

Prediction, iteration, and regression

Seung-Ho An, University of Arizona

Prediction

Loops

Evaluating the predictions

Time-series plot

Prediction (again)

Modeling with a line

Linear regression in R

1. Prediction

2016 election popular vote

- Clinton: 65,853,516 (48.2%)
- Trump: 62,984,825 (46.1%)

Why did Trump win? **Electoral college**

- Trump: 304, Clinton: 227

Election determined by 77,744 votes (margins in WI, MI, and PA)

- 0.056% of the electorate (~136 million)

Electoral college system

- Must win an absolute majority of 538 electoral votes
- $538 = 435$ (House of Representatives) + 100 (Senators) + 3 (DC)
- Must win at least 270 votes
- Nobody wins an absolute majority \rightsquigarrow House vote

Must predict winner of each state

Predict state-level support for each candidate using polls

Allocate electoral college votes of that state to its predicted winner

Aggregate EC votes across states to determine the predicted winner

Coding strategy:

- For each state, subset to polls within that state
- Further subset the latest polls
- Average the latest polls to estimate support for each candidate
- Allocate the electoral votes to the candidate who has greatest support
- Repeat this for all states and aggregate the electoral votes

Sounds like a lot of subsets :(

2. Loop

What if we wanted to know the number of unique values of each column of the `cces_2020` data?

```
library(TPDDdata)
cces_2020
```

```
## # A tibble: 51,551 x 6
##   gender race educ          pid3          turnout_self pres_vote
##   <fct> <fct> <fct>          <fct>          <dbl> <fct>
## 1 Male   White 2-year   Republican      1 Donald J. Trump (~
## 2 Female White Post-grad Democrat        NA <NA>
## 3 Female White 4-year   Independent     1 Joe Biden (Democr~
## 4 Female White 4-year   Democrat       1 Joe Biden (Democr~
## 5 Male   White 4-year   Independent     1 Other
## 6 Male   White Some college Republican      1 Donald J. Trump (~
## 7 Male   Black Some college Not sure        NA <NA>
## 8 Female White Some college Independent     1 Donald J. Trump (~
## 9 Female White High school graduate Republican     1 Donald J. Trump (~
## 10 Female White 4-year   Democrat       1 Joe Biden (Democr~
## # ... with 51,541 more rows
```


Manually changing values

```
length(unique(cces_2020$gender))
```

```
## [1] 2
```

```
length(unique(cces_2020$race))
```

```
## [1] 8
```

```
length(unique(cces_2020$educ))
```

```
## [1] 6
```

```
length(unique(cces_2020$pid3))
```

```
## [1] 5
```

```
length(unique(cces_2020$turnout_self))
```

```
## [1] 3
```

```
length(unique(cces_2020$pres_vote))
```

```
## [1] 7
```

Note tat we can also access variables with `[[]]`:

```
unique(cces_2020$gender)
```

```
## [1] Male   Female  
## Levels: Male Female skipped not asked
```

```
unique(cces_2020[[1]])
```

```
## [1] Male   Female  
## Levels: Male Female skipped not asked
```

```
unique(cces_2020$pid3)
```

```
## [1] Republican Democrat Independent Not sure Other  
## Levels: Democrat Republican Independent Other Not sure skipped not asked
```

```
unique(cces_2020[[4]])
```

```
## [1] Republican Democrat Independent Not sure Other  
## Levels: Democrat Republican Independent Other Not sure skipped not asked
```

Manually changing values, alternative

```
unique(cces_2020[[1]])
```

```
## [1] Male   Female  
## Levels: Male Female skipped not asked
```

```
unique(cces_2020[[2]])
```

```
## [1] White           Black           Other           Hispanic  
## [5] Two or more races Asian           Middle Eastern   Native American  
## 10 Levels: White Black Hispanic Asian Native American ... not asked
```

```
unique(cces_2020[[3]])
```

```
## [1] 2-year           Post-grad           4-year  
## [4] Some college     High school graduate No HS  
## 8 Levels: No HS High school graduate Some college 2-year 4-year ... not asked
```

```
unique(cces_2020[[4]])
```

```
## [1] Republican Democrat   Independent Not sure   Other  
## Levels: Democrat Republican Independent Other Not sure skipped not asked
```

```
unique(cces_2020[[5]])
```

```
## [1] 1 NA 0
```

```
unique(cces_2020[[6]])
```

What if you had more values? Not efficient!

Recognize the template:

```
length(unique(cces_2020[[<column number>]]))
```

Can we give R this template and a set of column numbers have it do our task repeatedly?

for loop provides a way to execute these templates multiple times:

```
output <- rep(NA, times = ncol(cces_2020))      # 1. output
for (i in seq_along(cces_2020)) {               # 2. sequence
  output[i] <- length(unique(cces_2020[[i]]))    # 3. body
}
output
```

```
## [1] 2 8 6 5 3 7
```

- Elements of a loop:
 - ① output: vector to hold the
 - ② i: placeholder name we'll use to swap values between iterations
 - ③ seq_along(cces_2020): vector of values we want the placeholder to take
 - ④ body: a set of expressions that will be repeatedly evaluated
 - ⑤ {}: curly braces to define beginning and end of the loop
- **Indentation** is important for readability of the code

Election data: pres20

Variable	Description
state	abbreviated name of state
biden	Biden's vote share (percentage)
trump	Trump's vote share (percentage)
ev	number of electoral college votes for the state

Polling data: polls20

Variable	Description
state	state in which poll was conducted
end_date	end date the period when poll was conducted
daysleft	number of days between end date and election date
pollster	name of organization conducting poll
sample_size	number of samples for each poll conducted
biden	predicted support for Biden (percentage)
trump	predicted support for Trump (percentage)

```
library(TPDDdata)

# calculate Trump's margin of victory
polls20 <- polls20 |>
  mutate(margin = biden - trump)

pres20 <- pres20 |>
  mutate(margin = biden - trump)

glimpse(polls20)
```

```
## Rows: 2,445
## Columns: 8
## $ end_date    <date> 2020-11-02, 2020-11-02, 2020-11-02, 2020-11-02, 2020-11-0~
## $ state       <chr> "FL", "PA", "FL", "FL", "NV", "GA", "SC", "MT", "ME", "AZ"~
## $ days_left   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ pollster    <chr> "The Political Matrix/The Listener Group", "Susquehanna", ~
## $ sample_size <dbl> 966, 499, 400, 1054, 1024, 1041, 817, 920, 1024, 610, 1261~
## $ biden       <dbl> 44.2, 48.4, 47.0, 47.3, 48.4, 45.4, 39.0, 45.0, 52.0, 50.0~
## $ trump       <dbl> 48.0, 49.2, 48.2, 49.4, 49.1, 49.7, 51.4, 50.0, 40.0, 47.5~
## $ margin      <dbl> -3.8, -0.8, -1.2, -2.1, -0.7, -4.3, -12.4, -5.0, 12.0, 2.5~
```

- Coding strategy:
 - ① For each state, subset to polls within that state
 - ② Further subset the latest polls
 - ③ Average the latest polls to estimate support for each candidate
 - ④ Allocate the electoral votes to the candidate who has greatest support
 - ⑤ Repeat this for all states and aggregate the electoral votes

Poll prediction for each state

```
poll_pred <- rep(NA, 51) # place holder

# get list of unique state names to iterate over
state_names <- sort(unique(polls20$state))

# add labels to holder
names(poll_pred) <- state_names

for (i in 1:51) {
  state_data <- subset(polls20, subset = (state == state_names[i]))

  latest <- state_data$days_left == min(state_data$days_left)

  poll_pred[i] <- mean(state_data$margin[latest])
}
head(poll_pred)
```

```
##      AK      AL      AR      AZ      CA      CO
## -9.00 -26.00 -23.00  4.25 26.00 11.00
```

```
poll_pred <- polls20 |>
  group_by(state) |>
  filter(days_left == min(days_left)) |>
  summarize(margin_pred = mean(margin))
poll_pred
```

```
## # A tibble: 51 x 2
##   state margin_pred
##   <chr>         <dbl>
## 1 AK             -9
## 2 AL            -26
## 3 AR            -23
## 4 AZ             4.25
## 5 CA             26
## 6 CO             11
## 7 CT             22
## 8 DC             89
## 9 DE             22
## 10 FL            0.0800
## # ... with 41 more rows
```

3. Evaluating the predictions

Prediction error = actual outcome - predicted outcome

```
poll_pred <- poll_pred |>
  left_join(pres20) |>
  mutate(errors = margin - margin_pred)
```

```
## Joining with `by = join_by(state)`
```

```
poll_pred
```

```
## # A tibble: 51 x 8
```

```
##   state margin_pred   ev biden trump  other  margin errors
##   <chr>      <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1 AK          -9      3  42.8  52.8  0.732 -10.1  -1.06
## 2 AL         -26      9  36.6  62.0  0.699 -25.5   0.538
## 3 AR         -23      6  34.8  62.4  0.257 -27.6  -4.62
## 4 AZ          4.25     11  49.4  49.1  0.263  0.309 -3.94
## 5 CA          26     55  63.5  34.3  0.244  29.2   3.16
## 6 CO          11      9  55.0  41.6  0.161  13.4   2.41
## 7 CT          22      7  59.3  39.2  0.129  20.1  -1.93
## 8 DC          89      3  92.1  5.40  0.491  86.8  -2.25
## 9 DE          22      3  58.7  39.8  0.0780 19.0  -3.03
## 10 FL          0.0800    29  47.9  51.2  0.0835 -3.36  -3.44
## # ... with 41 more rows
```

Bias: average prediction error

```
mean(poll_pred$errors)
```

```
## [1] -3.983248
```

Root mean-square error: average magnitude of the prediction error

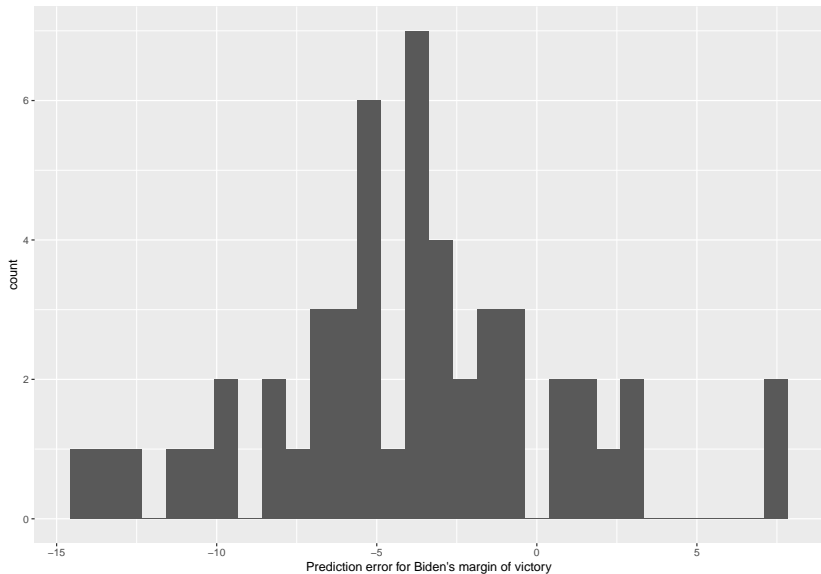
```
sqrt(mean(poll_pred$errors^2))
```

```
## [1] 6.06975
```

Histogram of the errors

```
ggplot(poll_pred, aes(x = errors)) +  
  geom_histogram() +  
  labs(  
    x = "Prediction error for Biden's margin of victory"  
  )
```

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



Sometimes we want plot text labels instead of point and we use `geom_text` and the `label` aesthetic:

```
## merge the actual results
```

```
ggplot(poll_pred, aes(x = margin_pred, y=margin)) +  
  geom_text(aes(label = state)) +  
  geom_abline(xintercept = 0, slope = 1, linetype=2) +  
  geom_hline(yintercept = 0, color = "grey50") +  
  geom_vline(xintercept = 0, color = "grey50")
```


Election prediction: need to predict winner in each state:

```
poll_pred |>
  filter(margin > 0) |>
    summarize(sum(ev)) |>
  pull()
```

```
## [1] 306
```

```
poll_pred |>
  filter(margin_pred > 0) |>
  summarize(sum(ev)) |>
  pull()
```

```
## [1] 328
```

- Prediction of binary outcome variable = **classification problem**
- Wrong prediction \approx misclassification
 - ① **true positive**: predict Trump wins when he actually wins
 - ② **false positive**: predict Trump wins when he actually loses
 - ③ **true negative**: predict Trump loses when he actually loses
 - ④ **false negative**: predict Trump loses when he actually wins
- Sometimes false negatives are more/less important: e.g. civil war

Classification based on polls

Accuracy: `sign()` returns 1 for a positive number, -1 for a negative number, and 0 for 0

```
poll_pred |>
  summarize(prop_correct = mean(sign(margin_pred) == sign(margin))) |>
  pull()
```

```
## [1] 0.9215686
```

Which states did polls call wrong?

```
poll_pred |>
  filter(sign(margin_pred) != sign(margin))
```

```
## # A tibble: 4 x 8
##   state margin_pred    ev biden trump  other margin errors
##   <chr>         <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1 FL           0.0800    29  47.9  51.2  0.0835 -3.36  -3.44
## 2 GA          -1.15     16  49.5  49.2  0.0759  0.236  1.39
## 3 NC           3.95     15  48.6  49.9  0.296  -1.35  -5.30
## 4 NV          -0.350      6  50.1  47.7  0.759   2.39   2.74
```

4. Time-series plot

We often want to show a time series of the national-level polls to get a sense of the popular vote:

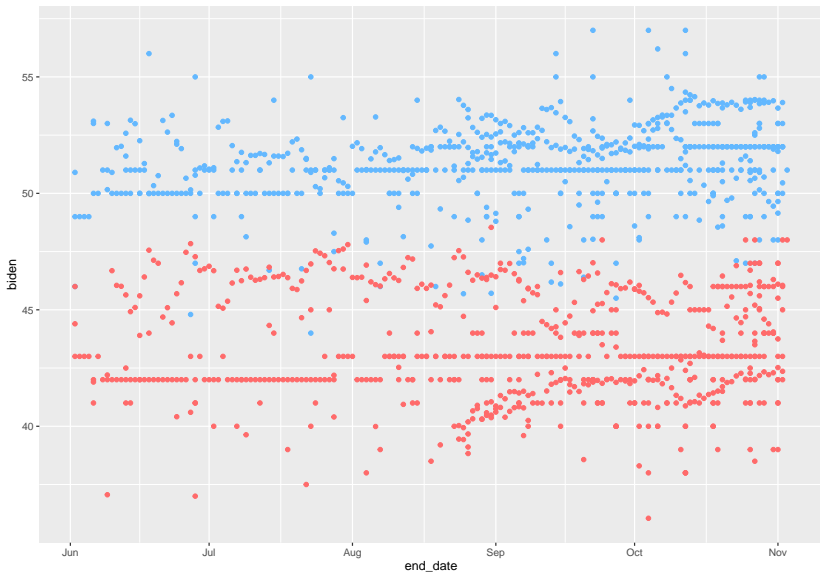
```
national_polls20
```

```
## # A tibble: 654 x 5
##   end_date   pollster sample_1 biden trump
##   <date>     <chr>    <dbl> <dbl> <dbl>
## 1 2020-11-03 Lake Research      2400  51    48
## 2 2020-11-02 Research Co.      1025  50    42
## 3 2020-11-02 YouGov           1363  53    43
## 4 2020-11-02 Ipsos              914  52    45
## 5 2020-11-02 SurveyMonkey     28240  52    46
## 6 2020-11-02 HarrisX          2297  52    48
## 7 2020-11-02 TIPP             1212  50.4  46.0
## 8 2020-11-02 USC Dornsife      5423  53.9  42.4
## 9 2020-11-01 John Zogby Strategies/EMI Research Solutions 1008  49.6  43.8
## 10 2020-11-01 Swayable          5174  51.8  46.1
## # ... with 644 more rows, and abbreviated variable name 1: sample_size
```

```
national_polls20 |>
  ggplot(aes(x = end_date)) +
  geom_point(aes(y = biden), color = "steelblue1") +
  geom_point(aes(y = trump), color = "indianred1")
```

Plotting the raw results

Fairly messy:



Clean the mess by taking moving averages

Goal: plot the average of polls in the last 7 days (very difficult with `dplyr`)

Loop over each day in the data and do:

- 1 subset to all polls in the previous 7 days of that day
- 2 calculate the average of these polls for Biden and Trump
- 3 save the results as a 1-row tibble

You can get R to properly understand dates and do arithmetic with them:

```
head(national_polls20$end_date)
```

```
## [1] "2020-11-03" "2020-11-02" "2020-11-02" "2020-11-02" "2020-11-02"
## [6] "2020-11-02"
```

```
head(national_polls20$end_date+3)
```

```
## [1] "2020-11-06" "2020-11-05" "2020-11-05" "2020-11-05" "2020-11-05"
## [6] "2020-11-05"
```

We can convert a string to a date using the lubridate package:

```
"2020-11-03" + 3 ## R doesn't know this is a date yet!
```

```
## Error in "2020-11-03" + 3: non-numeric argument to binary operator
```

```
lubridate::ymd("2020-11-03") + 3
```

```
## [1] "2020-11-06"
```

```
lubridate::mdy("11/03/2020") + 3
```

```
## [1] "2020-11-06"
```

Setup the vector of dates to cover:

```
election_day <- lubridate::ymd("2020-11-03")
all_dates <- seq(from = min(national_polls20$end_date) + 1,
                 to = election_day,
                 by = "days")
head(all_dates)
```

```
## [1] "2020-06-03" "2020-06-04" "2020-06-05" "2020-06-06" "2020-06-07"
## [6] "2020-06-08"
```

```
output <-vector("list", length=length(all_dates))
for (i in seq_along(all_dates)) {
  this_date <- all_dates[[i]]

  this_week <- national_polls20 |>
    filter(
      this_date - end_date >= 0,      # this_date is after end_date
      this_date - end_date < 7       # within a week
    )

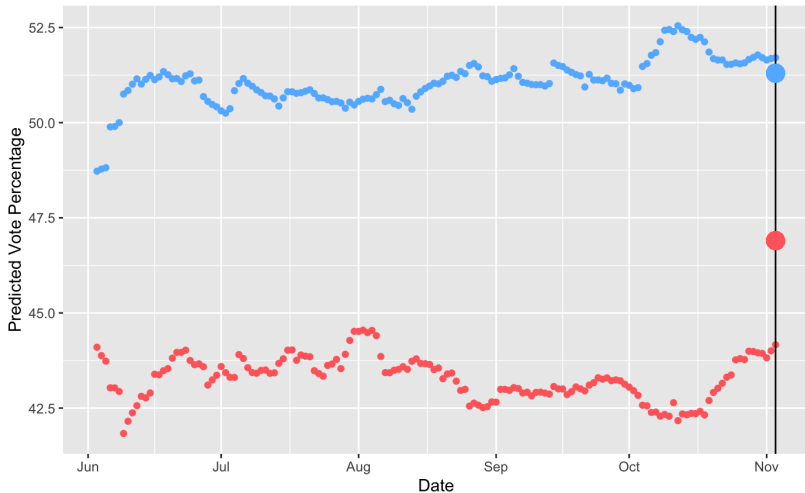
  output[[i]] <- this_week |>
    summarize(
      date = this_date,
      biden = mean(biden, na.rm = TRUE),
      trump = mean(trump, na.rm = TRUE)
    )
}
output <- bind_rows(output)
```

output

```
## [1] 2 8 6 5 3 7
```

```
output |>
  ggplot(aes(x = date)) +
  geom_point(aes(y = biden), color = "steelblue1") +
  geom_point(aes(y = trump), color = "indianred1") +
  geom_vline(xintercept = election_day) +
  geom_point(aes(x = election_day, y = 51.3), color = "steelblue1", size = 5) +
  geom_point(aes(x = election_day, y = 46.9), color = "indianred1", size = 5) +
  labs(
    x = "Date",
    y = "Predicted Vote Percentage"
  )
```

Let's plot



Prediction (again)

Predicting weight with activity: health data

Variable	Description
date	date of measurements
active_calories	calories burned
steps	number of steps taken (in, 1,000s)
weight	weight (lbs)
steps_lag	steps on day before (in 1,000s)
calories_lag	calories burned on day before

Predicting using bivariate relationship

- Goal: what's our best guess about Y_i if we know what X_i is?
 - What's our best guess about one's weight this morning? Would it be helpful if we know how many steps she/he took yesterday?
- Terminology:
 - **Dependent/outcome variable:** what we want to predict (weight)
 - **Independent/explanatory variable:** what we are using to predict (steps)

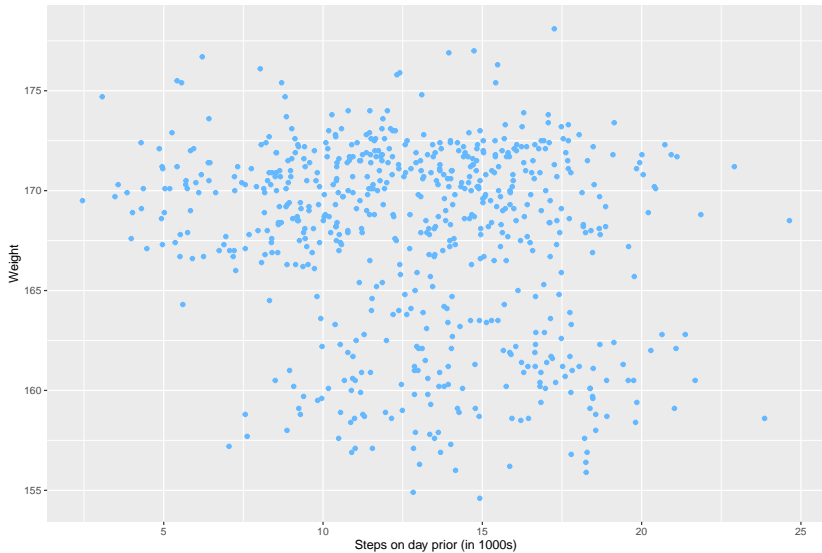
- Load the data:

```
library(TPDDdata)
health <- drop_na(health)
```

- Plot the data:

```
ggplot(health, aes(x = steps_lag, y = weight)) +
  geom_point(color = "steelblue1") +
  labs(
    x = "Steps on day prior (in 1000s)"
    y = "Weight",
    title = "Weight and Steps"
  )
```

Weight and Steps



Prediction on variable with another

- Prediction with access to just Y : average of the Y values
- Prediction with another variable: for any value of X , what's the best guess about Y ?
 - Need a function $y = f(x)$ that maps values of X into predictions
 - **Machine learning**: fancy ways to determine $f(x)$
- Example: what if did 5,000 steps today? What's my best guess about weight?

Start with looking at a narrow strip of X

Let's find all values that round to 5,000 steps:

```
health |>  
  filter(round(steps_lag) == 5)
```

```
## # A tibble: 12 x 6
```

##	date	active_calories	steps	weight	steps_lag	calorie_lag
##	<date>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	2015-09-08	1111.	15.2	169.	5.02	410.
## 2	2015-12-12	728.	14.7	167.	5.36	259.
## 3	2015-12-28	430.	8.94	170.	5.19	314
## 4	2016-01-29	475.	8.26	171.	4.95	314.
## 5	2016-02-14	264.	5.42	172.	4.86	297.
## 6	2016-02-15	892.	13.1	171.	5.42	264.
## 7	2016-05-02	627.	11.8	170.	5.04	283.
## 8	2016-06-27	352.	7.21	169.	4.93	212.
## 9	2016-07-22	766.	14.8	167.	4.96	251.
## 10	2016-11-25	452	9.4	173.	5.26	295
## 11	2016-11-28	577.	11.8	171.	4.97	304.
## 12	2016-12-30	621.	12.4	176.	5.42	371.

Best guess about Y for this X

Best prediction about weight for a step count of roughly 5,000 is the average weight for observations around that value:

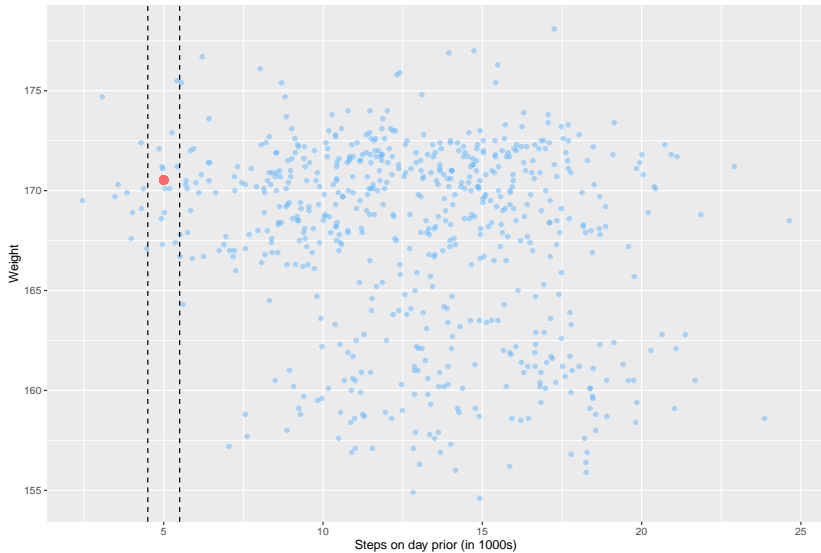
```
mean_wt_5k_steps <- health |>
  filter(round(steps_lag) == 5) |>
  summarize(mean(weight)) |>
  pull()
mean_wt_5k_steps
```

```
## [1] 170.5333
```

Plotting the best guess

```
ggplot(health, aes(x = steps_lag, y = weight)) +  
  geom_point(color = "steelblue1", alpha = 0.5) +  
  labs(  
    x = "Steps on day prior (in 1000s)",  
    y = "Weight",  
    title = "Weight and Steps") +  
  geom_vline(xintercept = c(4.5, 5.5), linetype = "dashed") +  
  geom_point(aes(x = 5, y = mean_wt_5k_steps), color = "indianred1",  
    size = 3)
```

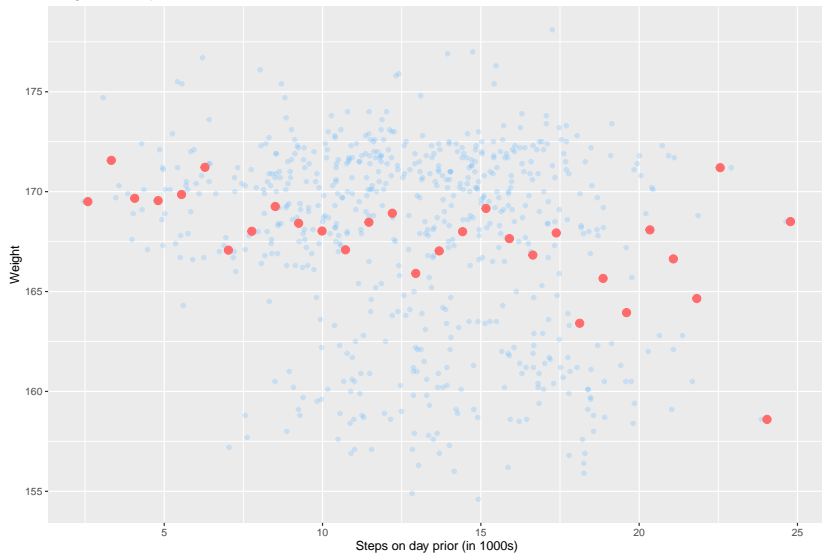

Weight and Steps



We can use a `stat_summary_bin()` to add these binned means all over the scatter plot:

```
ggplot(health, aes(x = steps_lag, y = weight)) +  
  geom_point(color = "steelblue1", alpha = 0.25) +  
  labs(  
    x = "Steps on day prior (in 1000s)",  
    y = "Weight",  
    title = "Weight and Steps"  
  ) +  
  stat_summary_bin(fun = "mean", color = "indianred1", size = 3,  
                  geom = "point", bandwidth = 1)
```

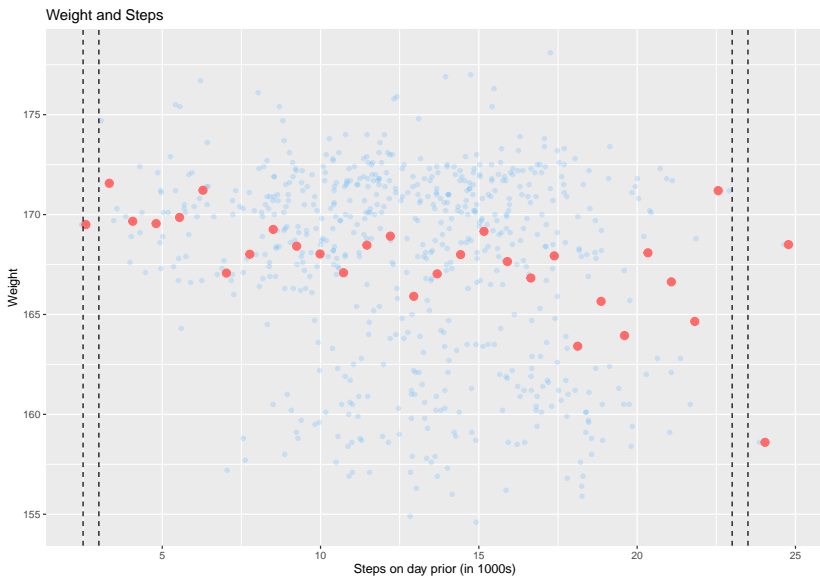
Weight and Steps



But what happens when we make the bins too small?

```
ggplot(health, aes(x = steps_lag, y = weight)) +  
  geom_point(color = "steelblue1", alpha = 0.25) +  
  labs(  
    x = "Steps on day prior (in 1000s)",  
    y = "Weight",  
    title = "Weight and Steps"  
  ) +  
  stat_summary_bin(fun = "mean", color = "indianred1", size = 3,  
                  geom = "point", bandwidth = 0.5) +  
  geom_vline(xintercept = c(2.5, 3, 23, 23.5), linetype = "dashed")
```

Gaps and bumps:



Modeling with a line

- Can we smooth out these binned means and close gaps? **A model**
- Simplest possible way to relate two variables: a line

$$y = mx + b$$

- Problem: for any line we draw, not all the data is on the line
 - Some points will be above the line, some below
 - Need a way to account for **chance variation** away from the line

- Model for the line of best fit:

$$Y_i = \underbrace{\alpha}_{\text{intercept}} + \underbrace{\beta}_{\text{slope}} \cdot X_i + \underbrace{\epsilon_i}_{\text{error term}}$$

- **Coefficients/parameters** (α, β) : true unknown intercept/slope of the line of best fit
- **Chance error** ϵ_i : accounts for the fact that the line doesn't perfectly fit the data
 - Each observation allowed to be off the regression line
 - Chance errors are 0 on average
- Useful fiction: this model represents the **data generating process**
 - George Box (British statistician): ***“all models are wrong, some are useful”***

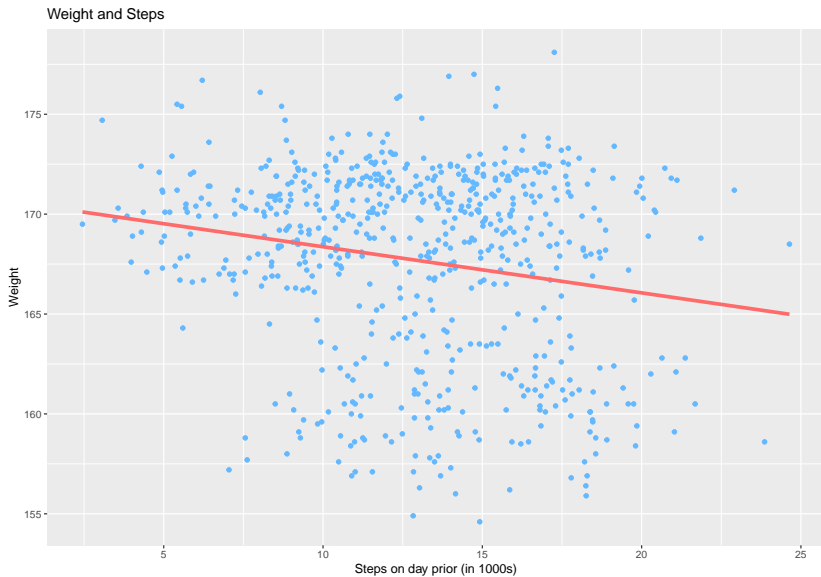
$$Y_i = \alpha + \beta \cdot X_i + \epsilon_i$$

- **Intercept** α : average value of Y when X is 0
 - Average weight when I take 0 steps the day prior
- **Slope** β : average change in Y when X increases by one unit
 - Average decrease in weight for each additional 1,000 steps

- Parameters: α, β
 - Unknown features of the **data-generating process**
 - Chance error makes these impossible to observe directly
- Estimates: $\hat{\alpha}, \hat{\beta}$
 - An **estimate** is our best guess about some parameter
- **Regression line:** $\hat{Y} = \hat{\alpha} + \hat{\beta} \cdot x$
 - Average value of Y when X is equal to x
 - Represents the best guess of **predicted value** of the outcome at x

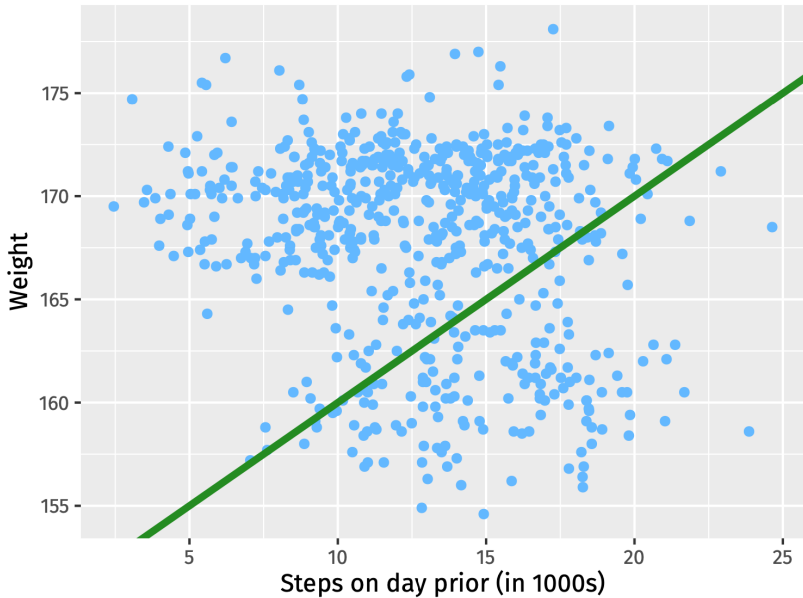
```
ggplot(health, aes(x = steps_lag, y = weight)) +  
  geom_point(color = "steelblue1") +  
  labs(  
    x = "Steps on day prior (in 1000s)",  
    y = "Weight",  
    title = "Weight and Steps")+  
  geom_smooth(method = "lm", se = FALSE, color = "indianred1", size = 1.5)
```

Line of best fit



Why not this line?

Weight and Steps



Let's understand the **prediction error** for a line with intercept a and slope b

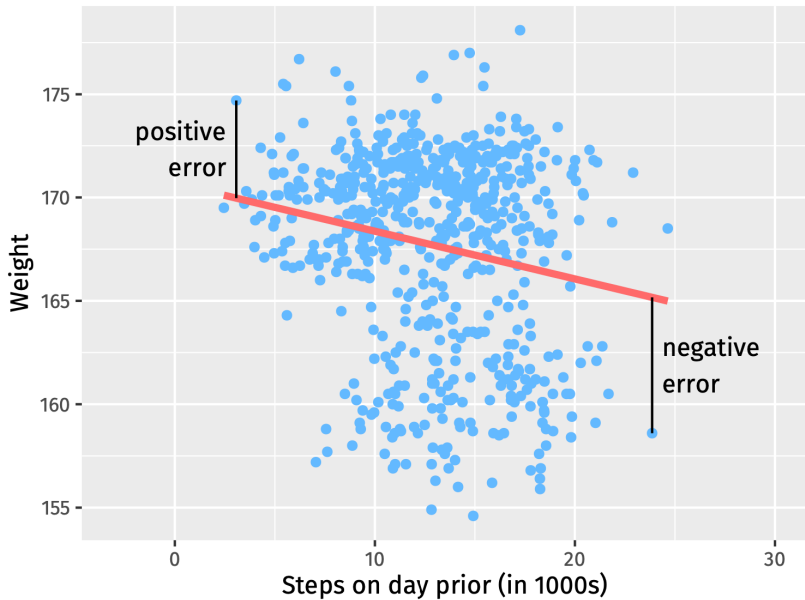
Fitted/predicted value for unit i :

$$a + b \cdot X_i$$

Prediction error (residual):

$$error = actual - predicted = Y_i - (a + b \cdot X_i)$$

Weight and Steps



- Get these estimates by the **least squares methods**
- Minimize the **sum of the squared residuals** (SSR):

$$SSR = \sum_{i=1}^n (\text{prediction error}_i)^2 = \sum_{i=1}^n (Y_i - a - b \cdot X_i)^2$$

- Finds the line that minimizes the magnitude of the prediction errors!

Linear regression in R

- R will calculate least squares line for a data set using `lm()`
 - Syntax: `lm(y ~ x, data = mydata)`
 - `y` is the name of the dependent variable
 - `x` is the name of the independent variable
 - `mydata` is the data.frame where they live

```
fit <- lm(weight ~ steps_lag, data=health)
fit
```

```
##
## Call:
## lm(formula = weight ~ steps_lag, data = health)
##
## Coefficients:
## (Intercept)      steps_lag
##    170.6750       -0.2308
```

use `coef()` to extract estimated coefficients:

```
coef(fit)
```

```
## (Intercept)    steps_lag
```

```
## 170.6749706   -0.2307681
```

Interpretation: a 1-unit increase in X (1,000 steps) is associated with a decrease in the average weight of 0.231 pounds

Question: what would this model predict about the change in average weight for a 10,000 step increase in steps?

The broom package can provide nice summaries of the regression output

`augment()` can show fitted values, residuals and other unit-level statistics:

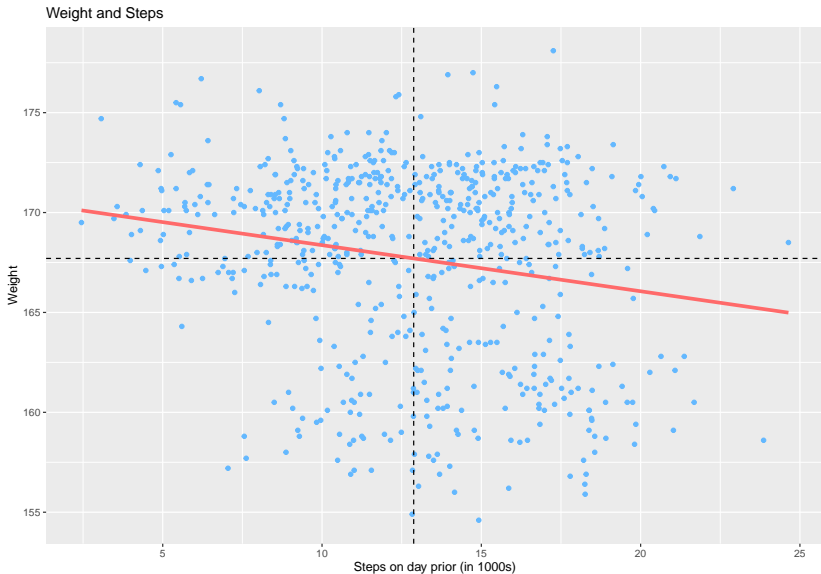
```
library(broom)
augment(fit) |>
  head()
```

```
## # A tibble: 6 x 8
##   weight steps_lag .fitted .resid   .hat .sigma   .cooksd .std.resid
##   <dbl>    <dbl>   <dbl> <dbl>   <dbl> <dbl>   <dbl>    <dbl>
## 1   169.    17.5    167.  2.46  0.00369 4.68 0.000513    0.526
## 2   168     18.4    166.  1.57  0.00463 4.68 0.000264    0.337
## 3   167.    19.6    166.  1.05  0.00609 4.68 0.000154    0.224
## 4   168.    10.4    168. -0.0750 0.00217 4.68 0.000000280 -0.0160
## 5   168.    18.7    166.  1.44  0.00496 4.68 0.000238    0.309
## 6   166.     9.14    169. -2.27  0.00296 4.68 0.000349   -0.485
```

Least squares line always goes through (\bar{X}, \bar{Y})

```
ggplot(health, aes(x = steps_lag, y = weight)) +  
  geom_point(color = "steelblue1") +  
  labs(  
    x = "Steps on day prior (in 1000s)",  
    y = "Weight",  
    title = "Weight and Steps") +  
  geom_hline(yintercept = mean(health$weight), linetype = "dashed") +  
  geom_vline(xintercept = mean(health$steps_lag), linetype = "dashed") +  
  geom_smooth(method = "lm", se = FALSE, color = "indianred1", size = 1.5)
```

Least squares line always goes through (\bar{X}, \bar{Y})



Estimated slope is related to correlation:

$$\hat{\beta} = (\text{correlation of X and Y}) \times \frac{\text{SD of Y}}{\text{SD of X}}$$

Mean of residuals is always 0

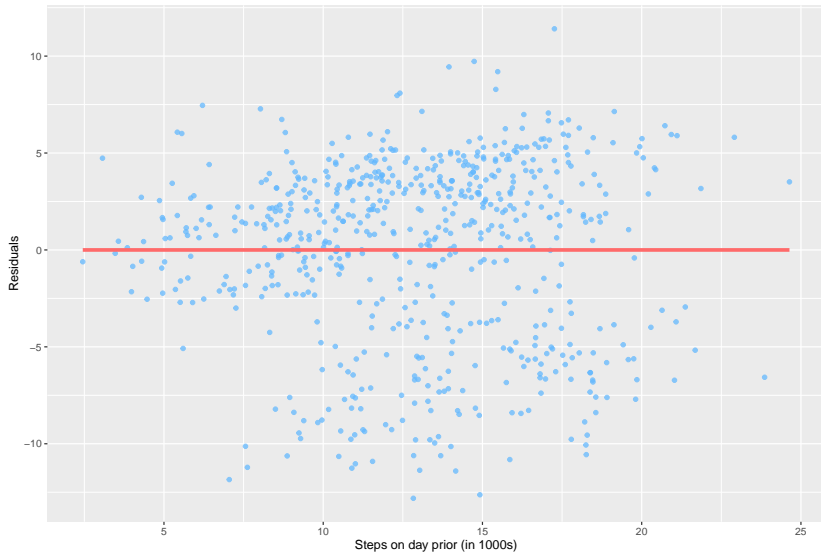
```
augment(fit) |>  
  summarize(mean(.resid))
```

```
## # A tibble: 1 x 1  
##   `mean(.resid)`  
##           <dbl>  
## 1      -1.21e-13
```

Plotting the residuals

```
augment(fit) |>
  ggplot(aes(x = steps_lag, y = .resid)) +
  geom_point(color = "steelblue1", alpha = 0.75) +
  labs(
    x = "Steps on day prior (in 1000s)",
    y = "Residuals",
    title = "Residual plot") +
  geom_smooth(method = "lm", se = FALSE, color = "indianred1", size = 1.5)
```


Residual plot



Another way to think of the regression line is a smoothed version of the binned means plot:

```
ggplot(health, aes(x = steps_lag, y = weight)) +  
  geom_point(color = "steelblue1", alpha = 0.25) +  
  labs(  
    x = "Steps on day prior (in 1000s)",  
    y = "Weight",  
    title = "Weight and Steps") +  
  stat_summary_bin(fun = "mean", color = "indianred1", size = 3,  
                   geom = "point", binwidth = 1) +  
  geom_smooth(method = "lm", se = FALSE, color = "indianred1", size = 1)
```

Weight and Steps

