W271 Assignment 7 (Solutions)

```
library(tidyverse)
library(magrittr)
library(patchwork)

library(lubridate)

library(tsibble)
library(feasts)
library(forecast)

library(sandwich)
library(lmtest)

library(nycflights13)
library(blsR)
```

```
theme_set(theme_minimal())
```

Question-1: AIC and BIC and "Stringency"

(4 points) Part-1

In the async lecture, Jeffrey says "BIC is in general more stringent than AIC or AICc". Let's illustrate that and reason about it.

1. Produce a dataset, `d`, that includes 100 observations of pure white-noise.
    - The outcome variable should be a variable `y` that has 100 draws from `rnorm`, with `mean=0` and `sd=1`.
    - The input variables should be variables `x1` ... `x10` that are also 100 draws from `rnorm` each with `mean=0` and `sd=1`.
    - There are fancy ways to write this code; the goal for this isn't to place a clever coding task in front of you, so feel free to use copy-paste to create the data object in any way that you can.
2. After producing data, fit 11 models against that data, stored as `model0` through `model10`. (The number appended to `model` corresponds to the number of parameters that you have used in your estimation).
3. After estimating your models, create a new dataset, `results_data`, that contains the number of parameters that you have used in an estimation, and the AIC and BIC values that you calculated for that number of parameters.
    1. Note – this is another place where the way that you create the data, and the way that the data is the most useful to use are incongruent.
    2. When we created the data, we created a dataset that has a column called `parameters`, a column called `aic` and a column called `bic`.
    3. However, it is much more useful to have "tidy" data that has these values stacked. If you find yourself creating the dataset in the "wide" form that we have described above, you can use the `dplyr::pivot_longer` function to pivot your data into a tidy format. Specifically, we used this call `pivot_longer(cols = c('aic', 'bic'))` with our input data structure.
4. Finally, produce a plot that shows the AIC and BIC values on the y-axis and the number of estimated parameters on the x-axis. In the subtitle to your plot, note whether a relatively higher or lower AIC

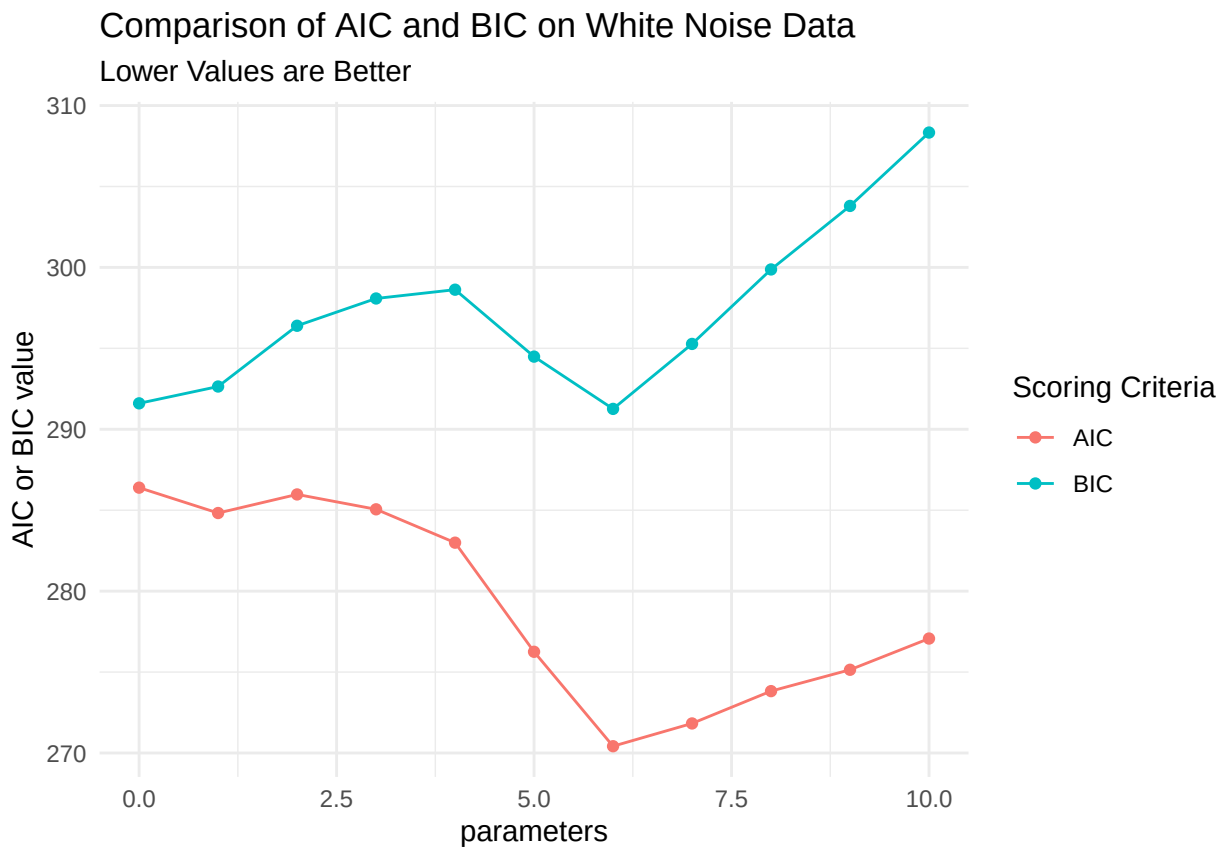
or BIC means that a model is performing better or worse (i.e. either "Higher values are better" or "Lower values are better"). What do you notice about these plots, and what does this tell you about the "stringency" of AIC vs. BIC?

```
d <- data.frame(
  y   = rnorm(n=100, mean=0, sd=1),
  x1  = rnorm(n=100, mean=0, sd=1),
  x2  = rnorm(n=100, mean=0, sd=1),
  x3  = rnorm(n=100, mean=0, sd=1),
  x4  = rnorm(n=100, mean=0, sd=1),
  x5  = rnorm(n=100, mean=0, sd=1),
  x6  = rnorm(n=100, mean=0, sd=1),
  x7  = rnorm(n=100, mean=0, sd=1),
  x8  = rnorm(n=100, mean=0, sd=1),
  x9  = rnorm(n=100, mean=0, sd=1),
  x10 = rnorm(n=100, mean=0, sd=1)
)
```

```
model_0  <- lm(y ~ 1, data = d)
model_1  <- lm(y ~ 1 + x1, data = d)
model_2  <- lm(y ~ 1 + x1 + x2, data = d)
model_3  <- lm(y ~ 1 + x1 + x2 + x3, data = d)
model_4  <- lm(y ~ 1 + x1 + x2 + x3 + x4, data = d)
model_5  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5, data = d)
model_6  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6, data = d)
model_7  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7, data = d)
model_8  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8, data = d)
model_9  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9, data = d)
model_10 <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10, data = d)
```

```
results_data <- data.frame(
  parameters = 0:10,
  AIC = c(AIC(model_0), AIC(model_1), AIC(model_2), AIC(model_3), AIC(model_4),
          AIC(model_5), AIC(model_6), AIC(model_7), AIC(model_8), AIC(model_9), AIC(model_10)),
  BIC = c(BIC(model_0), BIC(model_1), BIC(model_2), BIC(model_3), BIC(model_4),
          BIC(model_5), BIC(model_6), BIC(model_7), BIC(model_8), BIC(model_9), BIC(model_10))
)
```

```
results_data %>%
  pivot_longer(cols = c('AIC', 'BIC')) %>%
  ggplot() +
  aes(x=parameters, y=value, color=name) +
  geom_point() +
  geom_line() +
  labs(
    title = 'Comparison of AIC and BIC on White Noise Data',
    subtitle = 'Lower Values are Better',
    y = 'AIC or BIC value',
    color = 'Scoring Criteria'
  )
```



The first thing for us to note in this comparison is that there is no reason *within* a single set of parameters to compare an AIC score to a BIC score. That just isn't a comparison that is meaningful or interesting; it would quite literally be like comparing apples to organges. The second, and more imporant piece to note is that the rate of change of BIC is greater than the rate of change of AIC with respect to the addition of parameters. What does this mean? It means that as we're adding pure noise into the model, the BIC score is increasing faster (and so giving us a stronger feedback that the additional parameters are unuseful) compared to the AIC score.

(2 points) Part-2

Now, suppose that you had data that, *in the population model* actually held a relationship between the input features and the outcome feature. Specifically, suppose that for every unit increase in `x1` there was a `0.1` increase in the outcome, for every unit increase in `x2` there was a `0.2` increase in the outcome, ..., for every unit increase in `x10` there was a `1.0` unit increase in the outcome. Suppose that if all `x1` ... `x10` were zero, that the outcome would have an expectation of zero, but with white-noise around it with $\mu = 0$ and $\sigma = 1$.

- Modify the code that you wrote above to create data according to this schedule.
- Estimate 11 models as before.
- Produce a new dataset `resutls_data` that contains the AIC and BIC values from each of these models.
- Produce the same plot as you did before with the white noise series. Comment on what, if anything is similar or different between this plot, and the plot you created before.

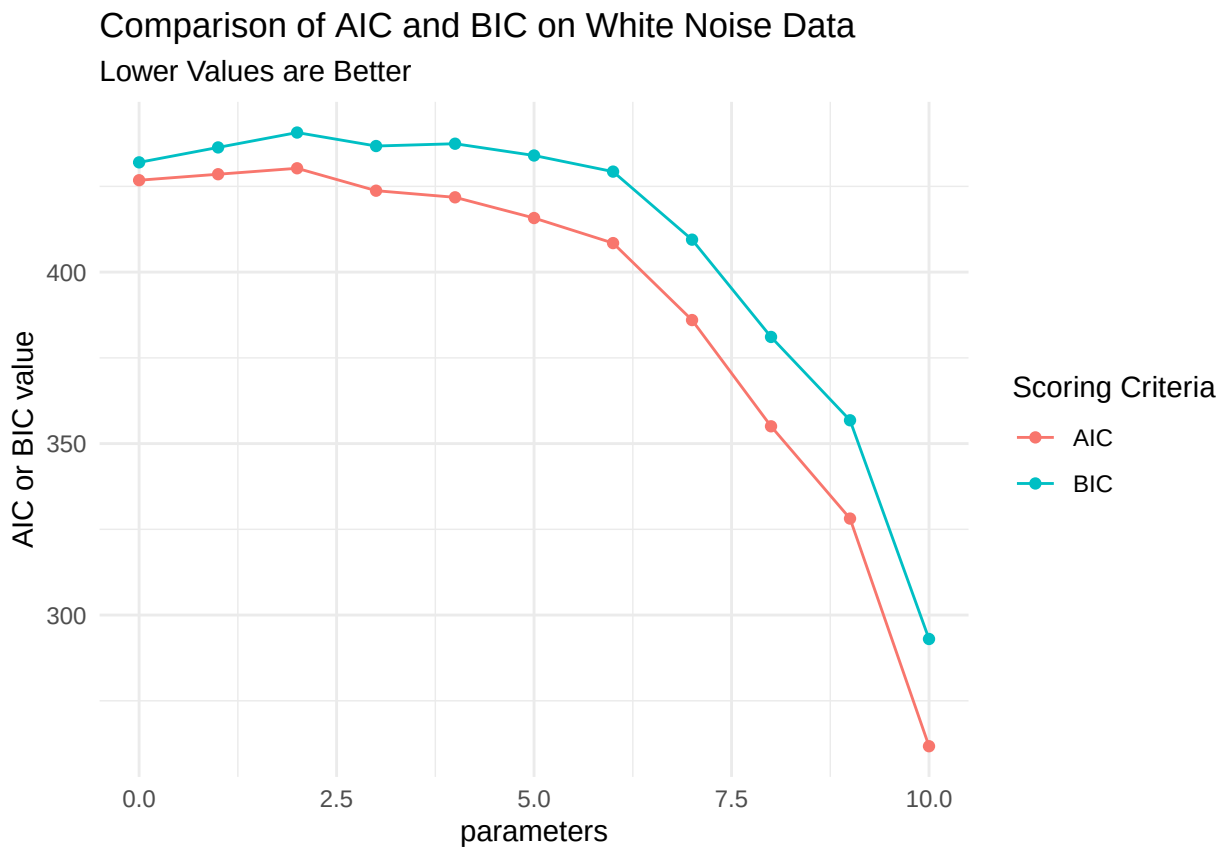
```
d <- data.frame(
  x1  = rnorm(n=100, mean=0, sd=1),
  x2  = rnorm(n=100, mean=0, sd=1),
  x3  = rnorm(n=100, mean=0, sd=1),
  x4  = rnorm(n=100, mean=0, sd=1),
```

```
  x5  = rnorm(n=100, mean=0, sd=1),
  x6  = rnorm(n=100, mean=0, sd=1),
  x7  = rnorm(n=100, mean=0, sd=1),
  x8  = rnorm(n=100, mean=0, sd=1),
  x9  = rnorm(n=100, mean=0, sd=1),
  x10 = rnorm(n=100, mean=0, sd=1)) %>%
  mutate(y = 0.1 * x1 + 0.2 * x2 + 0.3 * x3 + 0.4 * x4 + 0.5 * x5 +
             0.6 * x6 + 0.7 * x7 + 0.8 * x8 + 0.9 * x9 + 1.0 * x10 + rnorm(n=100, mean=0, sd=1)
         )
```

```
model_0  <- lm(y ~ 1, data = d)
model_1  <- lm(y ~ 1 + x1, data = d)
model_2  <- lm(y ~ 1 + x1 + x2, data = d)
model_3  <- lm(y ~ 1 + x1 + x2 + x3, data = d)
model_4  <- lm(y ~ 1 + x1 + x2 + x3 + x4, data = d)
model_5  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5, data = d)
model_6  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6, data = d)
model_7  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7, data = d)
model_8  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8, data = d)
model_9  <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9, data = d)
model_10 <- lm(y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10, data = d)
```

```
results_data <- data.frame(
  parameters = 0:10,
  AIC = c(AIC(model_0), AIC(model_1), AIC(model_2), AIC(model_3), AIC(model_4),
          AIC(model_5), AIC(model_6), AIC(model_7), AIC(model_8), AIC(model_9), AIC(model_10)),
  BIC = c(BIC(model_0), BIC(model_1), BIC(model_2), BIC(model_3), BIC(model_4),
          BIC(model_5), BIC(model_6), BIC(model_7), BIC(model_8), BIC(model_9), BIC(model_10))
)
```

```
results_data %>%
  pivot_longer(cols = c('AIC', 'BIC')) %>%
  ggplot() +
  aes(x=parameters, y=value, color=name) +
  geom_point() +
  geom_line() +
  labs(
    title = 'Comparison of AIC and BIC on White Noise Data',
    subtitle = 'Lower Values are Better',
    y = 'AIC or BIC value',
    color = 'Scoring Criteria'
  )
```



We notice a few things about this plot.

First and foremost, it does **not** make sense to compare the values from this plot directly to the values of the previous plot that we produced – these two datasets have very different outcomes and so we should expect that they have different values for the calculated AIC and BIC.

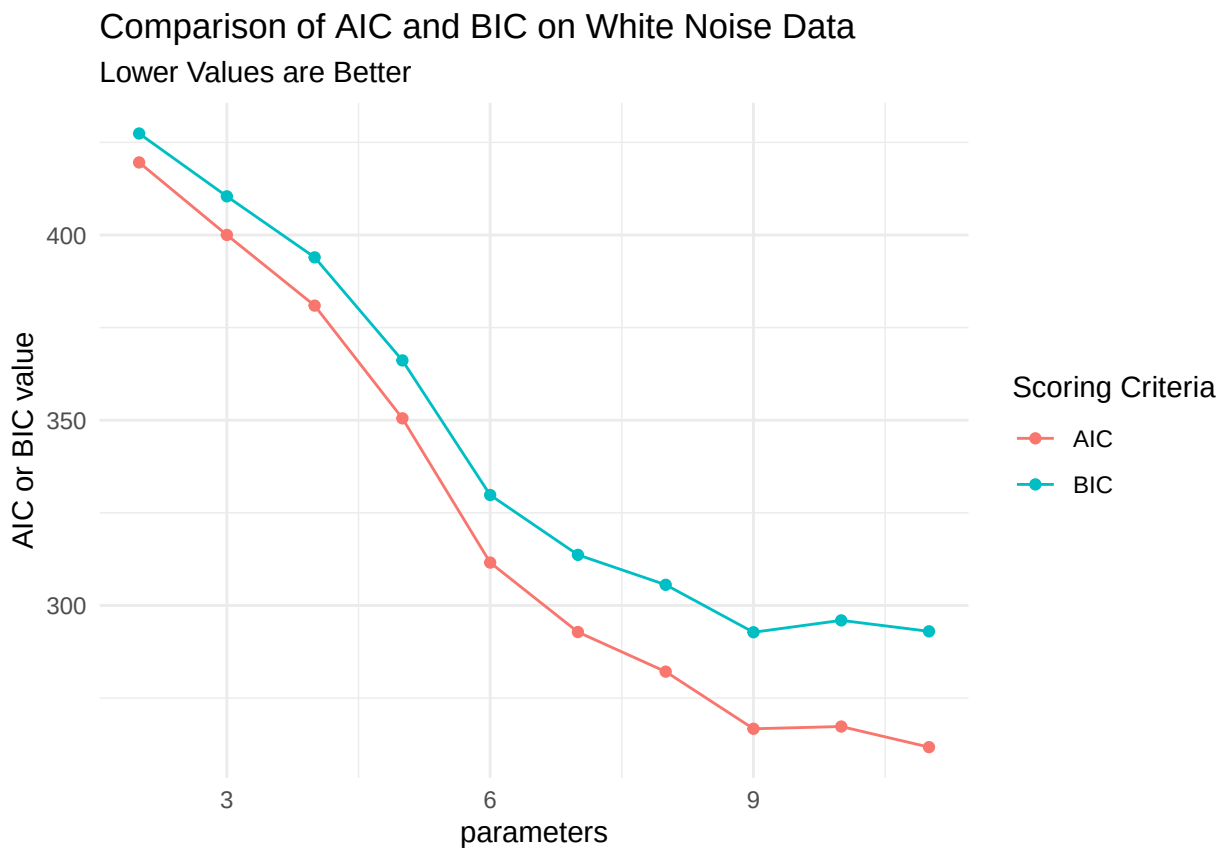
The second thing that we notice is that the plot seems to "fall off" on the higher parameters – specifically, maybe for parameters 5-10. With the models as we have estimated them, it is not easy to tell if this is because (a) we are explaining a lot of the data as we add more and more variables to the model; or (b) if the variables that have a larger impact – i.e. those whose β_i is larger, like 1.0, have a stronger effect. To try to split this, we propose fitting 11 more models, but adding the variables in the opposite order.

```
model_11  <- lm(y ~ 1 + x10, data = d)
model_12  <- lm(y ~ 1 + x10 + x9, data = d)
model_13  <- lm(y ~ 1 + x10 + x9 + x8, data = d)
model_14  <- lm(y ~ 1 + x10 + x9 + x8 + x7, data = d)
model_15  <- lm(y ~ 1 + x10 + x9 + x8 + x7 + x6, data = d)
model_16  <- lm(y ~ 1 + x10 + x9 + x8 + x7 + x6 + x5, data = d)
model_17  <- lm(y ~ 1 + x10 + x9 + x8 + x7 + x6 + x5 + x4, data = d)
model_18  <- lm(y ~ 1 + x10 + x9 + x8 + x7 + x6 + x5 + x4 + x3, data = d)
model_19  <- lm(y ~ 1 + x10 + x9 + x8 + x7 + x6 + x5 + x4 + x3 + x2, data = d)
model_20  <- lm(y ~ 1 + x10 + x9 + x8 + x7 + x6 + x5 + x4 + x3 + x2 + x1, data = d)
```

```
results_data_2 <- data.frame(
  parameters = 2:11,
  AIC = c(AIC(model_11), AIC(model_12), AIC(model_13), AIC(model_14), AIC(model_15),
          AIC(model_16), AIC(model_17), AIC(model_18), AIC(model_19), AIC(model_20)),
  BIC = c(BIC(model_11), BIC(model_12), BIC(model_13), BIC(model_14), BIC(model_15),
```

```
          BIC(model_16), BIC(model_17), BIC(model_18), BIC(model_19), BIC(model_20))
  )
```

```
results_data_2 %>%
  pivot_longer(cols = c('AIC', 'BIC')) %>%
  ggplot() +
  aes(x=parameters, y=value, color=name) +
  geom_point() +
  geom_line() +
  labs(
    title = 'Comparison of AIC and BIC on White Noise Data',
    subtitle = 'Lower Values are Better',
    y = 'AIC or BIC value',
    color = 'Scoring Criteria'
  )
```



With this second plot created, we can make a call – the AIC and BIC are moving in response to the magnitude of the effect of the variable on the outcome. This is really, really sensible, since both AIC and BIC are transformations of the SSE of the regression, and variables that have a relative larger impact on the outcomes have a relatively larger impact on the SSE of the regression. Neat!

Question-2: Weather in NYC

Our goals with this question are to:

- (If necessary) Clean up code that we've written before to re-use. This task of writing code, and then

coming back and using it later is often overlooked in the MIDS program. Here's a chance to practice!
- Estimate several different polynomial regressions against a time series and evaluate at what point we have produced a model with "enough complexity" that the model evaluation scores cease to tell us that additional model parameters are improving the model fit.

(1 point) Part-1: Load the Weather Data

Load the weather data in the same way as you did in the previous assignment, recalling that there was some weird duplication of data for one of the days. Then, create an object, `weather_weekly` that aggregates the data to have two variables `average_temperature` and `average_dewpoint` at the year-week level, for each airport. After your aggregation is complete, you should have a `tsibble` that has the following shape:

```
A tsibble: 159 x 4 [1W]
# Key:       origin [3]
   origin week_index average_temperature average_dewpoint
   <chr>      <week>               <dbl>            <dbl>
 1 EWR      2013 W01                34.3             19.4
 2 EWR      2013 W02                42.7             33.3
 3 EWR      2013 W03                39.6             26.5
```

```
weather_weekly <- weather %>%
  mutate(
    duplicated = case_when(
      month == 11 & day == 3 & hour == 1 ~ 'duped',
      TRUE                   ~ 'not-duped')) %>%
  filter(duplicated == 'not-duped') %>%
  mutate(time_index = make_datetime(year, month, day, hour)) %>%
  as_tsibble(
    key   = origin,
    index = time_index) %>%
  group_by(origin) %>%
  index_by(week_index = yearweek(time_index)) %>%
  summarise(
    average_temperature = mean(temp, na.rm = TRUE),
    average_dewpoint    = mean(dewp, na.rm = TRUE)
  )
```

(2 points) Part-2: Fit Polynomial Regression Models

For each of the `average_temperature` and `average_dewpoint` create ten models that include polynomials of increasing order.

- One issue that you're likely to come across is dealing with how to make the time index that you're using in your `tsibble` work with either `poly` or some other function to produce the polynomial terms; this arises because although the time index is ordered, it isn't really a "numeric" feature so when you call for something like, `poly(week_index, degree=2)` you will be met with an error.
- Cast the index to a numeric variable, where the first week is indexed to be `0`. Recall that Jeffrey notes that this form of translation only changes the way that the intercept is interpreted; we will note that because the `as.numeric(week_index)` creates input variables that are in the vicinity, it also changes the magnitude of the higher-order polynomial terms that are estimated, though it does not change the regression diagnostics and model scoring to transform (or not) these time index variables.

Additionally, you might recall that in 203, we actually recommended you away from using the `poly` function. That was a recommendation based on students' knowledge at the time, when we were considering fitting log and square root transformations of data. At this point, you can handle the additional complexity and can take the recommendation that `poly` is nice for working with polynomial translations of time.

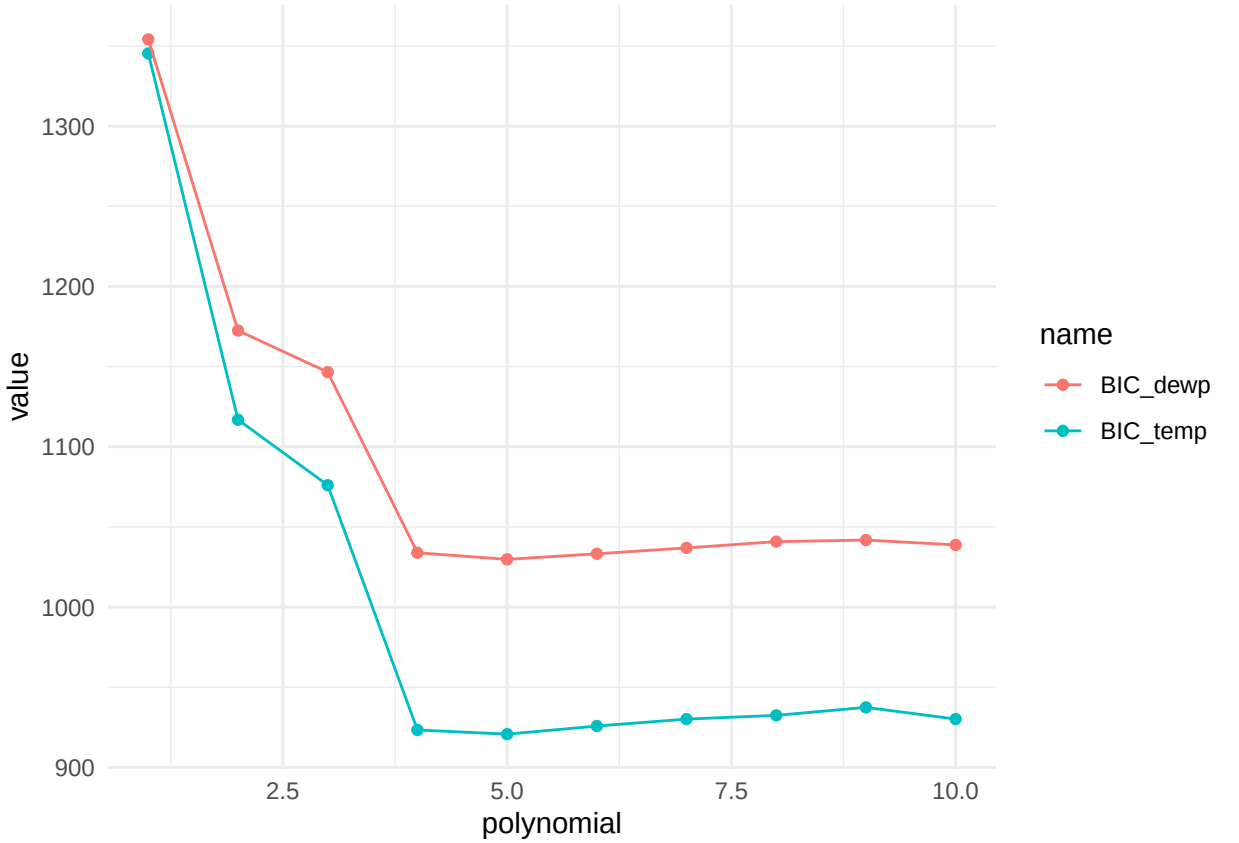
```
weather_weekly <- weather_weekly %>%
  mutate(numeric_week = as.numeric(week_index) - min(as.numeric(week_index)))
```

```
model_temperature_1  <- lm(average_temperature ~ numeric_week, data = weather_weekly)
model_temperature_2  <- lm(average_temperature ~ poly(numeric_week, degree = 2),  data = weather_weekly)
model_temperature_3  <- lm(average_temperature ~ poly(numeric_week, degree = 3),  data = weather_weekly)
model_temperature_4  <- lm(average_temperature ~ poly(numeric_week, degree = 4),  data = weather_weekly)
model_temperature_5  <- lm(average_temperature ~ poly(numeric_week, degree = 5),  data = weather_weekly)
model_temperature_6  <- lm(average_temperature ~ poly(numeric_week, degree = 6),  data = weather_weekly)
model_temperature_7  <- lm(average_temperature ~ poly(numeric_week, degree = 7),  data = weather_weekly)
model_temperature_8  <- lm(average_temperature ~ poly(numeric_week, degree = 8),  data = weather_weekly)
model_temperature_9  <- lm(average_temperature ~ poly(numeric_week, degree = 9),  data = weather_weekly)
model_temperature_10 <- lm(average_temperature ~ poly(numeric_week, degree = 10), data = weather_weekly)

model_dewpoint_1  <- lm(average_dewpoint ~ numeric_week, data = weather_weekly)
model_dewpoint_2  <- lm(average_dewpoint ~ poly(numeric_week, degree = 2),  data = weather_weekly)
model_dewpoint_3  <- lm(average_dewpoint ~ poly(numeric_week, degree = 3),  data = weather_weekly)
model_dewpoint_4  <- lm(average_dewpoint ~ poly(numeric_week, degree = 4),  data = weather_weekly)
model_dewpoint_5  <- lm(average_dewpoint ~ poly(numeric_week, degree = 5),  data = weather_weekly)
model_dewpoint_6  <- lm(average_dewpoint ~ poly(numeric_week, degree = 6),  data = weather_weekly)
model_dewpoint_7  <- lm(average_dewpoint ~ poly(numeric_week, degree = 7),  data = weather_weekly)
model_dewpoint_8  <- lm(average_dewpoint ~ poly(numeric_week, degree = 8),  data = weather_weekly)
model_dewpoint_9  <- lm(average_dewpoint ~ poly(numeric_week, degree = 9),  data = weather_weekly)
model_dewpoint_10 <- lm(average_dewpoint ~ poly(numeric_week, degree = 10), data = weather_weekly)
```

(2 points) Part-3: Evalute the model fits best for each outcomes

For each of the outcomes – `average_temperature` at the weekly level, and `average_dewpoint` at the weekly level – make an assessment based on either AIC or BIC for why one polynomial degree produces the best fitting model. In doing so, describe why you have chosen to use either AIC or BIC, what the particular scoring of this metric is doing (i.e. write the formula, and explain to your reader what is happening in that formula). Especially compelling in producing your argument for why you prefer a particular model form is to create a plot of the polynomial degree on the x-axis and the metric score on the y-axis.

```
model_bic <- data.frame(
  polynomial = 1:10,
  BIC_temp = c(BIC(model_temperature_1), BIC(model_temperature_2), BIC(model_temperature_3),
               BIC(model_temperature_4), BIC(model_temperature_5), BIC(model_temperature_6),
               BIC(model_temperature_7), BIC(model_temperature_8), BIC(model_temperature_9),
               BIC(model_temperature_10)),
  BIC_dewp = c(BIC(model_dewpoint_1), BIC(model_dewpoint_2), BIC(model_dewpoint_3),
               BIC(model_dewpoint_4), BIC(model_dewpoint_5), BIC(model_dewpoint_6),
               BIC(model_dewpoint_7), BIC(model_dewpoint_8), BIC(model_dewpoint_9),
               BIC(model_dewpoint_10))
  )

model_bic %>%
  pivot_longer(cols = c('BIC_temp', 'BIC_dewp')) %>%
  ggplot() +
  aes(x = polynomial, y = value, color = name) +
  geom_point() +
  geom_line()
```



Looking at these two BIC scoring criteria there seems to be a clear **lack** of improvement beyond a polynomial order of four. *Perhaps* moving from four to five would still increase the model's performance, but it is small compared to the polynomials 2-4. For us, if we were fitting this model, we would be likely to stop at `poly( , degree = 4)`.

Question-3: Smooth Moves

In the async lecture, Jeffrey proposes four different smoothers that might be used:

1. **Moving Average**: These moving average smoothers can be either symmetric or, often preferred, backward smoothers. Please use a backward smoother, and make the choice about the number of periods based off of some evaluation of different choices. You might consult [this page] in *Forecasting Principles and Practice 3*.
2. **Regression Smoothers**: Please use the polynomial regression that you stated you most preferred from your BIC analysis to the last question.
3. (Optional) **Spline Smoothers**: There is a reading in the repository that provides some more background (it is a review from 2019) on using spline smoothers. The current implementation that we prefer in R is the `splines2` library. For your spline smoother, use the `splines2::naturalSpline` function. Once you have fitted this spline, you can use the `predict` method to produce values. A good starting place for this is [here]. We'll note that this is the most challenging of the smoothers to get running in this assignment, and so getting it running successfully is optional.
4. **Kernel Smoothers**.: Please use the `ksmooth` function that is available to you in the `stats` library. Because `stats` is always loaded in R, we have not referred to it using the `::` notation.

(6 points, with 2 optional) Part-1: Create Smoothers

With the weekly weather data that you used for the previous question, produce a smoothed variable for `average_temperature` and `average_dewpoint` using each of the four smoothers described in the async. Three smoothers are required of this question – (1) Moving Average; (2) Regression Smoothers; and, (3) Kernel Smoothers. The fourth, splines, is optional but if you produce a spline smoother that is working effectively, you can earn two bonus points. (Note that the homework maximum score is still 100%.)

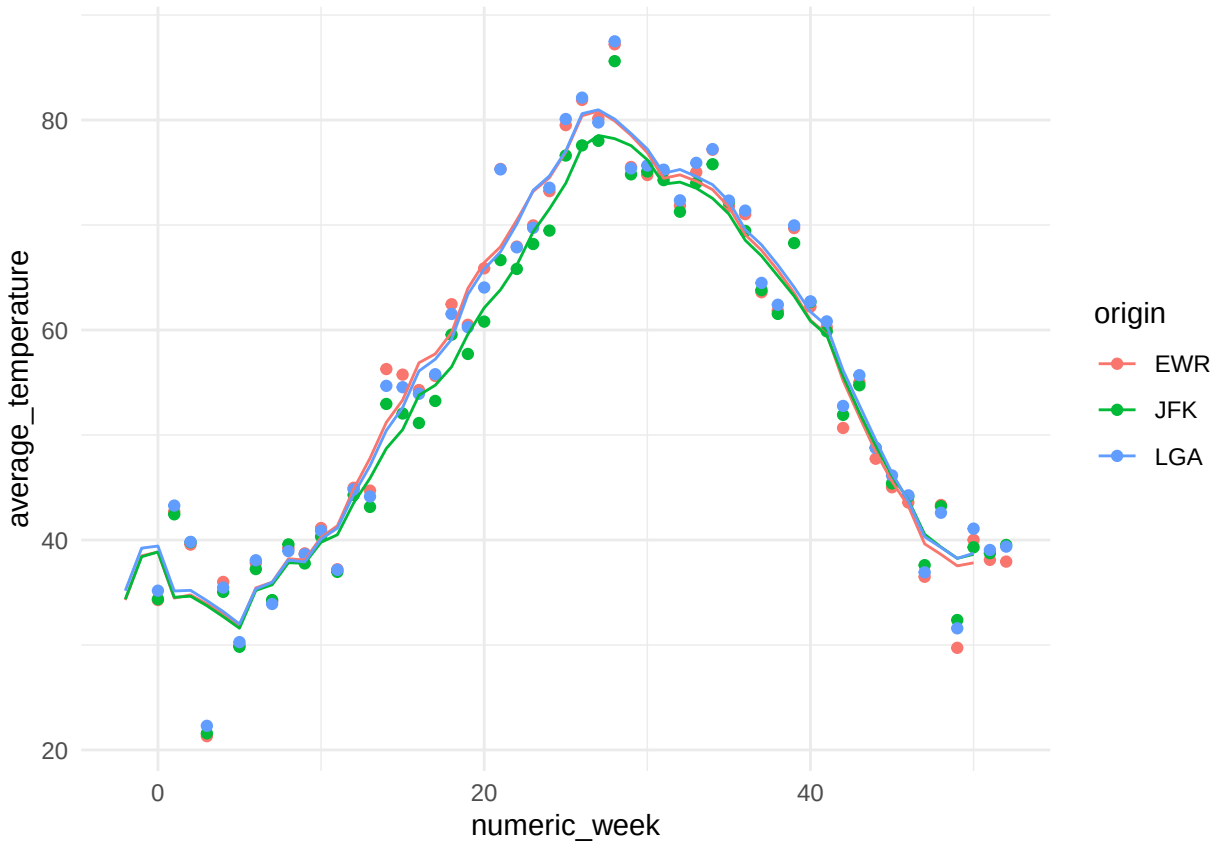
When you are done with this task, you should have created eight new variables that are each a smoothed version of this series.

For each smoother that you produce:

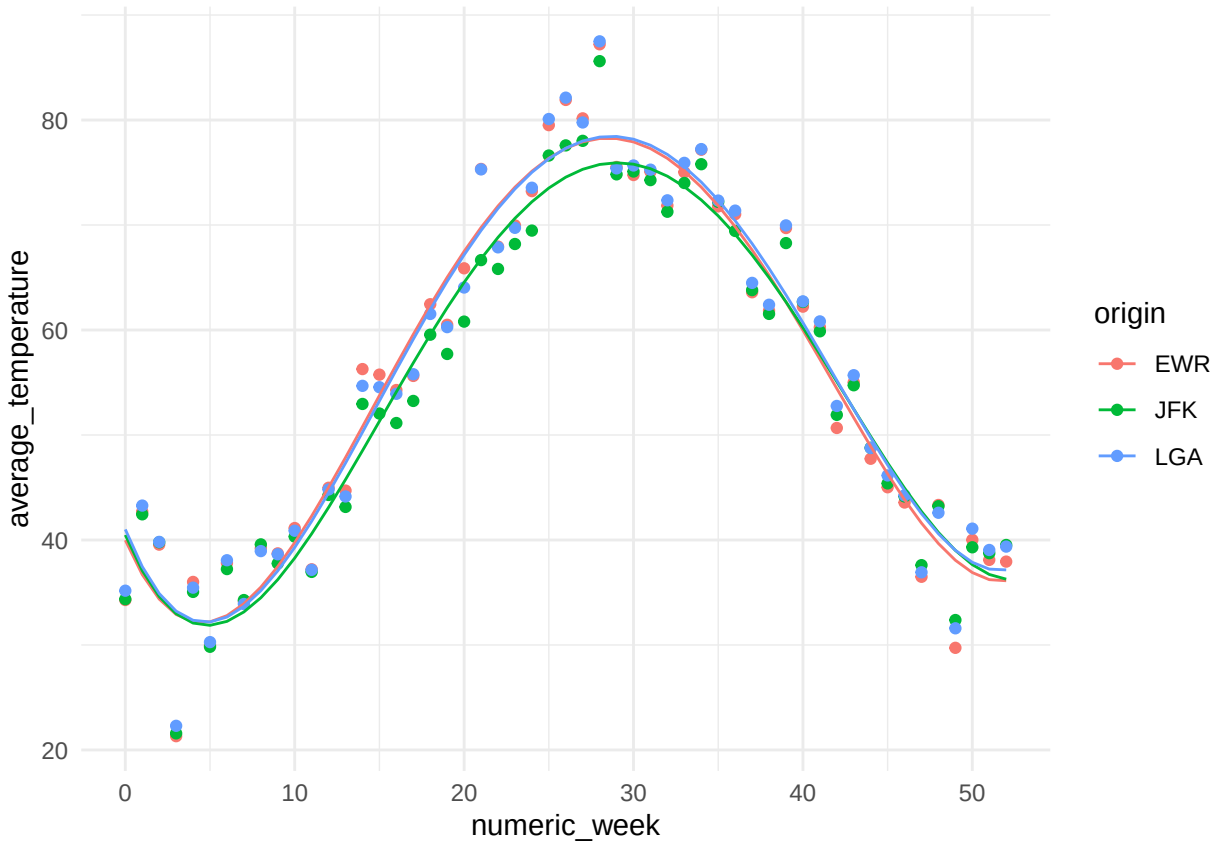
- Fit the smoother **within** each origin. That is, fit the smoother for JFK separately from LaGuardia and Newark.

- Attach the values that are produced by the smoother onto the `weekly_weather` dataframe.
- Produce a plot that shows the original data as `geom_point()`, and the smoother's predictions as `geom_line()`.
- Your goal is not to produce **any** smoother, but instead, for each class of smoother, the version that is doint the best job that is possible by this smoother. That is, you are working through the hyper-parametrs to these algorithms to produce their most effective output.

```
weather_weekly %>%
  group_by(origin) %>%
  mutate(moving_average = slider::slide_dbl(average_temperature, .f='mean', .before=4, .after=0)) %>%
  ggplot() +
  geom_point(aes(x=numeric_week, y=average_temperature, color = origin)) +
  geom_line(aes(x=numeric_week-2, y=moving_average, color = origin))
```




```
weather_weekly %>%
  group_by(origin) %>%
  mutate(regression_prediction = predict(lm(average_temperature ~ poly(as.numeric(week_index), degree=4
  ggplot() +
  geom_point(aes(x=numeric_week, y=average_temperature, color = origin)) +
  geom_line(aes(x=numeric_week, y=regression_prediction, color = origin))
```




```
plot_smoother <- function(band_width) {
  ewr_smoother <- weather_weekly %>%
    filter(origin == 'EWR') %$%
    ksmooth(x=as.numeric(week_index), y=average_temperature, bandwidth = band_width, n.points = length(w
  jfk_smoother <- weather_weekly %>%
    filter(origin == 'JFK') %$%
    ksmooth(x=as.numeric(week_index), y=average_temperature, bandwidth = band_width, n.points = length(w
  lga_smoother <- weather_weekly %>%
    filter(origin == 'LGA') %$%
    ksmooth(x=as.numeric(week_index), y=average_temperature, bandwidth = band_width, n.points = length(w

  smoothed_dataset <- data.frame(
    origin             = rep(c("EWR", "JFK", "LGA"), each = 53),
    week_index_numeric = c(ewr_smoother$x, jfk_smoother$x, lga_smoother$x),
    smoother_temp      = c(ewr_smoother$y, jfk_smoother$y, lga_smoother$y)
  )

  weather_weekly <- weather_weekly %>%
    mutate(week_index_numeric = as.numeric(week_index)) %>%
    left_join(smoothed_dataset, by = c('origin', 'week_index_numeric'))

  weather_weekly %>%
    ggplot() +
    geom_point(aes(x=week_index, y=average_temperature, color=origin)) +
    geom_line(aes( x=week_index, y=smoother_temp, color=origin))
```

```
  ## return(names(weather_weekly))
}

bandwidth_1 <- plot_smoother(band_width = 1)
bandwidth_2 <- plot_smoother(band_width = 2)
bandwidth_3 <- plot_smoother(band_width = 3)
bandwidth_4 <- plot_smoother(band_width = 4)
bandwidth_5 <- plot_smoother(band_width = 5)
bandwidth_6 <- plot_smoother(band_width = 6)


(bandwidth_1 + bandwidth_2) /
  (bandwidth_3 + bandwidth_4) /
  (bandwidth_5 + bandwidth_6)
```