Smoothing Temperature Data

Steve Bischoff

2023-12-28

Contents

1. Introduction

Smoothing methods are a popular way to identify the trend(s) in noisy data without having to construct a model of the data-generating process. Temperature data are a particularly appealing application of these methods. Figure 1 shows daily high temperatures in Austin, Texas in the late 90s.[1]

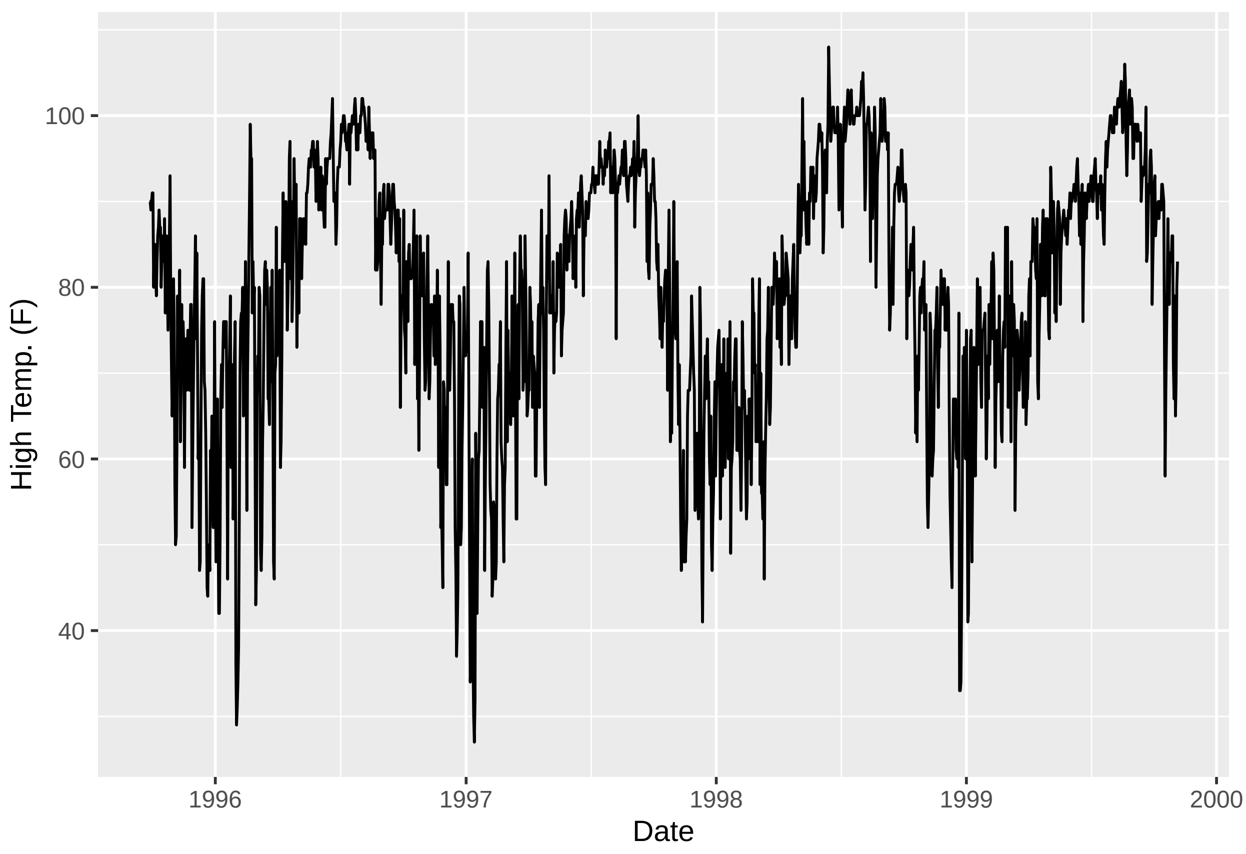

Figure 1: Austin daily high temperatures

The graph is visually unappealing due to daily temperature fluctuations. This noise also makes it difficult to generalize and identify trends. As just one example: the winter of 1996-1997 clearly has the lowest lows of these years, but was that cold spell an aberration or did it truly have the coldest winter?

This project investigates the application of automated smoothing methods to temperature data. Temperature data has several features that complicate the task. These features are detailed in Section 5, and potential solutions are discussed in subsequent sections. I find that *Super-Smoothing with K-Block Cross-Validation* does a pretty good job of smoothing the data where other methods fail.

1.1 Data

The data used in this project comes from the National Oceanic and Atmospheric Administration (NOAA).[1] I used the search tool to find stations that have had 100% Daily Summary coverage for many decades.

I wanted to investigate temperature smoothing across a wide variety of climates, so I requested data from 4 different cities. Table 1 provides a summary.

Table 1: Data source descriptions

City	Station ID	First Available
Austin	USW00013958	1938-06-01
San Diego	USW00023188	1939-07-01
Minneapolis	USW00014922	1938-04-09
Baltimore	USW00093721	1939-07-01

To make computation more manageable, I truncated each data set to 1993 - 2022. Figures 2 and 3 display time series and box plots of the truncated data sets.

2. Kernel Smoothing

Overview: Kernel smoothers apply a weighted average to the data. They take a *bandwidth* parameter that controls how smooth the output curve is.

Smoothing methods try to find a good *smoothing function* that cuts through the noise to give the underlying trend. The value of a smoothing function at a point is generally calculated as some type of weighted average of the values at surrounding points. For instance, the temperature given by the smoothing function on Oct. 1, 1996 is a weighted average of the actual temperatures on and around that date.

Perhaps the simplest smoother is the *constant-span running mean* ([2] p. 366). This smoother is given a parameter k that defines a window size for taking the mean. Suppose we apply the constant-span running mean with $k = 7$ to the temperature data. To calculate the smoothing function on Oct. 1, 1996, we take the mean temperature of the $\frac{k-1}{2} = 3$ days before, the day itself, and the $\frac{k-1}{2} = 3$ days after: in other words, we calculate the mean high temperature from Sep. 28 to Oct. 4.

While its simplicity is attractive, the constant-span running mean has some problems: the window parameter defines a sharp cutoff for the points it considers, and points at the edge of the window are given as much weight as points in the middle.

The smoothing method used in this project is known as *kernel smoothing* and tries to fix these problems. A *kernel* is a function that allows kernel smoothers to take truly *weighted* averages. When calculating the smooth function at a given point, the kernel weights observations based on how close they are to that point:

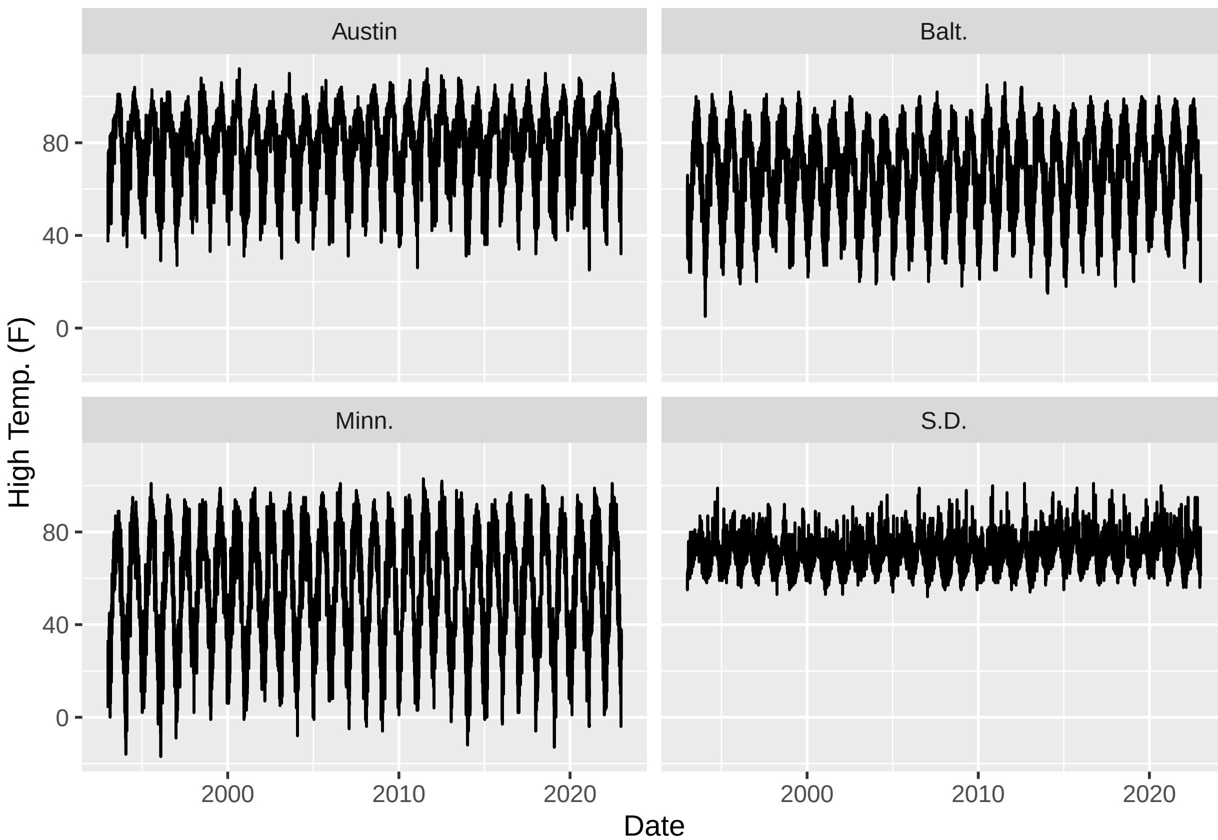

Figure 2: Daily high temperatures 1993-2002

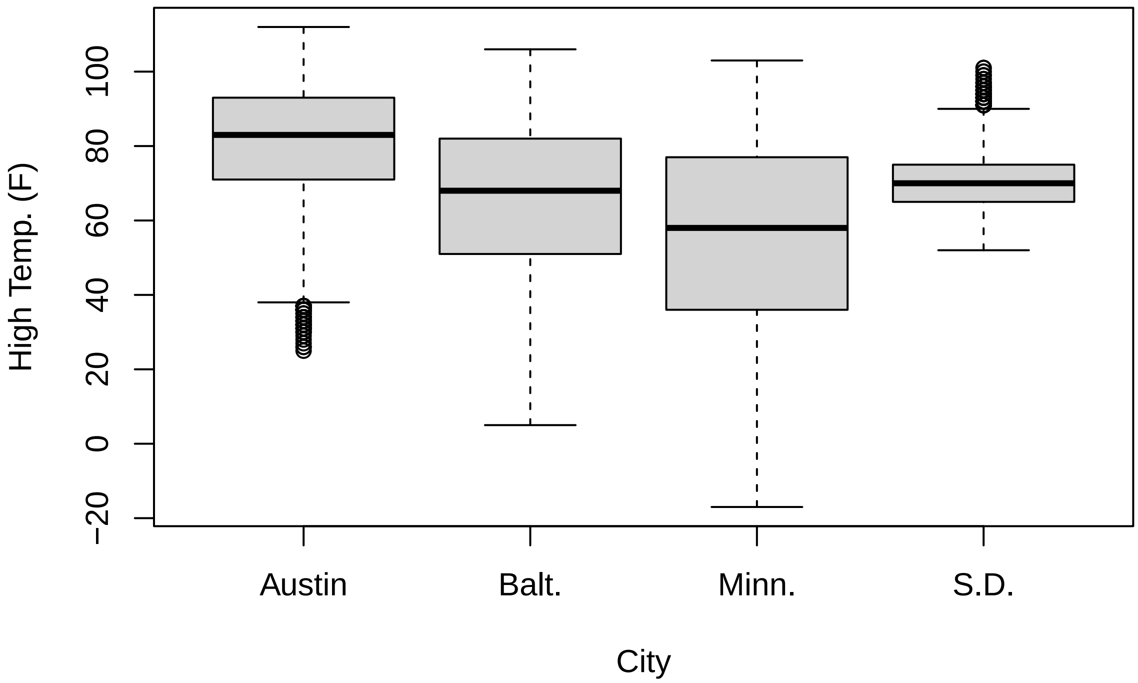

Figure 3: Daily high boxplots by city

the closer the observation, the higher the weight. This project uses a *normal kernel* that assigns weights around a given point in the bell shape of a normal probability distribution.

The effect of distance on weighting in controlled by the all-important *bandwidth* parameter h. If h is set to a low value, observations around the given point are weighted very highly, and weights steeply decrease as distance from the point increases. Conversely, if h is set to a high value, close points are given relatively less weight, and weights decrease less steeply with distance.

Suppose again that we're calculating the smoothed temperature on Oct. 1, 1996. Figure 4 shows how the temperatures on days around this date are weighted given different bandwidth parameters:

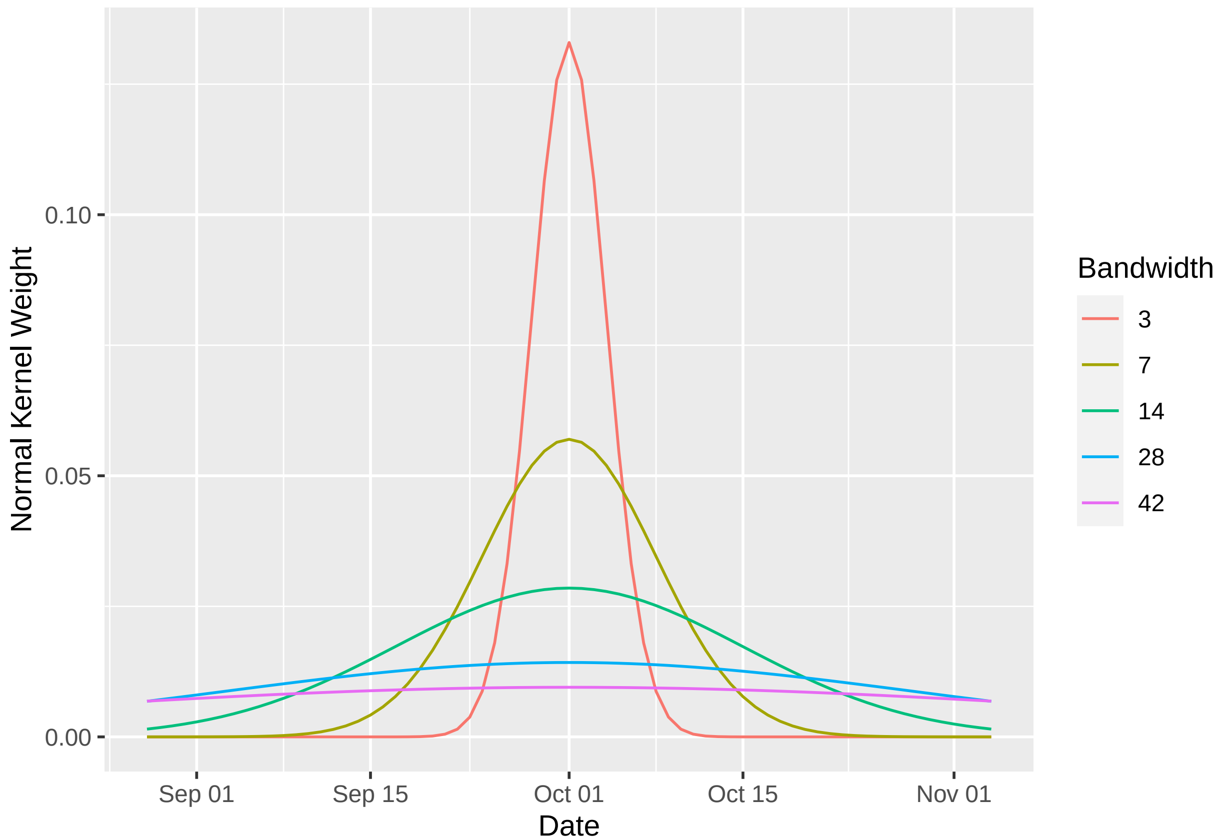

Figure 4: Normal Kernel Weights by Bandwidth around 1996/10/01

Given a bandwidth $h = 3$, the smoothed temperature on Oct. 1 mostly depends on the temperatures in a small window around that date. Given a bandwidth $h = 28$ or $h = 42$, the smoothed temperature on Oct. 1 is giving non-negligible weight to temperatures in August and November.

As seen in Figure 5, the bandwidth has a significant impact on the smooth:

The kernel smoother with $h = 3$ overfits the data, following its fluctuations too closely and failing to provide much of a smooth at all. The kernel smoother with $h = 42$ underfits the data, failing to fully capture the seasonal extremes in summer and winter.

3. Bandwidth Selection

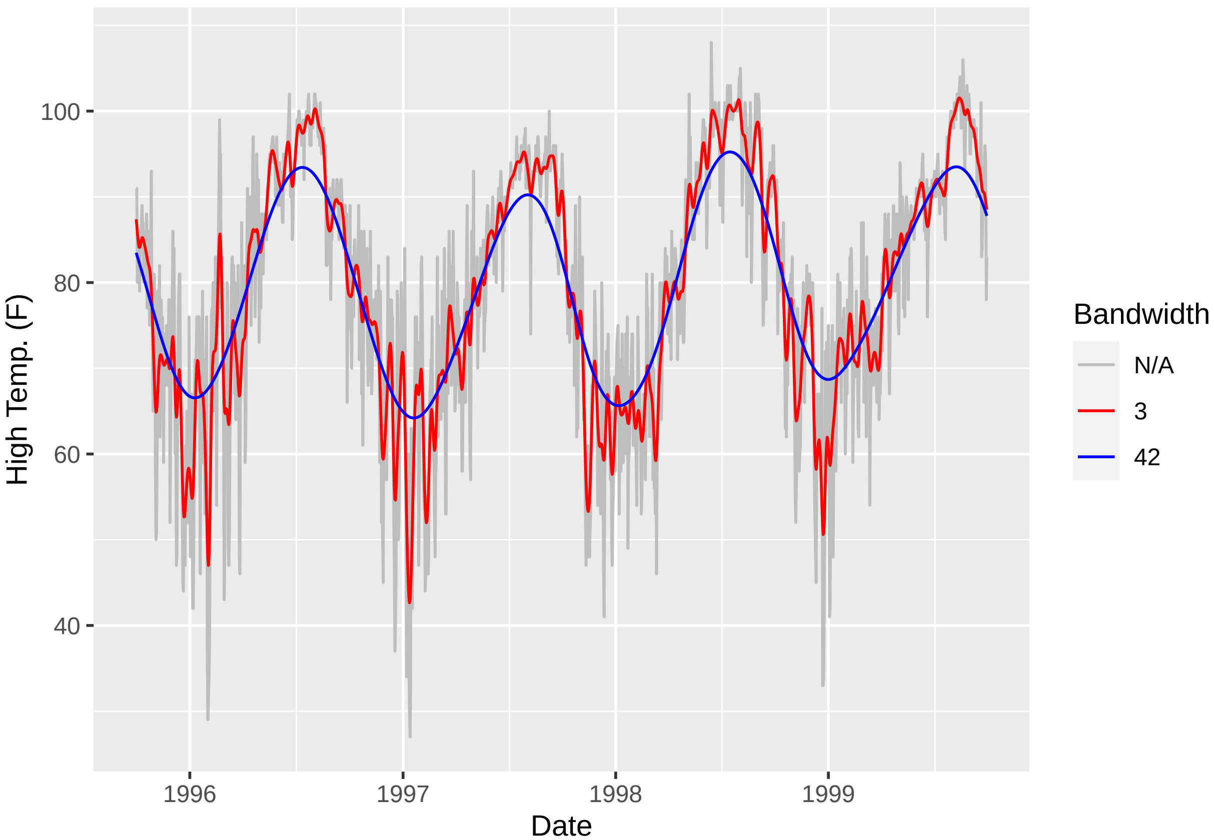

Figure 5: Austin Daily Highs with two smooths

Overview: Automated bandwidth selection methods scale better, so they're preferred to manual selection when we need to smooth many data sets and / or large data sets.

Selecting an appropriate bandwidth is crucially important for getting a good smooth of the data (it's generally viewed as much more important than the choice of kernel function). We can divide methods of bandwidth selection into two broad categories, manual and automated.

3.1 Manual Bandwidth Selection

A common way to manually select a bandwidth is by trial and error with visual inspection of the smooths given by different bandwidths. We plot various smooths over the data and see how well they capture the overall trend(s).

Figure 6 displays plots of a normal kernel smooth with $h = 14$ for all 4 cities:

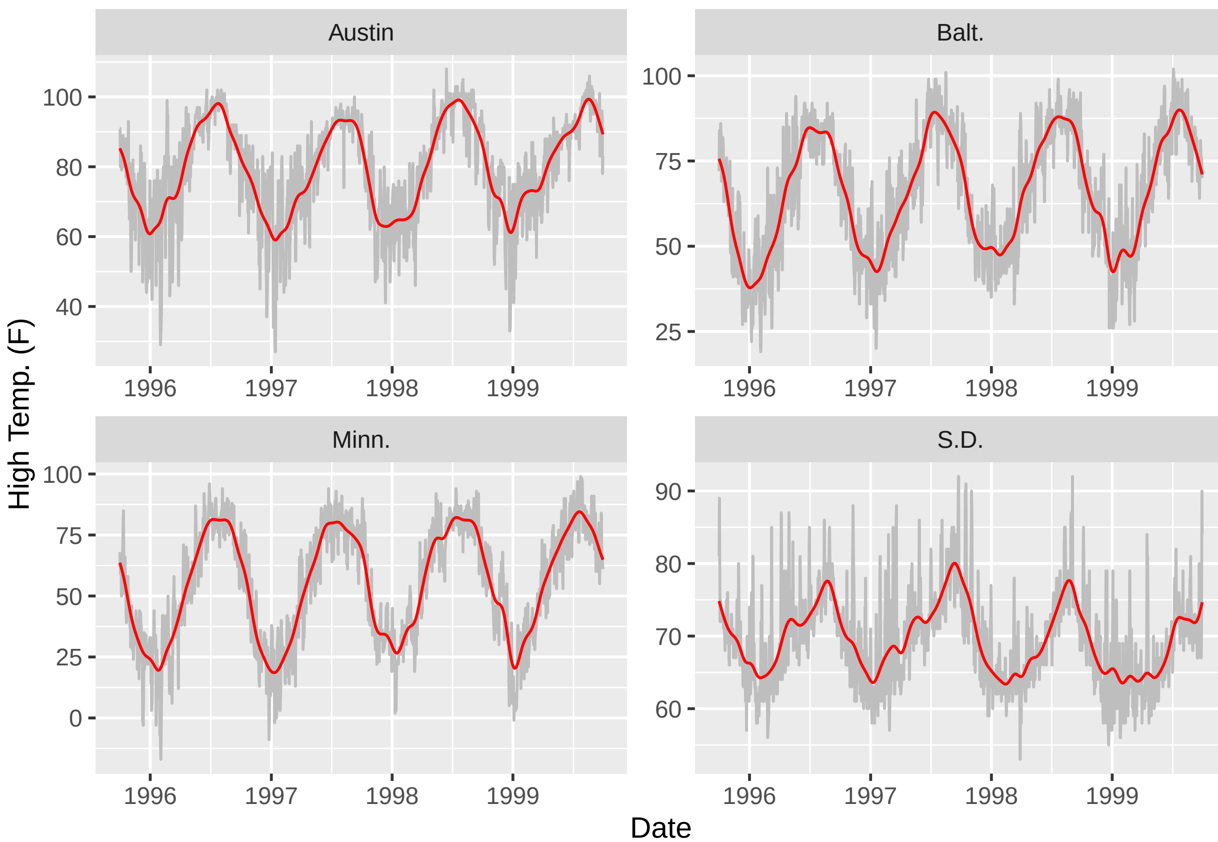

Figure 6: Daily highs with normal kernel smooth (h=14)

By visual inspection, a bandwidth $h = 14$ seems to do a pretty good job of smoothing the data, especially for Baltimore and Minnesota.

3.2 Automated Bandwidth Selection

A main problem with manual bandwidth selection is that it doesn't scale well in situations with many data sets and / or large data sets. There are reasons for this beyond the obvious time-intensiveness:

- Many data sets: Even if they're all about the same type of phenomenon, different data sets can have different properties that make different bandwidths more appropriate. We shouldn't assume that a bandwidth that works well on some of the data sets will work well on all of them. Notice above how the smooth with $h = 14$ is noticeably worse for San Diego, containing many small wiggles due to short-term temperature fluctuations. A higher bandwidth could help remove those wiggles and provide a better smooth.
- Large data sets: It's difficult to visually inspect the smooth over data sets that cover a wide "x-value" range. The above graphs only show 4 years out of 30, so we don't know how well the smooth does in other years.

Methods for automated bandwidth selection are explored in the rest of this project.

4. Leave-One-Out Cross-Validation (LOOCV)

Overview: Leave-one-out cross-validation (LOOCV) chooses a bandwidth by simulating how well different bandwidths extend to new data. It falls under the training set / testing set paradigm, with each test set having size 1. LOOCV fails to select good bandwidths for temperature data: given a set of candidate bandwidths, it selects the smallest candidate for each data set.

Given a data set of size n, leave-one-out cross-validation performs n training set / testing set splits. As indicated by the name, each LOOCV testing set contains only one data point that has been left out of the corresponding training set.

LOOCV chooses among a set of candidate bandwidths by measuring how well smoothers with those bandwidths generalize from the training sets to the test sets. Generalization ability is typically measured using some error function such as the mean-squared error (MSE) used in this project. For a given candidate bandwidth, the LOOCV algorithm iterates through the train / test splits. For a given split, it "trains" a smoother with that bandwidth on the training set and calculates the squared error when applied to the (singleton) test set.[fn.] The MSE across all splits is returned as the performance measure for that bandwidth. The candidate bandwidth with the smallest MSE is returned as the "optimal" choice.

[fn.] Fortunately, a mathematical shortcut can be applied so that each smoother doesn't actually have to be trained n times. See ([2] p. 371).

I applied LOOCV to each temperature data set, testing candidate bandwidths ranging from $h = 1$ to $h = 20$ in the integers (as we'll see, testing larger bandwidths is unnecessary).

4.1. Results

Figure 7 shows the test-set MSE at each candidate bandwidth for each data set. The test-set MSE monotonically increases with bandwidth size, so the LOOCV algorithm selects the smallest candidate ($h = 1$) for each city.

In other words, LOOCV fails to select good bandwidths for smoothing these data sets. As seen in Section 2, very small bandwidths barely smooth the data. By selecting the smallest candidate, LOOCV is simply trying to follow the data as closely as possible instead of smoothing it.

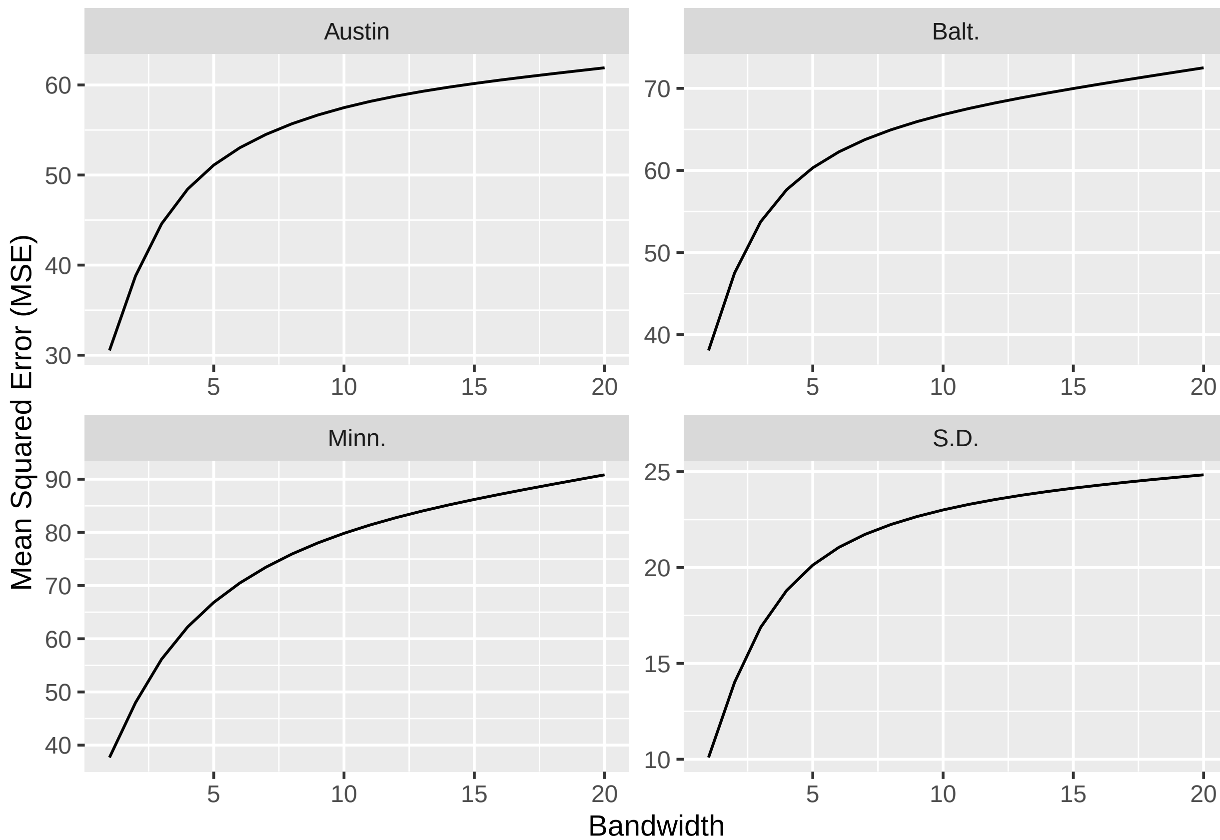

Figure 7: LOOCV MSE by bandwidth

5. Difficulties Posed By Temperature Data

Overview: Temperature data has at least two properties that pose difficulties for kernel smoothing and bandwidth selection:

- Non-Constant Variance
- Dependent Data

Understanding why LOOCV fails to smooth temperature data can help us pick better methods for automated bandwidth selection.

5.1. Non-Constant Variance

The smoothing methods examined up to now assume that the variance of the data is constant across its range. However, temperature data doesn't meet this assumption due to both long and short-term factors. Climate change is an example of a long-term factor affecting temperature variance: as the climate gets further disrupted, we see more temperature extremes and fluctuations.

The seasons are the primary short-term factor affecting temperature variance. Take another look at some of the Austin data in Figure 1. We can see from this graph (and I know from personal experience) that temperature variance is much higher in the winter than the summer. Austin winters generally fluctuate between 40F and 80F, often within the span of a couple days, while Austin summers generally stay in the upper 90s to low 100s.

As a result, different bandwidths are appropriate in different seasons. Higher bandwidths seem better in winter since they'll give smooths that are less affected by the short-term fluctuations. Relatively lower bandwidths seem better in summer since they can better capture the peak heat (usually in August).

5.2. Dependent Data

Most smoothing methods work better when deviations from the broader trend in the data are uncorrelated with each other. In the case of temperature data, this would occur if fluctuations around the broader trend were independent from one day to the next. The fluctuations would tend to balance each other out, allowing a smoother to easily cut through the middle of them.

Unfortunately, temperature data isn't like this. When temperatures deviate from the broader trend, they generally do so over the course of several days in a heat wave or cold spell. Abnormally hot days are likely to be followed by abnormally hot days; likewise for abnormally cold days.

Consider the February 1996 heat wave in Austin highlighted in Figure 8. Here, the positive deviations from the overall trend lasted about a week. Unless they have a very high bandwidth, smoothers won't be able to "cut through" this heat wave and stick to the broader trend. The prolonged spell of abnormally high temperatures will inevitably pull the smooth upwards, perhaps in dramatic fashion. Looking back at the smoother with $h = 14$ in Section 3, this is exactly what happens.

We get more evidence of the dependent nature of temperature data from the autocorrelation (ACF) plots of the residuals from the $h = 14$ smooths on each city. The ACF plots in Figure 9 display the correlations between different lags of the residuals. For each city, the residuals are positively correlated with the lag-1 and lag-2 residuals. For Minneapolis, the residuals are even positively correlated with the lag-3 residuals.

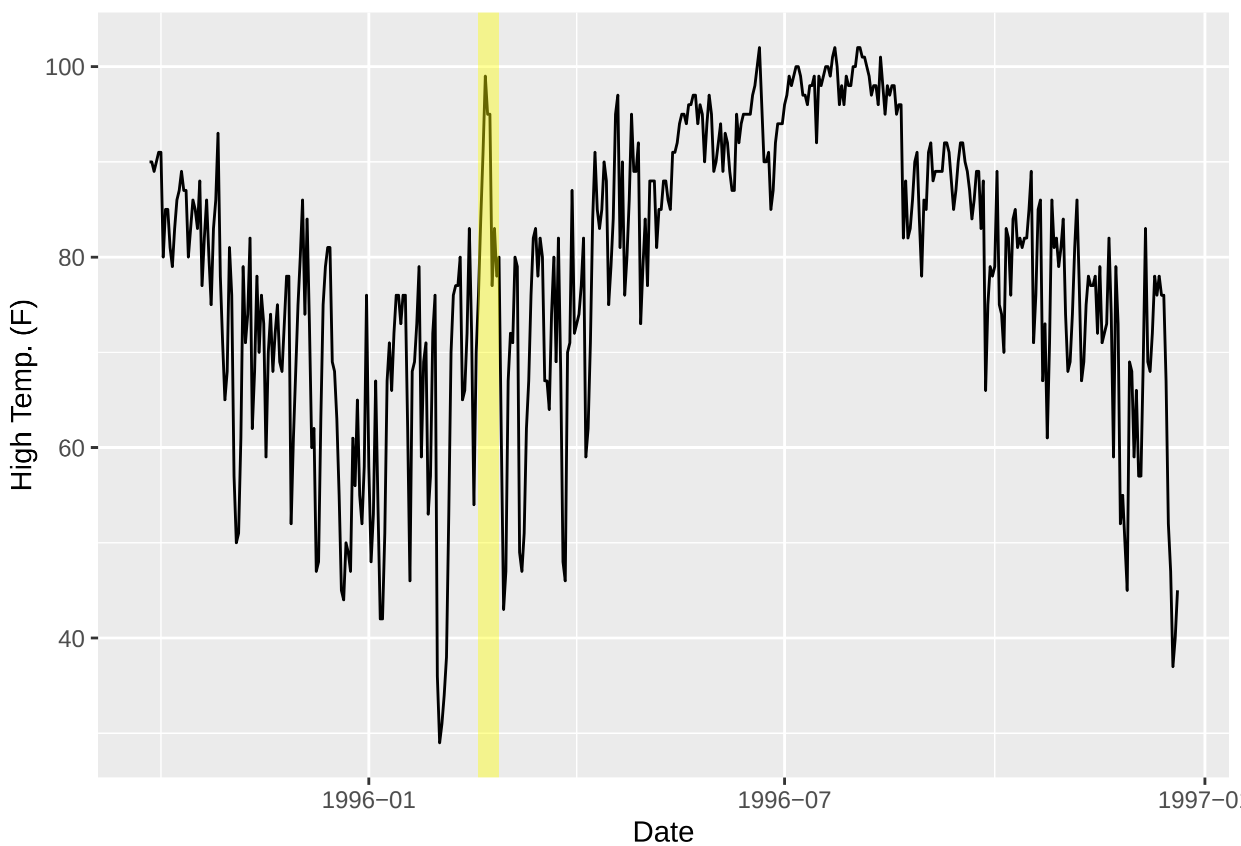

Figure 8: Austin daily highs with Feb. 1996 heat wave

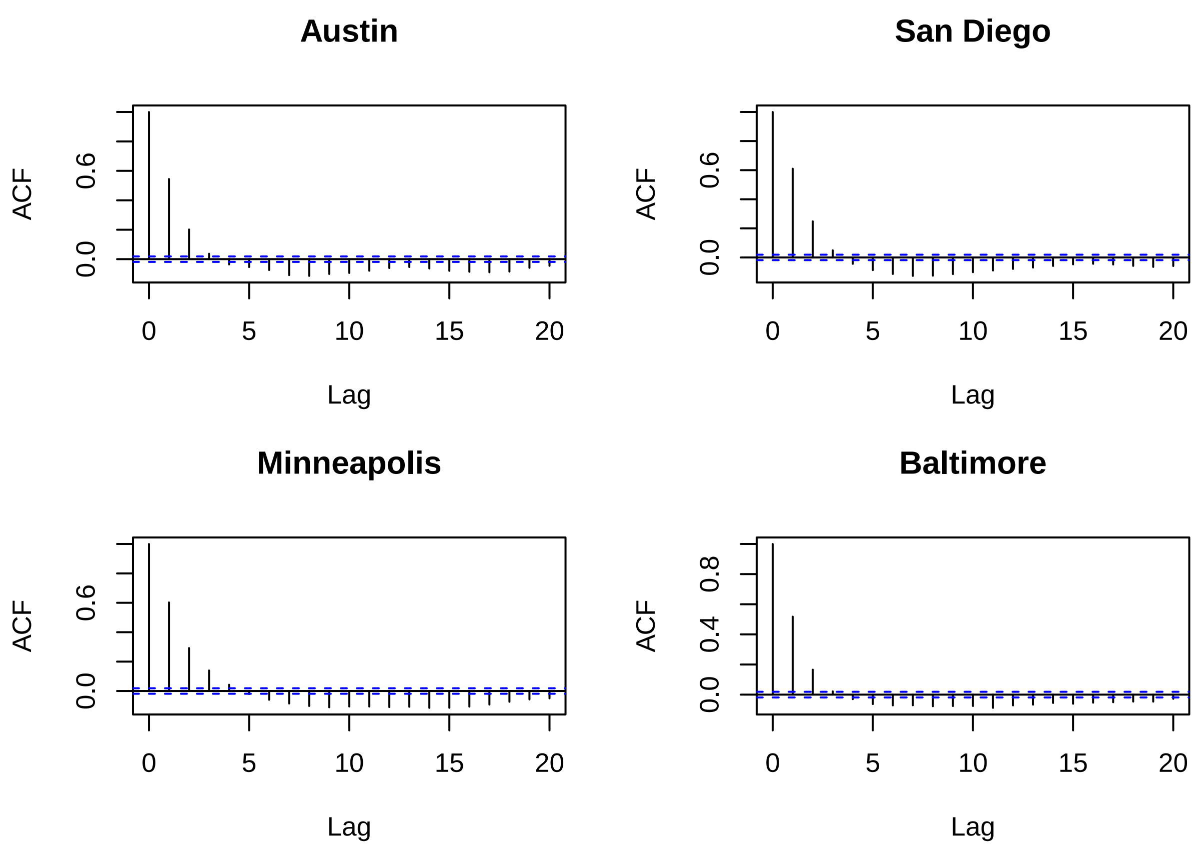

Figure 9: Autocorrelation plots for normal kernel smooth (h=14) residuals

6. Super-Smoother

Overview: The super-smoother helps with the problem of non-constant variance by selecting different "optimal" bandwidths at different points. LOOCV again performs poorly as the bandwidth selection method, choosing the lowest candidate bandwidth at most points.

Up to this point, I've only discussed smoothers that use a single bandwidth across an entire data set. These can struggle when the data have non-constant variance, as different bandwidths are appropriate in different regions.

The super-smoother tries to solve this problem by allowing the bandwidth to vary across the data.[3] Like the previous LOOCV algorithm, the super-smoother is given a set of candidate bandwidths and computes the LOOCV error at each point for each bandwidth. A second smooth is then applied to these errors. At each point in the data set, the bandwidth with the smallest smoothed LOOCV error is selected as the "optimal" bandwidth at that point. A third smooth is applied to these optimal bandwidths to ease the transitions between different variance regions. Finally, the smoothed optimal bandwidths are used to calculate the output values of the super-smoother.

Figure 10 (from [2]) shows an example non-constant variance data set with a super-smoother fit given 3 candidate bandwidths (0.05, 0.2, 0.5). Figure 11 (also from [2]) shows the optimal bandwidth selected by LOOCV at each point as well as the smooth applied to these bandwidths. Note that higher bandwidths are selected in higher-variance regions, allowing the smooth to cut through the noise, while lower bandwidths are selected in lower-variance regions, allowing the smooth to better capture the trend.

I tested the LOOCV super-smoother on each data set. After some trial-and-error, I used $H = (7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87)$ as the set of candidate bandwidths.

The decision to use a set of 11 candidate bandwidths introduced a minor complication. The paper introducing the super-smoother uses a set of 3 candidate bandwidths and recommends using the second-largest of these when applying the second and third smooths to the LOOCV errors and optimal bandwidths, respectively (call the bandwidth of these smooths h^*). This recommendation is ambiguous for a larger set of candidates: do we use the second-largest or the median bandwidth for the later smooths? I decided to test both options ($h^* = 15$ and $h^* = 47$).

6.1 Results

Unfortunately, the super-smoother with LOOCV bandwidth selection also mostly fails to give good smooths of the data. As seen in Table 2, for each city and choice of h^*, the mean of the bandwidths chosen by the super-smoother across the data set is fairly close to 7, the smallest candidate bandwidth. In other words, the LOOCV super-smoother selects the smallest possible bandwidth as "optimal" at most points in the data set. (I tested sets of candidate bandwidths that contained even smaller values and got the same result.)

Table 2: Mean LOOCV Super-Smoother Bandwidth by City

City	Mean Bandwidth (h*=47)	Mean Bandwidth (h*=15)
Austin	7.507918	13.079704
San Diego	7.201515	12.700100
Minneapolis	7.103678	8.391622
Baltimore	7.016794	9.609791

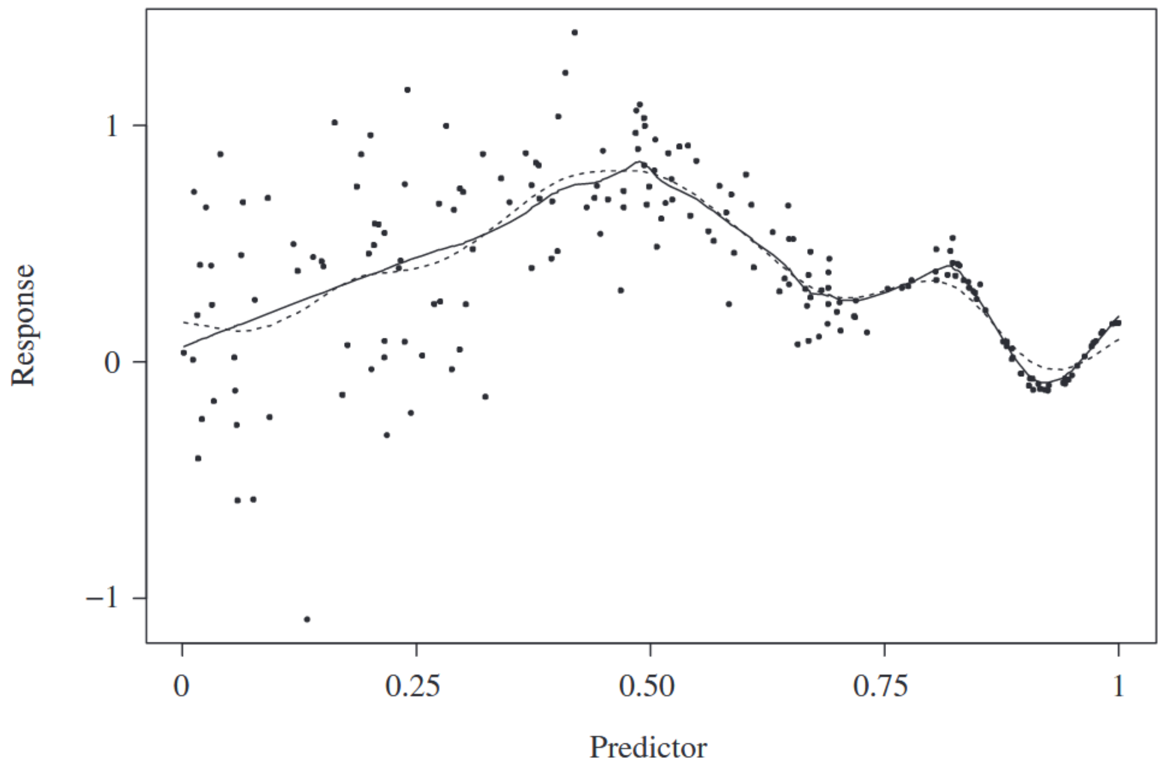

Figure 10: Non-constant variance data with LOOCV super-smooth (solid line)

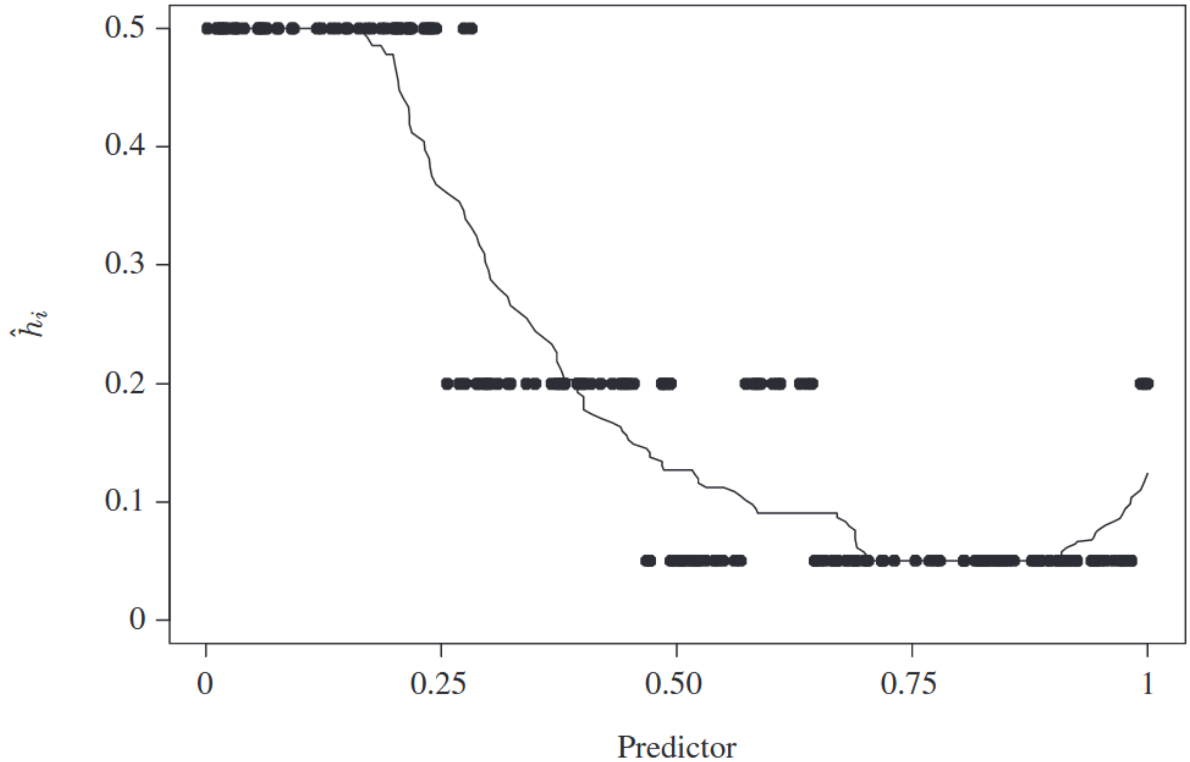

Figure 11: Selected bandwidths with smooth

Setting h^* to the smaller value of 15 gives a better smooth than $h^* = 47$, but not by much. Figure 12 displays parts of the smooths given by the LOOCV super-smoother with $h^* = 15$. The smooths are generally very choppy with only a few bright spots (e.g. Fall 1996 and Spring 1997 in Austin, Fall 1997 in San Diego). If lower bandwidths had been included in the set of candidates, the smooths would be even worse.

7. K-Block Cross-Validation

Overview: Using K-block cross-validation (KBCV) instead of LOOCV helps with dependent data. The super-smoother that uses KBCV for bandwidth selection produces good smoothing results.

The super-smoother is designed to overcome difficulties posed by data with non-constant variance, but it still fails on temperature data when using LOOCV to select bandwidths. This is due in part to the dependent nature of temperature data. If the high temperature on a given day deviates from the broader trend, the highs on days shortly before and after are likely to deviate from the broader trend in the same way. In LOOCV, surrounding days are used to predict the day that has been left out. As a result, the LOOCV prediction of the high temperature on a given day contains information about the broader trend we're trying to capture (good) but also information about how that day deviates from the trend (bad).

K-block cross-validation (KBCV) is a LOOCV alternative designed to help with this problem (the original authors call the technique h-block cross-validation, but I use "k" instead to avoid confusion with bandwidth

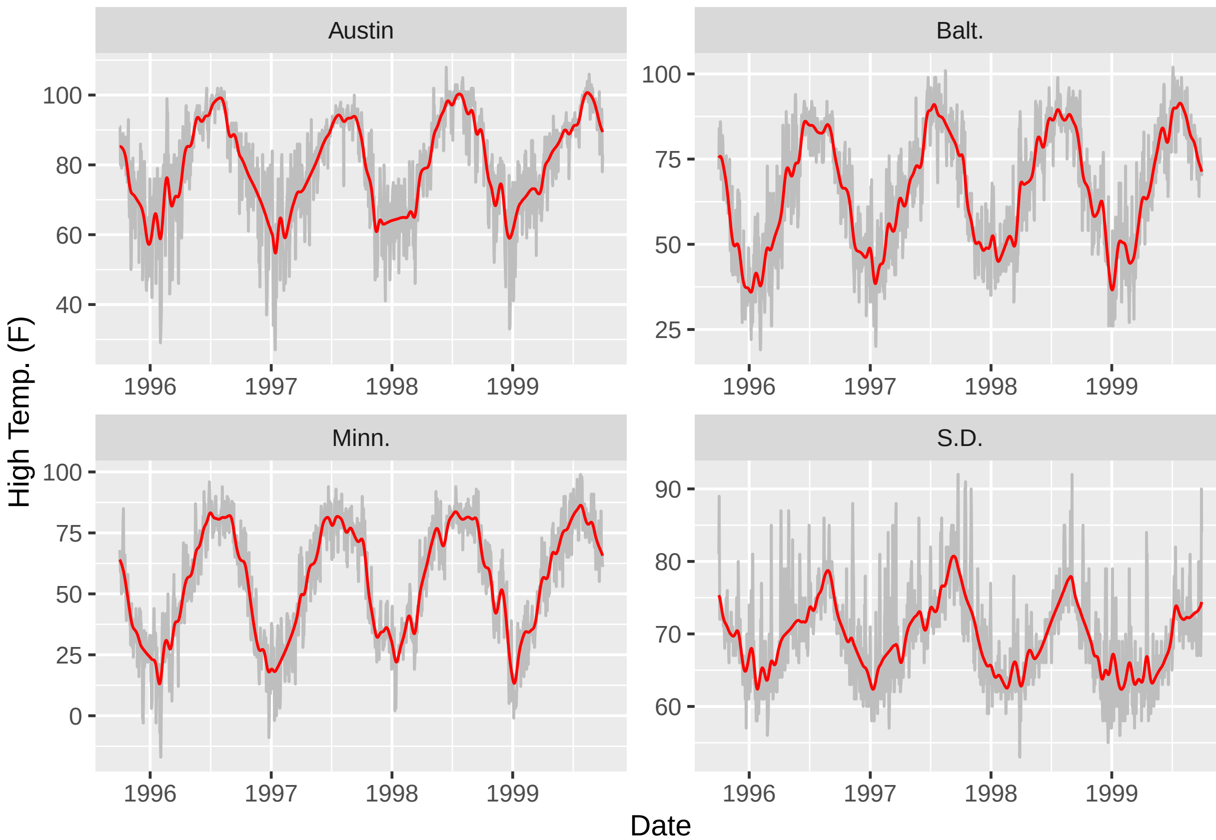

Figure 12: Daily High Temperatures with LOOCV Super-Smooth (h*=15)

parameters).[4] As with LOOCV, KBCV uses singleton test sets. The difference is in the training sets: LOOCV uses all $n-1$ remaining points in the training set, whereas KBCV uses $n-2k-1$ training points for a user-specified window parameter k. In addition to the testing point, KBCV training sets leave out a window of k points on either side.

I tested KBCV with a window parameter $k = 3$, which leaves out a total of 7 points (the testing point, 3 points before, and 3 points after) from each training set. I set $k = 3$ because the autocorrelation plots in Section 5.2 gave evidence of positive residual correlation up to lag 3. The super-smoother bandwidth used to smooth the cross-validation errors and optimal bandwidths was set to $h^* = 15$ since this gave better LOOCV results.

7.1 Results

Table 3 displays the means of the bandwidths chosen by the KBCV super-smoother across each data set. With KBCV instead of LOOCV as the bandwidth selection method, the super-smoother tends to select much higher bandwidths.

Table 3: Mean KBCV Super-Smoother Bandwidth by City

City	Mean Bandwidth (h*=15, k=3)
Austin	24.23029
San Diego	26.94622
Minneapolis	17.97655
Baltimore	19.89744

The generally higher bandwidths lead to much-improved smooths which can be seen in Figures 13 and 14. The smooths are much less wiggly than the LOOCV super-smooths in Figure 12 (Section 6.1), doing a better job of capturing broader seasonal trends instead of short-term deviations. For instance, the LOOCV smooth for Austin in Winter 1995-1996 contains 3 prominent local minima and is yanked upwards by the February 1996 heat wave. The KBCV smooth, on the other hand, contains only a single minimum in that period and is less sensitive to the heat wave.

Figures 13 and 14 also displays the bandwidths selected by the super-smoother at each point (along with the $h^* = 15$ smooth applied to them). We can see some seasonality in the bandwidth selections that match the advertised properties of the super-smoother. Higher bandwidths tend to be selected in higher-variance periods (e.g. Austin winters, Baltimore springs) and lower bandwidths tend to be selected in lower-variance periods (e.g. Austin summers).

8. Discussion and Further Research

9. References

[1] https://www.ncei.noaa.gov/cdo-web/

[2] Givens and Hoeting (2012), Computational Statistics.

[3] Friedman (1984), A variable span smoother.

[4] Chow and Nolan (1994), A cross-validatory method for dependent data.

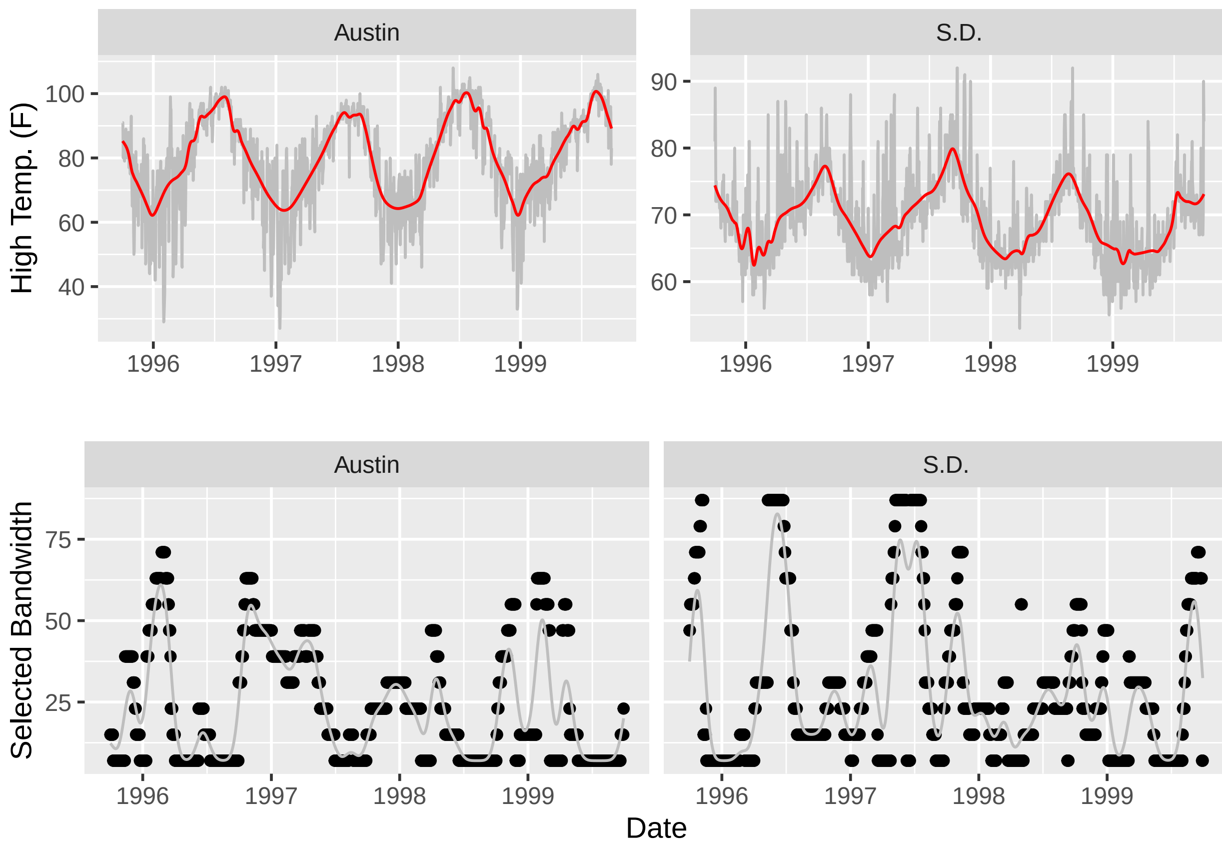

Figure 13: Above: daily highs with KBCV super-smooth (h*=15, k=3). Below: KBCV super-smoother selected bandwidths.

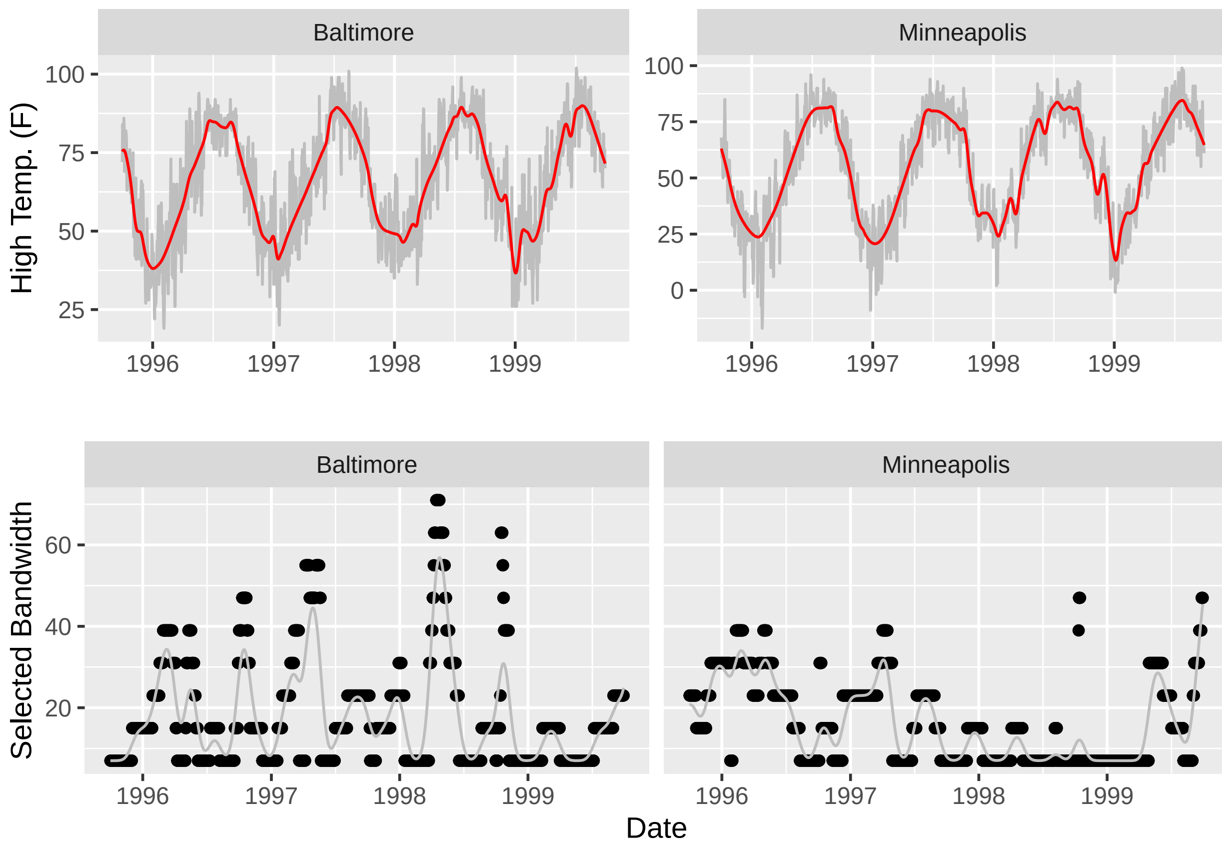

Figure 14: Above: daily highs with KBCV super-smooth (h*=15, k=3). Below: KBCV super-smoother selected bandwidths.