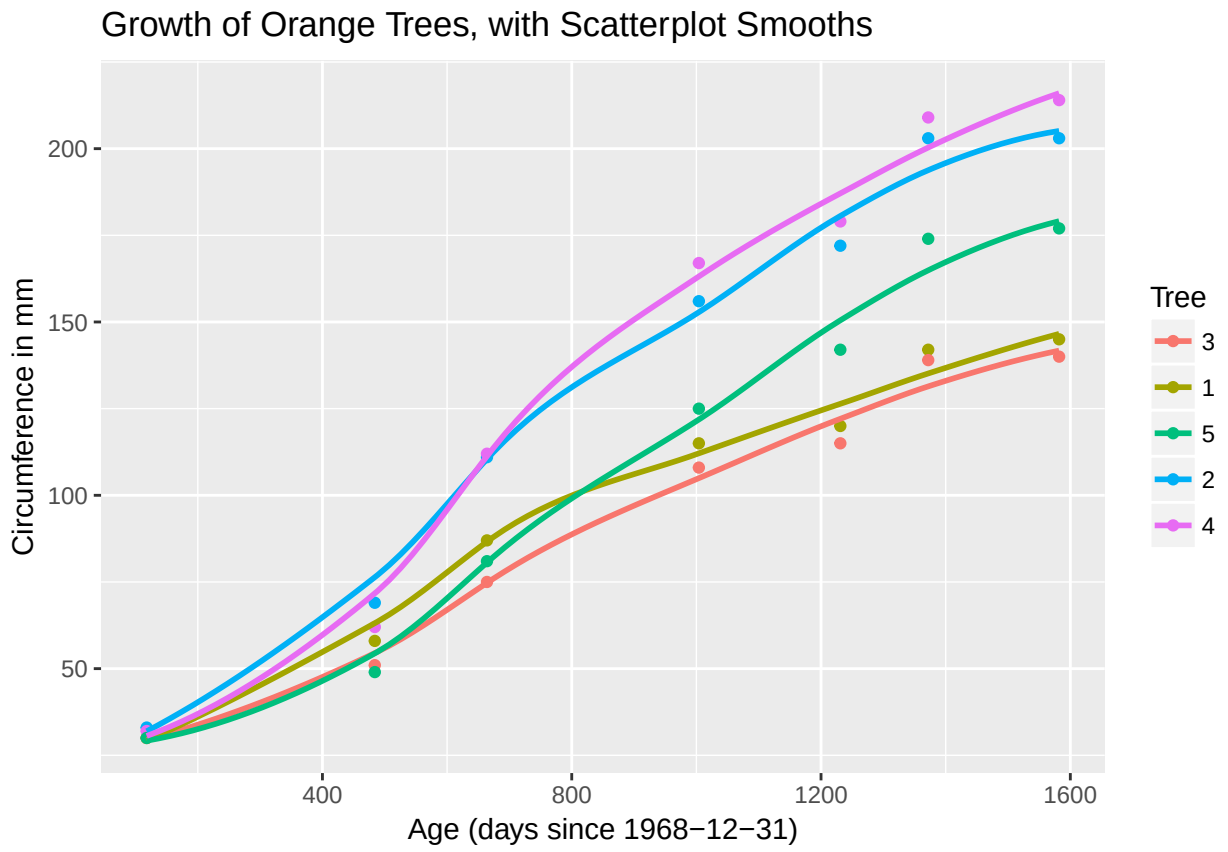
So the linear regression model is: circumference $= 17.4 + 0.107$ age.

So our predicted circumference for a tree of age 1000 days is 124 mm.

## 4.6 Some Other Scatterplots: Assessing each Tree separately

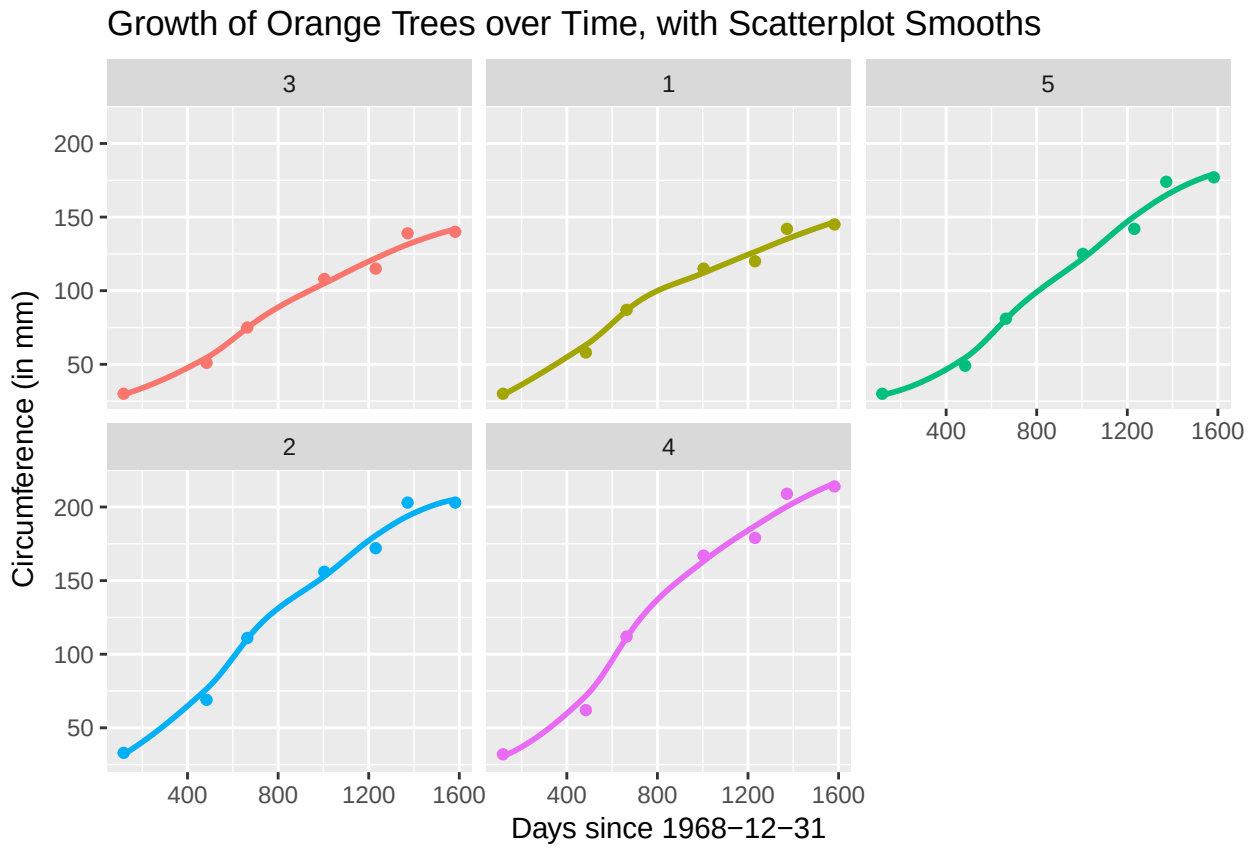
As a third option, let's fit separate smooth (`loess`) curves to each of the five individual trees, and plot each of them in different colors.

```
ggplot(orange, aes(x = age, y = circumference, col = Tree)) +
  geom_point() +
  geom_smooth(method = "loess", se = FALSE) +
  labs(title = "Growth of Orange Trees, with Scatterplot Smooths",
       x = "Age (days since 1968-12-31)", y = "Circumference in mm")
```



Or we could facet the plots, showing multiple scatterplots, one for each Tree.

```
ggplot(orange, aes(x = age, y = circumference, col = Tree)) +
  geom_point() +
  geom_smooth(method = "loess", se = FALSE) +
  facet_wrap(~ Tree) +
  guides(col = FALSE) +
  labs(title = "Growth of Orange Trees over Time, with Scatterplot Smooths",
       x = "Days since 1968-12-31", y = "Circumference (in mm)")
```



| | A | B | C | D |
|---|---|---|---|---|
| 1 | patient | drug | gender | response |
| 2 | MW | A | M | 23 |
| 3 | TT | B | F | 15 |
| 4 | KH | B | M | 18 |
| 5 | GC | A | M | 29 |
| 6 | DS | B | F | 34 |
| 7 | HJ | B | F | 15 |
| 8 | KM | A | M | 7 |
| 9 | RS | A | M | 19 |
| 10 | DG | A | F | 22 |
| 11 | | | | |

Figure 1: An Excel sheet with a tidy data set

# 5 Getting Data into R from Excel or another Software Package: The Fundamentals

The easiest way to get data from another software package into R is to save the file (from within the other software package) in a form that R can read. What you want is to end up with an Excel file that looks like this. . .

This *tidy* data set contains:

- one row for each subject
- variables that indicate characteristics of each of the subjects

The variable names are in the first row, and the data are in the remaining rows (2-10 in this small example). Categorical variables are most easily indicated by letters (drug A or B, for instance) while continuous variables, like response, are indicated by numbers. Leave missing cells blank or use the symbol `NA`, rather than indicating them with, say, `-99` or some other code.

Within Excel, this file can be saved as a `.csv` (comma-separated text file) or just as an Excel .XLS file, and then imported directly into R, via RStudio by clicking Import Dataset under the Workspace tab, then selecting From Text File. If you've saved the file in Excel as a `.csv` file, RStudio will generally make correct guesses about how to import the file. Once imported, you just need to save the workspace when you quit RStudio and you'll avoid the need to re-import.