Gov 50: 12. Prediction and Iteration

Matthew Blackwell

Harvard University

Roadmap

1. Prediction

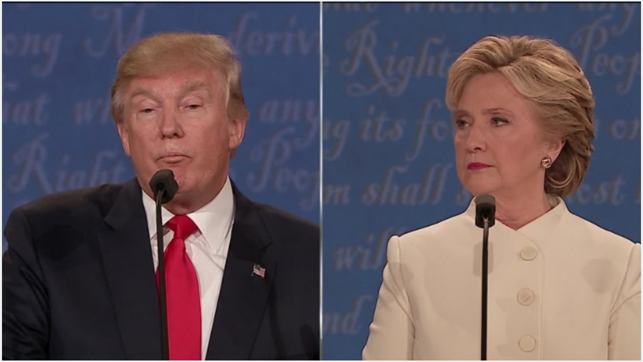
2. Loops

3. Evaluating the predictions

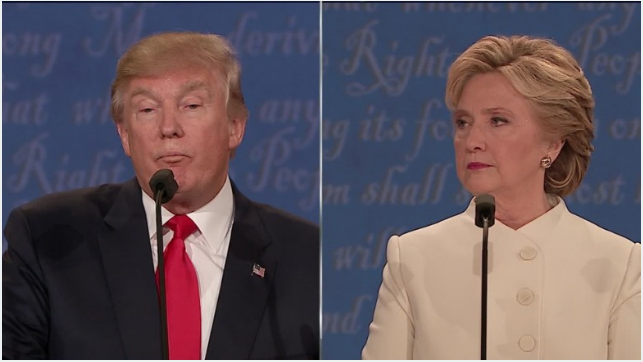
4. Time-series plot

1/ Prediction

2016 US Presidential Election

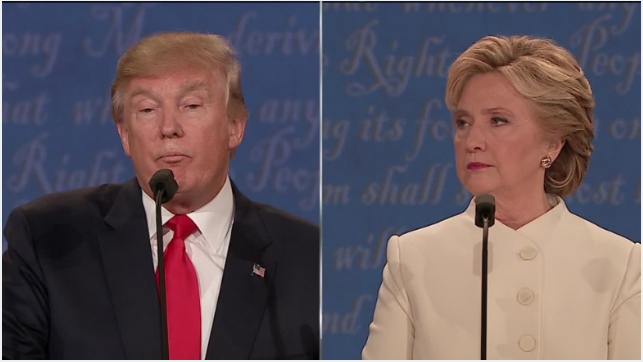


2016 US Presidential Election



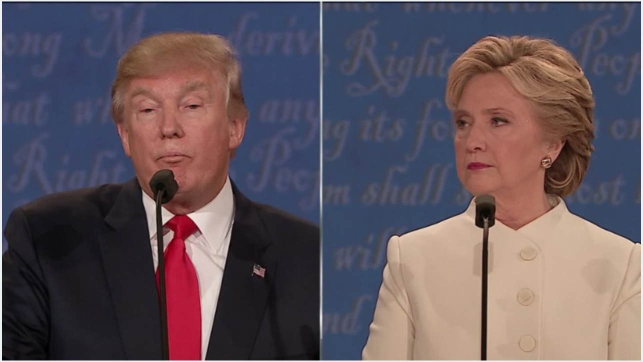
- 2016 election popular vote:

2016 US Presidential Election



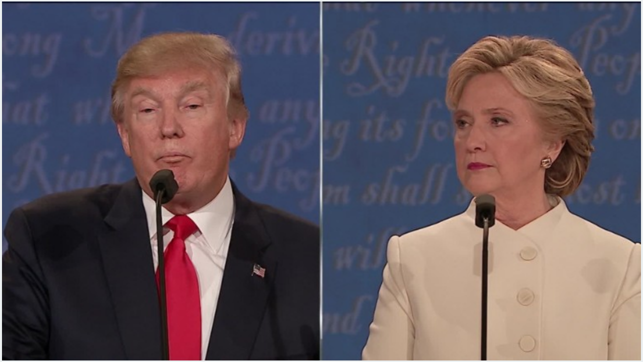
- 2016 election popular vote:
    - Clinton: 65,853,516 (48.2%)

2016 US Presidential Election



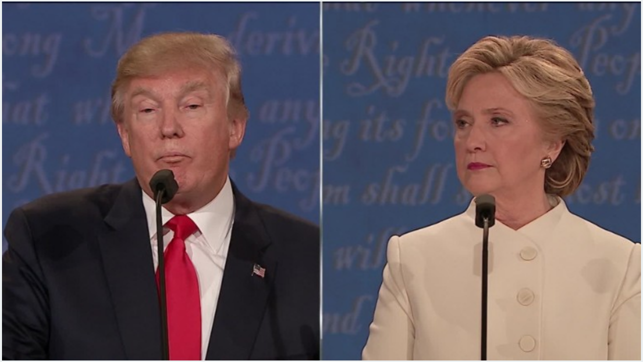
- 2016 election popular vote:
  - Clinton: 65,853,516 (48.2%)
  - Trump: 62,984,825 (46.1%)

2016 US Presidential Election



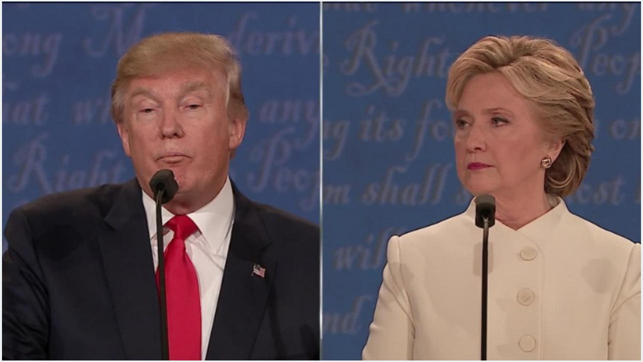
- 2016 election popular vote:
    - Clinton: 65,853,516 (48.2%)
    - Trump: 62,984,825 (46.1%)
- Why did Trump win? **Electoral college**

2016 US Presidential Election



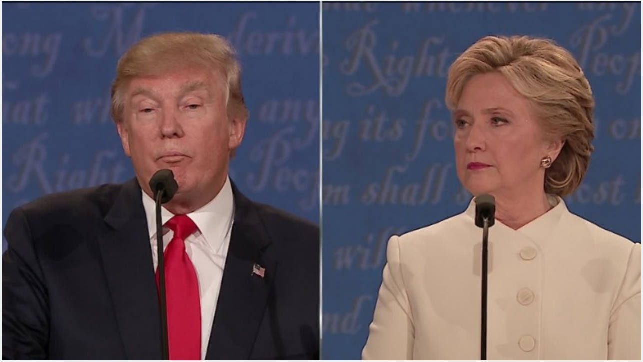
- 2016 election popular vote:
  - Clinton: 65,853,516 (48.2%)
  - Trump: 62,984,825 (46.1%)
- Why did Trump win? **Electoral college**
  - Trump: 304, Clinton: 227

2016 US Presidential Election



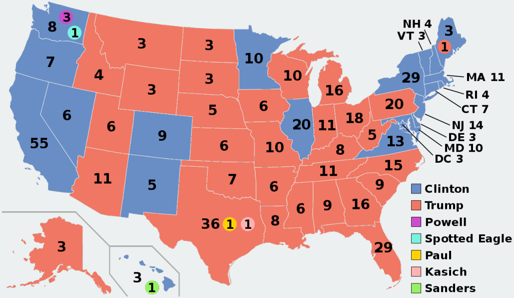
- 2016 election popular vote:
  - Clinton: 65,853,516 (48.2%)
  - Trump: 62,984,825 (46.1%)
- Why did Trump win? **Electoral college**
  - Trump: 304, Clinton: 227
- Election determined by 77,744 votes (margins in WI, MI, and PA)

2016 US Presidential Election



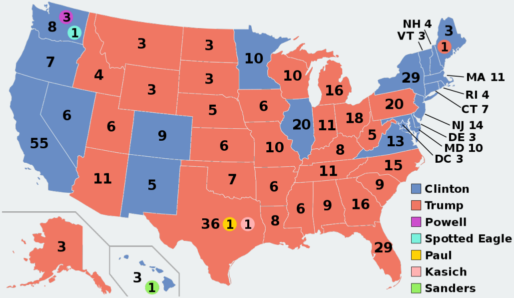
- 2016 election popular vote:
    - Clinton: 65,853,516 (48.2%)
    - Trump: 62,984,825 (46.1%)
- Why did Trump win? **Electoral college**
    - Trump: 304, Clinton: 227
- Election determined by 77,744 votes (margins in WI, MI, and PA)
    - 0.056% of the electorate (~136 million)

Predicting US Presidential Elections



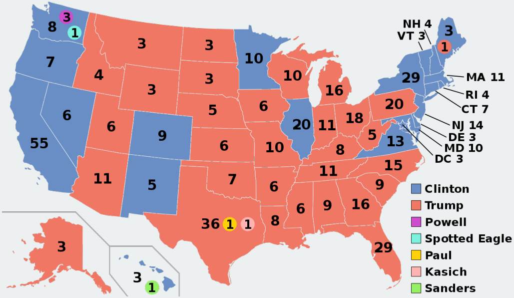
- **Electoral college system**

Predicting US Presidential Elections



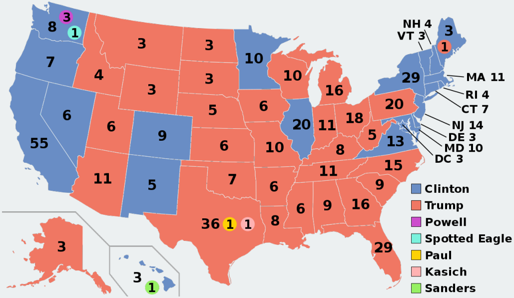
- **Electoral college system**
  - Must win an absolute majority of 538 electoral votes

Predicting US Presidential Elections



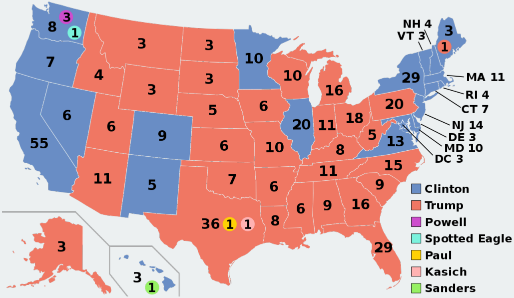
- **Electoral college system**
  - Must win an absolute majority of 538 electoral votes
  - 538 = 435 (House of Representatives) + 100 (Senators) + 3 (DC)

Predicting US Presidential Elections



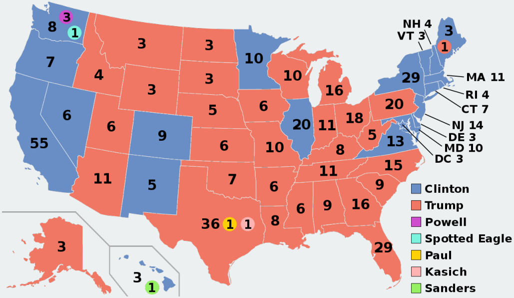
- **Electoral college system**
  - Must win an absolute majority of 538 electoral votes
  - 538 = 435 (House of Representatives) + 100 (Senators) + 3 (DC)
  - Must win at least 270 votes

Predicting US Presidential Elections



- **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

Predicting US Presidential Elections



- **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 ⇝ House vote
- Must predict winner of each state

Prediction strategy

- Predict state-level support for each candidate using polls

Prediction strategy

- Predict state-level support for each candidate using polls
- Allocate electoral college votes of that state to its predicted winner

Prediction strategy

- 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

Prediction strategy

- 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:

Prediction strategy

- 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:
    1. For each state, subset to polls within that state.

Prediction strategy

- 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:
    1. For each state, subset to polls within that state.
    2. Further subset the latest polls

Prediction strategy

- 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:
    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

Prediction strategy

- 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:

    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

Prediction strategy

- 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:
  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

Prediction strategy

- 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:
    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
- Sounds like a lot of subsets, ugh...

2/ Loops

A simple example

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

```r
library(gov50data)
cces_2020
```

```
## # A tibble: 51,551 x 6
##    gender race  educ                     pid3    turno~1 pres_~2
##    <fct>  <fct> <fct>                    <fct>     <dbl> <fct>
##  1 Male   White 2-year                   Republ~       1 Donald~
##  2 Female White Post-grad                Democr~      NA <NA>
##  3 Female White 4-year                   Indepe~       1 Joe Bi~
##  4 Female White 4-year                   Democr~       1 Joe Bi~
##  5 Male   White 4-year                   Indepe~       1 Other
##  6 Male   White Some college             Republ~       1 Donald~
##  7 Male   Black Some college             Not su~      NA <NA>
##  8 Female White Some college             Indepe~       1 Donald~
##  9 Female White High school graduate Republ~       1 Donald~
## 10 Female White 4-year                   Democr~       1 Joe Bi~
## # ... with 51,541 more rows, and abbreviated variable names
## #   1: turnout_self, 2: pres_vote
```

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
```

Subsetting with brackets

Note that 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
## [5] Other
## 7 Levels: Democrat Republican Independent ... not asked
```

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

```
## [1] Republican  Democrat    Independent Not sure
## [5] Other
## 7 Levels: Democrat Republican Independent ... not asked
```

Manually changing values, alternative

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

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

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

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

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

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

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

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

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

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

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

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

Recognizing the template

What if you had more values? Not efficient!

Recognizing the template

What if you had more values? Not efficient!

Recognize the template:

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

Recognizing the template

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?

Loops in R

for loop provide 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:

Loops in R

for loop provide 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

Loops in R

for loop provide 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.

Loops in R

for loop provide 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.

Loops in R

for loop provide 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.

Loops in R

for loop provide 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.

Loops in R

for loop provide 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.

2020 polling prediction

Election data: `pres20`

Name	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`:

Name	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 day
`pollster`	name of organization conducting poll
`sample_size`	name of organization conducting poll
`biden`	predicted support for Biden (percentage)
`trump`	predicted support for Trump (percentage)

Some preprocessing

```r
library(gov50data)

# 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, 2~
## $ state       <chr> "FL", "PA", "FL", "FL", "NV", "GA", "S~
## $ days_left   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
## $ pollster    <chr> "The Political Matrix/The Listener Gro~
## $ sample_size <dbl> 966, 499, 400, 1054, 1024, 1041, 817, ~
## $ biden       <dbl> 44.2, 48.4, 47.0, 47.3, 48.4, 45.4, 39~
## $ trump       <dbl> 48.0, 49.2, 48.2, 49.4, 49.1, 49.7, 51~
## $ margin      <dbl> -3.8, -0.8, -1.2, -2.1, -0.7, -4.3, -1~
```

Reminder of our goal

- 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

```r
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
```

Tidyverse alternative version

```
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

Polling errors

Prediction error = actual outcome − predicted outcome

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

```
## # A tibble: 51 x 8
##    state margin_pred    ev biden trump  other  margin errors
##    <chr>       <dbl> <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
```

Assessing the prediction error

Bias: average prediction error

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

```
## [1] -3.98
```

Assessing the prediction error

Bias: average prediction error

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

```
## [1] -3.98
```

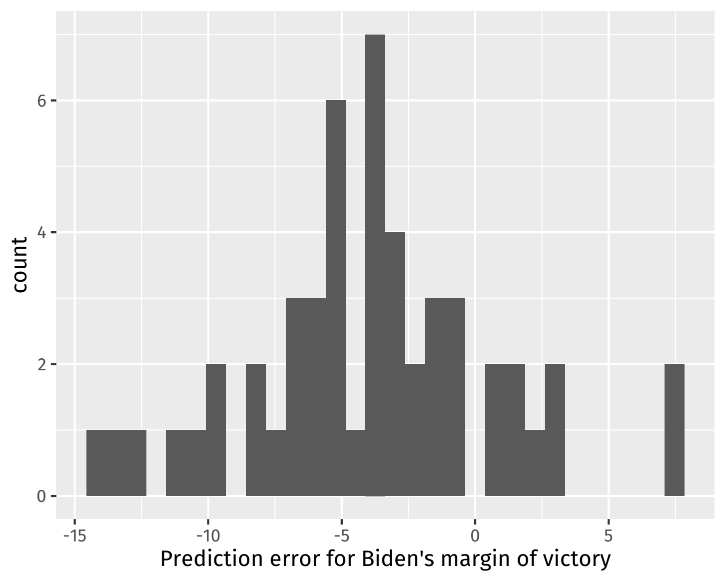
Root mean-square error: average magnitude of the prediction error

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

```
## [1] 6.07
```

Histogram of the errors

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



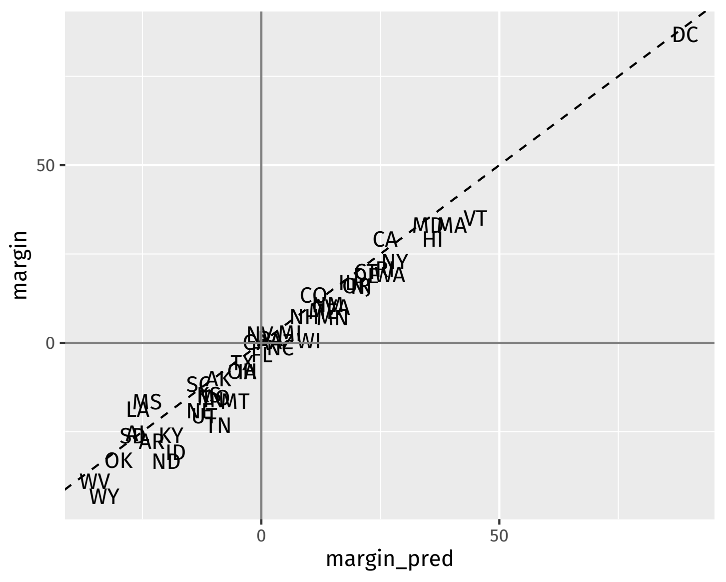
count
6
4
2
0
-15
-10
-5
0
5
Prediction error for Biden's margin of victory

Comparing polls to outcome

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")
```

Comparing polls to outcome



Classification

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
```

Classification

Election prediction: need to predict winner in each state:

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

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

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

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

- Prediction of binary outcome variable = **classification problem**

Classification

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 $\rightsquigarrow$ misclassification

Classification

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 $\rightsquigarrow$ misclassification
  1. **true positive**: predict Trump wins when he actually wins.

Classification

Election prediction: need to predict winner in each state:

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

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

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

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

- Prediction of binary outcome variable = **classification problem**
- Wrong prediction $\rightsquigarrow$ misclassification
    1. **true positive**: predict Trump wins when he actually wins.
    2. **false positive**: predict Trump wins when he actually loses.

Classification

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 $\rightsquigarrow$ 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.

Classification

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 $\rightsquigarrow$ 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.

Classification

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 $\rightsquigarrow$ 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] 0.922
```

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] 0.922
```

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>  <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

National polls

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

```
national_polls20
```

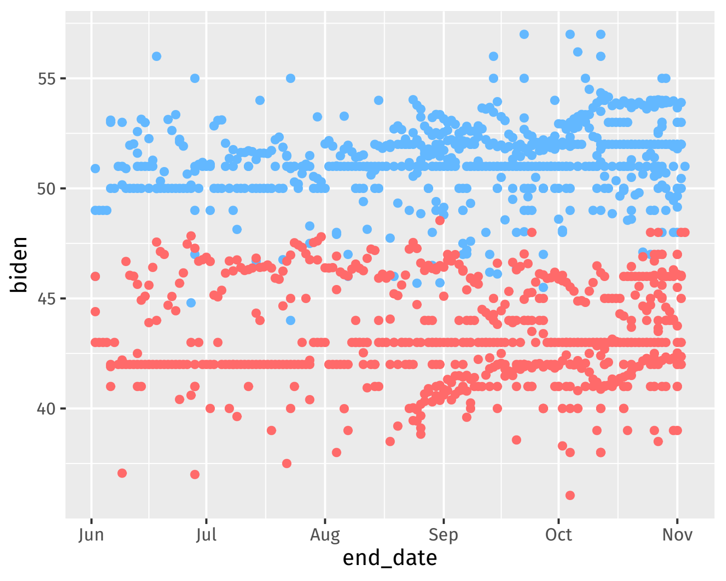
```
## # A tibble: 654 x 5
##    end_date   pollster                          sampl~1 biden trump
##    <date>     <chr>                               <dbl> <dbl> <dbl>
##  1 2020-11-03 Lake Research                        2400  51    48  
##  2 2020-11-02 Research Co.                         1025  50    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~           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
```

Plotting the raw results

```r
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.

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.

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 result as a 1-row tibble.

Dates in R

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"
## [5] "2020-11-02" "2020-11-02"
```

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

```
## [1] "2020-11-06" "2020-11-05" "2020-11-05" "2020-11-05"
## [5] "2020-11-05" "2020-11-05"
```

Lubridate to create dates

We can covert 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"
```

Getting a vector of dates

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"
## [5] "2020-06-07" "2020-06-08"
```

Moving window loop

```r
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)
```

Result

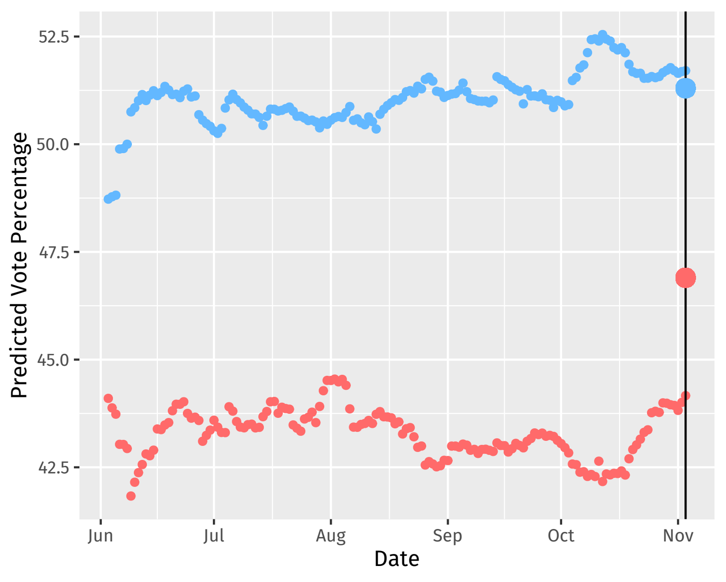
```
output
```

```
## # A tibble: 154 x 3
##    date       biden trump
##    <date>     <dbl> <dbl>
##  1 2020-06-03  48.7  44.1
##  2 2020-06-04  48.8  43.9
##  3 2020-06-05  48.8  43.7
##  4 2020-06-06  49.9  43.0
##  5 2020-06-07  49.9  43.0
##  6 2020-06-08  50    42.9
##  7 2020-06-09  50.8  41.8
##  8 2020-06-10  50.8  42.2
##  9 2020-06-11  51.0  42.4
## 10 2020-06-12  51.2  42.6
## # ... with 144 more rows
```

Let's plot

```
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 =
  geom_point(aes(x = election_day, y = 46.9), color = "indianred1", size =
  labs(
    x = "Date",
    y = "Predicted Vote Percentage"
  )
```

Let’s plot


52.5
50.0
47.5
45.0
42.5
Predicted Vote Percentage
Jun
Jul
Aug
Sep
Oct
Nov
Date