PSC4375: Loops & Predicting

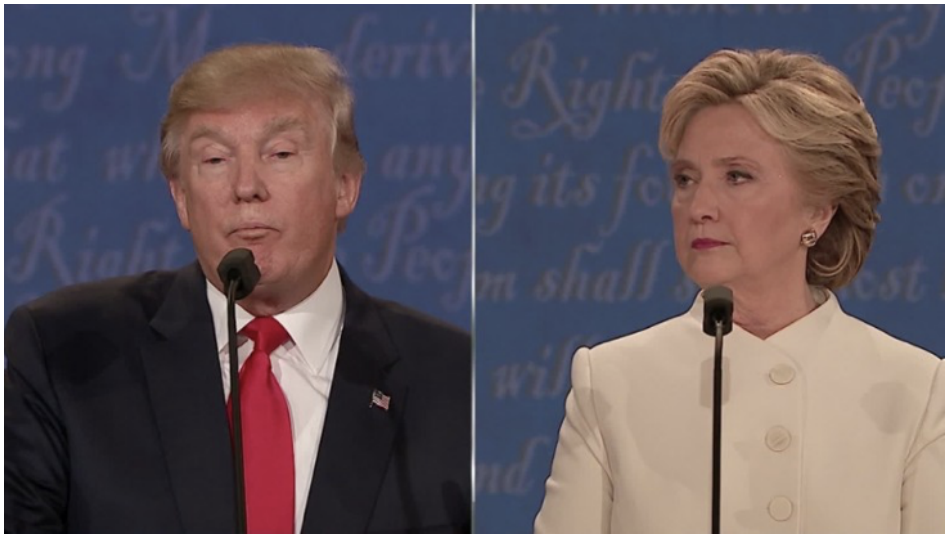
Week 5: Lecture 9

Prof. Weldzius

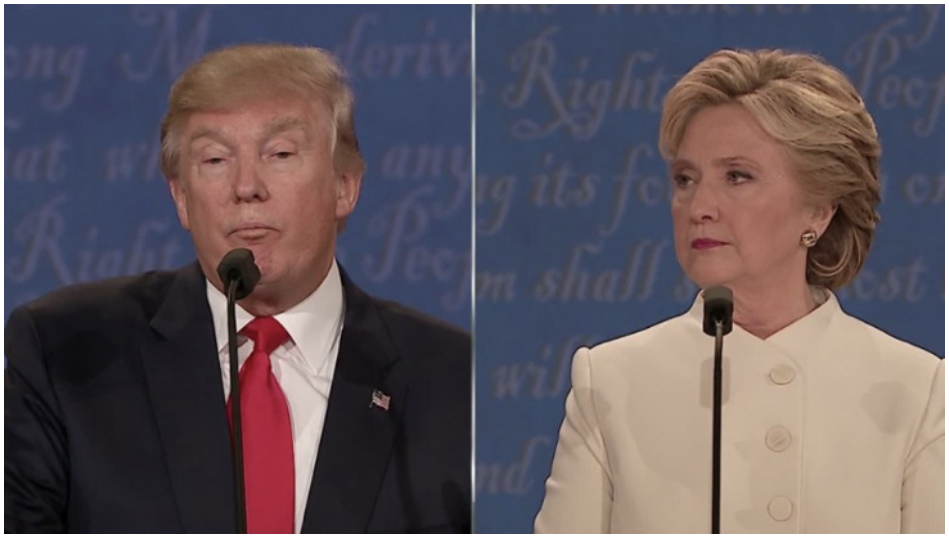
Villanova University

Slides Updated: 2025-02-17

2016 US Presidential Election

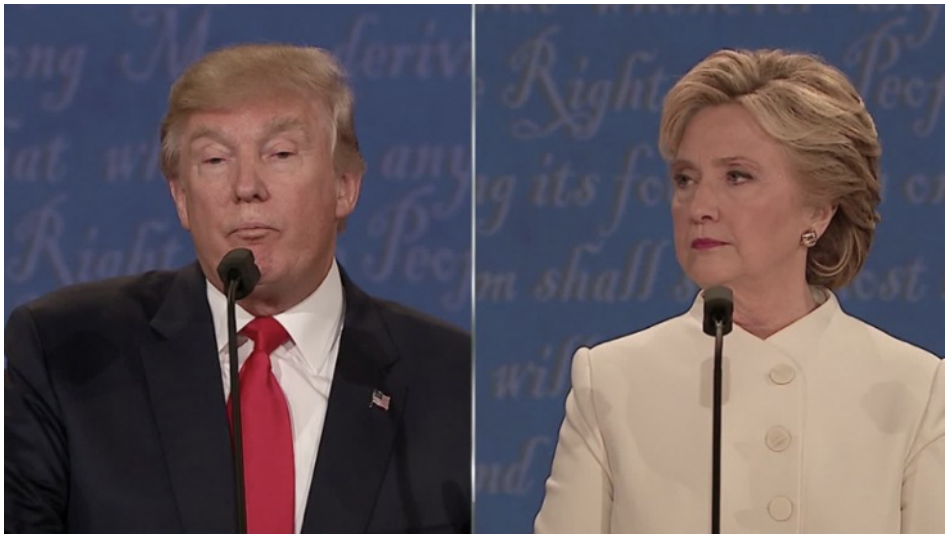


2016 US Presidential Election



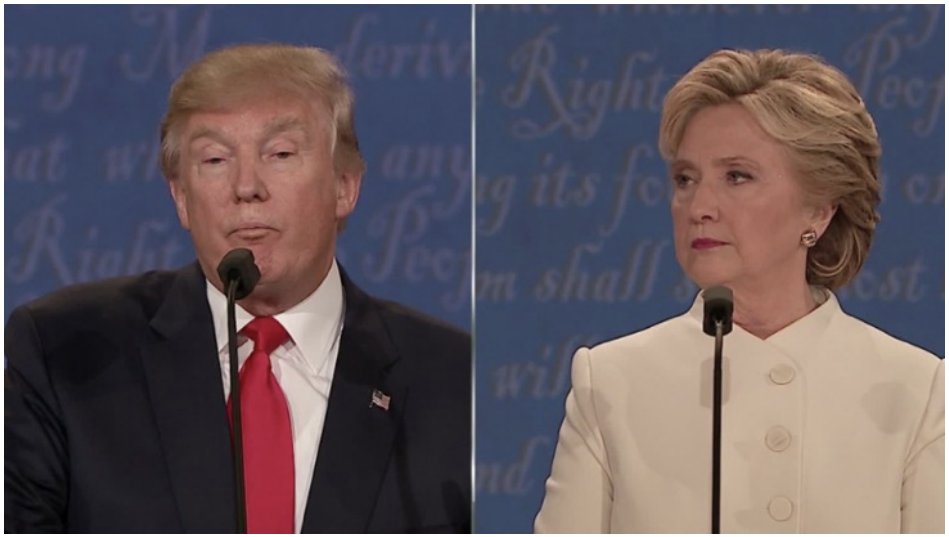
- 2016 election popular vote:

2016 US Presidential Election



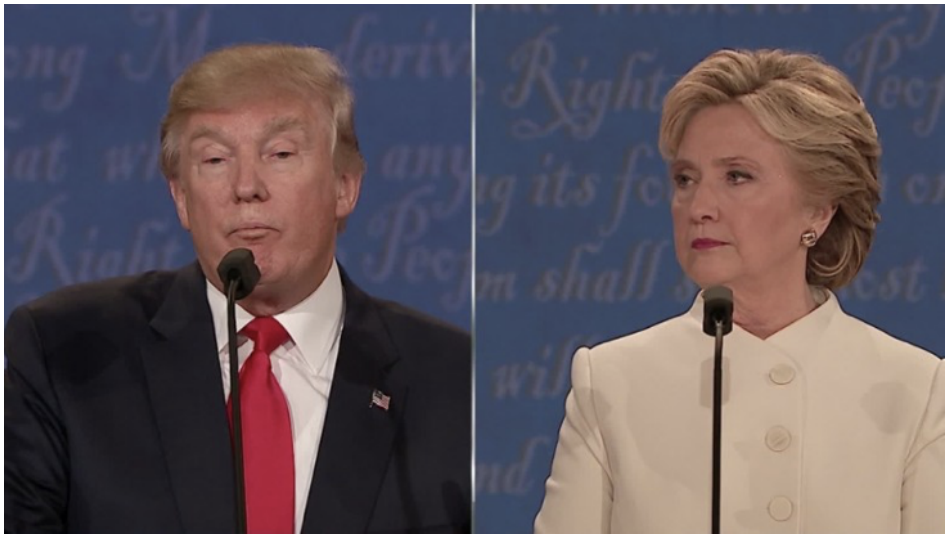
- 2016 election popular vote:
  - Clinton: 65,853,516 (48.2%)
  - Trump: 62,984,825 (46.1%)

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**
  - 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, PA)
  - 0.056% of the electorate (~136million)

Predicting US Presidential Elections

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

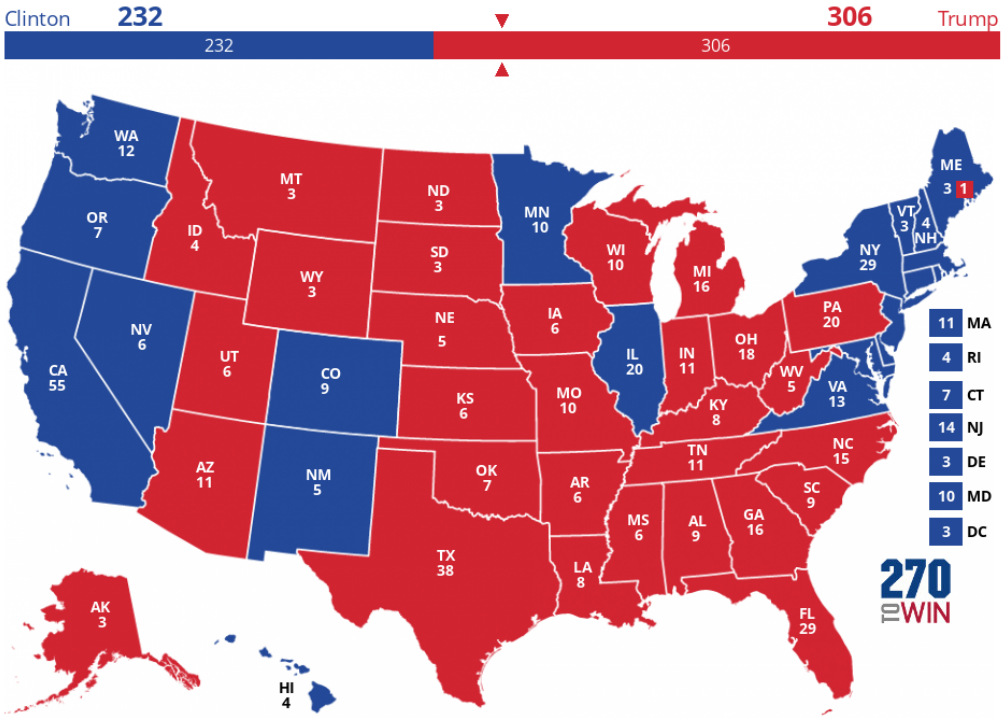
- **Electoral college system**
  - Must win an absolute majority of 538 electoral votes
  - 538 = 435 (House of Representatives) + 100 (Senators) + 3 (DC)
  - Must win at least 270 votes
  - nobody wins an absolute majority $\rightsquigarrow$ House vote

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 $\rightsquigarrow$ House vote
- Must predict winner of each state

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
- Must predict winner of each state



Prediction strategy

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...

Multiplication

Multiplication

- Let's create a new variable that multiples each value in a vector by 2:

Multiplication

- Let's create a new variable that multiples each value in a vector by 2:
  - Easy in R: `values * 2`

Multiplication

- Let's create a new variable that multiples each value in a vector by 2:
  - Easy in R: `values * 2`
  - Pretend you didn't know this approach

Multiplication

```
values <- c(2,4,6)
```

Multiplication

```r
values <- c(2,4,6)

## number of values
n <- length(values)
```

Multiplication

```r
values <- c(2,4,6)

## number of values
n <- length(values)

## create container to hold results
results <- rep(NA, times = n)
```

Multiplication

```r
values <- c(2,4,6)

## number of values
n <- length(values)

## create container to hold results
results <- rep(NA, times = n)

## multiply each value by 2
results[1] <- values[1] * 2
results[2] <- values[2] * 2
results[3] <- values[3] * 2
```

Multiplication

```
values <- c(2,4,6)

## number of values
n <- length(values)

## create container to hold results
results <- rep(NA, times = n)

## multiply each value by 2
results[1] <- values[1] * 2
results[2] <- values[2] * 2
results[3] <- values[3] * 2

results
```

```
## [1]  4  8 12
```

Loops in R

Loops in R

- What if you had more values? Not efficient!

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

```
for (i in X) {
  expression1
  expression2
  ...
  expression3
}
```

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

```
for (i in X) {
  expression1
  expression2
  ...
  expression3
}
```

- Elements of a loop:

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

```
for (i in X) {
  expression1
  expression2
  ...
  expression3
}
```

- Elements of a loop:
  - `i`: counter (can use any name)

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

```
for (i in X) {
  expression1
  expression2
  ...
  expression3
}
```

- Elements of a loop:
  - `i`: counter (can use any name)
  - `X`: vector containing a set of ordered values the counter takes

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

```
for (i in X) {
  expression1
  expression2
  ...
  expression3
}
```

- Elements of a loop:
  - `i`: counter (can use any name)
  - `X`: vector containing a set of ordered values the counter takes
  - `expression`: a set of expressions that will be repeatedly evaluated.

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

```
for (i in X) {
  expression1
  expression2
  ...
  expression3
}
```

- Elements of a loop:
  - `i`: counter (can use any name)
  - `X`: vector containing a set of ordered values the counter takes
  - `expression`: a set of expressions that will be repeatedly evaluated.
  - `{ }`: curly braces to define beginning and end of the loop.

Loops in R

- What if you had more values? Not efficient!
  - **for loop**: a way to iteratively execute the same code multiple times.
- Basic structure:

```
for (i in X) {
  expression1
  expression2
  ...
  expression3
}
```

- Elements of a loop:
  - `i`: counter (can use any name)
  - `X`: vector containing a set of ordered values the counter takes
  - `expression`: a set of expressions that will be repeatedly evaluated.
  - `{ }`: curly braces to define beginning and end of the loop.
- Indentation is important for readability of the code.

Loop example:

```r
values <- c(2,4,6)
n <- length(values)
results <- rep(NA, times = n)

## begin loop
for (i in 1:n) {
  results[i] <- values[i] * 2
  print(str_c(values[i], " times 2 is equal to ", results[i]))
}
```

```
## [1] "2 times 2 is equal to 4"
## [1] "4 times 2 is equal to 8"
## [1] "6 times 2 is equal to 12"
```

2016 polling prediction

- Election data: `pres.csv`

Name	Description
`state_abb`	abbreviated name of state
`clinton`	Clinton's vote share (percentage)
`trump`	Trump's vote share (percentage)

- Polling data `polls16.csv`

Name	Description
`state`	abbreviated name of state in which polls was conducted
`middate`	middate of the period when polls was conducted
`daysleft`	number of days between middate and election day
`pollster`	name of organization conducting poll
`clinton`	predicted support for Clinton (percentage)
`trump`	predicted support for Trump (percentage)

Some preprocessing

```
## download; don't forget to setwd()
pres16 <- read_csv("../data/pres2016.csv")
polls16 <- read_csv("../data/polls2016.csv")

## calculate Trump's margin of victory
polls16 <- polls16 %>%
  mutate(margin = Trump - Clinton)
pres16 <- pres16 %>%
  mutate(margin = Trump - Clinton)
```

What does the data look like?

```
head(polls16)
```

```
## # A tibble: 6 x 8
##      id state Clinton Trump days_to_election electoral_votes
##   <dbl> <chr>   <dbl> <dbl>             <dbl>           <dbl>
## 1 26255 TX         38    41                24              38
## 2 26253 WI         48    44                23              10
## 3 26252 VA         54    41                23              13
## 4 26251 NV         47    40                19               6
## 5 26250 TX         46    48                23              38
## 6 26249 NH         50    43                23               4
## # i 2 more variables: population <chr>, margin <dbl>
```

Poll prediction for each state

```r
## place holder
poll.pred <- rep(NA, 51)

## get list of unique state names to iterate over
state_names <- unique(polls16$state)

## add labels to place holder
names(poll.pred) <- state_names
```

Poll prediction for each state

```
for (i in seq_along(state_names)) {
```

Poll prediction for each state

```
for (i in seq_along(state_names)) {
  ## subset the ith state
  state.data <- polls16 %>%
    filter(state == state_names[i])
```

Poll prediction for each state

```
for (i in seq_along(state_names)) {
  ## subset the ith state
  state.data <- polls16 %>%
    filter(state == state_names[i])

  ## pull out the closest date (minimum days to election)
  min_days <- min(state.data$days_to_election)
```

Poll prediction for each state

```
for (i in seq_along(state_names)) {
  ## subset the ith state
  state.data <- polls16 %>%
    filter(state == state_names[i])

  ## pull out the closest date (minimum days to election)
  min_days <- min(state.data$days_to_election)

  ## subset only the latest polls within the state
  state.data <- state.data %>%
    filter(days_to_election == min_days)
```

Poll prediction for each state

```
for (i in seq_along(state_names)) {
  ## subset the ith state
  state.data <- polls16 %>%
    filter(state == state_names[i])

  ## pull out the closest date (minimum days to election)
  min_days <- min(state.data$days_to_election)

  ## subset only the latest polls within the state
  state.data <- state.data %>%
    filter(days_to_election == min_days)

  ## compute the mean of the latest polls and store it
  poll.pred[i] <- mean(state.data$margin)
}
head(poll.pred)
```

```
##  TX  WI  VA  NV  NH  PA
##   2  -8 -15  -7  -7  -6
```

Poll prediction for each state (my way)

```
poll.list <- list()
state_names <- unique(polls16$state)

for (i in seq_along(state_names)) {
  state.data <- polls16 %>%
    filter(state == state_names[i]) %>%
    filter(days_to_election == min(days_to_election)) %>%
    mutate(margin_poll = mean(margin)) %>%
    select(state, margin_poll)
  poll.list[[i]] <- state.data
  print(i)
}
PollPred <- do.call(rbind,poll.list)
head(PollPred)
```

Comparing polls to outcomes