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
Data Import / Export

- 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

- RStudio
  - Environment
  - Data Import
  - Data Preview
  - Options
  - Code

Data transformation tools

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>dates/times</td>
<td>readr</td>
<td>read_date()</td>
</tr>
<tr>
<td>factors</td>
<td>forcats</td>
<td>change_order()</td>
</tr>
<tr>
<td>strings</td>
<td>stringr</td>
<td>detect_matches()</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.

- Get and set components
  - Use an accessor function to get a component.
  - Assign into an accessor function to change a component in place.

Learn more at: [http://lubridate.tidyverse.org](http://lubridate.tidyverse.org)
**stringr: Manipulating strings**

**Detect Matches**
- `str_detect(pattern)` Detect the prevalence of a pattern match in a string.
- `str_detect(pattern, ignore_case)` Detect the prevalence of a pattern match in a string, ignoring case.

**Subset Strings**
- `str_subset(string, start = NA, end = NA)` Extract substrings from a character vector.
- `str_subset(string, pattern)` Extract substrings that contain a pattern match.
- `str_match(string, pattern)` Count the number of matches in a string.

**Mutate Strings**
- `str_replace(string, pattern, replacement)` Replace matched patterns in each string.
- `str_replace_all(string, pattern, replacement)` Replace all matched patterns in each string.

**Join and Split**
- `str_c(sep = c("", " "), collapse = "\n")` Join multiple strings into a single string.
- `str_split(string, regex)` Split a string into a vector of substrings.
- `str_duplicated(string)` Identify duplicated elements.

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.

**Factors**
- `factor()` Create a factor with specified levels.
- `factor(level)` Create a factor with specified levels.
- `factor() %in% levels()` Check if elements are in the levels.

**Change the order of levels**
- `factor(order = levels(factor), labels = c("A", "B", "C"))` Change the order of levels.
- `factor(order = levels(factor), labels = c("C", "B", "A"))` Change the order of levels.

**Inspect Factors**
- `factor(levels(factor))` Get the levels of a factor.
- `factor() %in% levels(factor)` Check if elements are in the levels.

Learn more at: [http://forcats.tidyverse.org](http://forcats.tidyverse.org)

---

**Tidy tools: overview**

- **Tidy operations**
  - Reshape long to wide synonym: `tidyr::pivot_longer()`
- **Reshape long to wide**
  - `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)` Join related variables into one

Reshape data to be tidy
Manipulate & summarize tidy data
Visualize me!
Data wrangling with dplyr & tidyr

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

This organization is easy in Excel
But, this makes data analysis and graphing hard

Using pipes: %>%

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

  > 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

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

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

Using pipes: %>%

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

Tidying: reshaping wide to long

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

```r
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 $50-75k
1 Agnostic  27   34    60     81    76   137
2 Atheist   12   27    37     52    35    70
3 Buddhist  27   21    30     34    33    58
4 Catholic  418  617   732    670   638

library(tidyr)

gather(pew1, "income", "frequency", 2:6)

religion income frequency
1  Agnostic <$10k        272   Atheist <$10k        123  Buddhist <$10k        274  Catholic <$10k       4185  ...      8114  Atheist $30-40k        5215 Buddhist $30-40k        3416 Catholic $30-40k       670…  …        …             …

Another solution, using reshape2::melt()

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

key value columns

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

Using pipes: %>%

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

• 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) %>%
mutate(income = ordered(income, levels=unique(income))) %>%
ggplot(aes(x=income, fill=religion)) +
geom_bar(aes(weight=frequency))
```

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

The result is piped to ggplot()

Tidying: separate() and unite()

It sometimes happens that several variables are crammed into one column, or parts of one variable are split across multiple columns

```
tidy: separate(storms, date, c("y","m","d"))
tidy: unite(data, col1, ..., sep)
```

For example, for the Pew data, we might want separate income into low & high

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

dpolyr: Subset observations (rows)

dpolyr implements a variety of verbs to select a subset of observations from a dataset

```
dplyr::filter(iris, Sepal.Length > 7)
Remove 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).
```

dpolyr: Subset variables (columns)

In a pipe expression, omit the dataset name

```
iris %>% filter(Sepal.Length > 7)
iris %>% filter(Species=="setosa")
iris %>% sample_n(10)
iris %>% slice(1:50) # setosa
```

Many helper functions in dpolyr allow selection by a function of variable names:

```
select(iris, contains("Sepal"))
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("\d\")
Select columns whose name matches a regular expression.
select(iris, num_range("[0-9]", 2:5)
Select columns named x1, x2, x3, x4, x5.
```
dplyr: `group_by()` and `summarise()`

- 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))
```

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?

```
Lat: factor(Lat, levels=-90:90)
# calculates all months data
nasa %>%
  filter(Lat < 90) %>%
  group_by(Lat) %>%
  summarise(Ann = mean(SolRad)) %>%
  ggplot(aes(x = Lat, y = Ann)) +
  geom_violin(fill = "pink", alpha = 0.3) +
  labs(x = "Latitude", y = "Solar radiation G(0) (kWh/m²)")
```

NASA data: solar radiation

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

```
> 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 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
$ Oct: num 111.11 11 11 11 11 11 11 11 11 11 ...
```

```
# 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)

> 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 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
$ Oct: num 111.11 11 11 11 11 11 11 11 11 11 ...
```

```
# Filter by year
nasa %>%
  filter(year == 2002) %>%
  ggplot(aes(x = Lat, y = mean(SolRad))) +
  geom_violin(fill = "pink", alpha = 0.3) +
  labs(x = "Latitude", y = "Solar radiation G(0) (kWh/m²)")
```
tidying the data

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

```
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

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

```
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

plotting the tidy data: smoothing

Here we treat Lat as quantitative.

```
ggplot(nasa_long, aes(x=Lat, y=solar)) +
  geom_smooth(color="blue") +
  facet_wrap(~ month) +
  theme_bw()
```

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

Can we express this as a statistical model?
build a 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)
```

**Family: gaussian**  
**Link function: identity**  
**Formula:**  
solar ~ Lon + month + s(Lat)

**Parametric coefficients:**

| Estimate (Pr(>|t|)) | 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 | -8.320 | <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) 8.997</td>
<td>9</td>
<td>37285</td>
<td>&lt;2e-16 ***</td>
</tr>
</tbody>
</table>

---

**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.

---

**visualize the model**

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 (R², 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 (R², df, AIC, BIC)  
  - `tidy(models)` extracts coefficients, SE, p-values  
  - `augment(models)` extracts observation-level info (residuals, ...)

Example: gapminder data

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

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

```
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+00   1.36e+00    3.91     9.5e-05 ***
log(gdpPercap)              -1.00e+00   1.60e+01   -0.63     0.5307
continentAmericas           8.74e+00   4.63e+01    1.88     0.0625 .
continentAsia               6.64e+00   4.09e+01    1.62     0.1060
continentEurope             1.23e+01   5.10e+00    2.41     0.0163 *
continentOceania            1.26e+01   1.27e+00    9.88    < 2e-16 ***
observation level

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

model level
```

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?

  ```r
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") %>%
  group_by(continent) %>%
  do(mod = lm(lifeExp ~ year + pop + log(gdpPercap), data=.)
  # view model summaries
  models %>% glance(mod)
  ``

```r
# A tibble: 4 x 12
# Groups:   continent [4]
## continent r.squared adj.r.squared sigma statistic p.value df logLik AIC  BIC deviance df.residual
1 Africa        0.500         0.498  6.48      207. 5.90e- 93     4 -2050. Americas      0.720         0.718  4.97      ...      466. 7.42e-123     4  -833.
2 Europe        0.578         0.574  4.99      195. 1.24e- 80     4 -1670. Oceania      0.962         0.944  2.17      191. 2.64e-115     4  -260.
3 Asia          0.633         0.626  4.77      208. 2.74e- 51     4 -1664. America     0.605         0.596  5.24      204. 2.26e- 55     4 -170.
```

- dplyr::do() allows us to store the result of an arbitrary computation in a tidy column

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

```r
# A tibble: 4 x 12
# Groups:   continent [4]
## continent r.squared adj.r.squared sigma statistic p.value df logLik AIC  BIC deviance df.residual
1 Africa        0.500         0.498  6.48      207. 5.90e- 93     4 -2050. Americas      0.720         0.718  4.97      ...      466. 7.42e-123     4  -833.
2 Europe        0.578         0.574  4.99      195. 1.24e- 80     4 -1670. Oceania      0.962         0.944  2.17      191. 2.64e-115     4  -260.
3 Asia          0.633         0.626  4.77      208. 2.74e- 51     4 -1664. America     0.605         0.596  5.24      204. 2.26e- 55     4 -170.
```

- dplyr::do() allows us to store the result of an arbitrary computation in a tidy column

```
```
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")
```

Visualizing coefficients

Coefficient plots are often useful, but these are on different scales.

```r
models %>% tidy(mod) %>%
  filter(term != "(Intercept)") %>%
  mutate(term=factor(term, levels=c("log(gdpPercap)", "year", "pop"))) %>%
  ggplot(aes(x=term, y=statistic, color=continent, group=continent)) +
  geom_point(size=5, alpha=0.5) +geom_line(size=1.5) +geom_hline(yintercept=c(-2, 0, 2), color = c("red", "black", "red")) +
  ylab("t statistic") +
  theme_minimal() + theme(legend.position=c(0.9, 0.8))
```

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

Any values outside $\pm 2$ are significant, $p < 0.5 !$

ggplot extensions

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

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

gapminder %>%
  select(-country, -year) %>%
  ggpairs(aes(color=continent))
```

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

**ggpubr**

The `ggpubr` package provides some easy-to-use functions for creating and customizing publication ready plots.

```r
ggvisnl(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)
```

see the examples at http://www.sthda.com/english/rpkgs/ggpubr/

**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")

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