Measurement and visualization, 2

Frank Edwards

9/17/2019

Prediction

- To descriptively learn about the future (weather, elections, economic changes)
- To validate theories or arguments:
    - Valid causal inference requires successful prediction of counterfactual claims
    - e.g. if X were different, what value of Y would we observe?

- Loops repeat the same set of operations a specified number of times
- Very useful when we need to apply a complex batch of code over a set of rows/columns/data.frames

```r
for(i in 1:3){ ### counter, counter range, open loop
  print(i+2) ### expression to evaluate
} ### close the loop
```

```
## [1] 3
## [1] 4
## [1] 5
```

1. Think through the procedure
2. Pre-allocate a vector/data.frame for output with correct dimensions for output
3. Run
4. Reshape output to integrate with other objects

Goal: Calculate products of 2 for consecutive integers between 1 and 5

```r
### create index vector
digits<-seq(from=1,to=5, by=1)
### allocate output vector of needed length
output<-rep(NA, length(digits))
for(i in 1:length(digits)){
  output[i]<-digits[i]*2 # store in output at position i
}
## view!
output
```

```
## [1]  2  4  6  8 10
```

- We can loop within loops!
- We can use if{} and else{} within loops
- Calculate $x = (2x)!$ for $x \in [1, 5]$

```
### create index vector
digits<-seq(from=1,to=5, by=1)
### allocate output vector of needed length
factorial<-rep(NA, length(digits))
for(i in 1:length(digits)){
  start_pt<-digits[i]*2 # factorial start point
  fact_out<-start_pt
  for(k in (start_pt-1):1){
    fact_out<-fact_out * k
  }
  factorial[i]<-fact_out
}
## view!
factorial
```

```
## [1]       2      24     720   40320 3628800
```

Whoa - what is this doing?

Add an iteration counter and output to check progress

```
## [1] "i= 1 , k = 1 , start_p= 2 , fact_out= 2"
## [1] "i= 2 , k = 3 , start_p= 4 , fact_out= 12"
## [1] "i= 2 , k = 2 , start_p= 4 , fact_out= 24"
## [1] "i= 2 , k = 1 , start_p= 4 , fact_out= 24"
## [1] "i= 3 , k = 5 , start_p= 6 , fact_out= 30"
## [1] "i= 3 , k = 4 , start_p= 6 , fact_out= 120"
## [1] "i= 3 , k = 3 , start_p= 6 , fact_out= 360"
## [1] "i= 3 , k = 2 , start_p= 6 , fact_out= 720"
## [1] "i= 3 , k = 1 , start_p= 6 , fact_out= 720"
## [1] "i= 4 , k = 7 , start_p= 8 , fact_out= 56"
## [1] "i= 4 , k = 6 , start_p= 8 , fact_out= 336"
## [1] "i= 4 , k = 5 , start_p= 8 , fact_out= 1680"
## [1] "i= 4 , k = 4 , start_p= 8 , fact_out= 6720"
## [1] "i= 4 , k = 3 , start_p= 8 , fact_out= 20160"
## [1] "i= 4 , k = 2 , start_p= 8 , fact_out= 40320"
## [1] "i= 4 , k = 1 , start_p= 8 , fact_out= 40320"
## [1] "i= 5 , k = 9 , start_p= 10 , fact_out= 90"
## [1] "i= 5 , k = 8 , start_p= 10 , fact_out= 720"
## [1] "i= 5 , k = 7 , start_p= 10 , fact_out= 5040"
## [1] "i= 5 , k = 6 , start_p= 10 , fact_out= 30240"
## [1] "i= 5 , k = 5 , start_p= 10 , fact_out= 151200"
## [1] "i= 5 , k = 4 , start_p= 10 , fact_out= 604800"
## [1] "i= 5 , k = 3 , start_p= 10 , fact_out= 1814400"
## [1] "i= 5 , k = 2 , start_p= 10 , fact_out= 3628800"
## [1] "i= 5 , k = 1 , start_p= 10 , fact_out= 3628800"
```

```r
polls<-read_csv("./data/polls2016.csv")
## if not in the .RMD slide file
# polls<-read_csv("./slides/data/polls2016.csv")
head(polls)

## # A tibble: 6 x 7
##       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

polls<-polls %>%
  filter(population=="Likely Voters") %>%
```

Goal: make a table with the mean, median, and SD for both candidates

```
## initiate a storage object with correct dimensions
descriptives_out<-data.frame("Candidate" = rep(NA,2),
                             "Mean" = rep(NA,2),
                             "Median" = rep(NA,2),
                             "SD" = rep(NA,2))
```

```r
## Create index vector for select()
columns<-c("Clinton", "Trump")

for(i in 1:length(columns)){ ## use vector length for counter range
  temp<-polls %>%
    pull(columns[i])
  descriptives_out$Candidate[i]<-columns[i]
  descriptives_out$Mean[i]<-mean(temp)
  descriptives_out$Median[i]<-median(temp)
  descriptives_out$SD[i]<-sd(temp)
}
```

```
descriptives_out
```

```
##   Candidate     Mean Median       SD
## 1   Clinton 44.14168     44 6.970659
## 2     Trump 42.95598     42 7.586107
```

- Loops will have bugs and will be frustrating!
- In the console, manually set an index: i.e. `i<-1`
- Then individually run lines *inside* the loop to verify that they work
- More advanced debugging tools for R scripts:
  `https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio`

Yes! We'll do it later in the lecture

You usually don't need to use loops with tidyverse coding, but it's still useful to learn.

- Generally, we have to pre-allocate an object with the correct dimensions for a loop
- e.g. If we want to store 5 rows of 2 columns, we need to make an object with those dimensions before the loop
- However, we can use lists to let R flexibly store loop output

```r
### create index vector
digits<-seq(from=1,to=5, by=1)
### allocate output vector of needed length
output<-list()
for(i in 1:length(digits)){
  output[[i]]<-digits[i]*2 # store in output at position i
}
str(output)
```

```
## List of 5
##  $ : num 2
##  $ : num 4
##  $ : num 6
##  $ : num 8
##  $ : num 10
```

```r
## make into vector after processing
unlist(output)
```

```
## [1]  2  4  6  8 10
```

```r
### use bind_rows() instead of unlist()
### if you are storing data frames
```

Data for today: election results

```r
results<-read_csv("./data/1976-2016-president.csv")
head(results)
```

```
## # A tibble: 6 x 14
##    year state state_po state_fips state_cen state_ic office candidate party
##   <dbl> <chr> <chr>         <dbl>     <dbl>    <dbl> <chr>  <chr>     <chr>
## 1  1976 Alab~ AL                1        63       41 US Pr~ Carter, ~ demo~
## 2  1976 Alab~ AL                1        63       41 US Pr~ Ford, Ge~ repu~
## 3  1976 Alab~ AL                1        63       41 US Pr~ Maddox, ~ amer~
## 4  1976 Alab~ AL                1        63       41 US Pr~ "Bubar, ~ proh~
## 5  1976 Alab~ AL                1        63       41 US Pr~ Hall, Gus comm~
## 6  1976 Alab~ AL                1        63       41 US Pr~ Macbride~ libe~
## # ... with 5 more variables: writein <lgl>, candidatevotes <dbl>,
## #   totalvotes <dbl>, version <dbl>, notes <lgl>
```

```r
results<-results %>%
  filter(year==2016) %>%
  filter(candidate=="Clinton, Hillary" |
           candidate=="Trump, Donald J.") %>%
  group_by(state_po, candidate) %>%
  summarise(pct_vote = sum(candidatevotes)/sum(totalvotes) * 100)
```

- We can join (or merger) two data frames together by common variables
- Joining variables must have identical column names, types, and values

*How are both datasets structured? What common variables could we join
on?*

```
glimpse(polls)
```

```
## Observations: 727
## Variables: 5
## $ state          <chr> "TX", "WI", "VA", "NV", "TX", "NH", "PA", "NV...
## $ Clinton        <dbl> 38, 48, 54, 47, 46, 50, 51, 47, 51, 46, 51, 4...
## $ Trump          <dbl> 41, 44, 41, 40, 48, 43, 42, 46, 42, 48, 43, 4...
## $ days_to_election <dbl> 24, 23, 23, 19, 23, 23, 23, 23, 23, 23, 23, 2...
## $ electoral_votes <dbl> 38, 10, 13, 6, 38, 4, 20, 6, 16, 18, 15, 6, 1...
```

```
glimpse(results)
```

```
## Observations: 102
## Variables: 3
## Groups: state_po [51]
## $ state_po  <chr> "AK", "AK", "AL", "AL", "AR", "AR", "AZ", "AZ", "CA"...
## $ candidate <chr> "Clinton, Hillary", "Trump, Donald J.", "Clinton, Hi...
## $ pct_vote  <dbl> 36.550871, 51.281512, 34.357946, 62.083092, 33.65312...
```

20

- State abbreviation is a common column for both
- Candidate is a column in results, and is spread over column names in polls
- We want to join, such that the election results for each candidate are joined onto each poll for a state.
- For example, Nevada poll results for Clinton should match onto Nevada election results
- Note that there is more than one poll available for most states, but only one election result

Rename state in polls to state_po to match across data.frames

```
polls<-polls %>%
  rename(state_po = state)
names(polls)
```

```
## [1] "state_po"          "Clinton"           "Trump"
## [4] "days_to_election"  "electoral_votes"
```

Spread candidate across columns in results

- Take the candidate column in results, and make one column for each candidate
- Note that this structure matches the structure of polls

```
results_wide<-results %>%
  mutate(candidate = case_when(
    candidate == "Clinton, Hillary" ~ "Clinton",
    candidate == "Trump, Donald J." ~ "Trump"
  )) %>%
  spread(key = candidate, value = pct_vote) %>%
  rename(clinton_vote = Clinton,
         trump_vote = Trump)

head(results_wide)

## # A tibble: 6 x 3
## # Groups:   state_po [6]
##   state_po clinton_vote trump_vote
##   <chr>           <dbl>      <dbl>
## 1 AK               36.6       51.3
## 2 AL               34.4       62.1
## 3 AR               33.7       60.6
## 4 AZ               45.1       48.7
## 5 CA               61.7       31.6
## 6 CO               48.2       43.3
```

- left_join() joins the object on the right to the object on the left, retaining all rows in the left hand object, but potentially removing rows in the right hand object.
- All columns are preserved.

```
polls_results<-polls %>%
  left_join(results_wide)

glimpse(polls_results)
```

```
## Observations: 727
## Variables: 7
## $ state_po       <chr> "TX", "WI", "VA", "NV", "TX", "NH", "PA", "NV...
## $ Clinton        <dbl> 38, 48, 54, 47, 46, 50, 51, 47, 51, 46, 51, 4...
## $ Trump          <dbl> 41, 44, 41, 40, 48, 43, 42, 46, 42, 48, 43, 4...
## $ days_to_election <dbl> 24, 23, 23, 19, 23, 23, 23, 23, 23, 23, 23, 2...
## $ electoral_votes <dbl> 38, 10, 13, 6, 38, 4, 20, 6, 16, 18, 15, 6, 1...
## $ clinton_vote   <dbl> 43.23526, 46.45384, 49.75135, 47.91782, 43.23...
## $ trump_vote     <dbl> 52.23469, 47.21818, 44.42765, 45.50070, 52.23...
```

Check data structure to ensure we didn't create duplicates in the final object

```
nrow(polls)
```

```
## [1] 727
```

```
nrow(polls_results)
```

```
## [1] 727
```

```
ncol(polls)
```

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

```
ncol(polls_results)
```

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

## Calculate prediction error

Error is a general term for how wrong our guess is. We can generally calculate error by subtracting the observation from our prediction.

prediction error = predicted value - observed value.

```r
polls_results<-polls_results %>%
  mutate(error.clinton = Clinton - clinton_vote,
         error.trump = Trump - trump_vote)

## format error data for plotting / faceting
## turn wide -> long. gather() is the inverse of spread()
plot_errors<-polls_results %>%
  select(error.clinton, error.trump) %>%
  gather(key = "candidate", value  = "error") %>%
  mutate(candidate = case_when(
    candidate == "error.clinton" ~ "Clinton",
    candidate == "error.trump" ~ "Trump"
  ))
```

Evaluate the errors

```
ggplot(plot_errors,
       aes(x = error)) +
  geom_histogram() +
  geom_vline(aes(xintercept = 0), lty=2) +
  facet_wrap(~candidate)
```
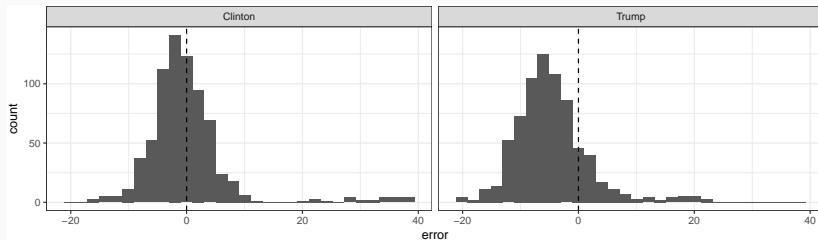
```
polls_results %>%
  summarise(error.clinton.mean = mean(error.clinton),
            error.trump.mean = mean(error.trump))


## # A tibble: 1 x 2
##   error.clinton.mean error.trump.mean
##                <dbl>            <dbl>
## 1            -0.0737            -4.66
```

Root Mean Square Error

RMSE provides a measure of absolute error, where positive and negative errors don't negate each other

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y} - y)^2}{n}}$$

```
polls_results %>%
  summarise(rmse.clinton = sqrt(mean(error.clinton^2)),
            rmse.trump = sqrt(mean(error.trump^2)))
```

```
## # A tibble: 1 x 2
##    rmse.clinton rmse.trump
##           <dbl>      <dbl>
## 1          7.34       7.91
```

1. Polls had similar magnitude of error for both candidates (RMSE)
2. Poll errors were consistently negative for Trump, were zero on average for Clinton.

How many polls called it right?

1. Make an average prediction for each state across polls
2. Whichever candidate has the highest average polling number is predicted the winner

Making a prediction based on the polls

```
polls_classify<-polls_results %>%
  group_by(state_po) %>%
  summarise(clinton_mn = mean(Clinton),
            trump_mn = mean(Trump)) %>%
  mutate(clinton_wins_pred = clinton_mn>trump_mn)

table(polls_classify$clinton_wins_pred)


##
## FALSE  TRUE
##    24    26
```

```
polls_classify %>%
  left_join(polls %>%
              select(state_po, electoral_votes) %>%
              distinct()) %>%
  summarise(clinton_ec_votes_share_pred = sum(clinton_wins_pred

## # A tibble: 1 x 1
##   clinton_ec_votes_share_pred
##                         <dbl>
## 1                       0.628

## actual result
227/538

## [1] 0.4219331
```

Bold cells are correct classifications.

	Positive, obs.	Negative, obs.
Positive, pred.	**True positive**	False positive
Negative, pred.	False negative	**True negative**

- First, join the election data onto our predictions
- Remove duplicate rows (because many polls are run per state, but only one election!)

```
polls_classify<-polls_classify %>%
  select(state_po, clinton_wins_pred) %>%
  left_join(polls_results %>%
              select(state_po, clinton_vote, trump_vote) %>%
              distinct())
```

- Then make an election binary outcome

```
polls_classify<-polls_classify%>%
  mutate(clinton_wins_vote = clinton_vote>trump_vote) %>%
  select(-clinton_vote, -trump_vote)
```

```r
## calculate proportion of accurate classifications
## i.e. clinton_wins_pred == clinton_wins_vote

polls_classify %>%
  summarise(mean(clinton_wins_pred == clinton_wins_vote))
```

```
## # A tibble: 1 x 1
##   `mean(clinton_wins_pred == clinton_wins_vote)`
##                                            <dbl>
## 1                                           0.88
```

Which ones did they get wrong?

```r
## Get misclassifications
polls_classify %>%
  filter(clinton_wins_pred!=clinton_wins_vote)
```

```
## # A tibble: 6 x 3
##   state_po clinton_wins_pred clinton_wins_vote
##   <chr>    <lgl>             <lgl>
## 1 FL       TRUE              FALSE
## 2 MI       TRUE              FALSE
## 3 NC       TRUE              FALSE
## 4 OH       TRUE              FALSE
## 5 PA       TRUE              FALSE
## 6 WI       TRUE              FALSE
```

What kind of classification error is this?

- Prediction and classification are core practices in statistics
- We can make predictions, then compare them to actual outcomes to evaluate our performance
- The best test of a theory is prediction. Keep predictive validation in mind when designing research and assessing theory.

NO HOMEWORK THIS WEEK

In lab, we will practice loops, joins, gathers and spreads