HW1: Forecasting Election Results

Summary

Learning goals in this HW are:

- gain more experience investigating and then implementing iteration within functions
- develop a habit of making commits of work and keeping your GitHub remote repo in sync with your local one

The scenario for this HW is the following. A notional election with two candidates, A and B, is going to be held in 5 weeks in a country with 52 states. We would like to make a probabilistic forecast for the outcome of the election based on the past 11 weeks of weekly polling results of the percent of voters in each state supporting candidate A. This is a popular election so we all we need to consider is the population-weighted aggregate of state-level vote shares for candidate A.

Instead of looking at any real polling numbers or populations, we're going to simulate 11 weeks of weekly polling results using a *random walk* (RW) model and also simulate 52 state fractions of the national population. We will then "forget" how we produced the polling data and fit a **new** RW model to the data with the goal of using this model along with the simulated population fractions to forecast the result of the election.

Side note: This is part of a common strategy in statistics for investigating and evaluating a modeling technique - use a model to create data and then see how well the model performs at some task when fit on that data.

For this HW, you are provided with a function (in the subdirectory `/R`) that simulates the polling data along with code (in this Rmarkdown file) for simulating the population fractions, using the fractions to aggregate the simulated polling data to the national level, and visualizing the state and derived national results together.

Your tasks are to

1. work through the current code and understand what it does
2. fit a RW model (as explained below) to the simulated polling data
3. forecast the outcome of the election in 5 weeks using that model

Step 0

A. Load libraries

B. Save functions in R sub-folder and source all.

Notes:

- in HW2, we will learn how to make the code base of a project easily loadable by setting it up as an R package; but for now we'll use the following "hack" to quickly get functions in the `/R` directory loaded.
- If you edit `simulate_data.R` or create other script files, you also might consider checking the box "Source on Save"

```
# this sources any .R file in the subfolder R
Rfiles <- list.files(file.path(paste0(getwd(),"/R/")), ".R")
Rfiles <- Rfiles[grepl(".R", Rfiles)]
sapply(paste0(paste0(getwd(),"/R/"), Rfiles), source)
```

```
##         /Users/aaron_new/advR2021/hw1/R/simulate_data.R
```

```
## value   ?
## visible FALSE
```

Step 1: simulate the data

Our approach to simulating the proportion $p(s,t)$ supporting candidate A in state s on week t:

- fix (randomly chosen) proportions $p(s,0)$ at $t = 0$
- in each state, for logit-transformed proportions, "randomly walk" with drift d and standard deviation sd_{rw} from each week to the next.

The random walk model is defined as follows:

$$\text{logit}(p(s,t)) = \text{logit}(p(s,t-1)) + \varepsilon(s,t)$$

where $\varepsilon(s,t) \sim N(d, sd_{rw})$. Or in pseudo R-code:

logit(p(s, t)) = logit(p(s, t-1)) + rnorm(1, mean = d, sd = sd_rw)

```
nstates <- 52

# props at t = 0:
set.seed(123)
rw0 <- runif(nstates, 45, 60)

# RW-based simulations
rw <- simulate_rw(rw0 = rw0, sd_rw = 0.05,
                  n_steps = 10, # weeks here
                  drift = 0.01,
                  seed = 1234)
rw
```

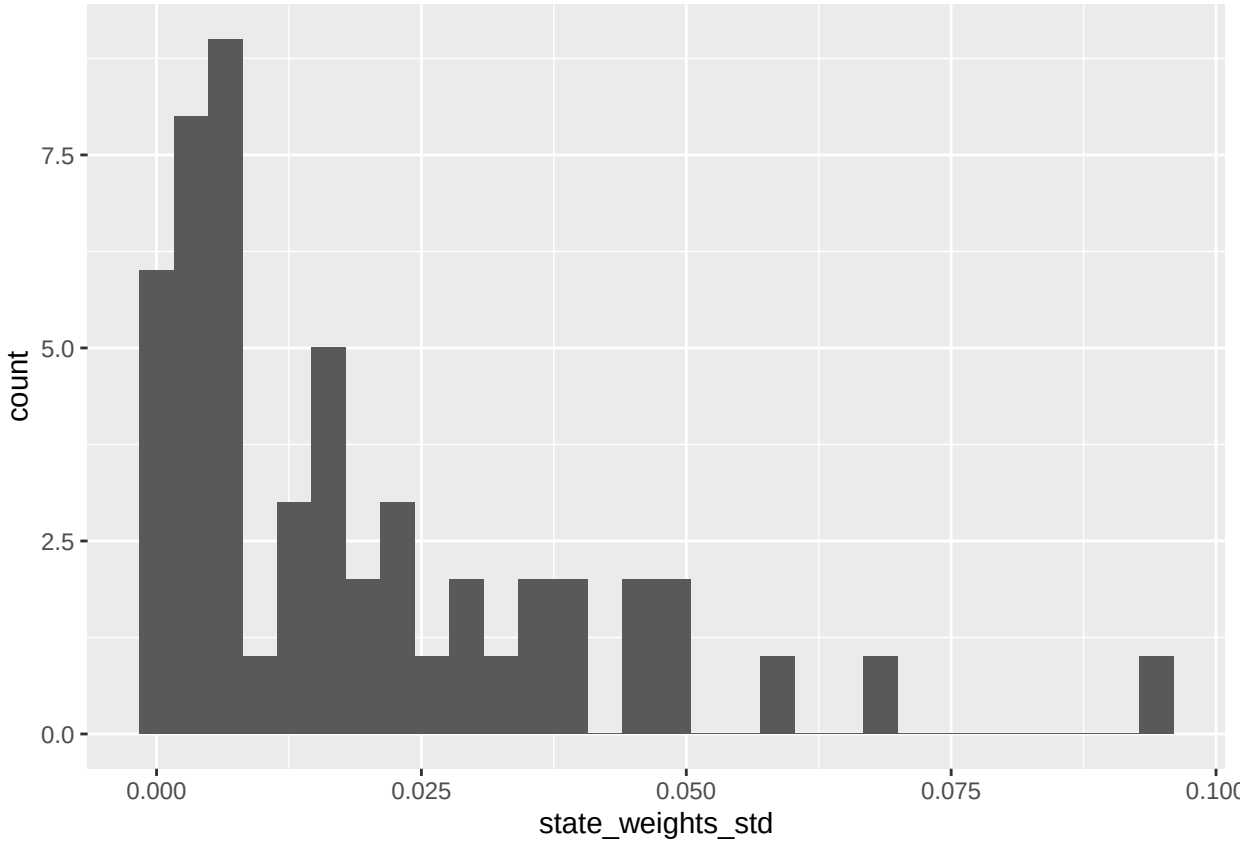
```
## # A tibble: 52 x 12
## # Groups:   state [52]
##    state   `0`   `1`   `2`   `3`   `4`   `5`   `6`   `7`   `8`   `9`  `10`
##    <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1  49.3  48.1  48.7  50.3  47.6  48.4  49.2  48.8  48.3  47.9  47.0
##  2     2  56.8  56.5  55.5  54.8  55.1  56.5  56.7  56.3  55.4  54.6  57.8
##  3     3  51.1  51.6  51.2  50.9  51.7  51.1  49.5  50.5  49.5  49.7  48.8
##  4     4  58.2  59.8  59.5  58.9  58.5  56.8  55.6  53.1  51.7  51.6  51.2
##  5     5  59.1  61.1  60.0  59.3  59.2  58.2  57.3  56.1  54.8  54.4  54.1
##  6     6  45.7  43.7  43.2  42.1  41.1  41.2  42.1  44.4  43.7  45.9  44.7
##  7     7  52.9  54.0  57.4  57.6  57.0  57.2  59.6  58.5  60.4  62.2  62.8
##  8     8  58.4  58.6  58.3  58.1  59.2  61.9  61.9  60.5  59.9  60.4  60.3
##  9     9  53.3  53.3  53.3  51.9  51.9  53.2  54.3  55.3  55.0  55.0  53.8
## 10    10  51.8  52.0  52.6  55.0  56.5  56.1  56.8  55.6  56.9  58.4  61.2
## # ... with 42 more rows
```

Approach to obtain state weights... here we sample from a Gamma distribution to get some positive outcomes, and then standardize

```
set.seed(123456)
state_weights <- rgamma(nstates, 1, 1)
state_weights_dat <- tibble(
  state = seq_len(nstates),
  state_weights_std = state_weights/sum(state_weights))
```

```
ggplot(state_weights_dat) +
  geom_histogram(aes(x = state_weights_std))
```

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



Step 2: calculate the national aggregate percentage.

The national aggregated percentage is the weighted average of the state-specific outcomes, with weights given by the standardized state weights.

Easy approach here: just go for a data set in the long format, then add average for each year.

Note: we could consider re-organizing this information to avoid repeated information but leaving things in long form usually the easiest way to hand data off to `ggplot`.

```
rw_long <-
  rw %>%
    left_join(state_weights_dat) %>%
    pivot_longer(-c(state, state_weights_std),
                 names_to = "t",
                 values_to = "percent") %>%
    mutate(t = as.numeric(t)) %>%
    group_by(t) %>%
    mutate(agg = sum(percent*state_weights_std))
```

```
## Joining, by = "state"
```

```
rw_long
```

```
## # A tibble: 572 x 5
## # Groups:   t [11]
##    state state_weights_std     t percent   agg
##    <int>             <dbl> <dbl>   <dbl> <dbl>
##  1     1            0.0225     0    49.3  52.7
##  2     1            0.0225     1    48.1  52.6
##  3     1            0.0225     2    48.7  52.6
##  4     1            0.0225     3    50.3  52.9
##  5     1            0.0225     4    47.6  53.3
##  6     1            0.0225     5    48.4  53.7
##  7     1            0.0225     6    49.2  54.4
##  8     1            0.0225     7    48.8  54.4
##  9     1            0.0225     8    48.3  54.6
## 10     1            0.0225     9    47.9  55.0
## # ... with 562 more rows
```

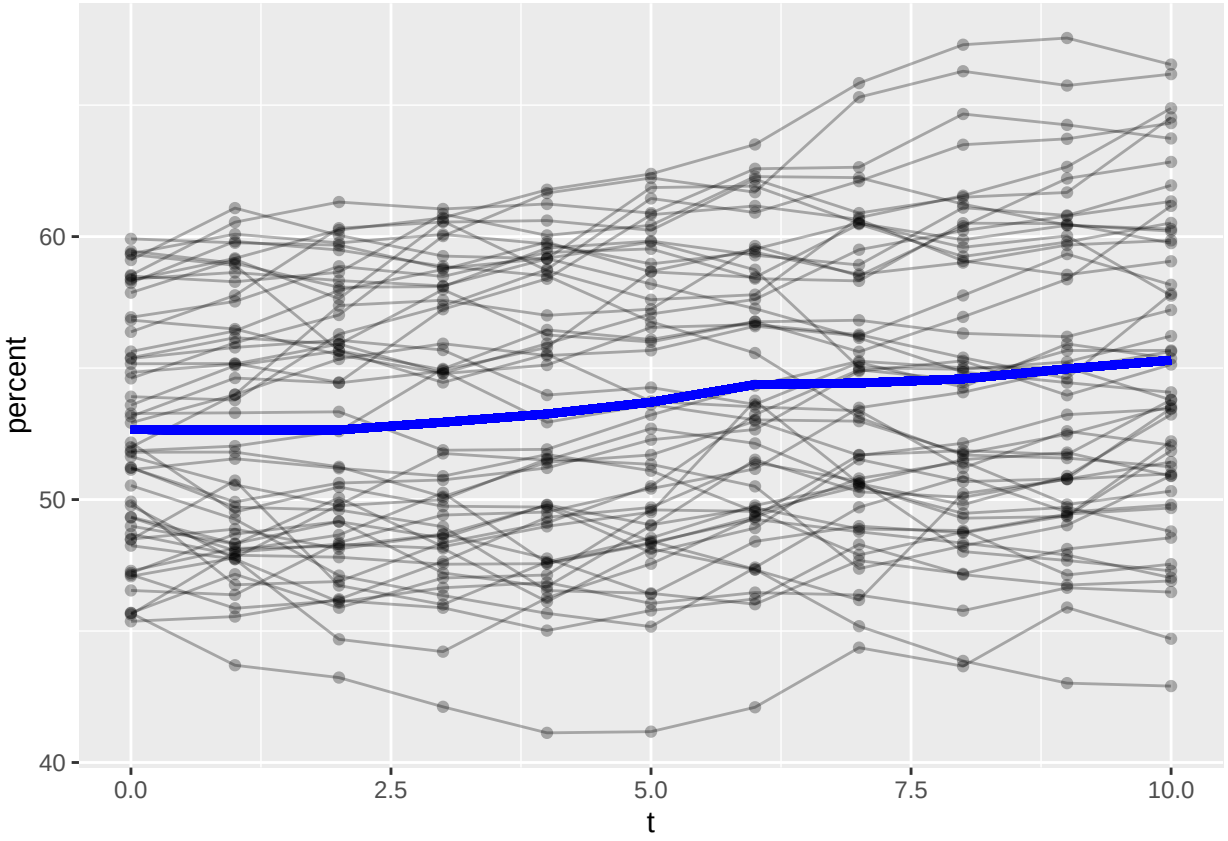
```
rw_long %>% arrange(t)
```

```
## # A tibble: 572 x 5
## # Groups:   t [11]
##    state state_weights_std     t percent   agg
##    <int>             <dbl> <dbl>   <dbl> <dbl>
##  1     1           0.0225      0    49.3  52.7
##  2     2           0.00578     0    56.8  52.7
##  3     3           0.00143     0    51.1  52.7
##  4     4           0.0600      0    58.2  52.7
##  5     5           0.0226      0    59.1  52.7
##  6     6           0.0332      0    45.7  52.7
##  7     7           0.0684      0    52.9  52.7
##  8     8           0.0297      0    58.4  52.7
##  9     9           0.00435     0    53.3  52.7
## 10    10           0.0481      0    51.8  52.7
## # ... with 562 more rows
```

Step 3: data visualization

Let's look at what we got so far

```
rw_long %>%
  ggplot(aes(x = t, y = percent,
        group = state)) +
    geom_point(alpha = 0.3) +
    geom_line(alpha = 0.3) +
    geom_line(aes(y = agg), color = "blue", size = 1.5)
```



Step 4: fit the RW model to your "data"

Now let's "forget" what RW parameters (drift and sd_rw) were used. Can you estimate them based on the available data?

The answer is yes!

Approach:

1. calculate the differences $e(s,t) = \text{logit}(p(s,t)) - \text{logit}(p(s,t-1))$.
2. your estimate for d is given by the mean of all $e(s,t)$
3. your estimate for sd_rw is given by the standard deviation of all $e(s,t)$

You should write a function returns the drift as well as the sd_rw estimate in a list, using, e.g., return(list(sd_rw = bla, drift = bla2))

Step 5: On to making forecasts...

With the estimated drift and sd for the random walk, we can forecast arbitrarily many future 5 week trajectories for each state, using the random walk equation

```
logit(p(s, t)) = logit(p(s, t-1)) + rnorm(1, mean = d, sd = sd_rw)
```

in order to generate each of the 5 weeks forecasted results from those of the preceeding week.

We can then aggregate those to the national level to get 5 weeks of a national forecast with uncertainty...

Your task:

- Write a function to make state forecasts.

- Forecast 5 weeks out, 100 trajectories per state.
- Aggregate to get 100 national level trajectories
- Visualize the mean and 5th and 95th percentiles for the national forecast.

We will work on the steps invloved in accomplishing this in class.