ggplot2: Going further in the tidyverse

Michael Friendly
Psych 6135

http://euclid.psych.yorku.ca/www/psy6135/
A larger view: Data science

- Data science treats statistics & data visualization as parts of a larger process
  - Data import: text files, data bases, web scraping, ...
  - Data cleaning → “tidy data”
  - Model building & visualization
  - Reproducible report writing
The tidyverse of R packages
Topics

• Data import / export
• Data wrangling: getting your data into shape
  ▪ dplyr & tidyr
  ▪ pipes: %>%
  ▪ grouping & summarizing
  ▪ Example: NASA data on solar radiation
• Visualizing models: broom
  ▪ Example: gapminder data
• ggplot2 extensions
• tables in R
• The `readr` package is the modern, tidy way to import and export data
  - **Tabular data:**
    - comma delimited (read.csv)
    - any other delimiters (";" = read.csv2; <tab> = read_tsv)
  - **Data types:**
    - specify column types or let functions guess

• **Other data formats**

<table>
<thead>
<tr>
<th>package</th>
<th>Data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>haven</td>
<td>SAS, SPSS, Stata</td>
</tr>
<tr>
<td>readxl</td>
<td>Excel files (.xls and .xlsx)</td>
</tr>
<tr>
<td>DBI</td>
<td>Databases (SQL, ...)</td>
</tr>
<tr>
<td>rvest</td>
<td>HTML (web scraping)</td>
</tr>
</tbody>
</table>
Data Import: RStudio

**file:**

```
C:/Users/friendly/Dropbox/Documents/6135/R/drugs.txt
```

**options:**

<table>
<thead>
<tr>
<th>Import Options</th>
<th>Code Preview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: drugs</td>
<td><code>library(readr)</code></td>
</tr>
<tr>
<td>Skip: 0</td>
<td><code>drugs &lt;- read_table2(&quot;R/drugs.txt&quot;)</code></td>
</tr>
<tr>
<td>Delimiter: Whitespace</td>
<td><code>view(drugs)</code></td>
</tr>
</tbody>
</table>

**code:**

```
library(readr)
drugs <- read_table2("R/drugs.txt")
view(drugs)
```
Data transformation tools

Some common data types can be messy when imported. Tidy tools are there to help

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Package</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>dates/times</td>
<td>lubridate</td>
<td>Read dates/times in various formats; extract components</td>
</tr>
<tr>
<td>factors</td>
<td>forcats</td>
<td>Change order of levels, drop levels, combine levels</td>
</tr>
<tr>
<td>strings</td>
<td>stringr</td>
<td>Detect matches, subset, replace</td>
</tr>
</tbody>
</table>
lubridate: Dates & times

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (y), month (m), day (d), hour (h), minute (m) and second (s) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

2017-11-28T14:02:00
2017-22-12 10:00:00
11/28/2017 1:02:03
1 Jan 2017 23:59:59
20170131
July 4th, 2000
4th of July '99
2001: Q3
2:01

ymd_hms(), ymd_hm(), ymd_h().
ymd_hms("2017-11-28T14:02:00")

ymd_hms(), ydm_hm(), ydm_h().
ydm_hms("2017-22-12 10:00:00")

mdy_hms(), mdy_hm(), mdy_h().
mdy_hms("11/28/2017 1:02:03")

dmy_hms(), dmy_hm(), dmy_h().
dmy_hms("1 Jan 2017 23:59:59")

ymd(), ydm(). ymd(20170131)

mdy(), myd(). mdy("July 4th, 2000")

dmy(), dym(). dmy("4th of July '99")

yq() Q for quarter. yq("2001: Q3")

hms::hms() Also lubridate::hms(),
hm() and ms(), which return periods: hms::hms(sec = 0, min=1, hours = 2)

2018-01-31 11:59:59
2018-01-31 11:59:59
2018-04-31 11:59:59
2018-01-31 11:59:59
2018-01-31 11:59:59

GET AND SET COMPONENTS

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

d #"2017-11-28"
day(d) # 28
day(d) <- 1
d #"2017-11-01"

date(x) Date component. date(dt)
year(x) Year. year(dt)
Isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.

month(x, label, abbr) Month. month(dt)

day(x) Day of month. day(dt)
wday(x,label,abbr) Day of week. qday(x) Day of quarter.

hour(x) Hour. hour(dt)

minute(x) Minutes. minute(dt)
second(x) Seconds. second(dt)

week(x) Week of the year. week(dt)
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

Learn more at: http://lubridate.tidyverse.org
stringr: Manipulating strings

**Detect Matches**

- `str_detect(string, pattern)` Detect the presence of a pattern match in a string.
- `str_which(string, pattern)` Find the indexes of strings that contain a pattern match.
- `str_count(string, pattern)` Count the number of matches in a string.
- `str_locate(string, pattern)` Locate the positions of pattern matches in a string. Also `str_locate_all`.

**Subset Strings**

- `str_sub(string, start = 1L, end = -1L)` Extract substrings from a character vector.
- `str_subset(string, pattern)` Return only the strings that contain a pattern match.
- `str_extract(string, pattern)` Return the first pattern match found in each string, as a vector. Also `str_extract_all` to return every pattern match.
- `str_match(string, pattern)` Return the first pattern match found in each string, as a matrix with a column for each grouping in pattern. Also `str_match_all`.

**Mutate Strings**

- `str_sub()` <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.
- `str_replace(string, pattern, replacement)` Replace the first matched pattern in each string.
- `str_replace_all(string, pattern, replacement)` Replace all matched patterns in each string.
- `str_to_lower(string, locale = "en")` Convert strings to lower case.
- `str_to_upper(string, locale = "en")` Convert strings to upper case.

**Join and Split**

- `str_c(..., sep = "", collapse = NULL)` Join multiple strings into a single string.
- `str_c(..., sep = "", collapse = "")` Collapse a vector of strings into a single string.
- `str_dups(string, times)` Repeat strings times.
- `str_split_fixed(string, pattern, n)` Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also `str_split_to` to return a list of substrings.
- `str_glue(..., sep = "", envir = parent.frame())` Create a string from strings and expressions to evaluate.

Learn more at: [http://stringr.tidyverse.org](http://stringr.tidyverse.org)
forcats: Working with factors

R represents categorical variables as factors, useful for analysis (e.g., ANOVA). In graphics, we often want to recode levels or reorder them.

Learn more at: http://forcats.tidyverse.org
Tidy tools: overview

**tidyr**
- gather()
- spread()

Reshape data to be tidy

**dplyr**
- filter()
- select()

Manipulate & summarize tidy data

**ggplot2**

Visualize me!
Tidy operations

Reshape long to wide
synonym: `tidyr::pivot_longer()`

Reshape long to wide
synonym: `tidyr::pivot_longer()`

tidyr::gather(cases, "year", "n", 2:4)
Gather columns into rows.

tidyr::spread(pollution, size, amount)
Spread rows into columns.

tidyr::separate(storms, date, c("y", "m", "d"))
Separate one column into several.

tidyr::unite(data, col, ..., sep)
Unite several columns into one.

Separate parts of a value into several variables

Join related variables into one
What is Tidy Data?
A dataset is said to be tidy if:
• observations are in rows
• variables are in columns
• each value is in its own cell.

A “messy” dataset: Survey of income by religion from Pew Research
• Values of income are in separate columns, not one variable
• Column headers are values, not variable names
• Cell values are frequencies--- implicit, not explicit

<table>
<thead>
<tr>
<th>religion</th>
<th>&lt;$10k</th>
<th>$10-20k</th>
<th>$20-30k</th>
<th>$30-40k</th>
<th>$40-50k</th>
<th>$50-75k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agnostic</td>
<td>27</td>
<td>34</td>
<td>60</td>
<td>81</td>
<td>76</td>
<td>137</td>
</tr>
<tr>
<td>Atheist</td>
<td>12</td>
<td>27</td>
<td>37</td>
<td>52</td>
<td>35</td>
<td>70</td>
</tr>
<tr>
<td>Buddhist</td>
<td>27</td>
<td>21</td>
<td>30</td>
<td>34</td>
<td>33</td>
<td>58</td>
</tr>
<tr>
<td>Catholic</td>
<td>418</td>
<td>617</td>
<td>732</td>
<td>670</td>
<td>638</td>
<td>1116</td>
</tr>
</tbody>
</table>

This organization is easy in Excel
But, this makes data analysis and graphing hard
Tidying: reshaping wide to long

We can tidy the data by reshaping from wide to long format using `tidyr::gather()`

```
> pew <- read.delim(
    file = "http://stat405.had.co.nz/data/pew.txt",
    header = TRUE,
    stringsAsFactors = FALSE, check.names = FALSE)

> (pew1 <- pew[1:4, 1:6])  # small subset

religion <$10k $10-20k $20-30k $30-40k $40-50k
1 Agnostic 27 34 60 81 76
2 Atheist 12 27 37 52 35
3 Buddhist 27 21 30 34 33
4 Catholic 418 617 732 670 638
```

Another solution, using `reshape2::melt()`

```
> library(reshape2)
> pew_tidy <- melt(
    data = pew1,
    id = "religion",
    variable.name = "income",
    value.name = "frequency"
)
```

NB: income is a character variable; we might want to create an ordered factor or numeric version
Using pipes: %>%

- R is a functional language
  - This means that \( f(x) \) returns a value, as in \( y \leftarrow f(x) \)
  - That value can be passed to another function: \( g(f(x)) \)
  - And so on: \( h(g(f(x))) \)

```r
> x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142)
> exp(diff(log(x)))
[1] 3.29 1.75 1.58 0.52 0.28
```

- This gets messy and hard to read, unless you break it down step by step

```r
> # Compute the logarithm of `x`, calculate lagged differences, return the exponential function of the result
> log(x)
[1] -2.216 -1.024 -0.462 -0.004 -0.664 -1.952
> diff(log(x))    #calculate lagged diffs
[1] 1.19 0.56 0.46 -0.66 -1.29
> exp(diff(log(x)))    # convert back to original scale
[1] 3.29 1.75 1.58 0.52 0.28
```
Using pipes: %>%

- Pipes (%>) change the syntax to make this easier

```r
> # use pipes
> x %>% log() %>% diff() %>% exp()
[1] 3.29 1.75 1.58 0.52 0.28
```

- Basic rules
  - `x %>% f()` passes object on left hand side as first argument (or . argument) of function on right hand side
    - `x %>% f()` is the same as `f(x)`
    - `x %>% f(y)` is the same as `f(x, y)`
    - `y %>% f(x, ., z)` is the same as `f(x, y, z)`
  - `x %<>% f()` does the same, but assigns the result to `x`
    - Shortcut for `x <- x %>% f()`
Using pipes: %>% ggplot()

For the Pew data, mutate income into an ordered factor and make a ggplot

```
pew1 %>%
  gather("income", "frequency", 2:6) %>%  # reshape
  mutate(income = ordered(income, levels=unique(income))) %>%  # make ordered
  ggplot(aes(x=income, fill=religion)) +  # plot
    geom_bar(aes(weight=frequency))  # as freq bars
```

mutate() calculates or transforms column variables
ordered(income) levels are now ordered appropriately.

The result is piped to ggplot()
It sometimes happens that several variables are crammed into one column, or parts of one variable are split across multiple columns.

```
pew_long %>%
  mutate(inc = gsub("[\$k]", "", income)) %>%
  mutate(inc = gsub("<", "0-", inc)) %>%
  separate(inc, c("low", "high"), ") %>%
  head()
```

<table>
<thead>
<tr>
<th>religion</th>
<th>income</th>
<th>frequency</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agnostic</td>
<td>&lt;$10k</td>
<td>27</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Atheist</td>
<td>&lt;$10k</td>
<td>12</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Buddhist</td>
<td>&lt;$10k</td>
<td>27</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Catholic</td>
<td>&lt;$10k</td>
<td>418</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Agnostic</td>
<td>$10-20k</td>
<td>34</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Atheist</td>
<td>$10-20k</td>
<td>27</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>
**dplyr: Subset observations (rows)**

dplyr implements a variety of verbs to select a subset of observations from a dataset.

- **dplyr::filter(iris, Sepal.Length > 7)**
  Extract rows that meet logical criteria.

- **dplyr::distinct(iris)**
  Remove duplicate rows.

- **dplyr::sample_frac(iris, 0.5, replace = TRUE)**
  Randomly select fraction of rows.

- **dplyr::sample_n(iris, 10, replace = TRUE)**
  Randomly select n rows.

- **dplyr::slice(iris, 10:15)**
  Select rows by position.

- **dplyr::top_n(storms, 2, date)**
  Select and order top n entries (by group if grouped data).

In a pipe expression, omit the dataset name:

```r
iris %>% filter(Sepal.Length > 7)
iris %>% filter(Species == "setosa")

iris %>% sample_n(10)
iris %>% slice(1:50)  # setosa
```
Many helper functions in dplyr allow selection by a `function` of variable names:

- `select(iris, contains("\."))`  
  Select columns whose name contains a character string.
- `select(iris, ends_with("Length"))`  
  Select columns whose name ends with a character string.
- `select(iris, everything())`  
  Select every column.
- `select(iris, matches("\.t\."))`  
  Select columns whose name matches a regular expression.
- `select(iris, num_range("x", 1:5))`  
  Select columns named x1, x2, x3, x4, x5.
- `select(iris, one_of(c("Species", "Genus")))`  
  Select columns whose names are in a group of names.
- `select(iris, starts_with("Sepal"))`  
  Select columns whose name starts with a character string.
- `select(iris, Sepal.Length:Petal.Width)`  
  Select all columns between Sepal.Length and Petal.Width (inclusive).
- `select(iris, -Species)`  
  Select all columns except Species.
Fundamental operations in data munging are:

- **grouping** a dataset by one or more variables
- calculating one or more summary measures
- **ungrouping**: expand to an ungrouped copy, if needed

```r
mtcars %>%
group_by(cyl) %>%
summarise(avg=mean(mpg))
```

```r
mtcars %>%
group_by(cyl) %>%
summarise(avg=mean(mpg)) %>%
ungroup()
```
Example: NASA data on solar radiation

How does solar radiation vary with latitude, over months of the year?

How to make this plot?

Q: what are the basic plot elements?
NASA data: solar radiation

This is easy to do for the total Annual solar radiation, a column in the data

```r
str(nasa)
'data.frame': 64800 obs. of 15 variables:
$ Lat: int -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 ...
$ Lon: int -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
$ Feb: num 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 ...
$ Mar: num 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 ...
$ Apr: num 0 0 0 0 0 0 0 0 0 0 ...
$ May: num 0 0 0 0 0 0 0 0 0 0 ...
$ Jun: num 0 0 0 0 0 0 0 0 0 0 ...
$ Jul: num 0 0 0 0 0 0 0 0 0 0 ...
$ Aug: num 0 0 0 0 0 0 0 0 0 0 ...
$ Sep: num 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
$ Nov: num 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
$ Dec: num 11 11 11 11 11 ...

nasa %>%
  filter(abs(Lat) < 60) %>%
  mutate(Latf = cut(Lat, pretty(Lat, n=10))) %>%
  ggplot(aes(x=Latf, y=Ann)) +
  geom_violin(fill="pink", alpha=0.3) +
  labs(x="Latitude", y="Solar radiation G(0) (kWh/m²)")
```
Faceting & tidy data

This is complicated to do for the separate months, because the data structure is **untidy**--- months were in separate variables (wide format)

```r
> str(nasa)
'data.frame': 64800 obs. of 15 variables:
$ Lat: int  -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 ...
$ Lon: int  -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
$ Feb: num  5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 ...
$ Mar: num  0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 ...
$ Apr: num  0 0 0 0 0 0 0 0 0 0 ...
$ May: num  0 0 0 0 0 0 0 0 0 0 ...
$ Jun: num  0 0 0 0 0 0 0 0 0 0 ...
$ Jul: num  0 0 0 0 0 0 0 0 0 0 ...
$ Aug: num  0 0 0 0 0 0 0 0 0 0 ...
$ Sep: num  0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
$ Nov: num  8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
$ Dec: num  11 11 11 11 11 11 11 11 11 11 ...
```
To plot solar radiation against latitude by month (separate panels), we need to:

- remove the Ann column
- reshape the data to long format, so solar is all in one column

```r
library(tidyr)
library(dplyr)
library(ggplot2)

nasa_long <- nasa %>%
  select(-Ann) %>%
  gather(month, solar, Jan:Dec, factor_key=TRUE) %>%
  filter( abs(Lat) < 60 ) %>%
  mutate( Lat_f = cut(Lat, pretty(Lat, 12)))
```

%>% “pipes” data to the next stage

- `select()` extracts or drops columns
- `gather()` collapses columns into key-value pairs
- `filter()` subsets observations
- `mutate()` creates new variables
tidying the data

> str(nasa_long)
'data.frame': 514080 obs. of 5 variables:
  $ Lon: int -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
  $ month: Factor w/ 12 levels "Jan","Feb","Mar",...: 1 1 1 1 1 1 1 1 1 1 ...
  $ solar: num 5.19 5.19 5.25 5.25 5.17 5.17 5.15 5.15 5.15 5.15 ...
  $ Lat_f: Factor w/ 12 levels "(-60,-50]","(-50,-40]",...: 1 1 1 1 1 1 1 1 1 1 ...

For ease of plotting, I created a factor version of Lat with 12 levels

> head(nasa_long)
  Lat Lon month solar    Lat_f
1  -59 -180   Jan  5.19   (-60,-50]
2  -59 -179   Jan  5.19   (-60,-50]
3  -59 -178   Jan  5.25   (-60,-50]
4  -59 -177   Jan  5.25   (-60,-50]
5  -59 -176   Jan  5.17   (-60,-50]
6  -59 -175   Jan  5.17   (-60,-50]

The data are now in a form where I can plot solar against Lat or Lat_f and facet by month
plotting the tidy data

Using `geom_violin()` shows the shapes of the distributions for levels of `Lat_f`

```r
ggplot(nasa_long, aes(x=Lat_f, y=solar)) + geom_violin(fill="pink") + facet_wrap(~ month) + theme_bw() + theme(axis.text.x = element_text(angle = 70, hjust = 1))
```

`facet_wrap(~month)` does the right thing

I had to adjust the x-axis labels for `Lat_f` to avoid overplotting
ggplot(nasa_long, aes(x=Lat, y=solar)) + geom_smooth(color="blue") + facet_wrap(~ month) + theme_bw()

Here we treat Lat as quantitative. geom_smooth() uses method = “gam” here because of large $n$

The variation in the smoothed trends over the year suggest quite lawful behavior

Can we express this as a statistical model?
What we saw in the plot suggests a **generalized additive model**, with a smooth, s(Lat)

```r
library(mgcv)
nasa.gam <- gam(solar ~ Lon + month + s(Lat), data=nasa_long)
summary(nasa.gam)
```

```r
Family: gaussian
Link function: identity

Formula:
solar ~ Lon + month + s(Lat)

Parametric coefficients:

| Estimate   | Std. Error | t value | Pr(>|t|) |
|------------|------------|---------|----------|
| (Intercept)| 4.691e+00  | 6.833e-03| 686.409  | < 2e-16 *** |
| Lon        | -1.713e-04 | 1.898e-05| -9.022   | < 2e-16 *** |
| monthFeb   | 1.195e-01  | 9.664e-03| 12.364   | < 2e-16 *** |
| ...        | ...        | ...     | ...      | ...       |
| monthDec   | -8.046e-02 | 9.664e-03| -8.326   | < 2e-16 *** |

---

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:

<table>
<thead>
<tr>
<th>edf</th>
<th>Ref.df</th>
<th>F</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s(Lat)</td>
<td>8.997</td>
<td>9.37285</td>
<td>&lt;2e-16 ***</td>
</tr>
</tbody>
</table>

---

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) = 0.398  Deviance explained = 39.8%
GCV = 2.0006  Scale est. = 2.0005  n = 514080
```

The violin plots suggest that variance is not constant. I’m ignoring this here by using the default gaussian model.

**Model terms:**
- Lon wasn’t included before
- month is a factor, for the plots
- s(Lat) fits a smoothed term in latitude, averaged over other factors

There are other model choices, but it is useful to visualize what we have done so far
Effect plots show the fitted relationship between the response and model terms, averaged over other predictors. The mgcv package has its own versions of these.

```r
plot(nasa.gam, cex.lab=1.25)
termplot(nasa.gam, terms="month", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
termplot(nasa.gam, terms="Lon", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
```

why the dip at the equator?  
effect of longitude is very small, but maybe interpretable  
month should be modeled as a time variable
Visualizing models

• R modeling functions [lm(), glm(), …] return model objects, but these are “messy”
  ▪ extracting coefficients takes several steps: data.frame(coef(mymod))
  ▪ some info ($R^2$, $F$, $p$.value) is computed in print() method, not stored
  ▪ can’t easily combine models

• Some have associated plotting functions
  ▪ plot(model): diagnostic plots
  ▪ car package: many model plot methods
  ▪ effects package: plot effects for model terms

• But what if you want to:
  ▪ make a table of model summary statistics
  ▪ fit a collection of models, compare, summarize or visualize them?
broom: visualizing models

- The broom package turns model objects into tidy data frames
  - `glance(models)` extracts model-level summary statistics ($R^2$, df, AIC, BIC)
  - `tidy(models)` extracts coefficients, SE, p-values
  - `augment(models)` extracts observation-level info (residuals, ...)

ggplot(aes(x = log(gdpPerCap), y=lifeExp, color=continent), data=gapminder) +
  geom_point() +
  geom_smooth(method = "loess")

How to model this?

How to extract & plot model statistics?

How to fit & display multiple models for subsets?
Example: gapminder data

Predict life expectancy from year, population, GDP and continent:

gapmod <- lm(lifeExp ~ year + pop + log(gdpPercap) + continent, data=gapminder)
summary(gapmod)

Call:
  lm(formula = lifeExp ~ year + pop + log(gdpPercap) + continent, data = gapminder)

Residuals:
   Min     1Q   Median     3Q    Max
-24.928 -3.285   0.314   3.699  15.221

Coefficients:
                     Estimate  Std. Error t value     Pr(>|t|)  
(Intercept)      -4.58e+02   1.67e+01   -27.43  < 2e-16 ***
year               2.38e-01   8.61e-03    27.58  < 2e-16 ***
pop                5.40e-09   1.38e-09     3.91  9.5e-05 ***
log(gdpPercap)    5.10e+00   1.60e-01   31.88  < 2e-16 ***
continentAmericas 8.74e+00   4.63e-01   18.86  < 2e-16 ***
continentAsia     6.64e+00   4.09e-01   16.22  < 2e-16 ***
continentEurope   1.23e+01   5.10e-01   24.11  < 2e-16 ***
continentOceania  1.26e+01   1.27e+00    9.88  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ‘ 1

Residual standard error: 5.79 on 1696 degrees of freedom
Multiple R-squared:  0.8,    Adjusted R-squared:  0.799
F-statistic: 969 on 7 and 1696 DF,  p-value: <2e-16

observation level
component level (coefficients)
model level
**glance()** gives the **model level** summary statistics

```r
> glance(gapmod)
  r.squared adj.r.squared sigma statistic p.value df logLik AIC  BIC deviance df.residual
1   0.8        0.7992 5.789      969       0  8 -5406 10830 10879  56835        1696
```

**tidy()** gives the **model component (term) statistics**

```r
> tidy(gapmod)

<table>
<thead>
<tr>
<th>term</th>
<th>estimate</th>
<th>std.error</th>
<th>statistic</th>
<th>p.value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>-4.585e+02</td>
<td>1.671e+01</td>
<td>-27.433</td>
<td>1.982e-137</td>
</tr>
<tr>
<td>year</td>
<td>2.376e-01</td>
<td>8.613e-03</td>
<td>27.584</td>
<td>1.122e-138</td>
</tr>
<tr>
<td>pop</td>
<td>5.403e-09</td>
<td>1.381e-09</td>
<td>3.912</td>
<td>9.496e-05</td>
</tr>
<tr>
<td>log(gdpPercap)</td>
<td>5.103e+00</td>
<td>1.601e-01</td>
<td>31.876</td>
<td>4.096e-175</td>
</tr>
<tr>
<td>continentAmericas</td>
<td>8.739e+00</td>
<td>4.635e-01</td>
<td>18.856</td>
<td>3.758e-72</td>
</tr>
<tr>
<td>continentAsia</td>
<td>6.635e+00</td>
<td>4.091e-01</td>
<td>16.219</td>
<td>4.167e-55</td>
</tr>
<tr>
<td>continentEurope</td>
<td>1.230e+01</td>
<td>5.102e-01</td>
<td>24.113</td>
<td>1.943e-110</td>
</tr>
<tr>
<td>continentOceania</td>
<td>1.256e+01</td>
<td>1.270e+00</td>
<td>9.884</td>
<td>1.943e-22</td>
</tr>
</tbody>
</table>
```

**augment()** gives the **observation level** statistics

```r
> augment(gapmod) %>% slice(1:5)

# A tibble: 5 x 12

<table>
<thead>
<tr>
<th>lifeExp</th>
<th>year</th>
<th>pop</th>
<th>log.gdpPercap.</th>
<th>continent</th>
<th>.fitted</th>
<th>.se.fit</th>
<th>.resid</th>
<th>.hat</th>
<th>.sigma</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;dbl&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;fct&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
</tr>
<tr>
<td>1</td>
<td>28.8</td>
<td>1952</td>
<td>8425333</td>
<td>Asia</td>
<td>46.0</td>
<td>0.408</td>
<td>-17.1</td>
<td>0.00496</td>
<td>5.78</td>
</tr>
<tr>
<td>2</td>
<td>30.3</td>
<td>1957</td>
<td>9240934</td>
<td>Asia</td>
<td>47.4</td>
<td>0.390</td>
<td>-17.1</td>
<td>0.00454</td>
<td>5.78</td>
</tr>
<tr>
<td>3</td>
<td>32.0</td>
<td>1962</td>
<td>10267083</td>
<td>Asia</td>
<td>48.8</td>
<td>0.376</td>
<td>-16.8</td>
<td>0.00423</td>
<td>5.78</td>
</tr>
<tr>
<td>4</td>
<td>34.0</td>
<td>1967</td>
<td>11537966</td>
<td>Asia</td>
<td>49.9</td>
<td>0.372</td>
<td>-15.9</td>
<td>0.00413</td>
<td>5.78</td>
</tr>
<tr>
<td>5</td>
<td>36.1</td>
<td>1972</td>
<td>13079460</td>
<td>Asia</td>
<td>50.5</td>
<td>0.382</td>
<td>-14.4</td>
<td>0.00435</td>
<td>5.78</td>
</tr>
</tbody>
</table>
```

#... with 2 more variables: .cooks <dbl>, .std.resid <dbl>
Going further: fitting multiple models

There may be different relations by continent (interactions)

• What if want to fit (and visualize) a separate model for each continent?
• dplyr::do() allows us to store the result of an arbitrary computation in a tidy column

# separate models for continents
models <- gapminder %>%
  filter(continent != "Oceania") %>% # only two countries
  group_by(continent) %>%
  do(mod = lm(lifeExp ~ year + pop + log(gdpPercap), data=.))

# view model summaries
models %>% glance(mod)

# A tibble: 4 x 12
# Groups:   continent [4]
   continent r.squared adj.r.squared sigma statistic   p.value df logLik
<fct>         <dbl>         <dbl> <dbl>     <dbl>     <dbl> <int>  <dbl>
1 Africa        0.500         0.498  6.48      207. 5.90e-93     4 -2050.
2 Americas      0.720         0.718  4.97      254. 1.39e-81     4  -904.
3 Asia          0.696         0.694  6.56      299. 5.27e-101     4 -1305.
4 Europe        0.797         0.795  2.46      466. 7.42e-123     4  -833.
# ... with 4 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
#   df.residual <int>
Going further: plotting multiple models

One visual summary might be a plot of $R^2$ values, ordered by continent.

```r
models %>%
  glance(mod) %>%
  ggplot(aes(r.squared, reorder(continent, r.squared))) +
  geom_point(size=4) +
  geom_segment(aes(xend = 0, yend = ..y..)) +
  ylab("Continent")
```
Coefficient plots are often useful, but these are on different scales.

Here, I plot the $t$-statistics, $t = b_{ij} / \text{se}(b_{ij})$ for all terms in all models.

Any values outside $\sim \pm 2$ are significant, $p < 0.5!$
There are a large number of ggplot extensions. See: [http://www.ggplot2-exts.org/](http://www.ggplot2-exts.org/)
ggplot extensions: GGally

GGally contains a large number of functions that extend ggplot2 to multivariate data

ggpairs() produces generalized scatterplot matrices, with lots of options

library(GGally)
library(dplyr)
library(ggplot2)
library(gapminder)

gapminder %>%
  select(-country, -year) %>%
  ggpairs(aes(color=continent))
ggplot extensions: gganimate

gganimate is a wrapper for the animation package with ggplot2.

It adds a frame= aesthetic, and animates the image as the frame variable changes.

```r
p5 <- ggplot(gapminder, aes(gdpPercap, lifeExp, size = pop, frame = year)) +
  geom_point() +
  geom_smooth(aes(group = year), method = "lm", show.legend = FALSE) +
  facet_wrap(~continent, scales = "free") +
  scale_x_log10()

gganimate(p5)
```
The **ggpubr** package provides some easy-to-use functions for creating and customizing publication ready plots.

```r
ggviolin(df, x = "dose", y = "len", fill = "dose",
        palette = c("#00AFBB", "#E7B800", "#FC4E07"),
        add = "boxplot", add.params = list(fill = "white")) +
stat_compare_means(comparisons = my_comparisons, label = "p.signif") +
stat_compare_means(label.y = 50)
```

Tables in R

- Not a ggplot topic, but it is useful to know that you can also produce beautiful tables in R
- There are many packages for this: See the CRAN Task View on Reproducible Research, [https://cran.r-project.org/web/views/ReproducibleResearch.html](https://cran.r-project.org/web/views/ReproducibleResearch.html)
  - xtable: Exports tables to LaTeX or HTML, with lots of control
  - stargazer: Well-formatted model summary tables, side-by-side
  - apaStyle: Generate APA Tables for MS Word
Tables in R: xtable

Just a few examples, stolen from xtable: vignette("xtableGallery.pdf")

```r
fm1 <- aov(tlimth ~ sex + ethnicity + grade + disadvg, data = tli)
xtable(fm1)
```

<table>
<thead>
<tr>
<th></th>
<th>Df</th>
<th>Sum Sq</th>
<th>Mean Sq</th>
<th>F value</th>
<th>Pr(&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sex</td>
<td>1</td>
<td>75.37</td>
<td>75.37</td>
<td>0.38</td>
<td>0.5417</td>
</tr>
<tr>
<td>ethnicity</td>
<td>3</td>
<td>2572.15</td>
<td>857.38</td>
<td>4.27</td>
<td>0.0072</td>
</tr>
<tr>
<td>grade</td>
<td>1</td>
<td>36.31</td>
<td>36.31</td>
<td>0.18</td>
<td>0.6717</td>
</tr>
<tr>
<td>disadvg</td>
<td>1</td>
<td>59.30</td>
<td>59.30</td>
<td>0.30</td>
<td>0.5882</td>
</tr>
<tr>
<td>Residuals</td>
<td>93</td>
<td>18682.87</td>
<td>200.89</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```r
fm3 <- glm(disadvg ~ ethnicity*grade, data = tli, family = binomial)
xtable(fm3)
```

|                          | Estimate | Std. Error | z value | Pr(>|z|) |
|--------------------------|----------|------------|---------|---------|
| (Intercept)              | 3.1888   | 1.5966     | 2.00    | 0.0458  |
| ethnicityHISPANIC        | -0.2848  | 2.4808     | -0.11   | 0.9086  |
| ethnicityOTHER           | 212.1701 | 22122.7093 | 0.01    | 0.9923  |
| ethnicityWHITE           | -8.8150  | 3.3355     | -2.64   | 0.0082  |
| grade                    | -0.5308  | 0.2892     | -1.84   | 0.0665  |
| ethnicityHISPANIC:grade  | 0.2448   | 0.4357     | 0.56    | 0.5742  |
| ethnicityOTHER:grade     | -32.6014 | 3393.4687  | -0.01   | 0.9923  |
| ethnicityWHITE:grade     | 1.0171   | 0.5185     | 1.96    | 0.0498  |

Too many decimals are used here, but you can control all that.