W271 Assignment 7

Emanuel Mejía

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

library(lubridate)

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

library(sandwich)
library(lmtest)

library(nycflights13)
library(blsR)
library(gamlr)
```

```r
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?

```r
# Creating dataset
# 1 outcome - 10 input variables
d <- data.frame(
  matrix(rnorm(1100, mean = 0, sd = 1) , nrow = 100)
)

colnames(d) <- c('y', 'x1', 'x2',
                 'x3', 'x4', 'x5',
                 'x6','x7','x8',
                 'x9','x10')
```

```r
# Fitting 11 models
model0 <- glm(
  formula = y ~ 1,
  data = d)

model1 <- lm(
  formula = y ~ x1,
  data = d)

model2 <- lm(
  formula = y ~ x1+x2,
  data = d)

model3 <- lm(
  formula = y ~ x1+x2+x3,
  data = d)

model4 <- lm(
  formula = y ~ x1+x2+x3+x4,
  data = d)

model5 <- lm(
  formula = y ~ x1+x2+x3+x4+x5,
  data = d)

model6 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6,
  data = d)

model7 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7,
  data = d)

model8 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7+x8,
  data = d)

model9 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9,
  data = d)

model10 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,
  data = d)
```

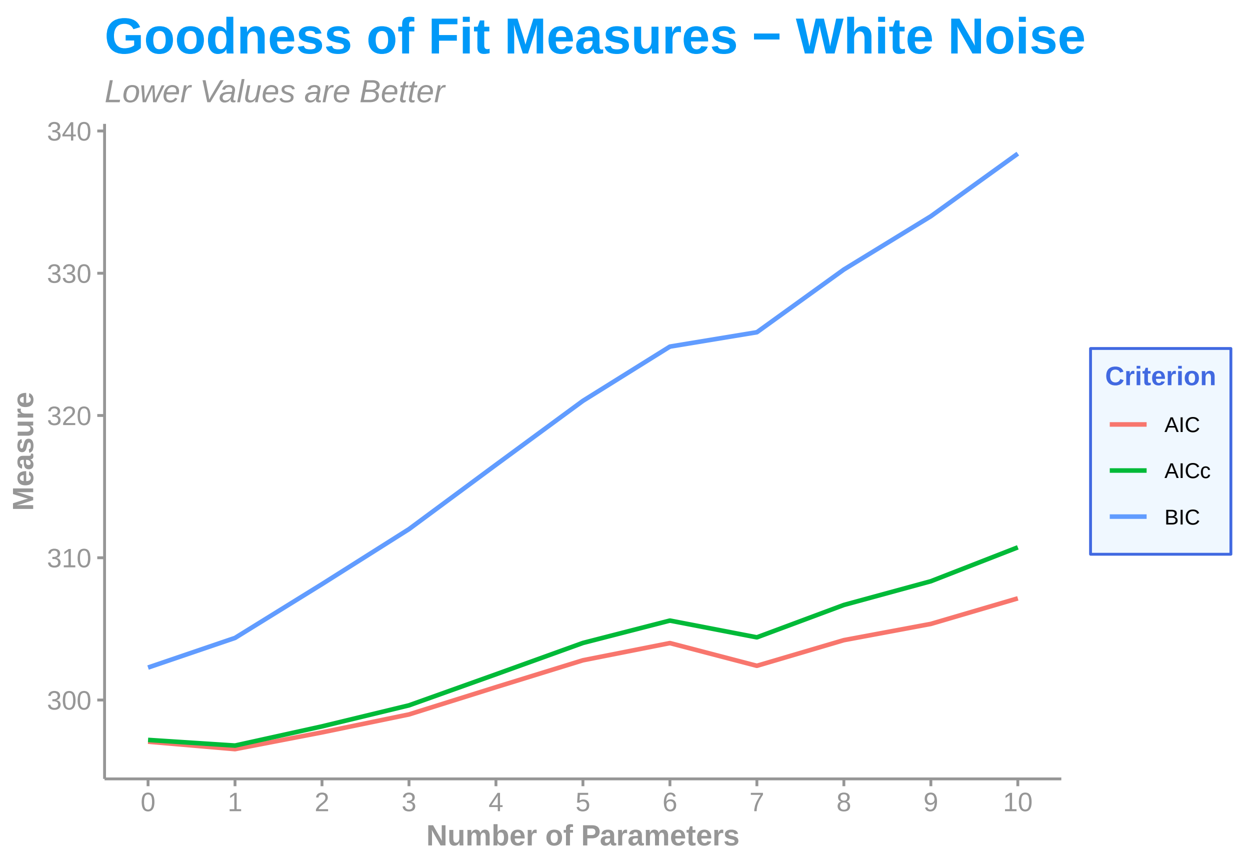
```r
# Creating results tibble for each model
results_data <- data.frame(matrix(ncol = 4, nrow = 0))
colnames(results_data) <- c('Params','AIC', 'AICc', 'BIC')

for (i in 0:10) {
  model <- eval(parse(text = paste0('model', i, sep = "")))
  results_data[i+1,] <- c(i, AIC(model), AICc(model), BIC(model))
}

results_data <- results_data %>%
 pivot_longer(cols = c('AIC', 'AICc', 'BIC'))

results_data
```

```
## # A tibble: 33 x 3
##    Params name  value
##     <dbl> <chr> <dbl>
##  1      0 AIC    297.
##  2      0 AICc   297.
##  3      0 BIC    302.
##  4      1 AIC    297.
##  5      1 AICc   297.
##  6      1 BIC    304.
##  7      2 AIC    298.
##  8      2 AICc   298.
##  9      2 BIC    308.
## 10      3 AIC    299.
## # i 23 more rows
```

```r
# Plotting the 3 measures
msr_plot <- ggplot(
  results_data,
  aes(
    x = Params,
    y = value
    )
  ) +
  geom_line(
    aes(color = name),
    size = 0.8
    ) +
  labs(
    title = "Goodness of Fit Measures - White Noise",
    subtitle = "Lower Values are Better",
    x = "Number of Parameters",
    y = "Measure",
    color = "Criterion",
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    legend.title = element_text(color = "royalblue",
                                size = 10,
                                face = "bold"),
    legend.background = element_rect(fill ="aliceblue",
                                     color ="royalblue"),
    legend.text=element_text(size=8)
    ) + scale_x_continuous(breaks=seq(0, 10, 1))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
msr_plot
```



What do you note?

We can notice that when using more parameters we decrease the model's performance when comparing it with less explaining params. Moreover, the higher the number of parameters, the bigger the gap between BIC and AIC, even its corrected and more stringent version (AICc), meaning that BIC has a higher penalization for using more explaining parameters to model the data.

(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.

```
# Creating related data
d <- transform(
  d,
  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
)
```

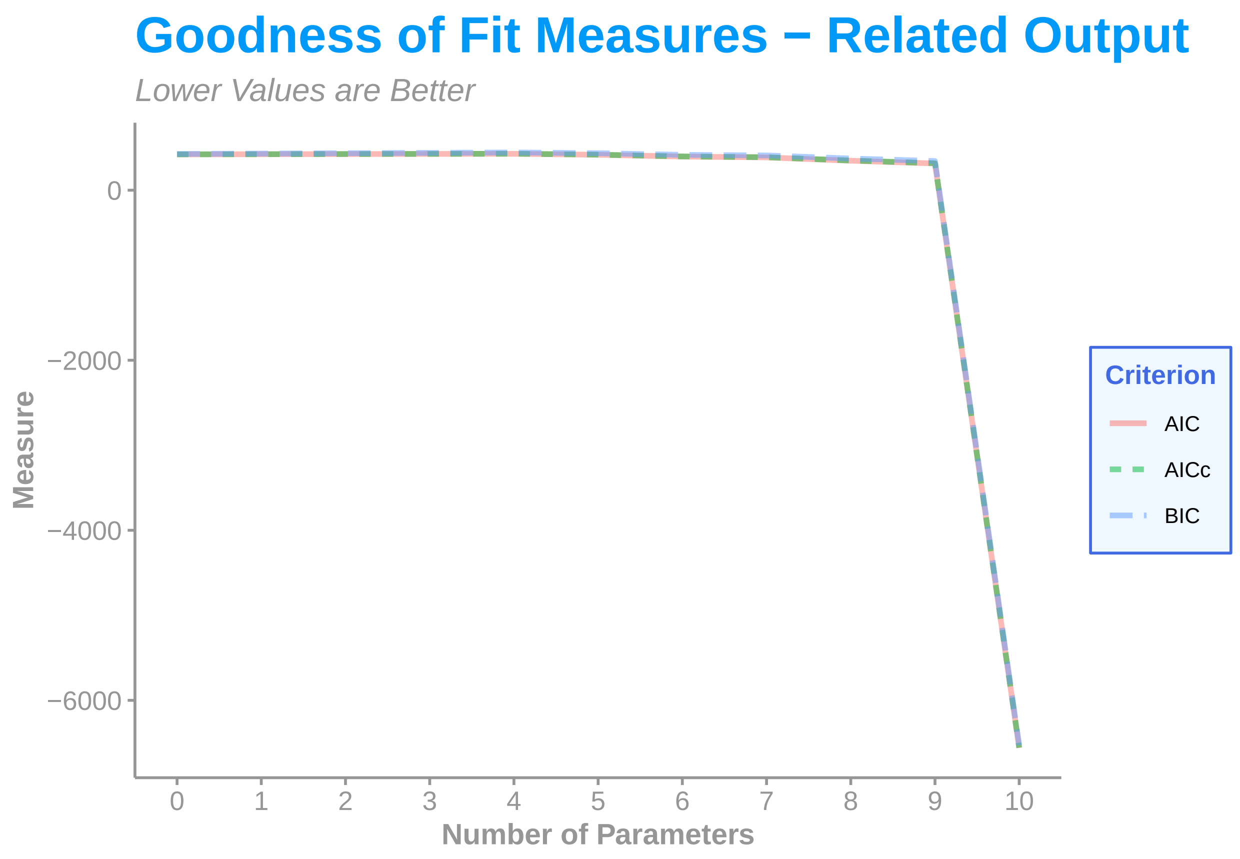
```
# Fitting 11 models
model0 <- glm(
  formula = y ~ 1,
  data = d)

model1 <- lm(
  formula = y ~ x1,
  data = d)

model2 <- lm(
  formula = y ~ x1+x2,
  data = d)

model3 <- lm(
  formula = y ~ x1+x2+x3,
  data = d)

model4 <- lm(
  formula = y ~ x1+x2+x3+x4,
  data = d)

model5 <- lm(
  formula = y ~ x1+x2+x3+x4+x5,
  data = d)

model6 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6,
  data = d)

model7 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7,
  data = d)

model8 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7+x8,
  data = d)

model9 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9,
  data = d)

model10 <- lm(
  formula = y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,
  data = d)
```

```
# Creating results tibble for each model
results_data <- data.frame(matrix(ncol = 4, nrow = 0))
colnames(results_data) <- c('Params','AIC', 'AICc', 'BIC')
```

```r
for (i in 0:10) {
  model <- eval(parse(text = paste0('model', i, sep = "")))
  results_data[i+1,] <- c(i, AIC(model), AICc(model), BIC(model))
}

results_data <- results_data %>%
 pivot_longer(cols = c('AIC', 'AICc', 'BIC'))

results_data
```

```
## # A tibble: 33 x 3
##    Params name  value
##     <dbl> <chr> <dbl>
##  1      0 AIC    421.
##  2      0 AICc   422.
##  3      0 BIC    427.
##  4      1 AIC    423.
##  5      1 AICc   423.
##  6      1 BIC    431.
##  7      2 AIC    425.
##  8      2 AICc   425.
##  9      2 BIC    435.
## 10      3 AIC    427.
## # i 23 more rows
```

```r
# Plotting the 3 measures
msr_plot <- ggplot(
  results_data,
  aes(
    x = Params,
    y = value
    )
  ) +
  geom_line(
    aes(color = name,
        linetype = name),
    size = 1,
    alpha = 0.5
    ) +
  labs(
    title = "Goodness of Fit Measures - Related Output",
    subtitle = "Lower Values are Better",
    x = "Number of Parameters",
    y = "Measure",
    color = "Criterion",
    linetype = "Criterion"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                  size = 12,
                                  face = "italic"),
    axis.title = element_text(color = "#969696",
                               size = 11,
                               face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    legend.title = element_text(color = "royalblue",
                                 size = 10,
                                 face = "bold"),
    legend.background = element_rect(fill ="aliceblue",
                                      color ="royalblue"),
    legend.text=element_text(size=8)
    ) + scale_x_continuous(breaks=seq(0, 10, 1))

msr_plot
```



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

For this specific plot we used a different style for each Criterion to demonstrate that all 3 perform similarly, slowly decreasing as we add variables even with the penalization for adding more explaining parameters, and when we actually use the 10 variables related to the output variable it drops dramatically, meaning that the model with all the variables performs significantly better than the others.

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
```

```
# Cleaning weather data
weather <- distinct(weather, origin, year, month, day, hour, .keep_all= TRUE)

# Creating weather tsibble
weather <- weather %>%
  mutate(time_index = make_datetime(year, month, day, hour)) %>%
  as_tsibble(index = time_index, key = origin)

# Creating weekly weather data
weather_wk <- weather %>% group_by_key() %>%
index_by(week_index = ~ yearweek(.)) %>%
summarise(
  average_temperature = mean(temp),
  average_dewpoint = mean(dewp)
)

weather_wk
```

```
## # 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.

```r
weather_wk <- weather_wk %>%
  mutate(numeric_week = as.numeric(week_index) - as.numeric(min(weather_wk$week_index)))
weather_wk
```

```
## # A tsibble: 159 x 5 [1W]
## # Key:       origin [3]
##    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
##  3 EWR      2013 W03                39.6            26.5             2
##  4 EWR      2013 W04                21.3             3.77            3
##  5 EWR      2013 W05                36.0            25.7             4
##  6 EWR      2013 W06                30.0            17.6             5
##  7 EWR      2013 W07                37.8            25.3             6
##  8 EWR      2013 W08                34.2            21.3             7
##  9 EWR      2013 W09                39.2            27.8             8
## 10 EWR      2013 W10                38.7            26.8             9
## # i 149 more rows
```

```r
# Creating a list of models for
# average temperature and average dewpoint
mod_temp = list()
mod_dewp = list()
```

```
for (i in 1:10) {
  mod_temp[[i]] <- lm(
  formula = average_temperature ~ poly(numeric_week, degree = i),
  data = weather_wk)

  mod_dewp[[i]] <- lm(
  formula = average_dewpoint ~ poly(numeric_week, degree = i),
  data = weather_wk)
}
```

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

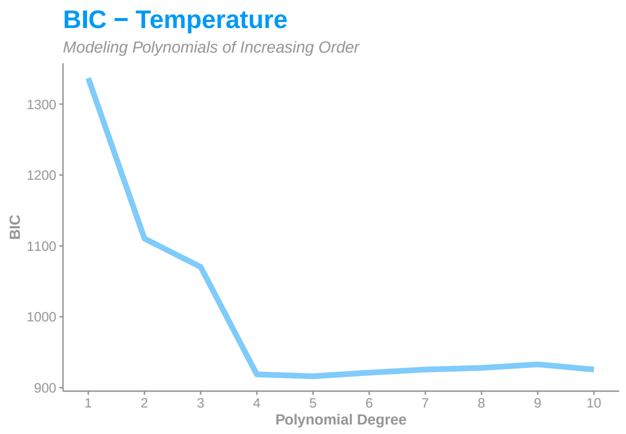
For each of the outcomes – `average_temperature` at the weekly level, and `average_dewpoint` at the weekly level – make an assessment based on either AIC or BIC for why one polynomial degree produces the best fitting model. In doing so, describe why you have chosen to use either AIC or BIC, what the 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.

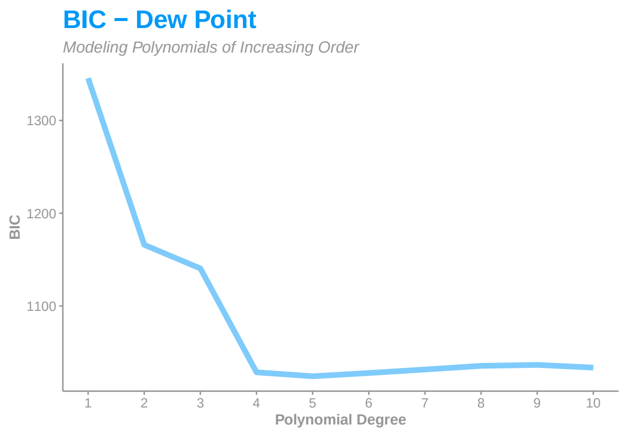
```
meas_temp <- data.frame(matrix(ncol = 2, nrow = 0))
colnames(meas_temp) <- c('Degree','BIC')

for (i in 1:10) {
  meas_temp[i,] <- c(i, BIC(mod_temp[[i]]))
}

temp_bic <- ggplot(
  meas_temp,
  aes(
    x = Degree,
    y = BIC
    )
  ) +
  geom_line(
    size = 2,
    color = "#0099F8",
    alpha = 0.5
    ) +
  labs(
    title = "BIC - Temperature",
    subtitle = "Modeling Polynomials of Increasing Order",
    x = "Polynomial Degree",
    y = "BIC"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
```

```r
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    ) + scale_x_continuous(breaks=seq(1, 10, 1))
```

```r
meas_dewp <- data.frame(matrix(ncol = 2, nrow = 0))
colnames(meas_dewp) <- c('Degree','BIC')

for (i in 1:10) {
  meas_dewp[i,] <- c(i, BIC(mod_dewp[[i]]))
}

dewp_bic <- ggplot(
  meas_dewp,
  aes(
    x = Degree,
    y = BIC
    )
  ) +
  geom_line(
    size = 2,
    color = "#0099F8",
    alpha = 0.5
    ) +
  labs(
    title = "BIC - Dew Point",
    subtitle = "Modeling Polynomials of Increasing Order",
    x = "Polynomial Degree",
    y = "BIC"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    ) + scale_x_continuous(breaks=seq(1, 10, 1))
```

```r
temp_bic
dewp_bic
```





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)`.

We used BIC instead of AIC or AICc because we know it is more stringent and has a higher penalization for using more explanatory variables. This is something desirable because even when a model's prediction could be closest to the real values, we could be losing the power to generalize a model to new data due to overfitting. In this specific case we're talking about the same explanatory variable, just using a higher exponent degree so, besides the risk of overfitting it becomes harder to explain as we keep adding degrees to our polynomial.

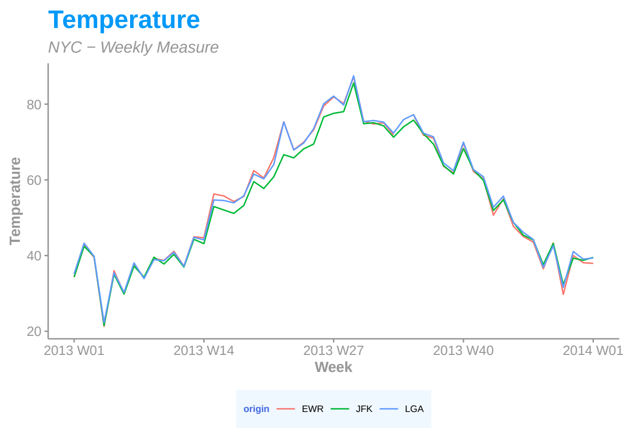
This Criterion is basically doing the following computation:

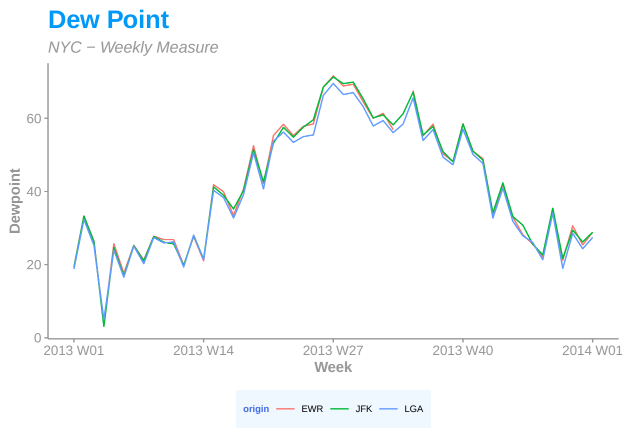
$$BIC = log(\hat{\sigma}_k^2) + \frac{n+k}{n-k-2}$$

where k is the number of parameters in the model (or in this case degrees of the polynomial), n denotes the sample size, and the estimated variance is $\hat{\sigma}_k^2 = \frac{SSE_k}{n}$.

Meaning that BIC is a measure of the difference between predicted and real values (left side of the equation), and a "penalization" term (right side of the equation) which grows as we keep adding parameters, and therefore increasing the whole Criterion measure, which we know that makes a model less efficient to explain the data.

The 4^{th}-degree polynomial (or even the 5^{th}) is the one producing the best result. And even when it is hard to explain what we mean by `week number` elevated to certain degree, it is understandable that the relationship between temperature/dewpoint and week number is not linear, nor always following the same trend, but we know that it exists: We can see in the time series graphs below temperature and dewpoint usually start low, increase towards the middle of the year, and then start decreasing again, which by the way is a well known behavior, so we need a higher degree polynomial to "explain" these relationships.





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:

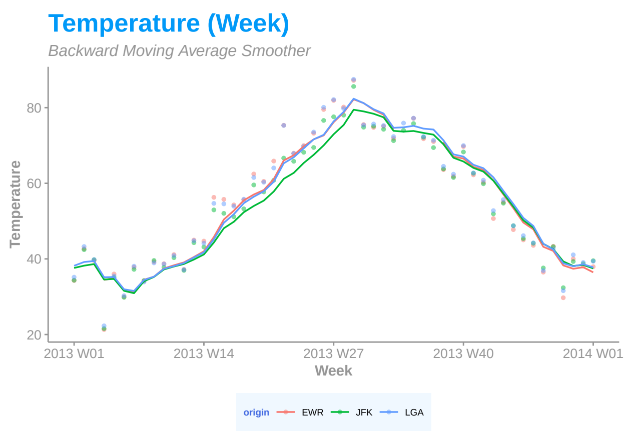
- Fit the smoother **within** each origin. That is, fit the smoother for JFK separately from LaGuardia and Newark.

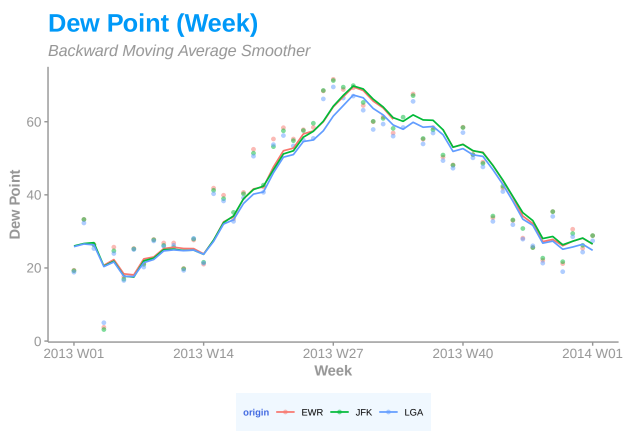
- Attach the values that are produced by the smoother onto the `weekly_weather` dataframe.
- Produce a plot that shows the original data as `geom_point()`, and the smoother's predictions as `geom_line()`.

- 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 hyperparameters to these algorithms to produce their most effective output.

```r
# creating a graphing function for smoothers
smooth_graph <- function(
    data = NULL, # Time Series to be used
    y.ax = NULL,    # y-axis variable
    y.smooth = NULL,   # smoothing column
    ttl = "Weekly Measure", # Plot Title
    subttl = "NYC - Weekly Measure", # Plot Subtitle
    y.lab = "y" # y-label
    ) {

  temp_plot <- ggplot(
  data,
  aes(
    x = week_index
    )
  ) +
  geom_point(
    aes(y = y.ax,
        color = origin),
    size = 1,
    alpha = 0.5
    ) +
  geom_line(
    aes(y = y.smooth,
        color = origin),
    size = 0.6
    ) +
  labs(
    title = ttl,
    subtitle = subttl,
    x = "Week",
    y = y.lab
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    legend.title = element_text(color = "royalblue",
                                size = 7,
                                face = "bold"),
    legend.background = element_rect(fill ="aliceblue"),
    legend.text=element_text(size=7),
    legend.position="bottom"
    )

  graph <- temp_plot

  return(graph)
}
```

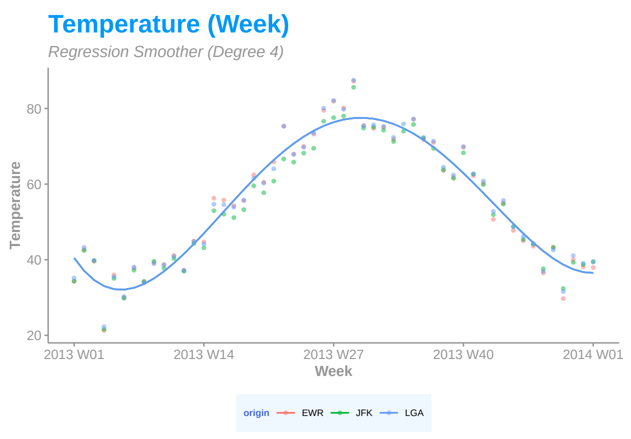
```
weather_wk <- weather_wk %>%
   mutate(
    mov_avg_temp = slider::slide_dbl(average_temperature, mean,
                                     .before = 3, .after = 0, .complete = TRUE),
    mov_avg_dewp = slider::slide_dbl(average_dewpoint, mean,
                                     .before = 3, .after = 0, .complete = TRUE),
  )

temp_mov <- smooth_graph(
  data = weather_wk,
  y.ax = weather_wk$average_temperature,
  y.smooth = weather_wk$mov_avg_temp,
  ttl = "Temperature (Week)",
  subttl = "Backward Moving Average Smoother",
  y.lab = "Temperature"
)

dewp_mov <- smooth_graph(
  data = weather_wk,
  y.ax = weather_wk$average_dewpoint,
  y.smooth = weather_wk$mov_avg_dewp,
  ttl = "Dew Point (Week)",
  subttl = "Backward Moving Average Smoother",
  y.lab = "Dew Point"
)

temp_mov
dewp_mov
```

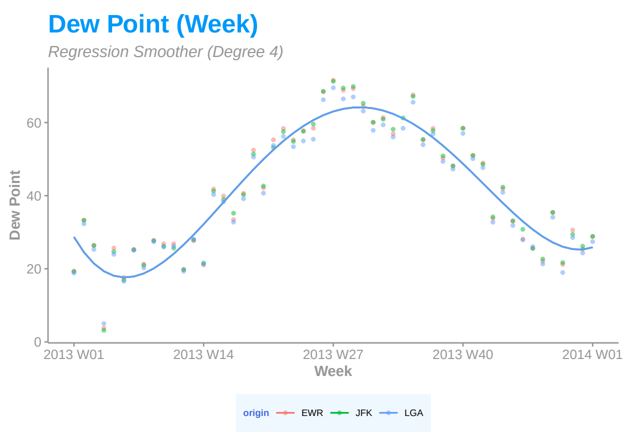




For the Backward Moving Average we'll only look at 3 weeks back (plus the current one) to produce the average. Although it doesn't really accomplish the smoothing job but, on the other hand, increasing the number of weeks produces lines that separate a lot from the actual datapoints because of the retarded effect.

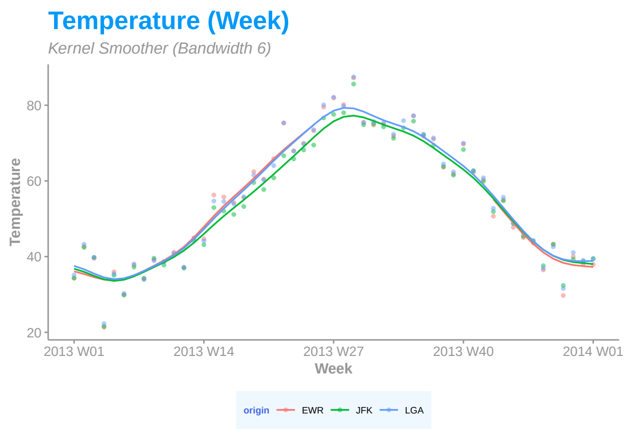
```
chosen_temp <- mod_temp[[4]]
chosen_dewp <- mod_dewp[[4]]

fit_temp <- c(fitted(chosen_temp))
fit_temp <- append(fit_temp, NA, after=33)

fit_dewp <- c(fitted(chosen_dewp))
fit_dewp <- append(fit_dewp, NA, after=33)

weather_wk$reg_smt_temp <- fit_temp
weather_wk$reg_smt_dewp <- fit_dewp

temp_reg <- smooth_graph(
  data = weather_wk,
  y.ax = weather_wk$average_temperature,
  y.smooth = weather_wk$reg_smt_temp,
  ttl = "Temperature (Week)",
  subttl = "Regression Smoother (Degree 4)",
  y.lab = "Temperature"
)

dewp_reg <- smooth_graph(
  data = weather_wk,
  y.ax = weather_wk$average_dewpoint,
  y.smooth = weather_wk$reg_smt_dewp,
  ttl = "Dew Point (Week)",
  subttl = "Regression Smoother (Degree 4)",
  y.lab = "Dew Point"
)

temp_reg
dewp_reg
```

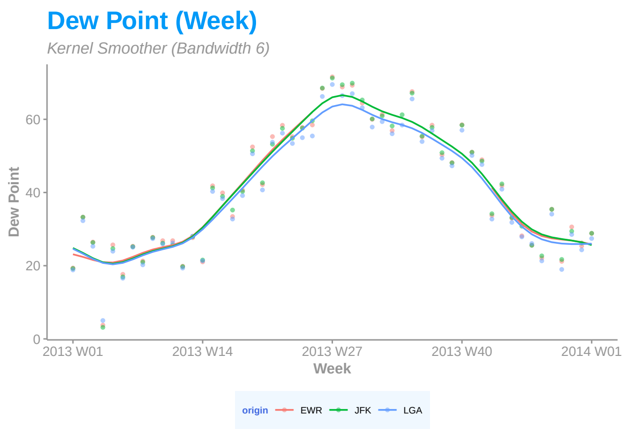




As we saw earlier, the chosen degree for our polynomial regression smoother is 4. As we can see in the plot, it seems to do a good job explaining the general trend while smoothing the datapoints as much as possible.

```r
weather_wk$ksmt_temp <- ksmooth(
  time(weather_wk$average_temperature),
  weather_wk$average_temperature,
  "normal",
  bandwidth = 6)$y

weather_wk$ksmt_dewp <- ksmooth(
  time(weather_wk$average_dewpoint),
  weather_wk$average_dewpoint,
  "normal",
  bandwidth = 6)$y

temp_ksm <- smooth_graph(
  data = weather_wk,
  y.ax = weather_wk$average_temperature,
  y.smooth = weather_wk$ksmt_temp,
  ttl = "Temperature (Week)",
  subttl = "Kernel Smoother (Bandwidth 6)",
  y.lab = "Temperature"
)

dewp_ksm <- smooth_graph(
  data = weather_wk,
  y.ax = weather_wk$average_dewpoint,
  y.smooth = weather_wk$ksmt_dewp,
  ttl = "Dew Point (Week)",
  subttl = "Kernel Smoother (Bandwidth 6)",
  y.lab = "Dew Point"
)

temp_ksm
dewp_ksm
```





Using a bandwidth of 6 for the kernel smoother seems to do a good job at smoothing the data and explaining the general trend while keeping enough information.

The final tsibble object we obtained is the following:

```
weather_wk
```

```
## # A tsibble: 159 x 11 [1W]
## # Key:       origin [3]
##    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
##  3 EWR      2013 W03                39.6            26.5             2
##  4 EWR      2013 W04                21.3             3.77            3
##  5 EWR      2013 W05                36.0            25.7             4
##  6 EWR      2013 W06                30.0            17.6             5
##  7 EWR      2013 W07                37.8            25.3             6
##  8 EWR      2013 W08                34.2            21.3             7
##  9 EWR      2013 W09                39.2            27.8             8
## 10 EWR      2013 W10                38.7            26.8             9
## # i 149 more rows
## # i 6 more variables: mov_avg_temp <dbl>, mov_avg_dewp <dbl>,
## #   reg_smt_temp <dbl>, reg_smt_dewp <dbl>, ksmt_temp <dbl>, ksmt_dewp <dbl>
```