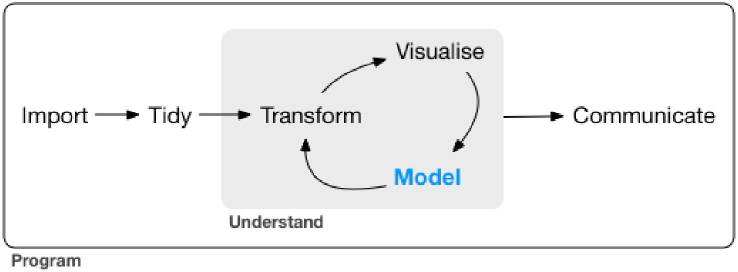
GR5206: lecture 8

Computational Statistics
And Introduction to Data Science

Thibault Vatter
Department of Statistics, Columbia University

11/01/2019

Today



- First:
  - how models work mechanistically (focus on linear models),
  - how to use models to find patterns in real data.
- Then:
  - how to use **many** simple models,
  - how to combine modeling and programming tools.
- As usual, material borrowed from R for data science.

Outline

1 Model basics

2 Model building

3 Many models

4 List-columns

Outline

1 Model basics

2 Model building

3 Many models

4 List-columns

Hypothesis generation/confirmation

- Each observation can either be used for exploration **OR** confirmation, not both.
- You can use an observation
  - as many times as you like for exploration,
  - only once for confirmation.
- When using an observation twice, switch from confirmation to exploration.

Models

- Goals:
  - Provide a simple low-dimensional summary of a dataset.
  - Often partition data into patterns and residuals.
  - Help peel back layers of structure (since strong patterns hide subtler trends).
- The two components of a model:
  - **Family of models**: a precise, but generic, pattern to capture.
    - A straight line, or a quadatric curve.
    - Equations like `y = a1 * x + a2` or `y = a1 * x ^ a2` (with `x` and `y` known variables and `a1` and `a2` parameters).
  - **Fitted model**: member of the family that is closest to the data.
    - `y = 3 * x + 7` or `y = 9 * x ^ 2`.

Word of caution

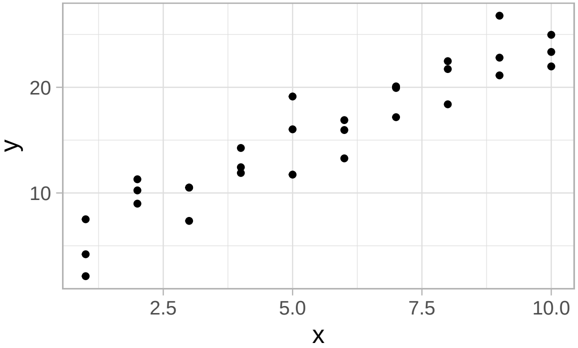
All models are wrong, but some are useful. —George Box

- A fitted model is "just" the closest model to the data from a family of models.
- The "best" model (according to some criteria):
  - isn't necessarily a good model,
  - isn't necessarily "true".
- **The goal is not to uncover truth, but to discover useful approximations.**

```r
library(tidyverse)
library(modelr)
```

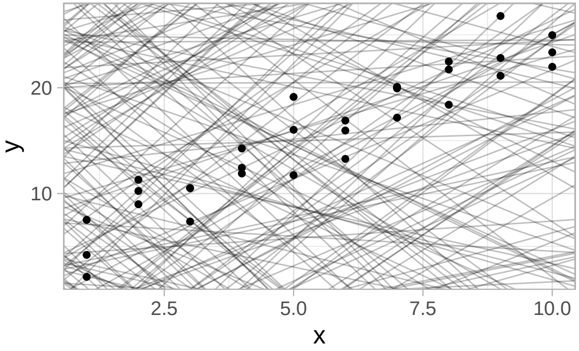
A simulated dataset

```r
ggplot(sim1, aes(x, y)) +
  geom_point(size = 2)
```

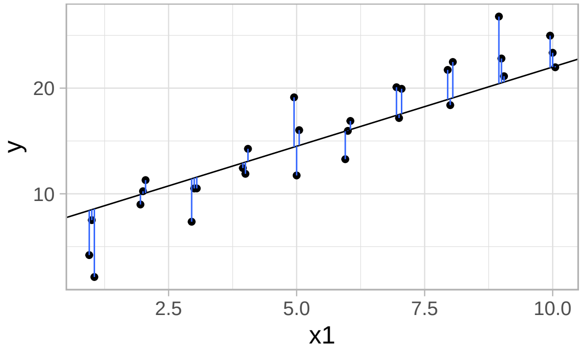


Linear family?

```r
models <- tibble(a1 = runif(250, -20, 40),
                 a2 = runif(250, -5, 5))

ggplot(sim1, aes(x, y)) +
  geom_point(size = 2) +
  geom_abline(aes(intercept = a1, slope = a2),
              data = models, alpha = 1/4)
```



Distance between data and model



- This distance is the difference between
  - the y value given by the model (the **prediction**),
  - and the actual y value in the data (the **response**).

Model family and RMSE

- The model family:

```
model1 <- function(a, data) a[1] + data$x * a[2]

model1(c(7, 1.5), sim1)
#>  [1]  8.5  8.5  8.5 10.0 10.0 10.0 11.5 11.5 11.5 13.0 13.0 13.0 14.5
#> [14] 14.5 14.5 16.0 16.0 16.0 17.5 17.5 17.5 19.0 19.0 19.0 20.5 20.5
#> [27] 20.5 22.0 22.0 22.0
```

- Root-mean-square error (RMSE):

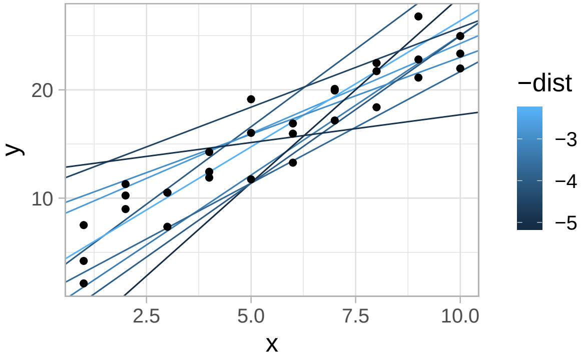
```
measure_distance <- function(mod, data) {
  diff <- data$y - model1(mod, data)
  sqrt(mean(diff ^ 2))}

measure_distance(c(7, 1.5), sim1)
#> [1] 2.67
```

RMSE for each model

```
sim1_dist <- function(a1, a2) measure_distance(c(a1, a2), sim1)

(models <- models %>% mutate(dist = map2_dbl(a1, a2, sim1_dist)))
#> # A tibble: 250 x 3
#>        a1      a2  dist
#>     <dbl>   <dbl> <dbl>
#>  1 -15.2   0.0889  30.8
#>  2  30.1  -0.827   13.2
#>  3  16.0   2.27    13.2
#>  4 -10.6   1.38    18.7
#>  5 -19.6  -1.04    41.8
#>  6   7.98  4.59    19.3
#>  7   9.87 -2.01    20.5
#>  8  -2.61 -4.50    46.9
#>  9  24.0   0.762   13.4
#> 10  26.4  -2.82    14.9
#> # ... with 240 more rows
```

The 10 best models

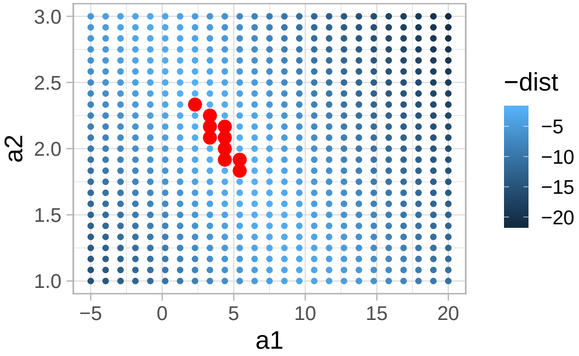
```r
ggplot(sim1, aes(x, y)) +
  geom_abline(aes(intercept = a1, slope = a2, color = -dist),
              data = models %>% filter(rank(dist) <= 10)) +
    geom_point(size = 2)
```



Grid search

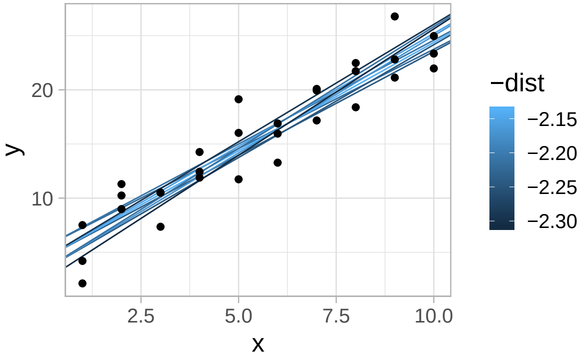
Columbia University in the City of New York

```r
grid <- expand.grid(a1 = seq(-5, 20, length = 25),
                    a2 = seq(1, 3, length = 25)) %>%
  mutate(dist = map2_dbl(a1, a2, sim1_dist))

ggplot(grid, aes(a1, a2)) +
  geom_point(aes(color = -dist)) +
  geom_point(data = grid %>% filter(rank(dist) <= 10),
             size = 4, color = "red")
```



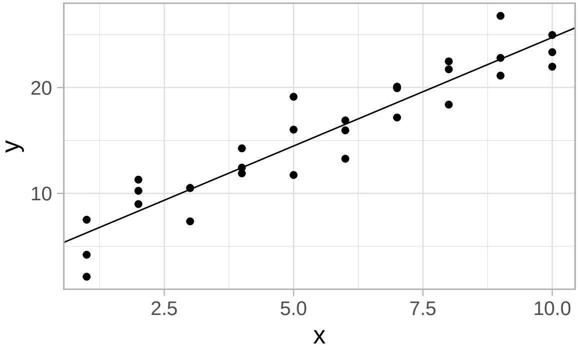
Grid search cont’d

```r
ggplot(sim1, aes(x, y)) +
  geom_abline(aes(intercept = a1, slope = a2, color = -dist),
              data = grid %>% filter(rank(dist) <= 10)) +
    geom_point(size = 2)
```



Newton-Raphson search

```r
(best <- optim(c(0, 0), measure_distance, data = sim1)$par)
#> [1] 4.22 2.05

ggplot(sim1, aes(x, y)) +
  geom_point(size = 2) +
  geom_abline(intercept = best[1], slope = best[2])
```



Linear models

- General form:
  - `y = a1 + a2 * x_1 + a3 * x_2 + ... + an * x_(n - 1)`.
- `lm()`:
  - Specify the model family using formulas!
  - E.g., `y ~ x` is translated to a function like `y = a1 + a2 * x`.

```
sim1_mod <- lm(y ~ x, data = sim1)
coef(sim1_mod)
#> (Intercept)           x
#>        4.22        2.05
```

- Two interesting quantities to look at:
  - The **predictions**.
  - The **residuals**.
- But before that...
  - A statistics digression.
  - Material borrowed from Prof. Avella-Medina.

The statistical model

- Assume the sample of pairs $(\mathbf{X}_i, Y_i)$ is such that

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_1 X_{i2} + \cdots + \beta_d X_{id} + \varepsilon_i, \; i = 1, \ldots, n$$

- Where
  - Response variable: Y_i
  - Covariates: $X_{i1}, X_{i2}, \ldots, X_{id}$
  - Noise term: ε_i, assumed to be i.i.d. with $\mathbb{E}[\varepsilon_i] = 0$ and $\mathrm{cov}(X_{ij}, \varepsilon_i) = 0$ for all $j = 1, \ldots d$.

- In matrix form:

$$\underbrace{\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} 1 & X_{11} & X_{12} & \ldots & X_{1d} \\ 1 & X_{21} & X_{22} & \ldots & X_{2d} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & X_{n1} & X_{n2} & \ldots & X_{nd} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}}_{\boldsymbol{\beta}} + \underbrace{\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}}_{\boldsymbol{\varepsilon}}$$

Least-squares estimation

- The least squares estimator (LSE) is defined as

$$\begin{aligned}\hat{\beta} &= \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{n}(Y_i - \beta_0 - \sum_{j=1}^{d} X_{ij}\beta_j)^2 = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{n}(Y_i - \mathbf{X}_i^T\beta)^2 \\ &= \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \{(\mathbf{Y} - \mathbf{X}^T\beta)^T(\mathbf{Y} - \mathbf{X}^T\beta)\} = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \|\mathbf{Y} - \mathbf{X}^T\beta\|_2^2\end{aligned}$$

- If $\operatorname{rank}(\mathbf{X}) = 1 + d = p$, we have a closed form solution

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

- Hence, under the model assumptions, $\hat{\beta}$ is unbiased since

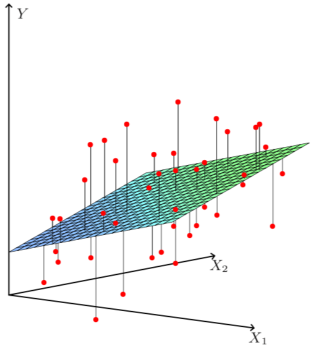
$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X}\beta + \varepsilon) = \beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\varepsilon$$

Least-squares estimation cont'd

- $\hat{Y} = \mathbf{X}\hat{\beta}$ has the geometric interpretation of being the projection of $\mathbf{Y}$ onto the plane spanned by the columns of:

$$\mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} = \mathbf{PY}$$

- Linear least squares fitting with $\mathbf{X} \in \mathbb{R}^2$. We seek the linear function of $\mathbf{X}$ that minimizes the sum of squared residuals from Y (Friedman, Hastie and Tibshirani 2008).



Statistical properties

Assuming we have an i.i.d sample of pairs $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$ one can establish we can establish theoretical guarantees for $\hat{\beta}$

- Consistency

$$\hat{\beta} \xrightarrow[n\to\infty]{\mathcal{P}} \beta$$

- Asymptotic normality

$$\sqrt{n}(\hat{\beta} - \beta) \xrightarrow[n\to\infty]{\mathcal{D}} N(\mathbf{0}, \sigma^2 \mathbf{Q}^{-1}),$$

where $\mathbf{Q} = \mathbb{E}[\mathbf{X}_1 \mathbf{X}_1^T]$.

Normal Linear Model

- Further assuming that the errors ε_i are i.i.d. $N(0, \sigma^2)$
  - $Y_i | \mathbf{X}_i = \mathbf{x}_i \sim N(\mathbf{x}_i^T \beta, \sigma^2)$.
  - $\hat{\beta}$ is also the MLE of β.
  - For a fixed design matrix $\mathbf{X}$

$$\hat{\beta} \sim N_p(\beta, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}).$$

  - $\hat{\sigma}^2 = \frac{1}{n-p}\|\mathbf{Y} - \mathbf{X}\hat{\beta}\|_2^2$ is an unbiased estimator of σ^2. Indeed, one can show that $(n-p)\frac{\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-p}$.
  - $(n-p)\frac{\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-p}$
  - $\hat{\beta}$ and $\hat{\sigma}^2$ are independent!
  - Hence

$$(\hat{\beta}_j - \beta_j)/\hat{\sigma}_{\hat{\beta}_j} \sim t_{n-p},$$

where $\hat{\sigma}^2_{\hat{\beta}_j}$ is the jth diagonal element of $\hat{\sigma}^2(\mathbf{X}^T\mathbf{X})^{-1}$.

Normal linear model cont'd

- Log likelihood function:

$$\ell_n(\beta) = \log L_n(\beta, \sigma^2) = -\frac{1}{2}\left\{ n \log \sigma^2 + \frac{1}{\sigma^2} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \beta)^2 \right\}$$

- Since $\hat{\beta} = \hat{\boldsymbol{\beta}}_{ML}$ we see that

$$\hat{\sigma}^2_{ML} = \frac{1}{n} \|\mathbf{Y} - \mathbf{X}\hat{\beta}\|_2^2$$

- Hence, log likelihood is maximized at

$$\begin{aligned} \ell_n(\beta) = \log L_n(\hat{\beta}, \hat{\sigma}^2_{ML}) &= -\frac{1}{2}\left\{ n \log \hat{\sigma}^2_{ML} + n \right\} \\ &= -\frac{1}{2}\left\{ n \log \|\mathbf{Y} - \mathbf{X}\hat{\beta}\|_2^2 + n - n \log n \right\} \end{aligned}$$

Normal linear model cont'd

- When comparing two nested models it is useful to write

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon = \begin{bmatrix}\mathbf{X}_1 & \mathbf{X}_2\end{bmatrix}\begin{bmatrix}\beta_1 \\ \beta_2\end{bmatrix} + \varepsilon = \mathbf{X}_1\beta_1 + \mathbf{X}_2\beta_2 + \varepsilon,$$

  where $\mathbf{X}_1$ is $n \times q$ and $\mathbf{X}_2$ is $n \times (p - q)$.
- Likelihood ratio statistic:

$$2\{\ell_n(\hat{\beta}) - \ell_n(\hat{\beta}_1^R)\} = n \log\left(\frac{\|\mathbf{Y} - \mathbf{X}_1^T\hat{\beta}_1^R\|_2^2}{\|\mathbf{Y} - \mathbf{X}^T\hat{\beta}\|_2^2}\right) = n \log\left(1 + \frac{p-q}{n-p}F\right)$$

- F-statistic:

$$F = \frac{n-p}{p-q}\frac{\|\mathbf{Y} - \mathbf{X}^T\hat{\beta}_1^R\|_2^2 - \|\mathbf{Y} - \mathbf{X}^T\hat{\beta}\|_2^2}{\|\mathbf{Y} - \mathbf{X}^T\hat{\beta}\|_2^2} \sim F_{p-q,n-p}$$

Normal linear model in R

```
summary(sim1_mod)
#>
#> Call:
#> lm(formula = y ~ x, data = sim1)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -4.147 -1.520  0.133  1.467  4.652
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)    4.221      0.869    4.86  4.1e-05 ***
#> x              2.052      0.140   14.65  1.2e-14 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.2 on 28 degrees of freedom
#> Multiple R-squared:  0.885,  Adjusted R-squared:  0.88
#> F-statistic:  215 on 1 and 28 DF,  p-value: 1.17e-14
```

Normal linear model in R cont'd

```
sim0_mod <- lm(y ~ 1, data = sim1)
anova(sim0_mod, sim1_mod)
#> Analysis of Variance Table
#>
#> Model 1: y ~ 1
#> Model 2: y ~ x
#>   Res.Df  RSS Df Sum of Sq   F  Pr(>F)
#> 1     29 1178
#> 2     28  136  1      1042 215 1.2e-14 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- ... end of the digression!
- Two interesting quantities to look at:
  - The **predictions**.
  - The **residuals**.

Visualizing predictions in 3 steps

- Step 1

```
(grid <- sim1 %>%
   modelr::data_grid(x))
#> # A tibble: 10 x 1
#>         x
#>     <int>
#>  1      1
#>  2      2
#>  3      3
#>  4      4
#>  5      5
#>  6      6
#>  7      7
#>  8      8
#>  9      9
#> 10     10
```

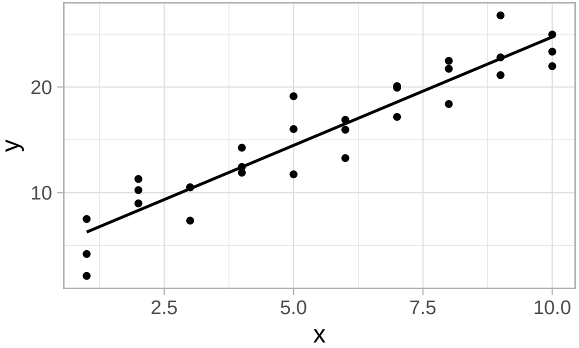
- Step 2

```
(grid <- grid %>%
   add_predictions(sim1_mod))
#> # A tibble: 10 x 2
#>         x  pred
#>     <int> <dbl>
#>  1      1  6.27
#>  2      2  8.32
#>  3      3 10.4
#>  4      4 12.4
#>  5      5 14.5
#>  6      6 16.5
#>  7      7 18.6
#>  8      8 20.6
#>  9      9 22.7
#> 10     10 24.7
```

Visualizing predictions in 3 steps

- Step 3

```
ggplot(sim1, aes(x, y)) +
  geom_point(size = 2) +
  geom_line(aes(y = pred), data = grid, size = 1)
```



Visualizing residuals in three steps

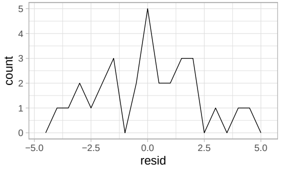
- Step 1

```
(sim1 <- sim1 %>%
   add_residuals(sim1_mod))
#> # A tibble: 30 x 3
#>         x     y    resid
#>     <int> <dbl>    <dbl>
#>  1      1  4.20 -2.07
#>  2      1  7.51  1.24
#>  3      1  2.13 -4.15
#>  4      2  8.99  0.665
#>  5      2 10.2   1.92
#>  6      2 11.3   2.97
#>  7      3  7.36 -3.02
#>  8      3 10.5   0.130
#>  9      3 10.5   0.136
#> 10      4 12.4   0.00763
#> # ... with 20 more rows
```

Visualizing residuals in three steps

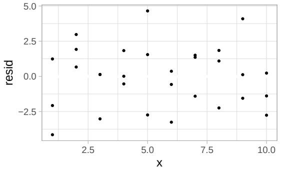
- Step 2

```
ggplot(sim1, aes(resid)) +
  geom_freqpoly(binwidth = 0.5)
```



- Step 3

```
ggplot(sim1, aes(x, resid)) +
  geom_ref_line(h = 0) +
  geom_point()
```



Formulas

- What's that?
  - A way of getting "special behavior".
  - "Capture variables" so they can be interpreted by the function.
  - Sometimes called "Wilkinson-Rogers notation" from Symbolic Description of Factorial Models for Analysis of Variance
- Behind the scenes:

```
df <- tribble(~y, ~x1, ~x2,
              4, 2, 5,
              5, 1, 6)
model_matrix(df, y ~ x1)
#> # A tibble: 2 x 2
#>   `(Intercept)`    x1
#>           <dbl> <dbl>
#> 1             1     2
#> 2             1     1
```

Formulas cont'd

- Without intercept:

```
model_matrix(df, y ~ x1 - 1)
#> # A tibble: 2 x 1
#>      x1
#>   <dbl>
#> 1     2
#> 2     1
```

- Adding a second variable:

```
model_matrix(df, y ~ x1 + x2)
#> # A tibble: 2 x 3
#>   `(Intercept)`    x1    x2
#>           <dbl> <dbl> <dbl>
#> 1             1     2     5
#> 2             1     1     6
```

Categorical variables

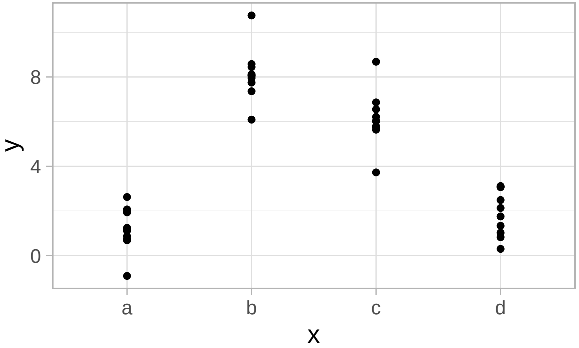
Columbia University in the City of New York

```
df <- tribble(~ sex, ~ response,
              "male", 1,
              "female", 2,
              "male", 1)
model_matrix(df, response ~ sex)
#> # A tibble: 3 x 2
#>   `(Intercept)` sexmale
#>           <dbl>   <dbl>
#> 1             1       1
#> 2             1       0
#> 3             1       1
```

- Why doesn't R also create a `sexfemale` column?

Another simulated dataset

```r
ggplot(sim2, aes(x, y)) +
  geom_point(size = 2)
```

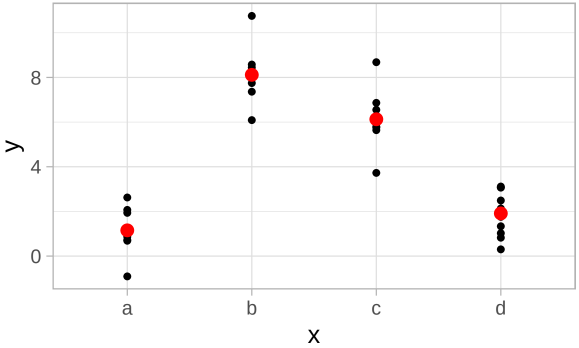


Linear model and predictions

```
mod2 <- lm(y ~ x, data = sim2)

(grid <- sim2 %>%
    data_grid(x) %>%
    add_predictions(mod2))
#> # A tibble: 4 x 2
#>   x      pred
#>   <chr> <dbl>
#> 1 a      1.15
#> 2 b      8.12
#> 3 c      6.13
#> 4 d      1.91
```

Visualize the results

```
ggplot(sim2, aes(x)) +
  geom_point(aes(y = y), size = 2) +
  geom_point(data = grid, aes(y = pred), color = "red", size = 4)
```

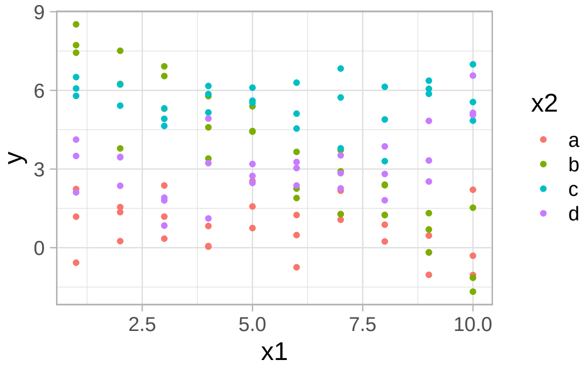


What’s happening here?

```
tibble(x = "e") %>%
  add_predictions(mod2)
#> Error in model.frame.default(Terms, newdata, na.action = na.action,
#> xlev = object$xlevels): factor x has new level e
```

Interactions (cont. and cat.)

```r
ggplot(sim3, aes(x1, y)) +
  geom_point(aes(color = x2))
```



Two possible models

```
mod1 <- lm(y ~ x1 + x2, data = sim3)
mod2 <- lm(y ~ x1 * x2, data = sim3)
```

- Note that:
  - `y ~ x1 + x2` becomes `y = a0 + a1 * x1 + a2 * x2`.
  - `y ~ x1 * x2` becomes `y = a0 + a1 * x1 + a2 * x2 + a12 * x1 * x2`.

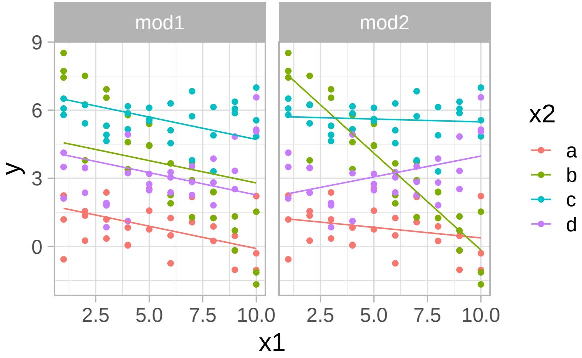
Two new tricks to visualize them

- Give `data_grid()` both variables.
- To generate predictions from both models simultaneously, use
  - `gather_predictions()` to add predictions as rows,
  - or `spread_predictions()` to add predictions as columns.

```
(grid <- sim3 %>%
   data_grid(x1, x2) %>%
   gather_predictions(mod1, mod2))
#> # A tibble: 80 x 4
#>    model    x1 x2     pred
#>    <chr> <int> <fct> <dbl>
#>  1 mod1      1 a      1.67
#>  2 mod1      1 b      4.56
#>  3 mod1      1 c      6.48
#>  4 mod1      1 d      4.03
#>  5 mod1      2 a      1.48
#>  6 mod1      2 b      4.37
#>  7 mod1      2 c      6.28
#>  8 mod1      2 d      3.84
#>  9 mod1      3 a      1.28
#> 10 mod1      3 b      4.17
#> # ... with 70 more rows
```

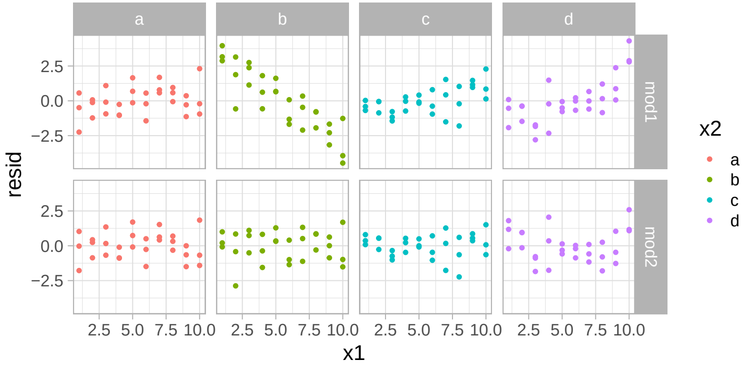
Using facetting

```
ggplot(sim3, aes(x1, y, color = x2)) +
  geom_point() +
  geom_line(data = grid, aes(y = pred)) +
  facet_wrap(~ model)
```



Which model is better?

```
sim3 <- sim3 %>% gather_residuals(mod1, mod2)

ggplot(sim3, aes(x1, resid, color = x2)) +
  geom_point() +
  facet_grid(model ~ x2)
```



Which model is better?

- Remember slide 23

```
anova(mod1, mod2)
#> Analysis of Variance Table
#>
#> Model 1: y ~ x1 + x2
#> Model 2: y ~ x1 * x2
#>   Res.Df RSS Df Sum of Sq    F Pr(>F)
#> 1    115 270
#> 2    112 118  3       153 48.5 <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

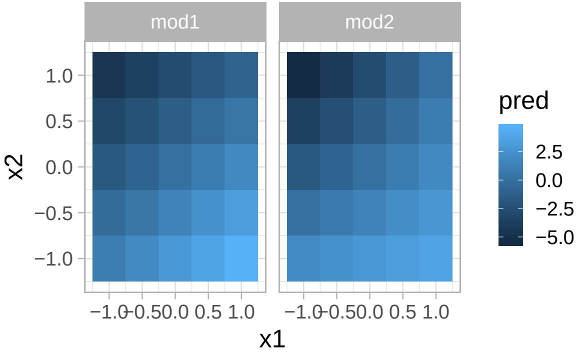
Interactions (two continuous)

```
mod1 <- lm(y ~ x1 + x2, data = sim4)
mod2 <- lm(y ~ x1 * x2, data = sim4)

(grid <- sim4 %>%
        data_grid(x1 = seq_range(x1, 5), x2 = seq_range(x2, 5)) %>%
        gather_predictions(mod1, mod2))
#> # A tibble: 50 x 4
#>    model    x1    x2   pred
#>    <chr> <dbl> <dbl>  <dbl>
#>  1 mod1   -1    -1    0.996
#>  2 mod1   -1    -0.5 -0.395
#>  3 mod1   -1     0   -1.79
#>  4 mod1   -1     0.5 -3.18
#>  5 mod1   -1     1   -4.57
#>  6 mod1   -0.5  -1    1.91
#>  7 mod1   -0.5  -0.5  0.516
#>  8 mod1   -0.5   0   -0.875
#>  9 mod1   -0.5   0.5 -2.27
#> 10 mod1   -0.5   1   -3.66
#> # ... with 40 more rows
```

`?seq_range` for other arguments (e.g., `pretty = TRUE` for tables).

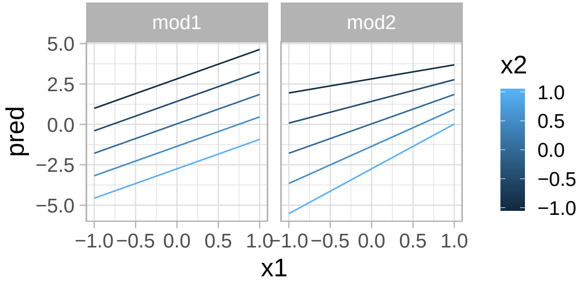
Visualize

```
ggplot(grid, aes(x1, x2)) + geom_tile(aes(fill = pred)) +
  facet_wrap(~ model)
```



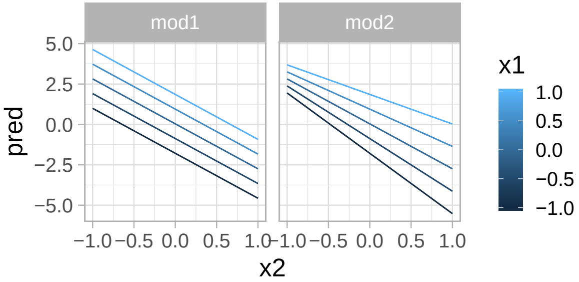
Slices with respect to x2

```r
ggplot(grid, aes(x1, pred, color = x2, group = x2)) +
  geom_line() +
  facet_wrap(~ model)
```



Slices with respect to x1

```
ggplot(grid, aes(x2, pred, color = x1, group = x1)) +
  geom_line() +
  facet_wrap(~ model)
```



Which model is better?

- Remember slide 23

```
anova(mod1, mod2)
#> Analysis of Variance Table
#>
#> Model 1: y ~ x1 + x2
#> Model 2: y ~ x1 * x2
#>   Res.Df  RSS Df Sum of Sq    F Pr(>F)
#> 1    297 1323
#> 2    296 1278  1      45.2 10.5 0.0014 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Transformations

- `log(y) ~ sqrt(x1) + x2` is transformed to `log(y) = a1 + a2 * sqrt(x1) + a3 * x2`.
- If the transformation involves `+`, `*`, `^`, or `-`, wrap it in `I()`:
  - `y ~ x + I(x ^ 2)` ≡ `y = a1 + a2 * x + a3 * x^2`.
  - `y ~ x ^ 2 + x` ≡ `y ~ x * x + x` ≡ `y = a1 + a2 * x`.

```
df <- tribble(~y, ~x, 1,  1, 2,  2, 3,  3)
model_matrix(df, y ~ x^2 + x)
model_matrix(df, y ~ I(x^2) + x)
#> # A tibble: 3 x 2
#>   `(Intercept)`     x
#>           <dbl> <dbl>
#> 1             1     1
#> 2             1     2
#> 3             1     3
#> # A tibble: 3 x 3
#>   `(Intercept)` `I(x^2)`     x
#>           <dbl>    <dbl> <dbl>
#> 1             1        1     1
#> 2             1        4     2
#> 3             1        9     3
```

Polynomial approximations with `poly`

- To get `y = a1 + a2 * x + a3 * x^2`:

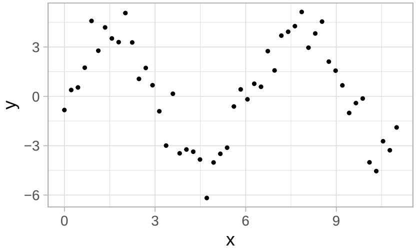
```
model_matrix(df, y ~ poly(x, 2))
#> # A tibble: 3 x 3
#>   `(Intercept)` `poly(x, 2)1` `poly(x, 2)2`
#>           <dbl>         <dbl>         <dbl>
#> 1             1     -7.07e- 1         0.408
#> 2             1     -7.85e-17        -0.816
#> 3             1      7.07e- 1         0.408
```

Natural splines

```
library(splines)
model_matrix(df, y ~ ns(x, 2))
#> # A tibble: 3 x 3
#>   `(Intercept)` `ns(x, 2)1` `ns(x, 2)2`
#>           <dbl>       <dbl>       <dbl>
#> 1             1       0           0
#> 2             1       0.566      -0.211
#> 3             1       0.344       0.771
```

A non-linear function

```r
sim5 <- tibble(x = seq(0, 3.5 * pi, length = 50),
               y = 4 * sin(x) + rnorm(length(x)))

ggplot(sim5, aes(x, y)) + geom_point()
```

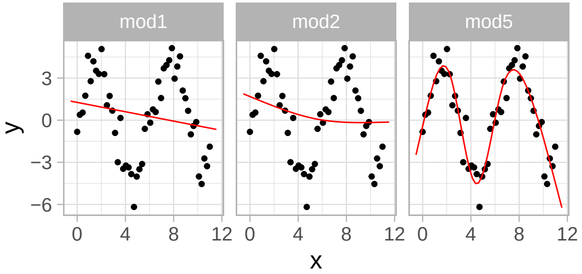


- Three models using splines:

```r
mod1 <- lm(y ~ ns(x, 1), data = sim5)
mod2 <- lm(y ~ ns(x, 2), data = sim5)
mod5 <- lm(y ~ ns(x, 5), data = sim5)
```

Visualize

```r
grid <- sim5 %>%
  data_grid(x = seq_range(x, n = 50, expand = 0.1)) %>%
  gather_predictions(mod1, mod2, mod5, .pred = "y")

ggplot(sim5, aes(x, y)) + geom_point() +
  geom_line(data = grid, color = "red") +
  facet_wrap(~ model)
```



Missing values

```
options(na.action = na.warn)
df <- tribble(~x, ~y,
              1, 2.2,
              2, NA,
              3, 3.5,
              4, 8.3,
              NA, 10)

mod <- lm(y ~ x, data = df)
#> Warning: Dropping 2 rows with missing values
mod <- lm(y ~ x, data = df, na.action = na.exclude)
nobs(mod)
#> [1] 3
```

Other model families

- **Generalized linear models**, e.g. `stats::glm()`:
  - While LMs assume continuous responses/Gaussian errors, GLMs extend them to other distributions, including non-continuous responses (e.g. binary data or counts).
- **Generalized additive models**, e.g. `mgcv::gam()`:
  - Extend GLMs to incorporate smooth functions
  - Formulas like `y ~ s(x)` become equations like `y = f(x)`.
- **Penalized linear models**, e.g. `glmnet::glmnet()`:
  - Add penalties to favor simpler models and "generalize" better.
- **Robust linear models**, e.g. `MASS:rlm()`:
  - Tweaks distance to downweight outliers.
  - Less sensitive to outliers, but sligthly worse without outliers.
- **Trees**, e.g. `rpart::rpart()`:
  - Piece-wise constant models splitting the data into small pieces.
  - Powerful when aggregated as **random forests** (e.g. `randomForest::randomForest()`) or **gradient boosting machines** (e.g. `xgboost::xgboost()`).

Outline

1 Model basics

2 Model building

3 Many models

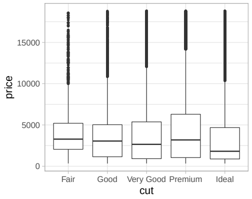
4 List-columns

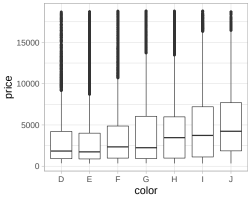
Model building

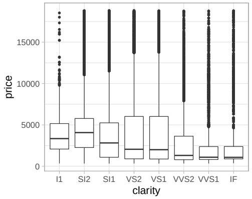
- To partition data into pattern and residuals:
  - Find patterns with visualization.
  - Make them concrete and precise with a model.
  - Repeat 1. and 2. after replacing the old response variable with the residuals from the model.
- How about large and complex datasets?
  - ML approaches "simply" focus on predictive ability.
  - Issues:
    - black boxes,
    - (sometimes) hard to use domain knowledge,
    - (often) difficult to assess whether or not the model will continue to work in the long-term
  - Usually, a combination of both approaches is preferred.

The diamonds dataset

```r
ggplot(diamonds, aes(cut, price)) + geom_boxplot()
ggplot(diamonds, aes(color, price)) + geom_boxplot()
ggplot(diamonds, aes(clarity, price)) + geom_boxplot()
```





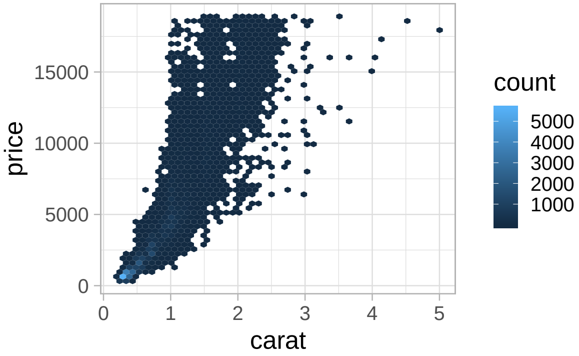


- Why are low quality diamonds more expensive?

Price and carat

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

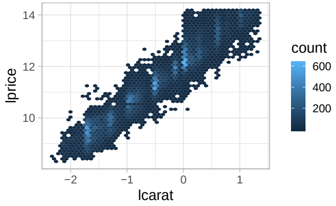
```r
ggplot(diamonds, aes(carat, price)) +
  geom_hex(bins = 50)
```



A couple of tweaks

- Focus on diamonds < 2.5 carats (99.7% of the data).
- Log-transform the carat and price.

```
diamonds2 <- diamonds %>%
  filter(carat <= 2.5) %>%
  mutate(lprice = log2(price), lcarat = log2(carat))

ggplot(diamonds2, aes(lcarat, lprice)) +
  geom_hex(bins = 50)
```

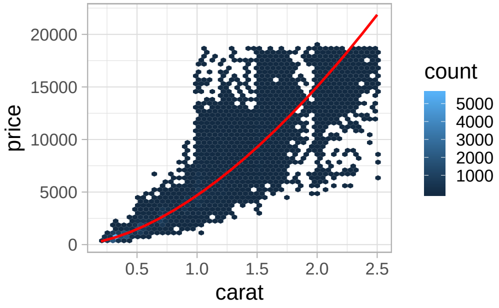


- A simple model:

```
mod_diamond <- lm(lprice ~ lcarat, data = diamonds2)
```

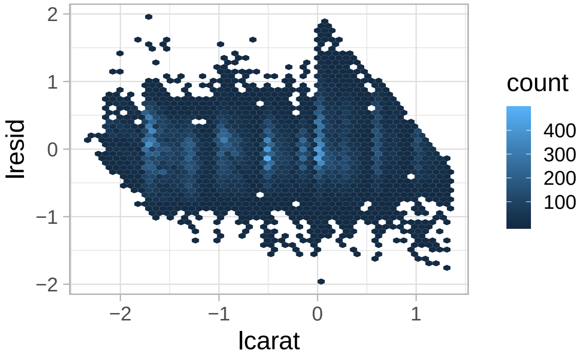
Visualize the predictions

```
grid <- diamonds2 %>%
  data_grid(carat = seq_range(carat, 20)) %>%
  mutate(lcarat = log2(carat)) %>%
  add_predictions(mod_diamond, "lprice") %>%
  mutate(price = 2 ^ lprice)

ggplot(diamonds2, aes(carat, price)) +
  geom_hex(bins = 50) +
  geom_line(data = grid, color = "red", size = 1)
```



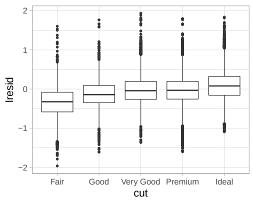
Visualize the residuals

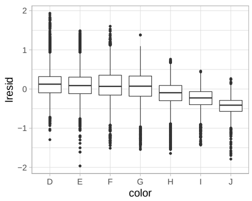
```
diamonds2 <- diamonds2 %>%
  add_residuals(mod_diamond, "lresid")

ggplot(diamonds2, aes(lcarat, lresid)) +
  geom_hex(bins = 50)
```

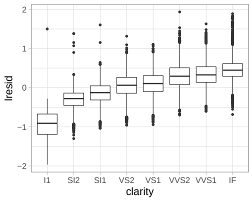


Replace price by residuals

```
ggplot(diamonds2, aes(cut, lresid)) + geom_boxplot()
ggplot(diamonds2, aes(color, lresid)) + geom_boxplot()
ggplot(diamonds2, aes(clarity, lresid)) + geom_boxplot()
```





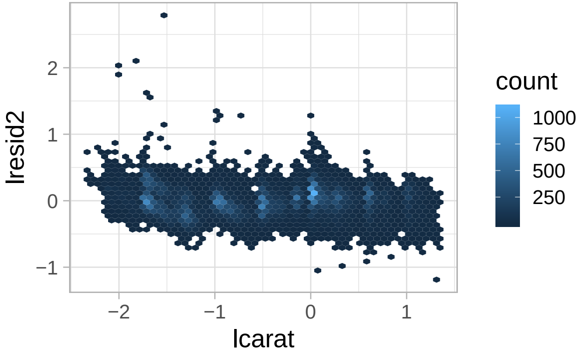


A more complicated model

```
mod_diamond2 <- lm(lprice ~ lcarat + color + cut + clarity,
                   data = diamonds2)

(grid <- diamonds2 %>%
        data_grid(cut,
                  lcarat = -0.515,
                  color = "G",
                  clarity = "SI1") %>%
        add_predictions(mod_diamond2))
#> # A tibble: 5 x 5
#>   cut       lcarat color clarity  pred
#>   <ord>      <dbl> <chr> <chr>   <dbl>
#> 1 Fair      -0.515 G     SI1      11.0
#> 2 Good      -0.515 G     SI1      11.1
#> 3 Very Good -0.515 G     SI1      11.2
#> 4 Premium   -0.515 G     SI1      11.2
#> 5 Ideal     -0.515 G     SI1      11.2
```

Visualize the residuals

```r
diamonds2 <- diamonds2 %>%
  add_residuals(mod_diamond2, "lresid2")

ggplot(diamonds2, aes(lcarat, lresid2)) +
  geom_hex(bins = 50)
```



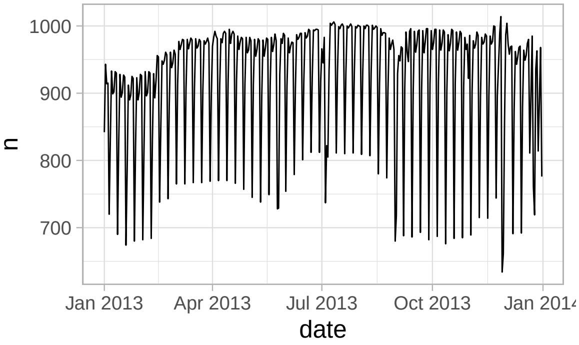
The number of daily flights

```
library(nycflights13)
library(lubridate)
daily <- flights %>%
     mutate(date = make_date(year, month, day)) %>%
  group_by(date) %>%
  summarize(n = n())
```

What affects this number?

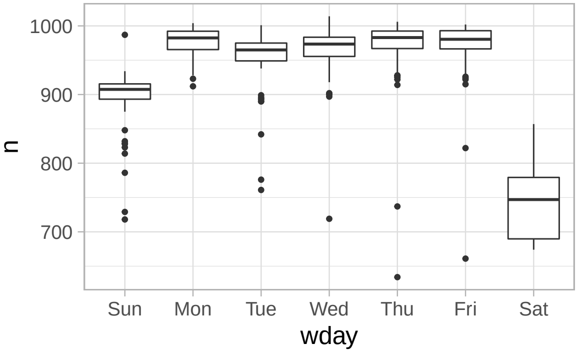
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
ggplot(daily, aes(date, n)) +
  geom_line()
```



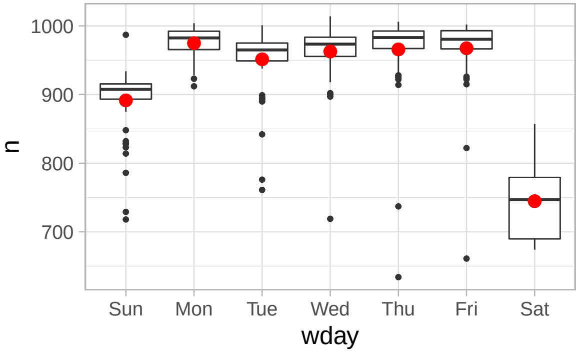
Day of week

```r
daily <- daily %>% mutate(wday = wday(date, label = TRUE))

ggplot(daily, aes(wday, n)) + geom_boxplot()

mod <- lm(n ~ wday, data = daily)
```



Visualize the predictions

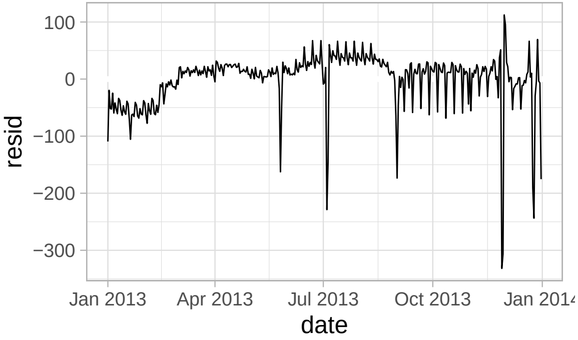
```r
grid <- daily %>%
  data_grid(wday) %>%
  add_predictions(mod, "n")

ggplot(daily, aes(wday, n)) +
  geom_boxplot() +
  geom_point(data = grid, color = "red", size = 4)
```



Visualize the residuals

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

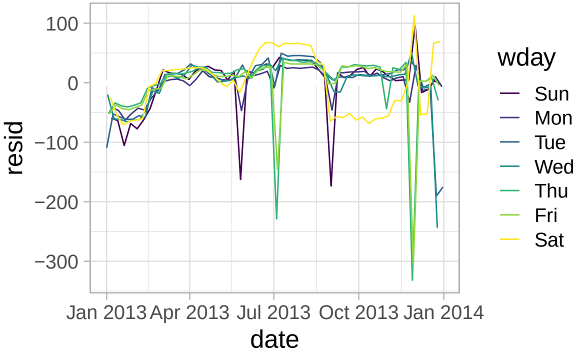
```
daily <- daily %>%
  add_residuals(mod)


daily %>%
  ggplot(aes(date, resid)) +
  geom_ref_line(h = 0) + geom_line()
```



What happens here?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```r
ggplot(daily, aes(date, resid, color = wday)) +
    geom_ref_line(h = 0) + geom_line()
```

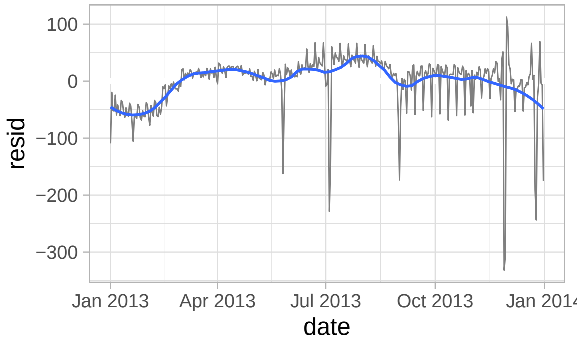


What happens here?

```
daily %>%
  filter(resid < -100)
#> # A tibble: 11 x 4
#>    date           n wday  resid
#>    <date>     <int> <ord> <dbl>
#>  1 2013-01-01   842 Tue   -109.
#>  2 2013-01-20   786 Sun   -105.
#>  3 2013-05-26   729 Sun   -162.
#>  4 2013-07-04   737 Thu   -229.
#>  5 2013-07-05   822 Fri   -145.
#>  6 2013-09-01   718 Sun   -173.
#>  7 2013-11-28   634 Thu   -332.
#>  8 2013-11-29   661 Fri   -306.
#>  9 2013-12-24   761 Tue   -190.
#> 10 2013-12-25   719 Wed   -244.
#> 11 2013-12-31   776 Tue   -175.
```

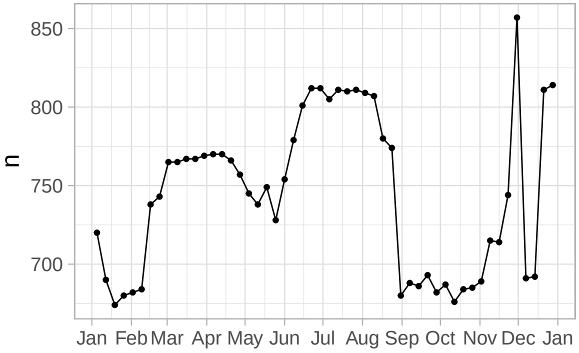
What happens here?

```
daily %>%
    ggplot(aes(date, resid)) +
    geom_ref_line(h = 0) +
    geom_line(color = "grey50") +
    geom_smooth(se = FALSE, span = 0.20)
#> `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Seasonal Saturday effect

```r
daily %>%
  filter(wday == "Sat") %>%
  ggplot(aes(date, n)) +
  geom_point() + geom_line() +
  scale_x_date(NULL, date_breaks = "1 month", date_labels = "%b")
```



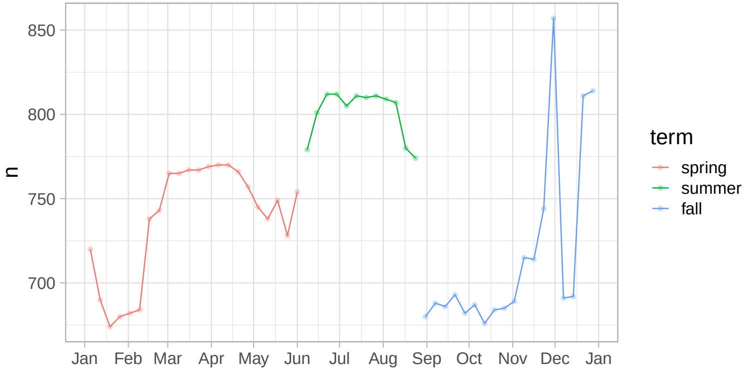
- State's school terms: summer break in 2013 was Jun 26–Sep 9.

The three school terms

```
term <- function(date) {
  cut(date, breaks = ymd(20130101, 20130605, 20130825, 20140101),
      labels = c("spring", "summer", "fall"))
}
daily <- daily %>%
  mutate(term = term(date))

daily
#> # A tibble: 365 x 5
#>    date           n wday   resid term
#>    <date>     <int> <ord>  <dbl> <fct>
#>  1 2013-01-01   842 Tue   -109.  spring
#>  2 2013-01-02   943 Wed    -19.7 spring
#>  3 2013-01-03   914 Thu    -51.8 spring
#>  4 2013-01-04   915 Fri    -52.5 spring
#>  5 2013-01-05   720 Sat    -24.6 spring
#>  6 2013-01-06   832 Sun    -59.5 spring
#>  7 2013-01-07   933 Mon    -41.8 spring
#>  8 2013-01-08   899 Tue    -52.4 spring
#>  9 2013-01-09   902 Wed    -60.7 spring
#> 10 2013-01-10   932 Thu    -33.8 spring
#> # ... with 355 more rows
```

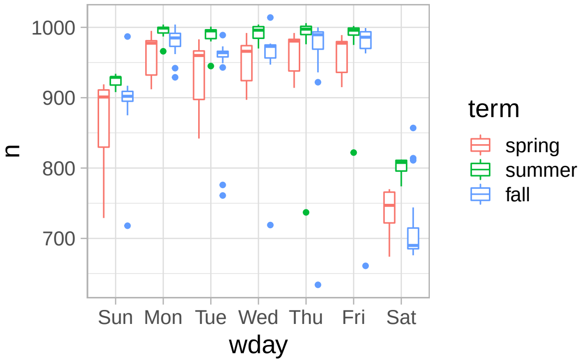
The three school terms cont'd

```
daily %>%
  filter(wday == "Sat") %>%
  ggplot(aes(date, n, color = term)) +
  geom_point(alpha = 1/3) +
  geom_line() +
  scale_x_date(NULL, date_breaks = "1 month", date_labels = "%b")
```



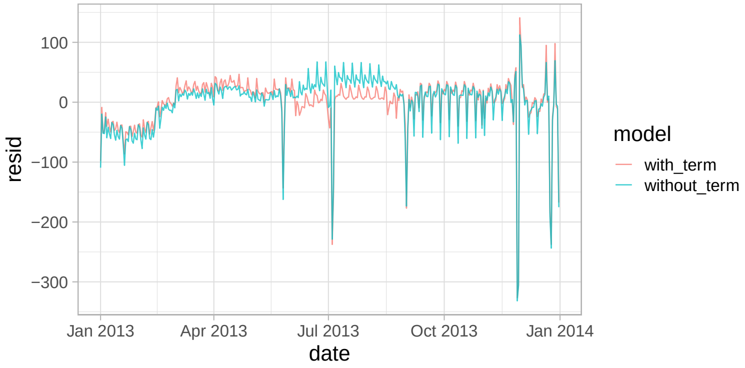
School terms and day of week

```
daily %>%
  ggplot(aes(wday, n, color = term)) +
  geom_boxplot()
```



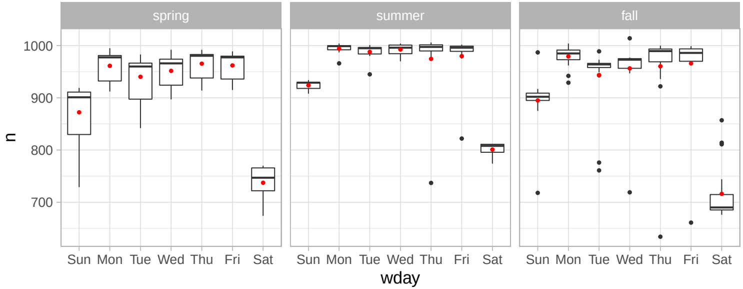
An improved model

```
mod1 <- lm(n ~ wday, data = daily)
mod2 <- lm(n ~ wday * term, data = daily)

daily %>%
  gather_residuals(without_term = mod1, with_term = mod2) %>%
  ggplot(aes(date, resid, color = model)) +
  geom_line(alpha = 0.75)
```



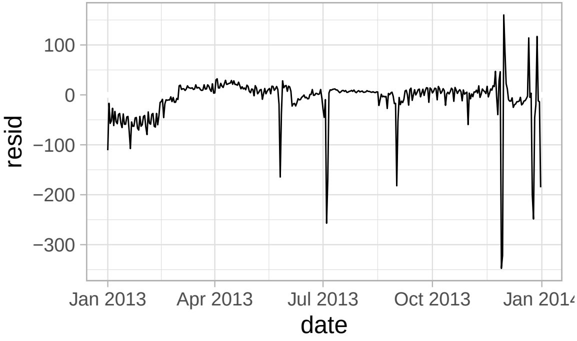
What's going on here?

```r
grid <- daily %>%
  data_grid(wday, term) %>%
  add_predictions(mod2, "n")

ggplot(daily, aes(wday, n)) +
  geom_boxplot() +
  geom_point(data = grid, color = "red") +
  facet_wrap(~ term)
```



Robust fit

```r
mod3 <- MASS::rlm(n ~ wday * term, data = daily)

daily %>%
  add_residuals(mod3, "resid") %>%
  ggplot(aes(date, resid)) +
  geom_hline(yintercept = 0, size = 2, color = "white") +
  geom_line()
```



Computed variables

- Either bundled up into a function:

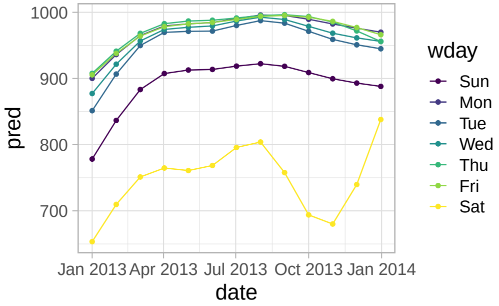
```r
compute_vars <- function(data) {
  data %>%
    mutate(term = term(date),
           wday = wday(date, label = TRUE))
}
```

- Or directly in the model formula:

```r
wday2 <- function(x) wday(x, label = TRUE)
mod3 <- lm(n ~ wday2(date) * term(date), data = daily)
```

Time of year: an alternative approach

```r
library(splines)
mod <- MASS::rlm(n ~ wday * ns(date, 5), data = daily)

daily %>%
  data_grid(wday, date = seq_range(date, n = 13)) %>%
  add_predictions(mod) %>%
  ggplot(aes(date, pred, color = wday)) +
  geom_line() +
  geom_point()
```



Outline

1 Model basics

2 Model building

3 Many models

4 List-columns

Many models

- To work with large numbers of models, use:
  - Many simple models to better understand complex datasets.
  - List-columns to store arbitrary data structures in a data frame.
  - The **broom** package to turn models into tidy data.
- Note that this part
  - is harder than the others,
  - requires a deeper internalization of ideas (e.g., about modeling, data structures, and iteration).

gapminder

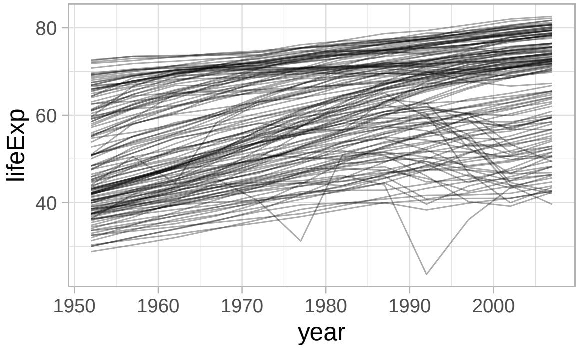
- Summarizes the progression of countries over time using variables like life expectancy and GDP.
- Popularized by Hans Rosling, a Swedish doctor and statistician, in a short video filmed in conjunction with the BBC

```
library(gapminder)
gapminder
#> # A tibble: 1,704 x 6
#>    country     continent  year lifeExp      pop gdpPercap
#>    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
#>  1 Afghanistan Asia       1952    28.8  8425333      779.
#>  2 Afghanistan Asia       1957    30.3  9240934      821.
#>  3 Afghanistan Asia       1962    32.0 10267083      853.
#>  4 Afghanistan Asia       1967    34.0 11537966      836.
#>  5 Afghanistan Asia       1972    36.1 13079460      740.
#>  6 Afghanistan Asia       1977    38.4 14880372      786.
#>  7 Afghanistan Asia       1982    39.9 12881816      978.
#>  8 Afghanistan Asia       1987    40.8 13867957      852.
#>  9 Afghanistan Asia       1992    41.7 16317921      649.
#> 10 Afghanistan Asia       1997    41.8 22227415      635.
#> # ... with 1,694 more rows
```

Focus on three variables

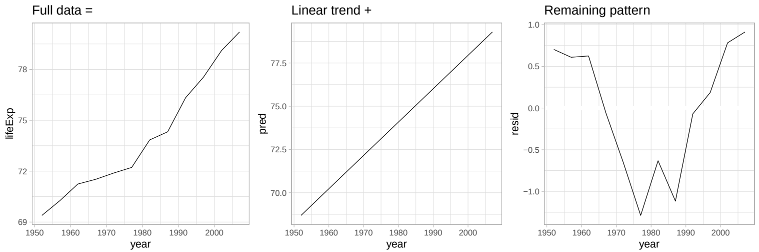
- How does life expectancy (lifeExp) change over time (year) for each country (country)?

```
gapminder %>%
  ggplot(aes(year, lifeExp, group = country)) +
    geom_line(alpha = 1/3)
```



Model for a single country

```r
nz <- filter(gapminder, country == "New Zealand")
nz %>% ggplot(aes(year, lifeExp)) + geom_line() + ggtitle("Full data = ")

nz_mod <- lm(lifeExp ~ year, data = nz)
nz %>% add_predictions(nz_mod) %>%
  ggplot(aes(year, pred)) + geom_line() + ggtitle("Linear trend + ")

nz %>% add_residuals(nz_mod) %>%
  ggplot(aes(year, resid)) +
    geom_hline(yintercept = 0, colour = "white", size = 3) +
    geom_line() + ggtitle("Remaining pattern")
```



Nested data

```
(by_country <- gapminder %>%
  group_by(country, continent) %>%
  nest())
#> # A tibble: 142 x 3
#> # Groups:   country, continent [710]
#>    country     continent               data
#>    <fct>       <fct>     <list<df[,4]>>
#>  1 Afghanistan Asia             [12 x 4]
#>  2 Albania     Europe           [12 x 4]
#>  3 Algeria     Africa           [12 x 4]
#>  4 Angola      Africa           [12 x 4]
#>  5 Argentina   Americas         [12 x 4]
#>  6 Australia   Oceania          [12 x 4]
#>  7 Austria     Europe           [12 x 4]
#>  8 Bahrain     Asia             [12 x 4]
#>  9 Bangladesh  Asia             [12 x 4]
#> 10 Belgium     Europe           [12 x 4]
#> # ... with 132 more rows
```

- In a grouped data frame, each row is an observation.
- In a nested data frame, each row is a group.

List-columns

- A model-fitting function applied to every country:

```
country_model <- function(df) lm(lifeExp ~ year, data = df)
models <- map(by_country$data, country_model)
```

- Or add an additional list-column:

```
(by_country <- by_country %>%
   mutate(model = map(data, country_model)))
#> # A tibble: 142 x 4
#> # Groups:   country, continent [710]
#>    country     continent             data model
#>    <fct>       <fct>     <list<df[,4]>> <list>
#>  1 Afghanistan Asia             [12 x 4] <lm>
#>  2 Albania     Europe           [12 x 4] <lm>
#>  3 Algeria     Africa           [12 x 4] <lm>
#>  4 Angola      Africa           [12 x 4] <lm>
#>  5 Argentina   Americas         [12 x 4] <lm>
#>  6 Australia   Oceania          [12 x 4] <lm>
#>  7 Austria     Europe           [12 x 4] <lm>
#>  8 Bahrain     Asia             [12 x 4] <lm>
#>  9 Bangladesh  Asia             [12 x 4] <lm>
#> 10 Belgium     Europe           [12 x 4] <lm>
#> # ... with 132 more rows
```

Why bother?

- Avoid leaving the list of models as a free-floating object.
- No need to manually keep them in sync when using e.g. `filter` or `arrange`.

```
by_country %>% filter(continent == "Europe")
#> # A tibble: 30 x 4
#> # Groups:   country, continent [710]
#>    country                continent           data model
#>    <fct>                  <fct>     <list<df[,4]>> <list>
#>  1 Albania                Europe          [12 x 4] <lm>
#>  2 Austria                Europe          [12 x 4] <lm>
#>  3 Belgium                Europe          [12 x 4] <lm>
#>  4 Bosnia and Herzegovina Europe          [12 x 4] <lm>
#>  5 Bulgaria               Europe          [12 x 4] <lm>
#>  6 Croatia                Europe          [12 x 4] <lm>
#>  7 Czech Republic         Europe          [12 x 4] <lm>
#>  8 Denmark                Europe          [12 x 4] <lm>
#>  9 Finland                Europe          [12 x 4] <lm>
#> 10 France                 Europe          [12 x 4] <lm>
#> # ... with 20 more rows
```

Adding residuals

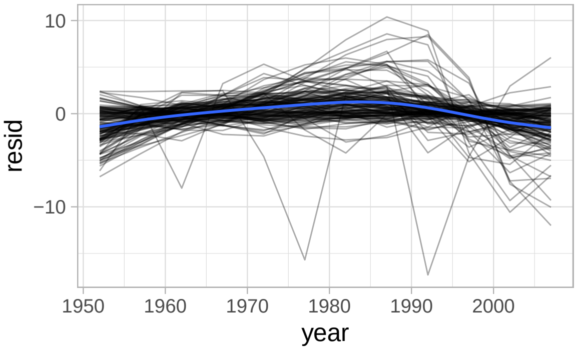
```
(by_country <- by_country %>%
  mutate(resids = map2(data, model, add_residuals)))
#> # A tibble: 142 x 5
#> # Groups:   country, continent [710]
#>    country     continent           data model  resids
#>    <fct>       <fct>      <list<df[,4]>> <list> <list>
#>  1 Afghanistan Asia            [12 x 4] <lm>   <tibble [12 x 5]>
#>  2 Albania     Europe          [12 x 4] <lm>   <tibble [12 x 5]>
#>  3 Algeria     Africa          [12 x 4] <lm>   <tibble [12 x 5]>
#>  4 Angola      Africa          [12 x 4] <lm>   <tibble [12 x 5]>
#>  5 Argentina   Americas        [12 x 4] <lm>   <tibble [12 x 5]>
#>  6 Australia   Oceania         [12 x 4] <lm>   <tibble [12 x 5]>
#>  7 Austria     Europe          [12 x 4] <lm>   <tibble [12 x 5]>
#>  8 Bahrain     Asia            [12 x 4] <lm>   <tibble [12 x 5]>
#>  9 Bangladesh  Asia            [12 x 4] <lm>   <tibble [12 x 5]>
#> 10 Belgium     Europe          [12 x 4] <lm>   <tibble [12 x 5]>
#> # ... with 132 more rows
```

Unnesting

```
resids <- unnest(by_country, resids)
resids
#> # A tibble: 1,704 x 9
#> # Groups:   country, continent [710]
#>    country continent     data model  year lifeExp    pop gdpPercap
#>    <fct>   <fct>      <list<d> <lis> <int>   <dbl>  <int>     <dbl>
#>  1 Afghan~ Asia       [12 x 4] <lm>   1952    28.8 8.43e6      779.
#>  2 Afghan~ Asia       [12 x 4] <lm>   1957    30.3 9.24e6      821.
#>  3 Afghan~ Asia       [12 x 4] <lm>   1962    32.0 1.03e7      853.
#>  4 Afghan~ Asia       [12 x 4] <lm>   1967    34.0 1.15e7      836.
#>  5 Afghan~ Asia       [12 x 4] <lm>   1972    36.1 1.31e7      740.
#>  6 Afghan~ Asia       [12 x 4] <lm>   1977    38.4 1.49e7      786.
#>  7 Afghan~ Asia       [12 x 4] <lm>   1982    39.9 1.29e7      978.
#>  8 Afghan~ Asia       [12 x 4] <lm>   1987    40.8 1.39e7      852.
#>  9 Afghan~ Asia       [12 x 4] <lm>   1992    41.7 1.63e7      649.
#> 10 Afghan~ Asia       [12 x 4] <lm>   1997    41.8 2.22e7      635.
#> # ... with 1,694 more rows, and 1 more variable: resid <dbl>
```

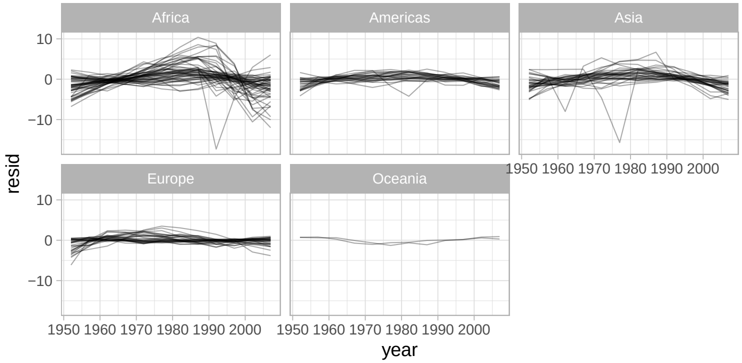
Visualizing the residuals

```r
resids %>%
  ggplot(aes(year, resid)) +
    geom_line(aes(group = country), alpha = 1 / 3) +
    geom_smooth(se = FALSE)
```



Visualizing the residuals cont'd

```r
resids %>%
  ggplot(aes(year, resid, group = country)) +
    geom_line(alpha = 1 / 3) +
  facet_wrap(~continent)
```



Model quality

```
library(broom)
glance(nz_mod)
#> # A tibble: 1 x 11
#>   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC
#>       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>  <dbl> <dbl>
#> 1     0.954         0.949 0.804      205. 5.41e-8     2  -13.3  32.6
#> # ... with 3 more variables: BIC <dbl>, deviance <dbl>,
#> #   df.residual <int>
by_country %>%
  mutate(glance = map(model, glance)) %>%
  unnest(glance) %>%
  print(n = 3)
#> # A tibble: 142 x 16
#> # Groups:   country, continent [710]
#>   country continent     data model resids r.squared adj.r.squared
#>   <fct>   <fct>     <list<d> <lis> <list>     <dbl>         <dbl>
#> 1 Afghan~ Asia      [12 x 4] <lm>  <tibb~     0.948         0.942
#> 2 Albania Europe    [12 x 4] <lm>  <tibb~     0.911         0.902
#> 3 Algeria Africa    [12 x 4] <lm>  <tibb~     0.985         0.984
#> # ... with 139 more rows, and 9 more variables: sigma <dbl>,
#> #   statistic <dbl>, p.value <dbl>, df <int>, logLik <dbl>,
#> #   AIC <dbl>, BIC <dbl>, deviance <dbl>, df.residual <int>
```

Or better

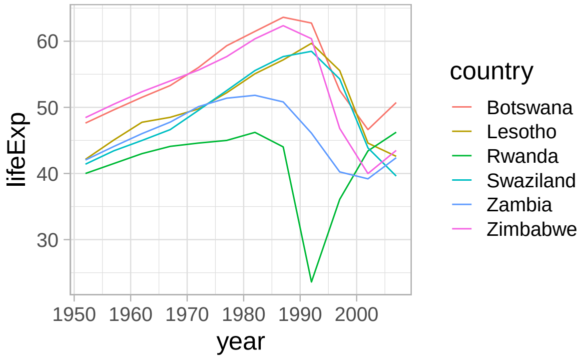
```
by_country_glance <- by_country %>%
  mutate(glance = map(model, glance)) %>%
  unnest(glance)
by_country_glance
#> # A tibble: 142 x 16
#> # Groups:   country, continent [710]
#>    country continent     data model resids r.squared adj.r.squared
#>    <fct>   <fct>     <list<d> <lis> <list>     <dbl>         <dbl>
#>  1 Afghan~ Asia      [12 x 4] <lm>  <tibb~     0.948         0.942
#>  2 Albania Europe    [12 x 4] <lm>  <tibb~     0.911         0.902
#>  3 Algeria Africa    [12 x 4] <lm>  <tibb~     0.985         0.984
#>  4 Angola  Africa    [12 x 4] <lm>  <tibb~     0.888         0.877
#>  5 Argent~ Americas  [12 x 4] <lm>  <tibb~     0.996         0.995
#>  6 Austra~ Oceania   [12 x 4] <lm>  <tibb~     0.980         0.978
#>  7 Austria Europe    [12 x 4] <lm>  <tibb~     0.992         0.991
#>  8 Bahrain Asia      [12 x 4] <lm>  <tibb~     0.967         0.963
#>  9 Bangla~ Asia      [12 x 4] <lm>  <tibb~     0.989         0.988
#> 10 Belgium Europe    [12 x 4] <lm>  <tibb~     0.995         0.994
#> # ... with 132 more rows, and 9 more variables: sigma <dbl>,
#> #   statistic <dbl>, p.value <dbl>, df <int>, logLik <dbl>,
#> #   AIC <dbl>, BIC <dbl>, deviance <dbl>, df.residual <int>
```

Which models don't fit well?

```
by_country_glance %>%
  arrange(r.squared)
#> # A tibble: 142 x 16
#> # Groups:   country, continent [710]
#>    country continent     data model resids r.squared adj.r.squared
#>    <fct>   <fct>     <list<d> <lis> <list>     <dbl>         <dbl>
#>  1 Rwanda  Africa    [12 x 4] <lm>  <tibb~    0.0172       -0.0811
#>  2 Botswa~ Africa    [12 x 4] <lm>  <tibb~    0.0340       -0.0626
#>  3 Zimbab~ Africa    [12 x 4] <lm>  <tibb~    0.0562       -0.0381
#>  4 Zambia  Africa    [12 x 4] <lm>  <tibb~    0.0598       -0.0342
#>  5 Swazil~ Africa    [12 x 4] <lm>  <tibb~    0.0682       -0.0250
#>  6 Lesotho Africa    [12 x 4] <lm>  <tibb~    0.0849       -0.00666
#>  7 Cote d~ Africa    [12 x 4] <lm>  <tibb~    0.283         0.212
#>  8 South ~ Africa    [12 x 4] <lm>  <tibb~    0.312         0.244
#>  9 Uganda  Africa    [12 x 4] <lm>  <tibb~    0.342         0.276
#> 10 Congo,~ Africa    [12 x 4] <lm>  <tibb~    0.348         0.283
#> # ... with 132 more rows, and 9 more variables: sigma <dbl>,
#> #   statistic <dbl>, p.value <dbl>, df <int>, logLik <dbl>,
#> #   AIC <dbl>, BIC <dbl>, deviance <dbl>, df.residual <int>
```

Visualize

```r
bad_fit <- filter(by_country_glance, r.squared < 0.25)

gapminder %>%
  semi_join(bad_fit, by = "country") %>%
  ggplot(aes(year, lifeExp, colour = country)) +
  geom_line()
```



Making tidy data with broom

- `broom::glance(model)`
  - A row for each model.
  - Columns give a model summary (measure of model quality, complexity, or combination of both).
- `broom::tidy(model)`
  - A row for each coefficient in the model.
  - Columns give information about the estimate or its variability.
- `broom::augment(model, data)`
  - A row for each row in `data`.
  - Adds extra values like residuals, and influence statistics.

```r
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
lm_D9 <- lm(weight ~ group)
```

`broom::glance()` and `broom::tidy()`

- `broom::glance(model)`
  - A row for each model.
  - Columns give a model summary.

```
tidy(lm_D9)
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)    5.03      0.220     22.9  9.55e-15
#> 2 groupTrt      -0.371     0.311     -1.19 2.49e- 1
```

- `broom::tidy(model)`
  - A row for each coefficient in the model.
  - Columns give information about the estimate or its variability.

```
tidy(lm_D9)
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)    5.03      0.220     22.9  9.55e-15
#> 2 groupTrt      -0.371     0.311     -1.19 2.49e- 1
```

broom::augment()

- broom::augment(model, data)
  - A row for each row in data.
  - Adds extra values like residuals, and influence statistics.

```
augment(lm_D9) %>%
  print(n = 10)
#> # A tibble: 20 x 9
#>    weight group .fitted .se.fit .resid  .hat .sigma .cooksd
#>     <dbl> <fct>   <dbl>   <dbl>  <dbl> <dbl>  <dbl>   <dbl>
#>  1   4.17 Ctl      5.03   0.220 -0.862  0.10  0.682 0.0946
#>  2   5.58 Ctl      5.03   0.220  0.548  0.10  0.703 0.0382
#>  3   5.18 Ctl      5.03   0.220  0.148  0.1   0.716 0.00279
#>  4   6.11 Ctl      5.03   0.220  1.08   0.1   0.661 0.148
#>  5   4.5  Ctl      5.03   0.220 -0.532  0.1   0.704 0.0360
#>  6   4.61 Ctl      5.03   0.220 -0.422  0.1   0.708 0.0227
#>  7   5.17 Ctl      5.03   0.220  0.138  0.1   0.716 0.00242
#>  8   4.53 Ctl      5.03   0.220 -0.502  0.1   0.705 0.0321
#>  9   5.33 Ctl      5.03   0.220  0.298  0.1   0.713 0.0113
#> 10   5.14 Ctl      5.03   0.220  0.108  0.1   0.716 0.00148
#> # ... with 10 more rows, and 1 more variable: .std.resid <dbl>
```

Outline

1 Model basics

2 Model building

3 Many models

4 List-columns

List-columns

```
tibble(x = list(1:3, 3:5),
       y = c("1, 2", "3, 4, 5"))
#> # A tibble: 2 x 2
#>   x         y
#>   <list>    <chr>
#> 1 <int [3]> 1, 2
#> 2 <int [3]> 3, 4, 5

tribble(~x, ~y,
        1:3, "1, 2",
        3:5, "3, 4, 5")
#> # A tibble: 2 x 2
#>   x         y
#>   <list>    <chr>
#> 1 <int [3]> 1, 2
#> 2 <int [3]> 3, 4, 5
```

An effective list-column pipeline

- Create the list-column:
  - With `nest()` to convert a grouped data frame into a nested data frame where you have list-column of data frames.
  - With `mutate()` and vectorised functions that return a list.
  - With `summarize()` and summary functions that return multiple results.
- Create other intermediate list-columns by transforming existing list columns with `map()`, `map2()` or `pmap()`.
- Simplify the list-column back down to a data frame or atomic vector.

Create with nesting I

```
gapminder %>%
  group_by(country, continent) %>%
  nest()
#> # A tibble: 142 x 3
#> # Groups:   country, continent [710]
#>    country     continent            data
#>    <fct>       <fct>     <list<df[,4]>>
#>  1 Afghanistan Asia             [12 x 4]
#>  2 Albania     Europe           [12 x 4]
#>  3 Algeria     Africa           [12 x 4]
#>  4 Angola      Africa           [12 x 4]
#>  5 Argentina   Americas         [12 x 4]
#>  6 Australia   Oceania          [12 x 4]
#>  7 Austria     Europe           [12 x 4]
#>  8 Bahrain     Asia             [12 x 4]
#>  9 Bangladesh  Asia             [12 x 4]
#> 10 Belgium     Europe           [12 x 4]
#> # ... with 132 more rows
```

Create with nesting II

```
gapminder %>%
  nest(data = year:gdpPercap)
#> # A tibble: 142 x 3
#>    country     continent            data
#>    <fct>       <fct>      <list<df[,4]>>
#>  1 Afghanistan Asia             [12 x 4]
#>  2 Albania     Europe           [12 x 4]
#>  3 Algeria     Africa           [12 x 4]
#>  4 Angola      Africa           [12 x 4]
#>  5 Argentina   Americas         [12 x 4]
#>  6 Australia   Oceania          [12 x 4]
#>  7 Austria     Europe           [12 x 4]
#>  8 Bahrain     Asia             [12 x 4]
#>  9 Bangladesh  Asia             [12 x 4]
#> 10 Belgium     Europe           [12 x 4]
#> # ... with 132 more rows
```

Create from vectorized functions

```
df <- tribble(~x1, "a,b,c", "d,e,f,g")
df %>%
  mutate(x2 = stringr::str_split(x1, ","))
#> # A tibble: 2 x 2
#>   x1      x2
#>   <chr>   <list>
#> 1 a,b,c   <chr [3]>
#> 2 d,e,f,g <chr [4]>

sim <- tribble(~f,      ~params,
               "runif", list(min = -1, max = -1),
               "rnorm", list(sd = 5),
               "rpois", list(lambda = 10))
sim %>%
  mutate(sims = invoke_map(f, params, n = 10))
#> # A tibble: 3 x 3
#>   f     params            sims
#>   <chr> <list>            <list>
#> 1 runif <named list [2]> <dbl [10]>
#> 2 rnorm <named list [1]> <dbl [10]>
#> 3 rpois <named list [1]> <int [10]>
```

Create from multivalued summaries II

- What's wrong here?

```
mtcars %>%
  group_by(cyl) %>%
  summarize(q = quantile(mpg))
#> Error: Column `q` must be length 1 (a summary value), not 5
```

- Use list-columns:

```
mtcars %>%
  group_by(cyl) %>%
  summarize(q = list(quantile(mpg)))
#> # A tibble: 3 x 2
#>     cyl q
#>   <dbl> <list>
#> 1     4 <dbl [5]>
#> 2     6 <dbl [5]>
#> 3     8 <dbl [5]>
```

Create from multivalued summaries II

```
probs <- c(0.01, 0.25, 0.5, 0.75, 0.99)
mtcars %>%
  group_by(cyl) %>%
  summarize(p = list(probs),
            q = list(quantile(mpg, probs))) %>%
  unnest(cols = c(p, q)) %>%
  print(n = 7)
#> # A tibble: 15 x 3
#>     cyl     p     q
#>   <dbl> <dbl> <dbl>
#> 1     4  0.01  21.4
#> 2     4  0.25  22.8
#> 3     4  0.5   26
#> 4     4  0.75  30.4
#> 5     4  0.99  33.8
#> 6     6  0.01  17.8
#> 7     6  0.25  18.6
#> # ... with 8 more rows
```

Simplifying list-columns

- If you want a single value, use `mutate()` with `map_lgl()`, `map_int()`, `map_dbl()`, and `map_chr()` to create an atomic vector.
- If you want many values, use `unnest()` to convert list-columns back to regular columns, repeating the rows as many times as necessary.

List to vector

```
df <- tribble(~x, letters[1:5], 1:3, runif(5))

df %>%
  mutate(type = map_chr(x, typeof),
         length = map_int(x, length))
#> # A tibble: 3 x 3
#>   x         type      length
#>   <list>    <chr>      <int>
#> 1 <chr [5]> character      5
#> 2 <int [3]> integer        3
#> 3 <dbl [5]> double         5
```

Unnesting

- Columns with the same number of elements:

```
tibble(x = 1:2,
       y = list(1:4, 1)) %>%
  unnest(y)
#> # A tibble: 5 x 2
#>       x     y
#>   <int> <dbl>
#> 1     1     1
#> 2     1     2
#> 3     1     3
#> 4     1     4
#> 5     2     1
```

- Columns with different number of elements:

```
tribble(~x, ~y,         ~z,
        1, "a",         1:2,
        2, c("b", "c"), 3
        ) %>%
  unnest(c(y, z))
#> # A tibble: 4 x 3
#>       x y         z
#>   <dbl> <chr> <dbl>
#> 1     1 a         1
#> 2     1 a         2
#> 3     2 b         3
#> 4     2 c         3
```