W271 Assignment 7

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

library(lubridate)

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

library(sandwich)
library(lmtest)

library(nycflights13)
library(blsR)

#install.packages("splines2")
library(splines2)
library(slider)

set.seed(1719)
```

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

Note: I used AI to help certain code chunks in Questions 2 and 3, but I went through every single line of code, fiddled with it, and made extensive edits.

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?

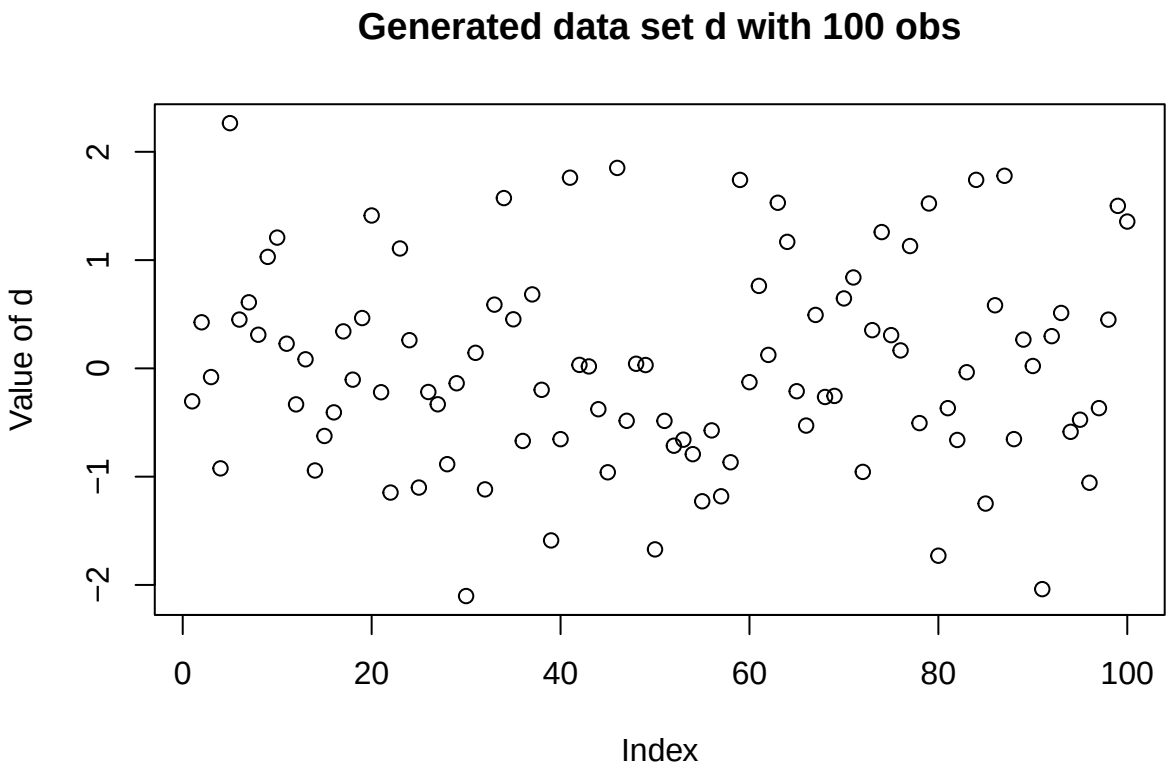
Answer

(Code is after comments and answers)

Note that both the AIC and BIC values increase (worse fit) as the number of estimated parameters (k) increases. Since the outcome variable (y) is pure white noise, none of the input variables (x1 through x10) have a true relationship with it. The slight variation in the scores is likely by chance, but the overall trend is an increase because the improved fit (lower residual sum of squares) from adding random predictors is outweighed by the complexity penalty.

Also, the BIC curve is consistently above the AIC curve and increases more rapidly. The penalty term in AIC is 2k, whereas the penalty in BIC is ln(n) k. With a sample size of n=100, ln(100) is about 4.6, meaning BIC's marginal penalty of k is more than twice as large as AIC's penalty for each additional parameter. This means BIC imposes a harsher significant penalty for adding parameters. Therefore, BIC is more stringent in preferring the simpler model (model0 with k=1) in this scenario where added complexity does not lead to a substantially better fit. AIC only mey lead us to model2 (3 parameters).

```
y <- rnorm(100, mean = 0, sd = 1)
plot(y,
     xlab = "Index",
     ylab = "Value of d",
     main = "Generated data set d with 100 obs")
```




```
input_vars <- list()

for (i in 1:10) {
  var <- rnorm(100, mean = 0, sd = 1)

  var_names <- paste0("x", i)
  input_vars[[var_names]] <- var
}
```

```r
x <- as_tibble(input_vars)

d <- bind_cols(y = y, x)
```

```r
# First the intercept only model, or model0
model0 <- lm(formula = y ~ 1,
             data = d)

summary(model0)
```

```
##
## Call:
## lm(formula = y ~ 1, data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.12026 -0.64938 -0.07568  0.51180  2.24686
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.01777    0.09308   0.191    0.849
##
## Residual standard error: 0.9308 on 99 degrees of freedom
```

```r
# Models 1-10: Adding one more X variable sequentially (k+1 parameters)
# I write the model explicitly (not using loop) so that I can check more easily

model1 <- lm(y ~ x1, data = d)
model2 <- lm(y ~ x1 + x2, data = d)
model3 <- lm(y ~ x1 + x2 + x3, data = d)
model4 <- lm(y ~ x1 + x2 + x3 + x4, data = d)
model5 <- lm(y ~ x1 + x2 + x3 + x4 + x5, data = d)
model6 <- lm(y ~ x1 + x2 + x3 + x4 + x5 + x6, data = d)
model7 <- lm(y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7, data = d)
model8 <- lm(y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8, data = d)
model9 <- lm(y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9, data = d)
model10 <- lm(y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10, data = d)

# Store models in a list
models_list <- list(
  model0 = model0, model1 = model1, model2 = model2, model3 = model3,
  model4 = model4, model5 = model5, model6 = model6, model7 = model7,
  model8 = model8, model9 = model9, model10 = model10
)
```

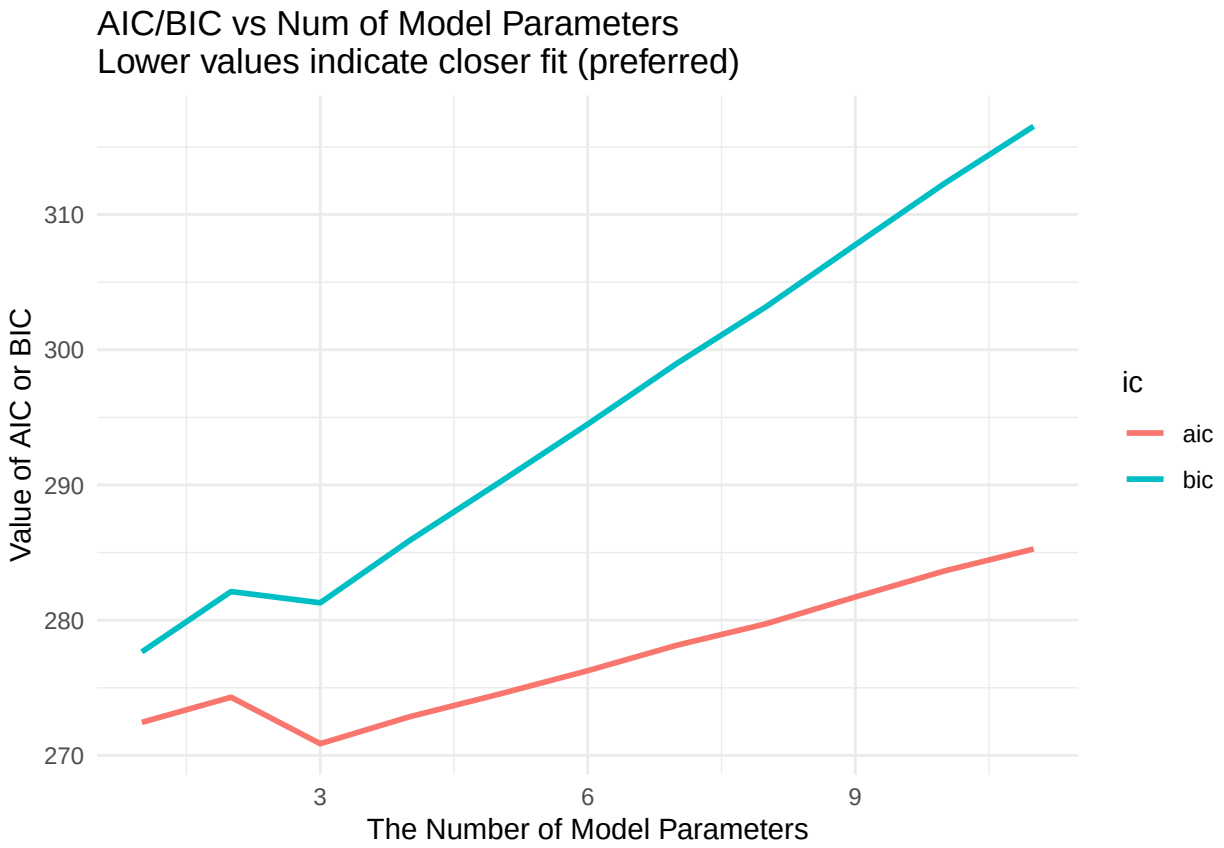
```r
# Part 1.3: Make results data
results_data_wide <- tibble(
  parameters = 1:11,
  # Extract AIC and BIC values for each model
  aic = map_dbl(models_list, AIC),
  bic = map_dbl(models_list, BIC)
)

results_data <- pivot_longer(results_data_wide,
             cols=c("aic", "bic"),
```

```
            names_to = "ic",
            values_to = "score")
```

```
results_data
```

```
## # A tibble: 22 x 3
##    parameters ic    score
##         <int> <chr> <dbl>
##  1          1 aic    272.
##  2          1 bic    278.
##  3          2 aic    274.
##  4          2 bic    282.
##  5          3 aic    271.
##  6          3 bic    281.
##  7          4 aic    273.
##  8          4 bic    286.
##  9          5 aic    275.
## 10          5 bic    290.
## # i 12 more rows
```

```
results_data %>%
  ggplot(aes(
    x = parameters,
    y = score,
    color = ic
  )) +
  geom_line(linewidth = 1) +
  labs(
    title = "AIC/BIC vs Num of Model Parameters\nLower values indicate closer fit (preferred)",
    x = "The Number of Model Parameters",
    y = "Value of AIC or BIC"
  )
```



What do you note? Fill this in! Already did this.

(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.

Answer

In this plot (shown below in the code blocks), both AIC and BIC scores decrease as the number of parameters increases from k=1 (after k = 6) to k=11.

The minimum of both curves occurs at k=11 (model10), which includes all ten x variables. This is the correctly specified population model, so the criteria are correctly identifying the most complex model as the best one.

The difference between this plot and the white-noise plot is that in the white-noise case, both metrics penalized complexity because it *didn't* improve the fit; in this population model, both metrics reward complexity because each additional parameter significantly improves the model's fit (i.e., significantly decreases the residual sum of squares). Similarity to the first plot, the BIC curve is still more stringent.

This demonstrates that when there is a true, significant relationship in the data, the goodness-of-fit term dominates the complexity penalty term, leading both AIC and BIC to favor the most complex (and correctly specified) model.

```
# Because now we generate y as a function of Xs and the white noise,
# we shall generate the X first
data_list_2 = list()

wn <- rnorm (n = 100, mean = 0, sd = 1)

for (i in 1:10) {
  var <- rnorm(n = 100, mean = 0, sd = 1)
  var_name <- paste0("x", i)

  data_list_2[[var_name]] <- var
}

x_2 = as_tibble(data_list_2)


beta_vec <- seq(from = 0.1, to = 1.0, length.out = 10)
beta_vec
```

```
##  [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
y_2 <- (as.matrix(x_2) %*% beta_vec) + wn

d_2 <- bind_cols(y = y_2, x_2)
```

```r
model0 <- lm(formula = y_2 ~ 1,
             data = d_2)

summary(model0)
```

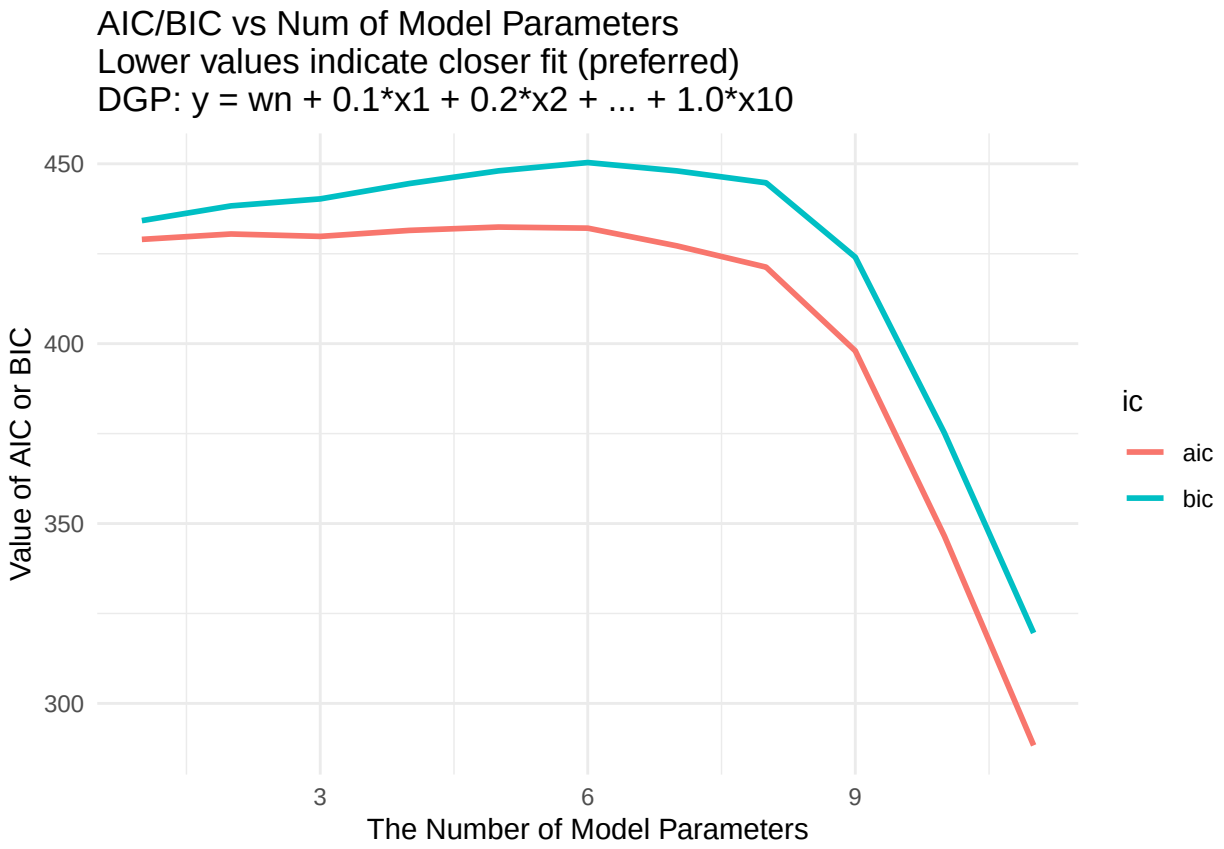
```
##
## Call:
## lm(formula = y_2 ~ 1, data = d_2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.0030 -1.3098 -0.1977  1.5195  4.1944
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.2277     0.2036  -1.118    0.266
##
## Residual standard error: 2.036 on 99 degrees of freedom
```

```r
# Models 1-10: Adding one more X variable sequentially (k+1 parameters)
# I write the model explicitly (not using loop) so that I can check more easily

model1 <- lm(y_2~ x1, data = d_2)
model2 <- lm(y_2~ x1 + x2, data = d_2)
model3 <- lm(y_2~ x1 + x2 + x3, data = d_2)
model4 <- lm(y_2~ x1 + x2 + x3 + x4, data = d_2)
model5 <- lm(y_2~ x1 + x2 + x3 + x4 + x5, data = d_2)
model6 <- lm(y_2~ x1 + x2 + x3 + x4 + x5 + x6, data = d_2)
model7 <- lm(y_2~ x1 + x2 + x3 + x4 + x5 + x6 + x7, data = d_2)
model8 <- lm(y_2~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8, data = d_2)
model9 <- lm(y_2~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9, data = d_2)
model10 <- lm(y_2 ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10, data = d_2)

# Store models in a list
models_list <- list(
  model0 = model0, model1 = model1, model2 = model2, model3 = model3,
  model4 = model4, model5 = model5, model6 = model6, model7 = model7,
  model8 = model8, model9 = model9, model10 = model10
)
```

```r
# Part 2.3: Make results data
results_data_wide_2 <- tibble(
  parameters = 1:11,
  # Extract AIC and BIC values for each model
  aic = map_dbl(models_list, AIC),
  bic = map_dbl(models_list, BIC)
)

results_data_2 <- pivot_longer(results_data_wide_2,
             cols=c("aic", "bic"),
             names_to = "ic",
             values_to = "score")
```

```r
results_data_2 %>%
  ggplot(aes(
```

```
    x = parameters,
    y = score,
    color = ic
  )) +
  geom_line(linewidth = 1) +
  labs(
    title = "AIC/BIC vs Num of Model Parameters\nLower values indicate closer fit (preferred)\nDGP: y =
    x = "The Number of Model Parameters",
    y = "Value of AIC or BIC"
  )
```



What do you notice about this plot and the model performance overall?

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
```

Answer

The code below aggregates the weather data to the weekly level (yearweek) and computes the average temperature and dewpoint for each of the three airports (origin). The resulting tsibble is keyed by origin, which is necessary for the next step.

```
weather
```

```
## # A tibble: 26,115 x 15
##    origin  year month   day  hour  temp  dewp humid wind_dir wind_speed
##    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
##  1 EWR     2013     1     1     1  39.0  26.1  59.4      270      10.4
##  2 EWR     2013     1     1     2  39.0  27.0  61.6      250       8.06
##  3 EWR     2013     1     1     3  39.0  28.0  64.4      240      11.5
##  4 EWR     2013     1     1     4  39.9  28.0  62.2      250      12.7
##  5 EWR     2013     1     1     5  39.0  28.0  64.4      260      12.7
##  6 EWR     2013     1     1     6  37.9  28.0  67.2      240      11.5
##  7 EWR     2013     1     1     7  39.0  28.0  64.4      240      15.0
##  8 EWR     2013     1     1     8  39.9  28.0  62.2      250      10.4
##  9 EWR     2013     1     1     9  39.9  28.0  62.2      260      15.0
## 10 EWR     2013     1     1    10  41    28.0  59.6      260      13.8
## # i 26,105 more rows
## # i 5 more variables: wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour <dttm>
```

```
weather %>%
  duplicates(key = origin, index = time_hour)
```

```
## # A tibble: 0 x 15
## # i 15 variables: origin <chr>, year <int>, month <int>, day <int>, hour <int>,
## #   temp <dbl>, dewp <dbl>, humid <dbl>, wind_dir <dbl>, wind_speed <dbl>,
## #   wind_gust <dbl>, precip <dbl>, pressure <dbl>, visib <dbl>,
## #   time_hour <dttm>
```

```
is_tsibble(weather)
```

```
## [1] FALSE
```

```
weather_ts <-
  weather %>%
  as_tsibble(
    key = origin,
    index = time_hour
```

```
  )

weekly_td <- weather_ts %>%
  index_by(week_index = yearweek(time_hour)) %>%
  group_by(origin) %>%
  summarize(average_temperature = mean(temp, na.rm = TRUE),
            average_dewpoint = mean(dewp, na.rm = TRUE))
weekly_td
```

```
## # 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
##  4 EWR      2013 W04                 21.3              3.77
##  5 EWR      2013 W05                 36.0             25.7
##  6 EWR      2013 W06                 30.0             17.6
##  7 EWR      2013 W07                 37.8             25.3
##  8 EWR      2013 W08                 34.2             21.3
##  9 EWR      2013 W09                 39.2             27.8
## 10 EWR      2013 W10                 38.7             26.8
## # i 149 more rows
```

(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.

Answer

The code below first creates a numeric_week index starting at 0 to be used as the predictor in the poly() function. Models are then fitted by nesting the data by origin, allowing for iteratively estimate 10 polynomial models (degree p=1 to p=10) for each airport independently (I did it first by looping then adpated the functional coding style suggested by AI).

```
first_week_num <-
  weekly_td %>%
  pull(week_index) %>%
  min() %>%
  as.numeric()

weekly_td <-
  weekly_td %>%
  mutate(numeric_week = as.numeric(week_index) - first_week_num)




print(head(weekly_td, n = 2))
```

```
## # A tsibble: 2 x 5 [1W]
## # Key:       origin [1]
##   origin week_index average_temperature average_dewpoint numeric_week
##   <chr>      <week>               <dbl>            <dbl>        <dbl>
## 1 EWR      2013 W01                34.3             19.4            0
## 2 EWR      2013 W02                42.7             33.3            1
```

```
# Fit 10 polynomial (in time) models for each of the two outcomes
# I will fit models within each airport (origin) using nest/map.

# Prepare the data for nested model fitting
weather_nested <- weekly_td %>%
  group_by(origin) %>%
  nest()
```

```r
weather_nested
```

```
## # A tibble: 3 x 2
## # Groups:   origin [3]
##   origin data
##   <chr>  <list>
## 1 EWR    <tbl_ts [53 x 4]>
## 2 JFK    <tbl_ts [53 x 4]>
## 3 LGA    <tbl_ts [53 x 4]>
```

```r
# Function to fit 10 polynomial models for a given outcome (y_var)
fit_polynomial_models <- function(data, y_var) {
  # Create a list to store models 1 through 10
  models_list <- list()

  # Base formula (vector) for the linear model
  base_formula <- paste0(y_var, " ~ poly(numeric_week, degree = ", 1:10, ")")

  # Fit models using map so I can pass a vector and pass it to
  # lm() for iterations
  models_list <- map(base_formula, ~ lm(as.formula(.x), data = data))

  # Name the list elements
  names(models_list) <- paste0("model", 1:10)

  return(models_list)
}

# Fit models for average_temperature
weather_models_temp <- weather_nested %>%
  mutate(temp_models = map(data, ~ fit_polynomial_models(.x, "average_temperature")))

# Fit models for average_dewpoint
weather_models_dewp <- weather_nested %>%
  mutate(dewp_models = map(data, ~ fit_polynomial_models(.x, "average_dewpoint")))

# Combine the results for convenience (optional, but helpful for the next part)
weather_models_all <- weather_models_temp %>%
  left_join(weather_models_dewp %>% select(origin, dewp_models), by = "origin")

print("Models fitted and stored in weather_models_all$temp_models and $dewp_models")
```

```
## [1] "Models fitted and stored in weather_models_all$temp_models and $dewp_models"
```

```r
summary(weather_models_temp[1, ]$temp_models[[1]]$model3)
```

```
##
## Call:
## lm(formula = as.formula(.x), data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.1349  -5.6023  -0.0636   2.6529  15.0694
##
## Coefficients:
```

```
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                      55.1947     0.9606  57.461  < 2e-16 ***
## poly(numeric_week, degree = 3)1   27.7540     6.9930   3.969 0.000236 ***
## poly(numeric_week, degree = 3)2 -104.7144     6.9930 -14.974  < 2e-16 ***
## poly(numeric_week, degree = 3)3  -25.9221     6.9930  -3.707 0.000534 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.993 on 49 degrees of freedom
## Multiple R-squared:  0.8381, Adjusted R-squared:  0.8282
## F-statistic: 84.57 on 3 and 49 DF,  p-value: < 2.2e-16
```

(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.

Answer

Analysis Rationale for AIC vs BIC I will choose to use the BIC (Bayesian Information Criterion) for model selection.

Formula and Explanation:

$$BIC = 2\log(\hat{\sigma}_k) + \frac{k\log(n)}{n}$$

Where:

- $\hat{\sigma}_k^2$ is the MLE for the variance;
- k is the number of estimated parameters (including intercept and sigma square);
- n is the number of observations used in the estimation.

The BIC provides a stronger penalty for model complexity ($k \cdot \log(n)$) than AIC (2k), as shown in my answer to Question 1, which is useful when the goal is to select the better model out of a set of candidates. Since the true functional form of the weather cycle is unknown, BIC helps guard against overfitting by favoring simpler models that still capture the essential trend (e.g., the annual seasonality).

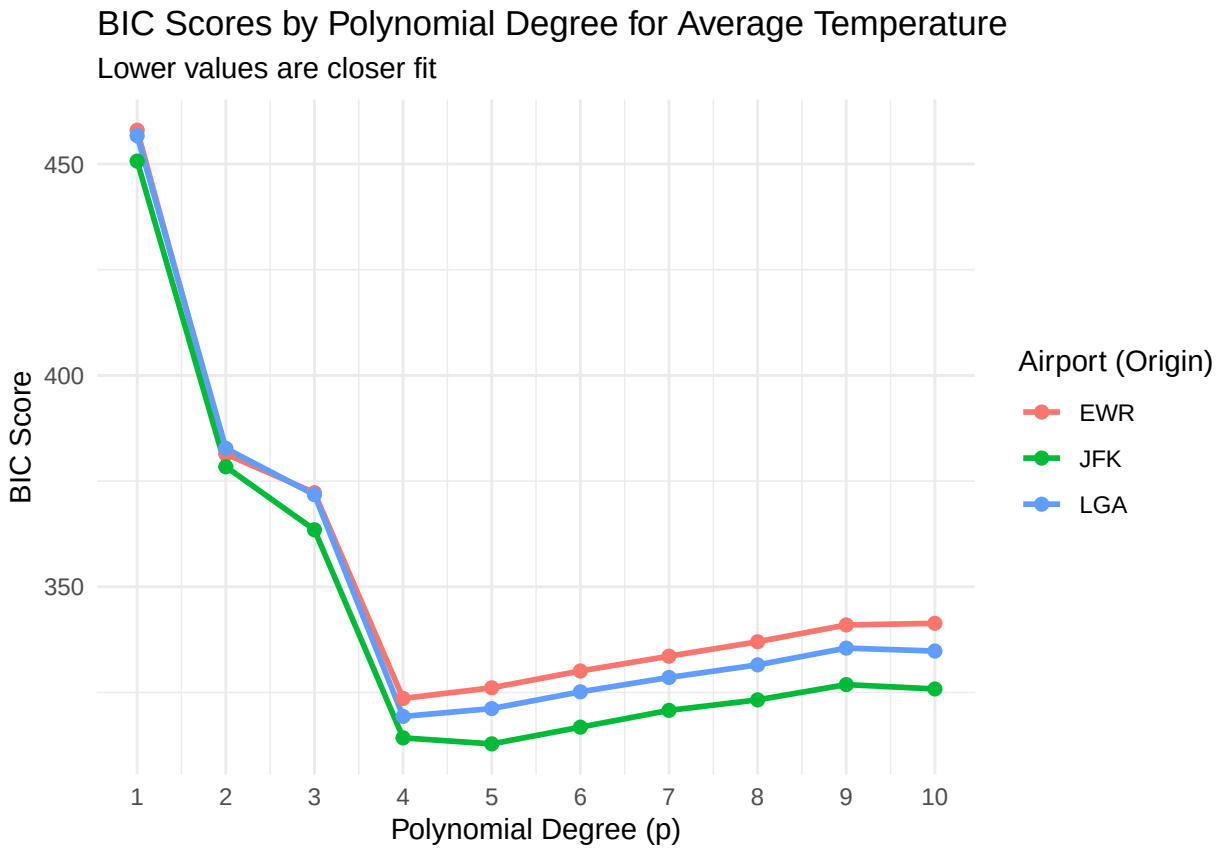
Evaluation for Average Temperature: The plot below shows that the BIC scores decrease sharply for the initial increase in polynomial degree, indicating that a linear or quadratic model is insufficient. For all three airports, the BIC score reaches its minimum around a degree of p=4 or p=5. After this point, the curve flattens out, and then begins to slightly increase, indicating that the complexity penalty is starting to outweigh the marginal improvement in fit.

Best Degree: The 4th or 5th degree polynomial is the best fitting model according to BIC for all origins. This degree is sufficient to capture the main seasonal (annual) cycle, which typically requires a pair of sine/cosine terms or, in this case, a 4th-degree polynomial. I will select the 4th degree as the preferred model for the next question, as it is the most parsimonious option among the best-fitting ones.

Evaluation for Average Dewpoint: Similar to temperature, the BIC scores for dewpoint show a sharp initial decrease. The BIC score reaches its minimum around a degree of p=4 or p=5 for all three airports. The curves show that a degree of p=4 is highly effective at capturing the trend. For EWR and LGA, the minimum is p=4; for JFK, the minimum is p=5. Given the goal of parsimony, and the minimal difference between the p=4 and p=5 scores, I will choose the 4th degree polynomial as the preferred model for the next question.

```
# Write a function to extract BIC for a list of models
get_bic_data <-
  function(model_list, outcome_name) {
  map_df(model_list,
         ~ tibble(
            parameters = length(coef(.x)),
            bic_score = BIC(.x)),
    .id = "model_name") %>%
  mutate(
```

```
    degree = parameters - 1,
    outcome = outcome_name
    )
}

# Get BIC for all temperature models across 3 origins
bic_temp_data <-
  weather_models_all %>%
  rowwise() %>%
  mutate(bic_data = list(get_bic_data(temp_models, "Average Temperature"))) %>%
  ungroup() %>%
  unnest(bic_data)

# Plot the BIC scores
bic_temp_plot <- bic_temp_data %>%
  ggplot(aes(x = degree, y = bic_score, color = origin)) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  labs(
    title = "BIC Scores by Polynomial Degree for Average Temperature",
    subtitle = "Lower values are closer fit",
    x = "Polynomial Degree (p)",
    y = "BIC Score",
    color = "Airport (Origin)"
  ) +
  scale_x_continuous(breaks = 1:10)

bic_temp_plot
```

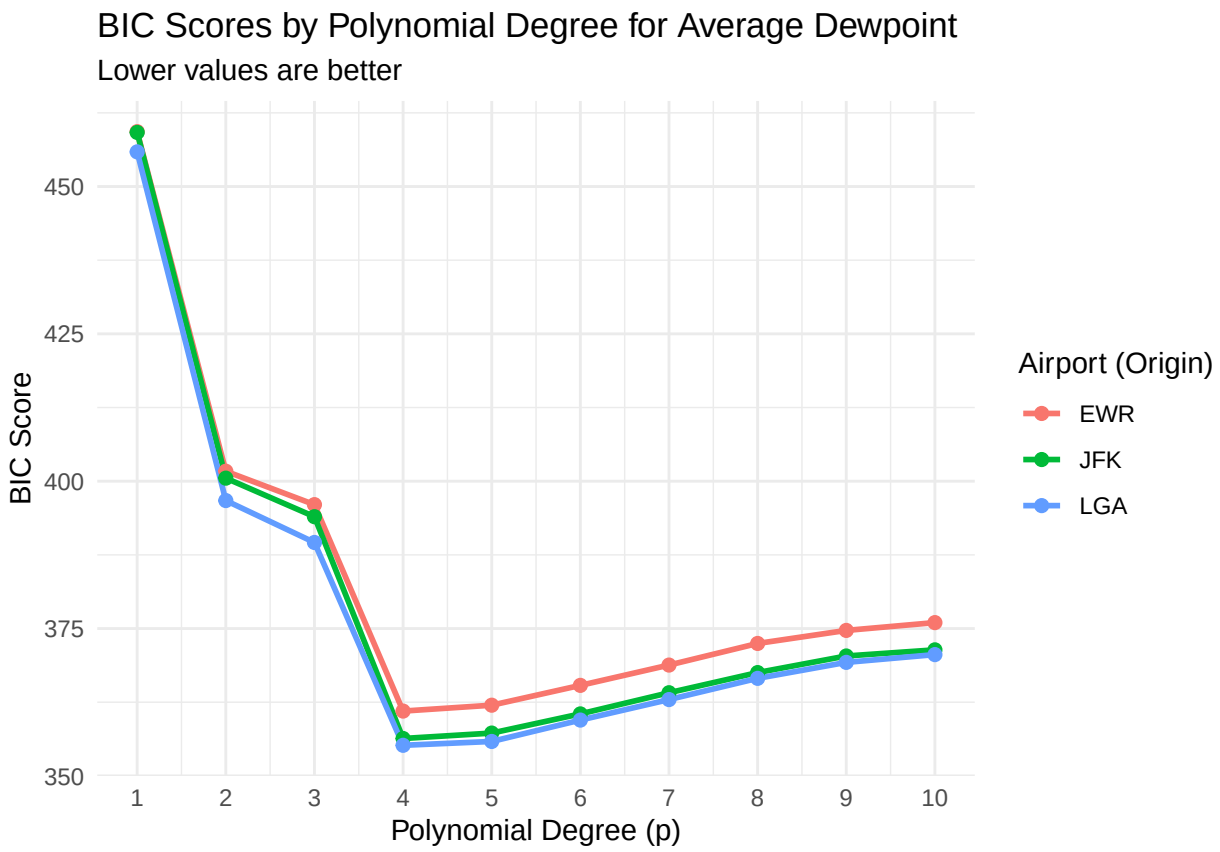



```
# Find the minimum BIC degree for each origin
best_temp_models <- bic_temp_data %>%
  group_by(origin) %>%
  filter(bic_score == min(bic_score)) %>%
  select(origin, best_degree_temp = degree, min_bic_temp = bic_score)

print(best_temp_models)
```

```
## # A tibble: 3 x 3
## # Groups:   origin [3]
##   origin best_degree_temp min_bic_temp
##   <chr>             <dbl>        <dbl>
## 1 EWR                   4         324.
## 2 JFK                   5         313.
## 3 LGA                   4         319.
```

```
# Write a function to extract BIC for a list of models
bic_dewp_data <- weather_models_all %>%
  rowwise() %>%
  mutate(bic_data = list(get_bic_data(dewp_models, "Average Dewpoint"))) %>%
  ungroup() %>%
  unnest(bic_data)

# Plot the BIC scores
bic_dewp_plot <- bic_dewp_data %>%
  ggplot(aes(x = degree, y = bic_score, color = origin)) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
```

```
  labs(
    title = "BIC Scores by Polynomial Degree for Average Dewpoint",
    subtitle = "Lower values are better",
    x = "Polynomial Degree (p)",
    y = "BIC Score",
    color = "Airport (Origin)"
  ) +
  scale_x_continuous(breaks = 1:10)

bic_dewp_plot
```




```
# Find the minimum BIC degree for each origin
best_dewp_models <- bic_dewp_data %>%
  group_by(origin) %>%
  filter(bic_score == min(bic_score)) %>%
  select(origin, best_degree_dewp = degree, min_bic_dewp = bic_score)

print(best_dewp_models)
```

```
## # A tibble: 3 x 3
## # Groups:   origin [3]
##   origin best_degree_dewp min_bic_dewp
##   <chr>             <dbl>        <dbl>
## 1 EWR                   4         361.
## 2 JFK                   4         356.
## 3 LGA                   4         355.
```

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.

Answer

The plots demonstrate that all four smoothers capture the dominant annual seasonality in the weekly average temperature.

The Moving Average (k=4) produces the most locally responsive and lag-heavy (due to the backward window) line, resulting in a somewhat non-smooth curve.

The Regression Smoother (p=4) yields a very smooth curve which captures the overall wave-like shape but may slightly miss peak/trough timing due to its global nature.

The Spline Smoother (df=5) provides a curve that is both smooth and responsive, conforming the peaks and valleys in previous curves.

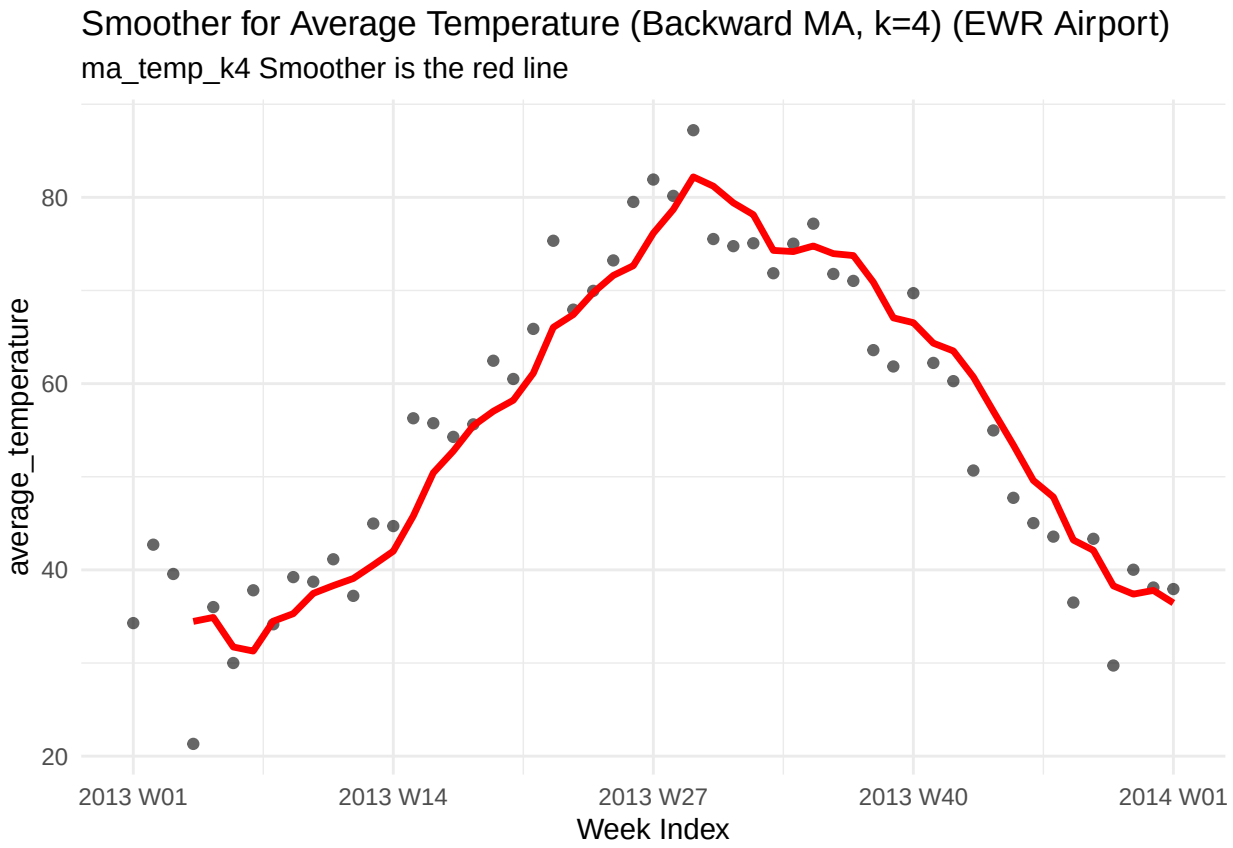
The Kernal Smoother (k=10) produces a curve that dampens the noise, showing a good balance between responsiveness and smoothness.

All four methods provide a useful less noisy component (of the trend/seasonal cycle).

```
# Note: For the smoothing plots, it is nicer to limit the origins to one for a clearer visual.
# I will use EWR for the plots, but fit the smoothers to ALL origins as requested.

# Prepare data for smoothing functions
# Function to get the smoother predictions, nested by origin
get_smoother_predictions <- function(data, smoother_col, smoother_name) {
  data %>%
    select(origin, week_index, numeric_week, average_temperature, average_dewpoint, !!sym(smoother_col))
    rename(predicted_value = !!sym(smoother_col)) %>%
    mutate(smoother = smoother_name)
}

# Base plotting function
plot_smoother <- function(data, smoother_col, outcome_col, title_suffix) {
  # Filter to a single origin for plot clarity
  data %>%
    filter(origin == "EWR") %>%
```

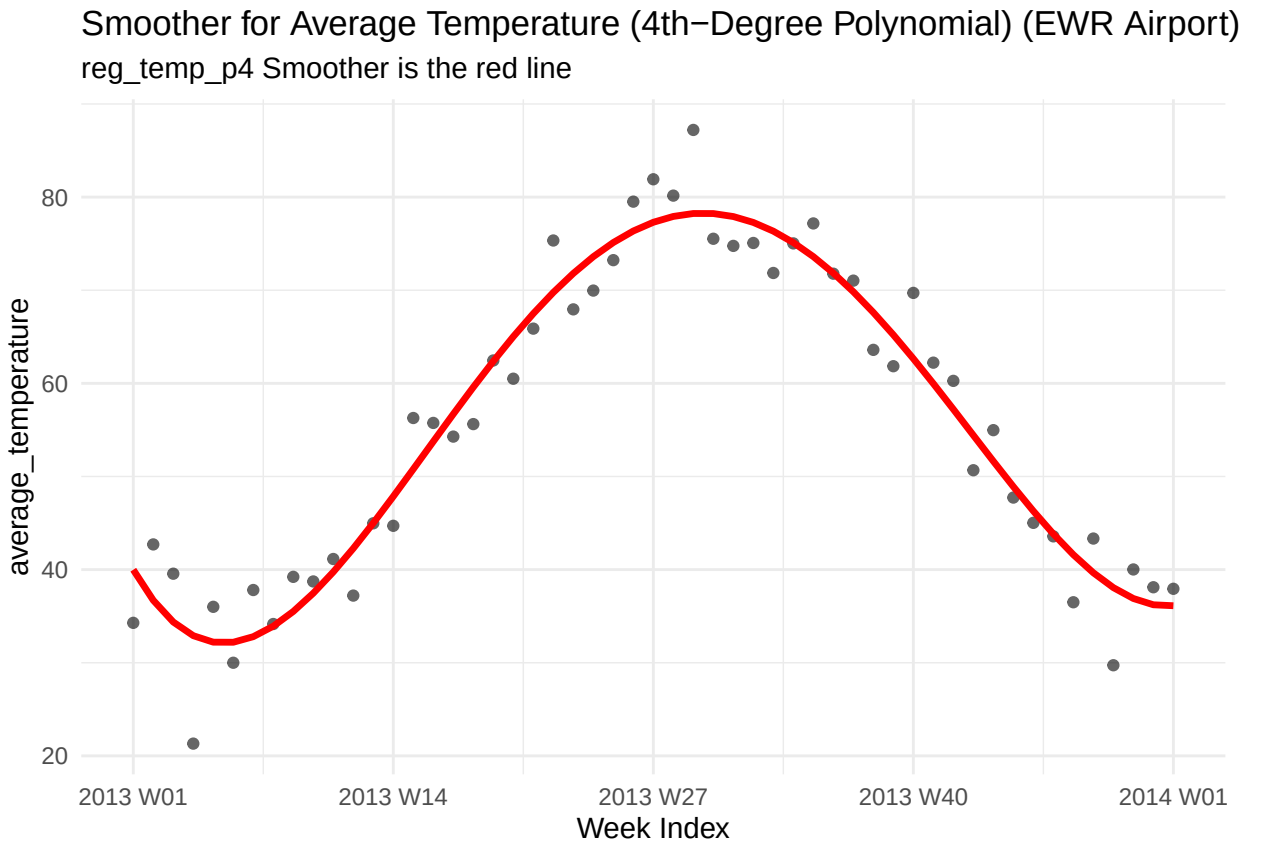
```
    ggplot(aes(x = week_index)) +
    geom_point(aes(y = !!sym(outcome_col)), alpha = 0.6) +
    geom_line(aes(y = !!sym(smoother_col)), color = "red", linewidth = 1.2) +
    labs(
      title = paste("Smoother for", title_suffix, "(EWR Airport)"),
      subtitle = paste(smoother_col, "Smoother is the red line"),
      x = "Week Index",
      y = outcome_col
    )
}

# 1. Moving Average Smoother (Backward-looking, k periods)
# Consultation with FPP3: A symmetric MA of order k=2m+1 provides a centered MA.
# For backward-looking (causal) smoothers, I'll use the 'rollmean' function
# We choose k=4 (a 4-week moving average) as a reasonable parameter to capture a weekly average with so

# Function to compute backward rolling mean
compute_backward_ma <- function(x, k) {
  # 'align = right' makes it a backward (causal) smoother
  zoo::rollmean(x, k = k, align = "right", fill = NA)
}

# Apply the moving average smoother (k=4)
weather_weekly <- weekly_td %>%
  group_by(origin) %>%
  mutate(
    ma_temp_k4 = compute_backward_ma(average_temperature, k = 4),
    ma_dewp_k4 = compute_backward_ma(average_dewpoint, k = 4)
  ) %>%
  ungroup()

# Produce plot for EWR (Example)
plot_smoother(weather_weekly, "ma_temp_k4", "average_temperature", "Average Temperature (Backward MA, k=
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



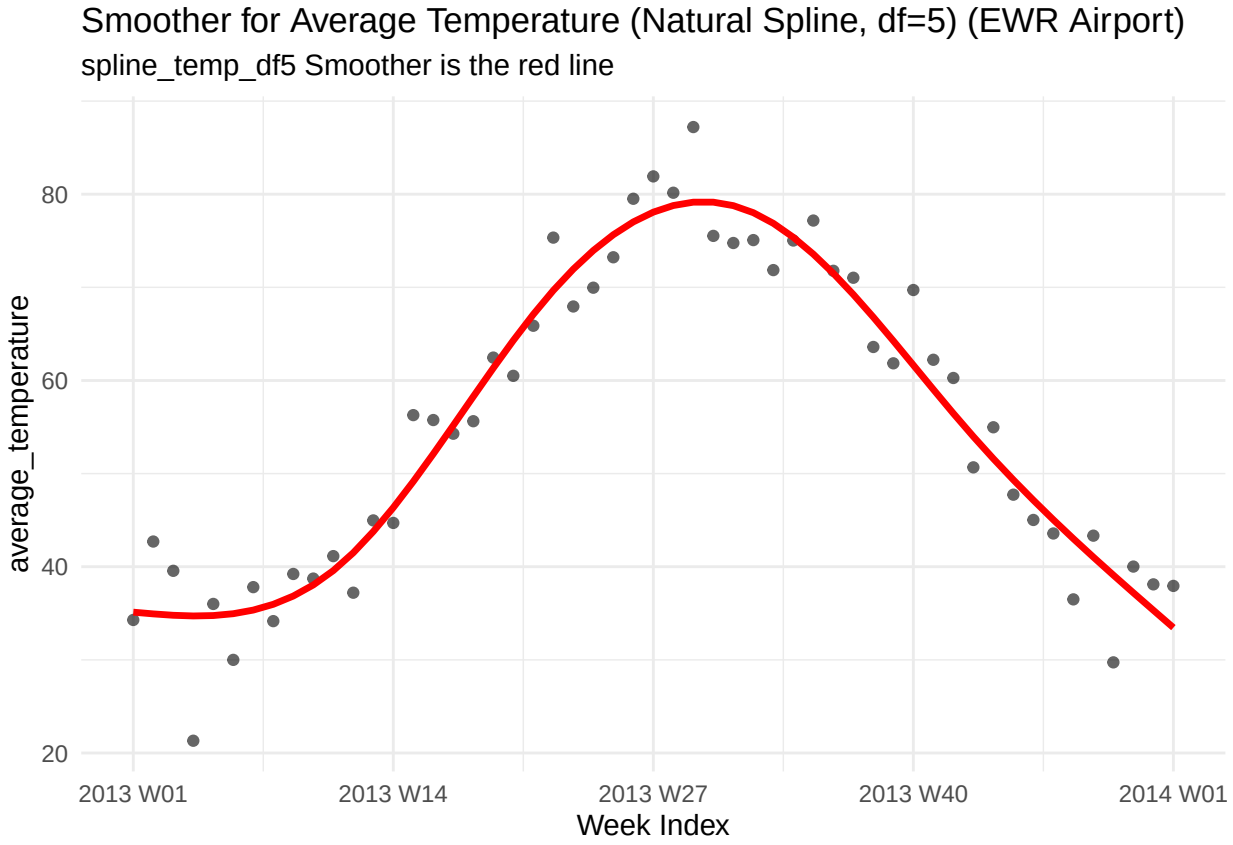

```
# 2. Regression Smoother (4th-degree polynomial chosen by BIC)

# Fit the chosen 4th-degree polynomial models within each origin
get_reg_pred <- function(data, outcome) {
  formula <- paste0(outcome, " ~ poly(numeric_week, degree = 4)")
  model <- lm(as.formula(formula), data = data)
  predict(model, newdata = data)
}

# Define the function (P4 is a common choice, adjust 'degree' if needed)
get_reg_pred <- function(data, y_var, degree = 4) {

  # 1. Construct the specific formula for the desired degree
  model_formula <- as.formula(paste0(y_var, " ~ poly(numeric_week, degree = ", degree, ")"))

  # 2. Fit the model using ONLY the 'data' tibble passed to the function
  fitted_model <- lm(model_formula, data = data)

  # 3. Predict using the fitted model and the same 'data' tibble (53 rows)
  predictions <- predict(fitted_model, newdata = data)

  # 4. Return the prediction vector
  return(predictions)
}

weather_weekly <- weather_weekly %>%
  group_by(origin) %>%
```

```
  filter(origin == "EWR") %>%
  mutate(
    reg_temp_p4 = get_reg_pred(., "average_temperature"),
    reg_dewp_p4 = get_reg_pred(., "average_dewpoint")
  ) %>%
  ungroup()

# Produce plot for EWR
plot_smoother(weather_weekly, "reg_temp_p4", "average_temperature", "Average Temperature (4th-Degree Pol
```




```
# I'll use df=5.

# Fit the natural cubic spline (df=5) within each origin
get_spline_pred <- function(data, outcome, df_val = 5) {
  # Ensure splines2 is loaded

  # The formula: y ~ naturalSpline(x, df)
  formula <- paste0(outcome, " ~ naturalSpline(numeric_week, df = ", df_val, ")")
  model <- lm(as.formula(formula), data = data)

  # Predict using the original data
  predict(model, newdata = data)
}

weather_weekly <- weather_weekly %>%
  group_by(origin) %>%
#  filter(origin == "EWR")
```

```
  mutate(
    spline_temp_df5 = get_spline_pred(., "average_temperature", df_val = 5),
    spline_dewp_df5 = get_spline_pred(., "average_dewpoint", df_val = 5)
  ) %>%
  ungroup()

# Produce plot for EWR (Example)
plot_smoother(weather_weekly, "spline_temp_df5", "average_temperature", "Average Temperature (Natural Sp
```




```
# 4. Rolling Median Smoother works
# I use a window size (period) of 10 weeks.


get_rolling_median_pred <- function(data, outcome, window_size = 10) {
  # This uses a backward-looking rolling median over 'window_size' weeks
  slide_dbl(
    .x = pull(data, !!sym(outcome)),
    .f = median,
    .before = window_size - 1,
    .after = 0,
    .complete = TRUE
  )
}

# Apply the Rolling Median smoother (k=10 weeks)
weather_weekly <- weather_weekly %>%
  group_by(origin) %>%
```

```
  mutate(
    kernel_temp_r10 = get_rolling_median_pred(., "average_temperature", window_size = 10),
    kernel_dewp_r10 = get_rolling_median_pred(., "average_dewpoint", window_size = 10)
  ) %>%
  ungroup()

# Produce plot for EWR
plot_smoother(weather_weekly, "kernel_temp_r10", "average_temperature", "Average Temperature (Rolling M
```

```
## Warning: Removed 9 rows containing missing values or values outside the scale range
## (`geom_line()`).
```