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**

- 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

---

**ggplot2: Going further in the tidyverse**

Michael Friendly
Psych 6135
http://euclid.psych.yorku.ca/www/psy6135/
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
• 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

We can tidy the data by reshaping from wide to long format using tidyr::gather()
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)`
  - `y %>% f(x, 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()`

```r
9
> # use pipes
> x %>% log() %>% diff() %>% exp()
[1] 3.29 1.75 1.58 0.52 0.28
```

Using pipes: %>% ggplot()

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

```r
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 new column variables.
The levels of income 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

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

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

<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>

dpolyr: Subset observations (rows)

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

In a pipe expression, omit the dataset name

```r
dplyr::filter(iris, Sepal.Length > 7)
dplyr::distinct(iris)
dplyr::sample_frac(iris, 0.5, replace = TRUE)
dplyr::sample_n(iris, replace = TRUE)
dplyr::slice(iris, 10:15)
dplyr::top_n(storms, 2, date)
```

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

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

- select(): Select columns by name or helper function.
- contains(): Select columns whose name contains a character string.
- ends_with(): Select columns whose name ends with a character string.
- everything(): Select every column.
- matches(): Select columns whose name matches a regular expression.

Example: NASA data on solar radiation

How does solar radiation vary with latitude, over months of the year? How to make this plot?

Example code:
```r
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)"
```

NASA data: solar radiation

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

Example code:
```r
nasa %>%
  group_by(Latf) %>%
  summarise(Ann=mean(Ann))
```
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 0 0 0 0 0 0 0 0 0 ...
  $ Oct: num  0 0 0 0 0 0 0 0 0 0 ...
  $ Nov: num  0 0 0 0 0 0 0 0 0 0 ...
  $ Dec: num  11 11 11 11 11 11 11 11 11 11 ...
```

It is much easier to 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)) )
```

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

```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)) )
```

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

```r
> 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  5.19 (-60,-50]
4 -59  -177   Jan  5.19 (-60,-50]
5 -59  -176   Jan  5.19 (-60,-50]
6 -59  -175   Jan  5.19 (-60,-50]
```

Faceting & tidy data

plotting the tidy data
plotting the tidy data: smoothing

```r
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.

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)
```

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)
tempplot(nasa.gam, term="month", se=TRUE, lw=3, cex.lab=1.25)
tempplot(nasa.gam, term="Lon", se=TRUE, lw=3, 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?
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
# Predict life expectancy from year, population, GDP and continent:

# 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)
```

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
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))
```

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

Here, I plot the $t$-statistics, $t = \frac{b_j}{se(b_j)}$ 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/

ggplot extensions: GGally

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

```r
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: ggrepel

```r
devtools::install_github("slowkow/ggrepel")
library(ggplot2)
library(ggrepel)
ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point(color = "red") + geom_text_repel(aes(label = rownames(mtcars))) + theme_classic(base_size = 16)
```

Plotting text labels is often difficult. ggrepel provides geoms for ggplot2 to repel overlapping text labels.

---

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.

Install from github:

```r
devtools::install_github("dgrtwo/gganimate")
```

```r
p5 <- ggplot(gapminder, aes(x = gdpPercap, y = 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
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)
```


---

ggthemes

**ggthemes** provides a large number of extra geoms, scales, and themes for ggplot.

```r
install.packages('ggthemes', dependencies = TRUE)
```

```r
+ theme_tufte() + theme_economist() + theme_fivethirtyeight()
```
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

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

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