Prepping Data

I am assuming some familiarity with the `tidymodels` framework here—focusing on `workflowsets`. I would guide folks to Julia Silge's incredible tutorials to learn more about the `tidymodels` ecosystem.

After downloading 500 pages' worth of reviews from each film, I read them in below. I found that I had reviews for all three films from April 9th 2012 and on, so I filter just to these reviews, and I combine them for one dataset. Since we need the date to be a numeric predictor, I center at the first day available: So, April 9th 2012 is coded 0, the next day is coded 1, and so on. I also split these into training and testing data, using the default 75% training/25% testing option in `rsample::initial_split()`.

```
# prep ------------------------------------------------------------
-------------
library(tidyverse)
library(tidymodels)
library(workflowsets)
set.seed(1839)

dat <- list.files(pattern = "^ep") %>%
  map_dfr(~read_csv(.x) %>% mutate(film = str_sub(.x, 1, 3))) %>%
  transmute(film, date = lubridate::mdy(date), stars) %>%
  na.omit() %>% # recent days don't parse and are NA
  filter(date >= "2012-04-09") %>%
  mutate(date = as.numeric(date - lubridate::ymd("2012-04-09"))) %>%
  initial_split()

dat_train <- training(dat)
dat_test <- testing(dat)
```

And now, we are set to define the models and data processing steps. This is a very simple case, as I have one outcome (the score, from 0.5 to 5.0 stars), and two predictors (date and film).

Models and Preprocessing

For the splines, I define typical OLS regression as the estimator. For the random forest, I am using the `ranger` package, and I will tune the number of variables it'll use (a little silly, because here we only have two candidates, but it's what I would do in a larger dataset, so I'm just being consistent with the practice here) and the minimum allowed data points in a terminal node. Lastly, we have a k-nearest neighbors model using the `kknn` package, tuning for neighbors, distance power (Manhattan being 1, Euclidean being 2), and type of kernel function for weighting distances.

```
# set models ------------------------------------------------------------
-------
lm_mod <- linear_reg() %>%
  set_engine("lm")

rf_mod <- rand_forest(
  mode = "regression",
  trees = 1000,
  mtry = tune(),
  min_n = tune()
) %>%
  set_engine("ranger")
```

```
nn_mod <- nearest_neighbor(
  mode = "regression",
  neighbors = tune(),
  dist_power = tune(),
  weight_func = tune()
) %>%
  set_engine("kknn")
```

Then, I turn to pre-processing steps—what are called “recipes” in the `tidymodels` framework. I start with a base recipe, which is simply predicting stars from date and film, and then dummy scoring the categorical film variable. Then, I make a recipe for the regression model with a natural spline on date. I allow the degrees of freedom to be tuned in crossvalidation, and then I make a step allowing for date and film to interact with one another. Lastly, I make a z-scored recipe.

```
# make recipes ------------------------------------------------------------
-----
rec_base <- recipe(stars ~ date + film, dat_train) %>%
  step_dummy(film)

rec_ns <- rec_base %>%
  step_ns(date, deg_free = tune()) %>%
  step_interact(~ starts_with("date"):starts_with("film"))

rec_zs <- rec_base %>%
  step_normalize(all_predictors())
```

And here’s where the cool part comes in: I can now define a “workflow set.” I specify the recipes with `preproc` and the models with `models`. One can use `cross = TRUE` to look at all combinations of recipes and models. Here, I am setting that to `FALSE` so that the natural spline recipe is used for the linear regression, the base recipe is used for the random forest, and the standardized recipe is used for the nearest neighbors.

```
# workflowset -------------------------------------------------------------
------
(wfs <- workflow_set(
  preproc = list(spline = rec_ns, base = rec_base, zscored = rec_zs),
  models = list(lm_mod, rf_mod, nn_mod),
  cross = FALSE
))
## # A workflow set/tibble: 3 x 4
##   wflow_id                 info                 option    result
##
## 1 spline_linear_reg
## 2 base_rand_forest
## 3 zscored_nearest_neighbor
```

And we see we have our three different workflows set up in one unified place. In practice, one could define a large number of workflows and recipe/model combinations to test against one another.

Crossvalidation

I then turn to tuning the aforementioned hyperparameters using 10-fold crossvalidation. After I make the folds, I simply pipe my workflow set to a specific `purrr::map()`-esque function for workflows. I tell it to run the `tune::tune_grid()` function on each, defining a random grid of 30 different combinations of parameters for each of the three workflows. I’m doing grid search here, and the `tidymodels` folks have set reasonable ranges to explore for the hyperparameter space, which is why the code here looks so bare in defining what those grids cover.

```
# cross validation ------------------------------
-------------------------------
folds <- vfold_cv(dat_train, v = 10)

cv_res <- wfs %>%
  workflow_map("tune_grid", seed = 1839, grid = 30, resamples = folds)
```

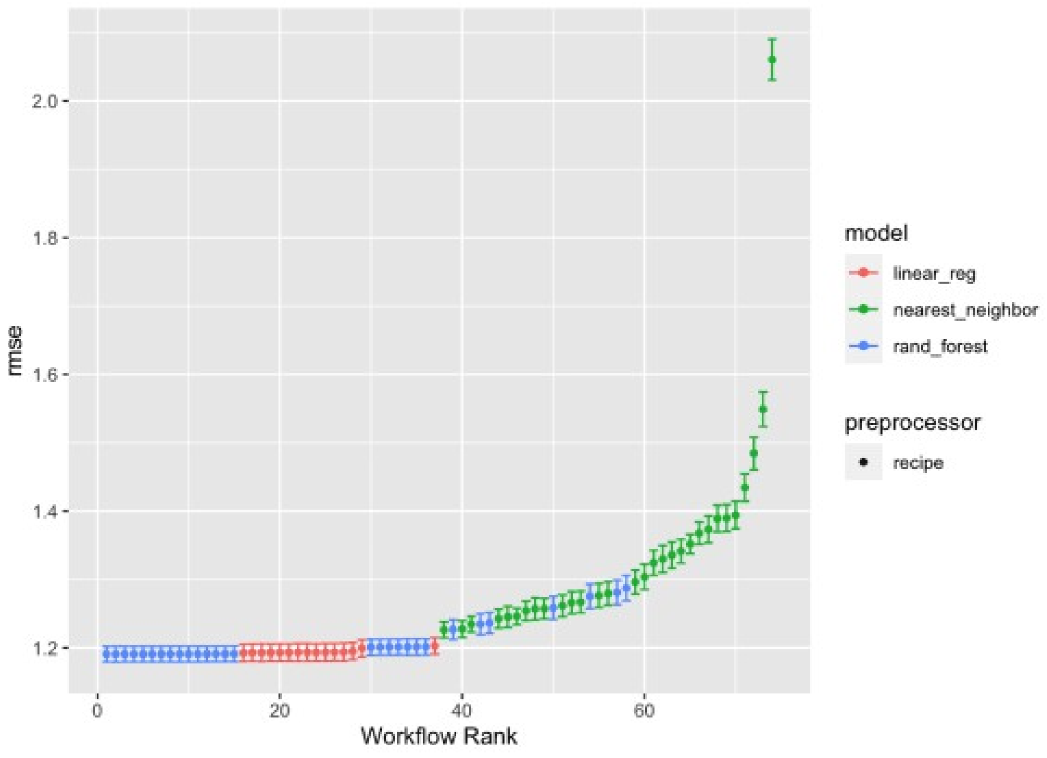
Compare Models

We can then look at the results. What are the top 10 workflows?

```
# compare models and params ------------------------------
----------------------
cv_res %>%
  rank_results() %>%
  filter(.metric == "rmse")
## # A tibble: 74 x 9
##    wflow_id    .config     .metric  mean std_err     n preprocessor model
rank
##
##  1 base_rand_… Preprocess… rmse     1.19 0.00689    10 recipe       rand_…
1
##  2 base_rand_… Preprocess… rmse     1.19 0.00697    10 recipe       rand_…
2
##  3 base_rand_… Preprocess… rmse     1.19 0.00691    10 recipe       rand_…
3
##  4 base_rand_… Preprocess… rmse     1.19 0.00686    10 recipe       rand_…
4
##  5 base_rand_… Preprocess… rmse     1.19 0.00689    10 recipe       rand_…
5
##  6 base_rand_… Preprocess… rmse     1.19 0.00687    10 recipe       rand_…
6
##  7 base_rand_… Preprocess… rmse     1.19 0.00694    10 recipe       rand_…
7
##  8 base_rand_… Preprocess… rmse     1.19 0.00692    10 recipe       rand_…
8
##  9 base_rand_… Preprocess… rmse     1.19 0.00696    10 recipe       rand_…
9
## 10 base_rand_… Preprocess… rmse     1.19 0.00694    10 recipe       rand_…
10
## # … with 64 more rows
```

All of them are random forests, and all of them have the same average root-mean-square error. I can also easily call a plot to compare all possible combinations of models explored in the grid search:

```
autoplot(cv_res, rank_metric = "rmse", metric = "rmse")
```



The random forests tended to do the best, with the linear regression with splines and an interaction defined doing about as well. It was k-nearest neighbors that really lagged behind.

Select Best Model

To actually examine my hypothesis, I'll pull the best hyperparameter values for the random forest, and then I will finalize my workflow, fit it on the training data, and then predict it on the 25% testing holdout data.

```
# predict -----------------------------------------------------------
----------
(best_results <- cv_res %>%
  pull_workflow_set_result("base_rand_forest") %>%
  select_best(metric = "rmse"))
## # A tibble: 1 x 3
##    mtry min_n .config
##
## 1     2     5 Preprocessor1_Model12
final_fit <- cv_res %>%
  pull_workflow("base_rand_forest") %>%
  finalize_workflow(best_results) %>%
  fit(dat_train)

dat_test <- bind_cols(dat_test, predict(final_fit, dat_test))
```

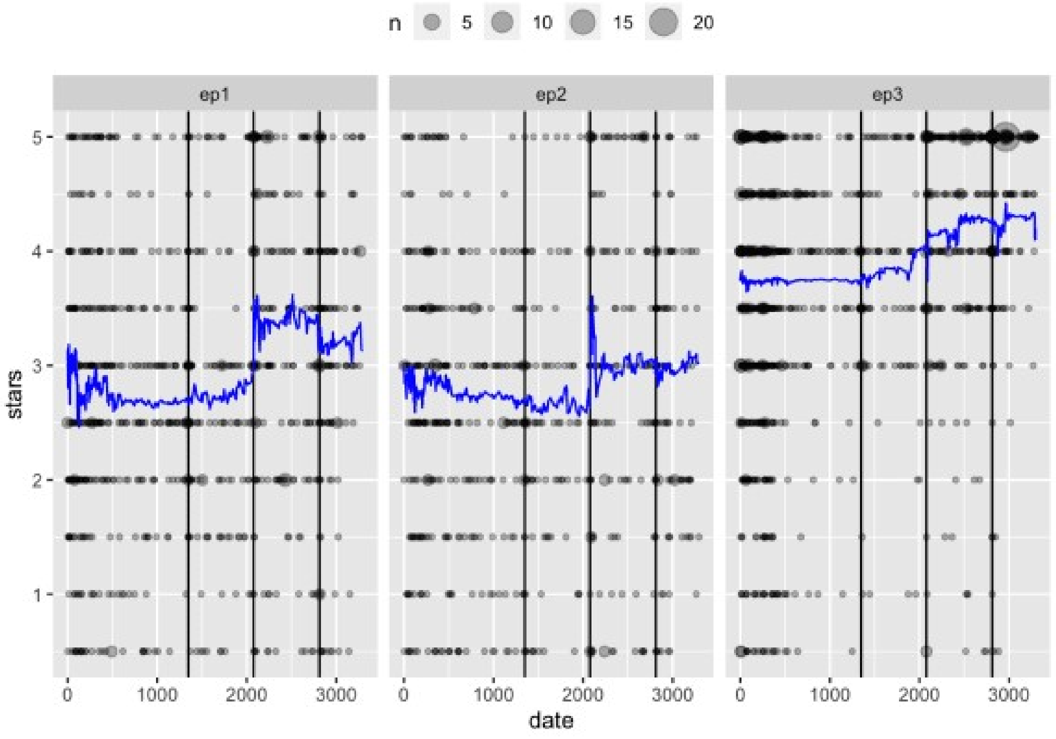
Results

So, what does it look like? The three vertical lines below are for the three sequel movies: *The Force Awakens*, *The Last Jedi*, and *The Rise of Skywalker*, respectively. Remember that I've scored date so that 0 is April 9th 2012 (the first review I have for all three films).

```
# plotting ------------------------------------------------------------
```

```
---------
tfa <- as.numeric(lubridate::ymd("2015-12-18") - lubridate::ymd("2012-04-09"))
tlj <- as.numeric(lubridate::ymd("2017-12-15") - lubridate::ymd("2012-04-09"))
tros <- as.numeric(lubridate::ymd("2019-12-20") -
lubridate::ymd("2012-04-09"))

ggplot(dat_test, aes(x = date, y = stars)) +
  facet_wrap(~ film) +
  geom_vline(aes(xintercept = tfa)) +
  geom_vline(aes(xintercept = tlj)) +
  geom_vline(aes(xintercept = tros)) +
  geom_count(alpha = .3) +
  geom_line(aes(x = date, y = .pred), color = "blue") +
  theme(legend.position = "top")
```



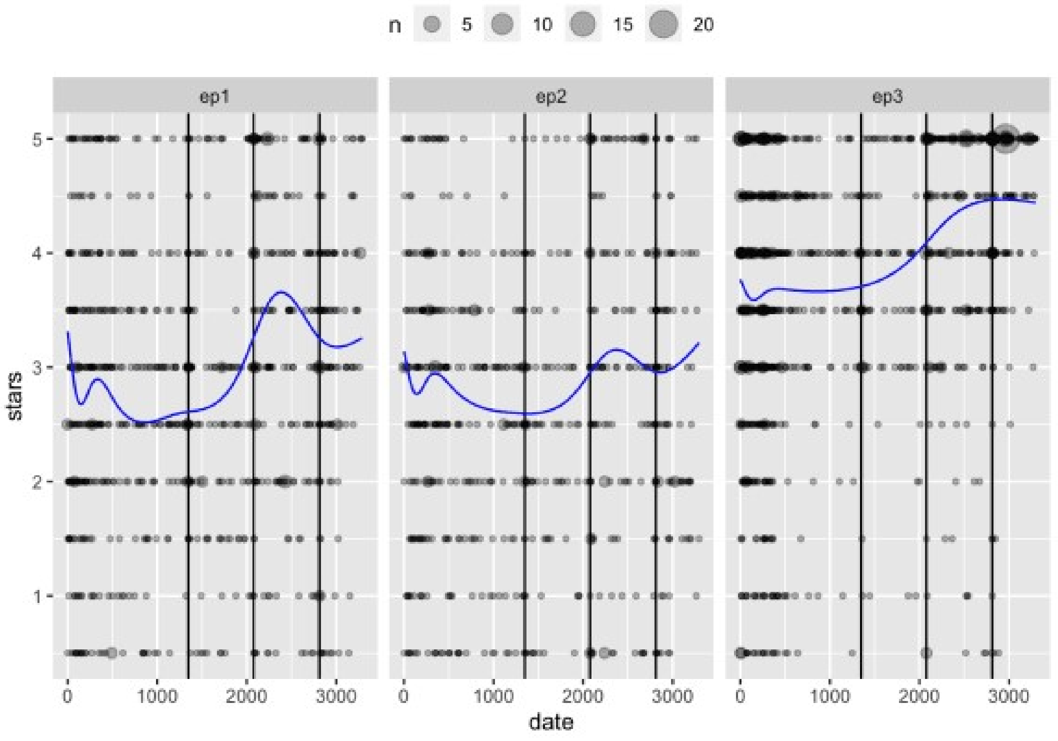
For *Episode I: The Phantom Menace* and *Episode II: Attack of the Clones*, we see a huge spike up at the release of Rian Johnson's expectation-subverting masterpiece, *Episode VIII: The Last Jedi*. We see this a bit for *Episode III: Revenge of the Sith*, as well, but less so—this movie was already probably the best-regarded film from the prequel era.

Some might be wondering: The spline on date interacting with the film dummy variables had basically the same performance, what does that model look like?

```
# spline ------------------------------------------------------------
-----------
(best_results <- cv_res %>%
  pull_workflow_set_result("spline_linear_reg") %>%
  select_best(metric = "rmse"))
## # A tibble: 1 x 2
##   deg_free .config
##
## 1        8 Preprocessor12_Model1
final_fit <- cv_res %>%
  pull_workflow("spline_linear_reg") %>%
  finalize_workflow(best_results) %>%
  fit(dat_train)

dat_test$pred_spline <- predict(final_fit, dat_test)$.pred

ggplot(dat_test, aes(x = date, y = stars)) +
  facet_wrap(~ film) +
  geom_vline(aes(xintercept = tfa)) +
  geom_vline(aes(xintercept = tlj)) +
  geom_vline(aes(xintercept = tros)) +
  geom_count(alpha = .3) +
  geom_line(aes(x = date, y = pred_spline), color = "blue") +
  theme(legend.position = "top")
```



We see a similar trend, albeit smoother. I would probably use this model if I had to do prediction, given the noisiness of the random forest prediction (I would also round predictions to the nearest half star, which would help stabilize things). But that isn't really what I was trying to do here. I was looking for some obvious evidence that people started judging the prequels better once the sequels were released.

And the data here pass the best statistical test of all: the interocular trauma test. The data are obvious in that positive reviews spiked after the release of *The Last Jedi.* To my knowledge, Joe Berkson coined the name of this test, and it first shows up in 1963 in Edwards, Lindman, and Savage's "Bayesian Statistical Inference for Psychological Research" in *Psychological Review*, 70(3):

"The preceding paragraph illustrates a procedure that statisticians of all schools find important but elusive. It has been called the interocular traumatic test; you know what the data mean when the conclusion hits you between the eyes."

The underlying mechanism is less clear. My first thought was nostalgia is evoked while people contrast the present to the past. But so-called "anti-fandom" could also be evoked when doing that contrast. Or it could be run-of-the-mill fanboy anger at contrasting the movies. I do think contrast plays a big role here, though, as we see the biggest effect after *The Last Jedi*, which brilliantly defied a lot of expectations about where the saga could go. Let's hope we finally do see Rian Johnson's own Star Wars trilogy someday.

Appendix

```
library(rvest)
library(RSelenium)
library(tidyverse)

# global params ------------------------------------------------------------
----
# url <- "https://www.rottentomatoes.com/m/star_wars_episode_i_the_
phantom_menace/reviews?type=user"
# url <- "https://www.rottentomatoes.com/m/star_wars_episode_ii_attack_of_the_clones/reviews?
type=user"
# url <- "https://www.rottentomatoes.com/m/star_wars_episode_iii_revenge_of_the_sith/reviews?
type=user"
pages_to_scrape <- 500

# funs ------------------------------------------------------------
-------------
get_stars <- function(the_html) {
  stars_text <- the_html %>%
    html_nodes(".audience-reviews__score") %>%
    as.character()

  stars <- map_dbl(stars_text, function(x) {
    score <- 0
    score <- score + str_count(x, "star-display__filled")
    score <- score + (str_count(x, "star-display__half") / 2)
  })

  return(stars)
}

get_dates <- function(the_html) {
  date_text <- the_html %>%
    html_nodes(".audience-reviews__duration") %>%
    as.character()

  dates <- map_chr(date_text, function(x) {
    str_split(x, ">") %>%
      `[[`(1) %>%
      `[`(2) %>%
      str_remove("<- function(the_html) {
  text_text <- the_html %>%
    html_nodes(".js-clamp") %>%
    as.character() %>%
    `[`(seq(1, length(.), by = 2))

  texts <- map_chr(text_text, function(x) {
    str_split(x, ">") %>%
      `[[`(1) %>%
      `[`(2) %>%
      str_remove("[<].*")
  })

  return(texts)
}
```

```
get_reviews <- function(the_html) {

  reviews <- tibble(
    date = get_dates(the_html),
    stars = get_stars(the_html),
    text = get_texts(the_html)
  )

  return(reviews)
}

# scrape -------------------------------------------------------------
-----------
res <- tibble()
driver <- rsDriver(browser = "firefox", port = 1839L)
driver$client$navigate(url)

for (i in seq_len(pages_to_scrape)) {
  cat("page", i, "\n")

  res <- res %>%
    bind_rows(get_reviews(read_html(driver$client$getPageSource()[[1]])))

  more <- driver$client$findElement(
    "css selector",
    ".prev-next-paging__button-right"
  )

  more$clickElement()
  Sys.sleep(2)
}

driver$client$closeall()

# write out ----------------------------------------------------------
--------
# res %>%
#   unique() %>%
#   write_csv("ep1.csv")

# res %>%
#   unique() %>%
#   write_csv("ep2.csv")

# res %>%
#   unique() %>%
#   write_csv("ep3.csv")
```