# Week 9: Flexible Regression Pt. 2

PLSC 40502 - Statistical Models

## Review

# Previously

- **Flexible functional forms**
  - Model $E[Y|X] = f(x)$ as a linear combination of **basis functions**
  - Polynomial bases
  - Step-function bases
  - Splines as a compromise between the two!
- **Regularization**
  - Many regressors $\rightsquigarrow$ overfitting
  - Penalize fits with many *extreme* covariates -- pull estimates towards zero.
  - L2 penalty ("ridge regression") - equivalent to Bayesian normal regression
  - L1 penalty ("lasso") - sparse solutions w/ many coefficients set to zero.

## This week

- **Kernels**
  - "Kernel smoothing" - Kernels as a "localization" tool
  - "Kernel methods" - Kernels as regression inputs
  - Two *different* uses of the term/concept - but some interesting relations
- **Forests**
  - Tree-based models for classification and regression
  - Ensembles/averages of trees via **random forests**
  - Diagnostics via "variable smoothing"
  - Extensions to detecting causal heterogeneity

## Kernel smoothing

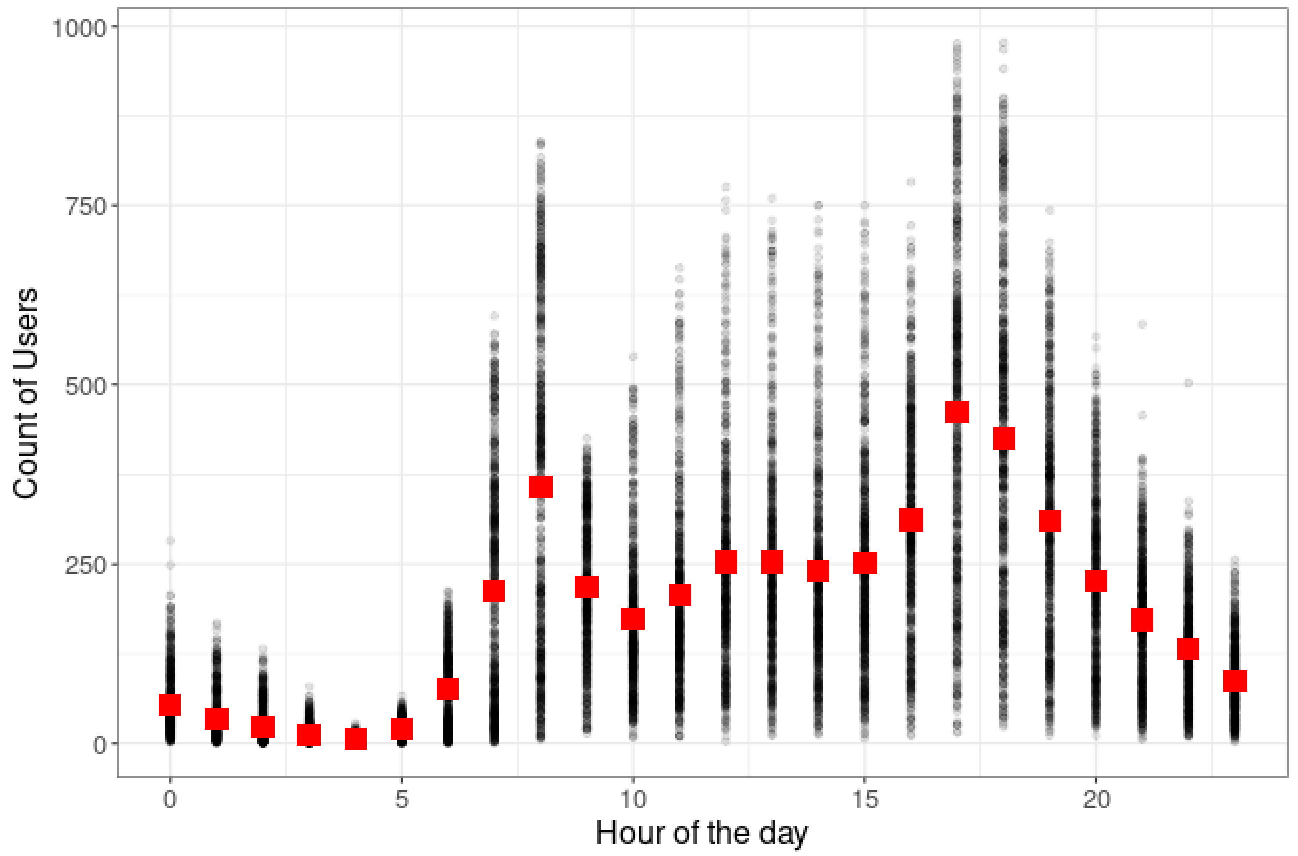
# Example: Modeling Bike Rentals

- Recall again our **bikeshare usage** dataset
  - Fanaee-T, Hadi, and Gama, Joao, "Event labeling combining ensemble detectors and background knowledge", *Progress in Artificial Intelligence* (2013): pp. 1-15

```
bike <- read_csv("data/bikes_hour.csv")
bike_by_hour <- bike %>% group_by(hr) %>% summarize(cnt = mean(cnt))
```

# Example: Modeling Bike Rentals

- Our most intuitive approach to estimating the CEF was to just take the **average** for a given hour of the day



# Kernel smoothing

- One downside - I can only evaluate the function at each discrete hour.
  - What would I predict is the usage at 7:45pm?
  - Intuitively, I would want to take the average of the "nearest data points"
- **Kernel smoothing** methods give us a straightforward, non-parametric approach to estimating $f(x) = E[Y|X = x]$ without making strong assumptions about the functional form of the CEF.
  - "weighted average" of "similar" observations
- We define the similarity between two points $x_0$ and $x$ by way of a **kernel function** $K_h()$

$$K_h(x_0, x) = D\left(\frac{|x - x_0|}{h(x_0)}\right)$$

- $D$ is our particular choice of kernel function and $h(x_0)$ is the "bandwidth" or "window size"
  - Often use a constant bandwidth that does not depend on the inputs $h(x_0) = h$
- The kernel function is **symmetric** and **non-negative**

## Kernel smoothing

- Uniform kernel

$$D(u) = \begin{cases} \frac{1}{2} & \text{if } |u| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- Triangular kernel

$$D(u) = \begin{cases} 1 - |u| & \text{if } |u| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

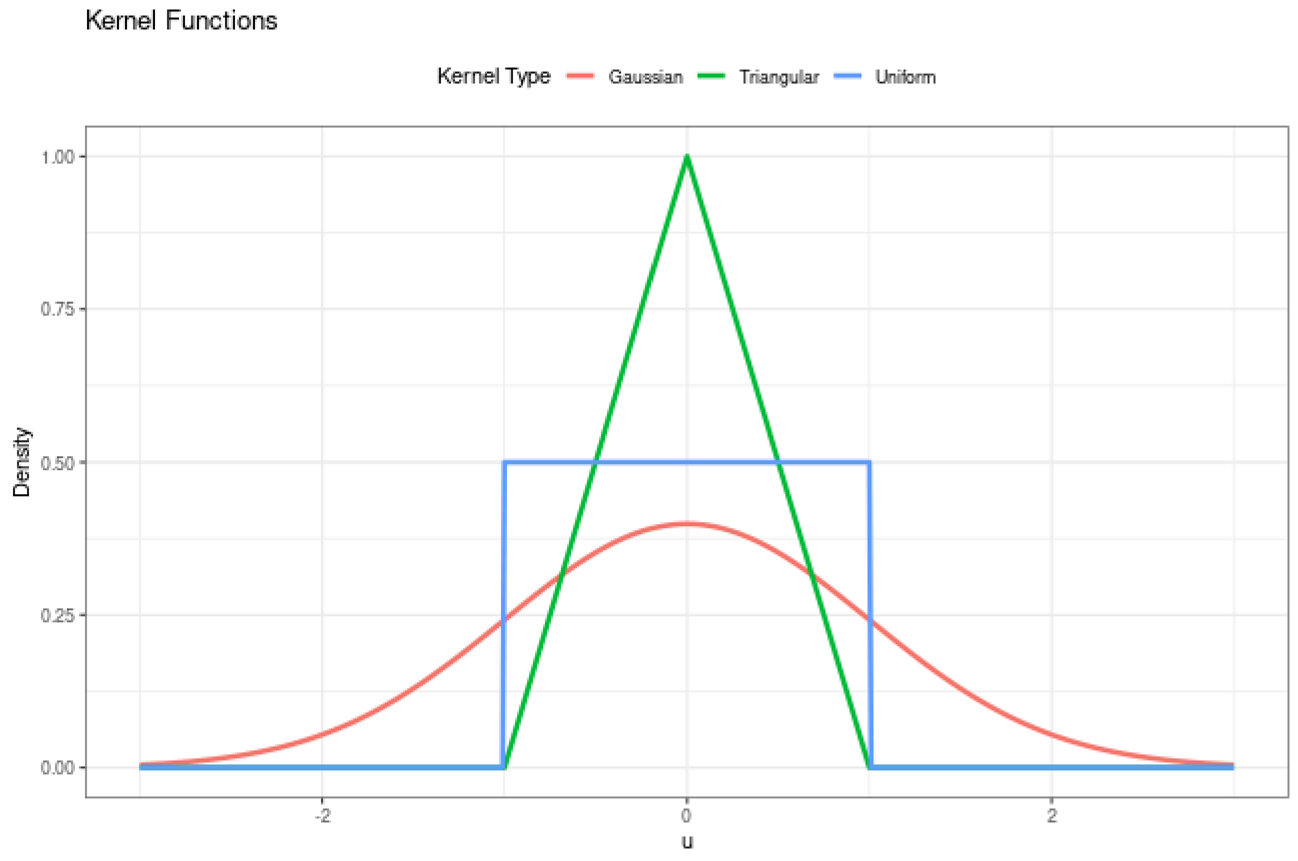
- Gaussian (or "radial basis") kernel

$$D(u) = \frac{1}{2\pi} \exp\left( - \frac{1}{2} u^2 \right)$$

## Kernel smoothing

```r
uniform_kernel <- function(x) {
  ifelse(abs(x) <= 1, 1/2, 0)
}

triangular_kernel <- function(x) {
  ifelse(abs(x) <= 1, 1 - abs(x), 0)
}

gaussian_kernel <- function(x) {
  (1 / sqrt(2 * pi)) * exp(-0.5 * x^2)
}
```

## Kernel smoothing



## Nadaraya-Watson estimator

- For a set of $N$ observations indexed by $i$ $\{y_i, x_i\}$ and given a kernel function $K_h()$ and a chosen bandwidth $h$, the **Nadarya-Watson** estimator of $f(x) = E[Y|X = x]$ is:

$$\hat{f}(x) = \frac{\sum_{i=1}^{n} K_h(x - x_i) y_i}{\sum_{i=1}^{n} K_h(x - x_i)}$$

- Our best prediction for $Y$ at $x$ is the average of the observed $y_i$, weighted by the "closeness" of their corresponding $x_i$ to the input $x$.

- Some intuitions:
  - **Larger bandwidths** = more influence from "further" points (lower variance, higher bias)
  - **Smaller bandwidths** = only "close" points have meaningful weight (higher variance, lower bias)

# Example: Modeling Bike Rentals

- In R, local kernel smoothing methods are implemented in the package **np**

```
library(np)
```

```
## Nonparametric Kernel Methods for Mixed Datatypes (version 0.60-18)
## [vignette("np_faq",package="np") provides answers to frequently asked questions]
## [vignette("np",package="np") an overview]
## [vignette("entropy_np",package="np") an overview of entropy-based methods]
```

- We'll start with a uniform kernel and a bandwidth of 3

```
kernel_reg <- npreg(cnt ~ hr, data=bike, ckertype="uniform", bws = 3, regtype='lc')
```

```
## Warning in rbandwidth(bw = tbw$bw, regtype = tbw$regtype, bwmethod =
## tbw$method, : ignoring kernel order specified with uniform kernel type
```

```
summary(kernel_reg)
```

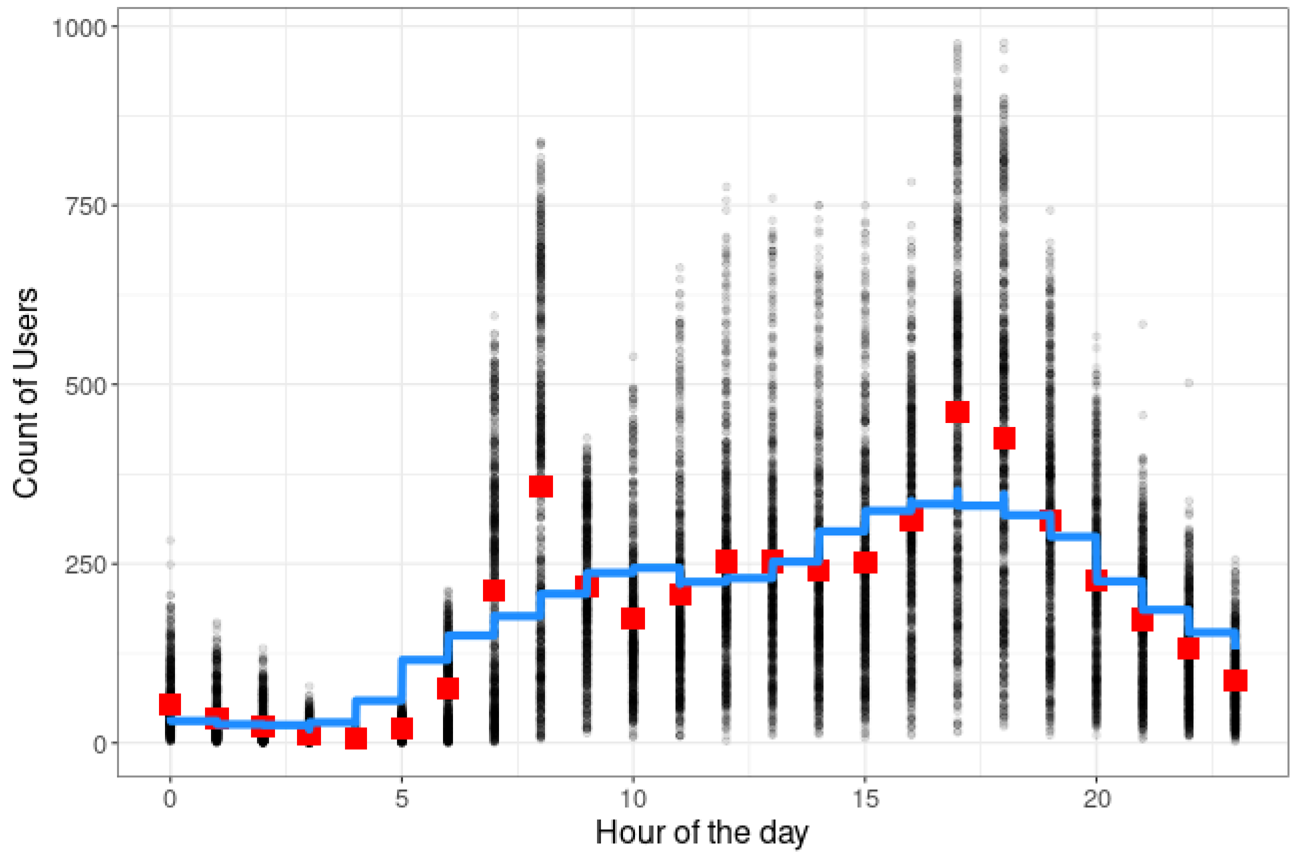
```
##
## Regression Data: 17379 training points, in 1 variable(s)
##               hr
## Bandwidth(s):  3
##
## Kernel Regression Estimator: Local-Constant
```

# Example: Modeling Bike Rentals

- Generate predictions on a sequence of points and plot the line

```
kernel_predict <- data.frame(hr = seq(0, 23, by=.01),
                             cnt = predict(kernel_reg, newdata=data.frame(hr = seq(0, 23, by=.0
```

## Example: Modeling Bike Rentals



# Example: Modeling Bike Rentals

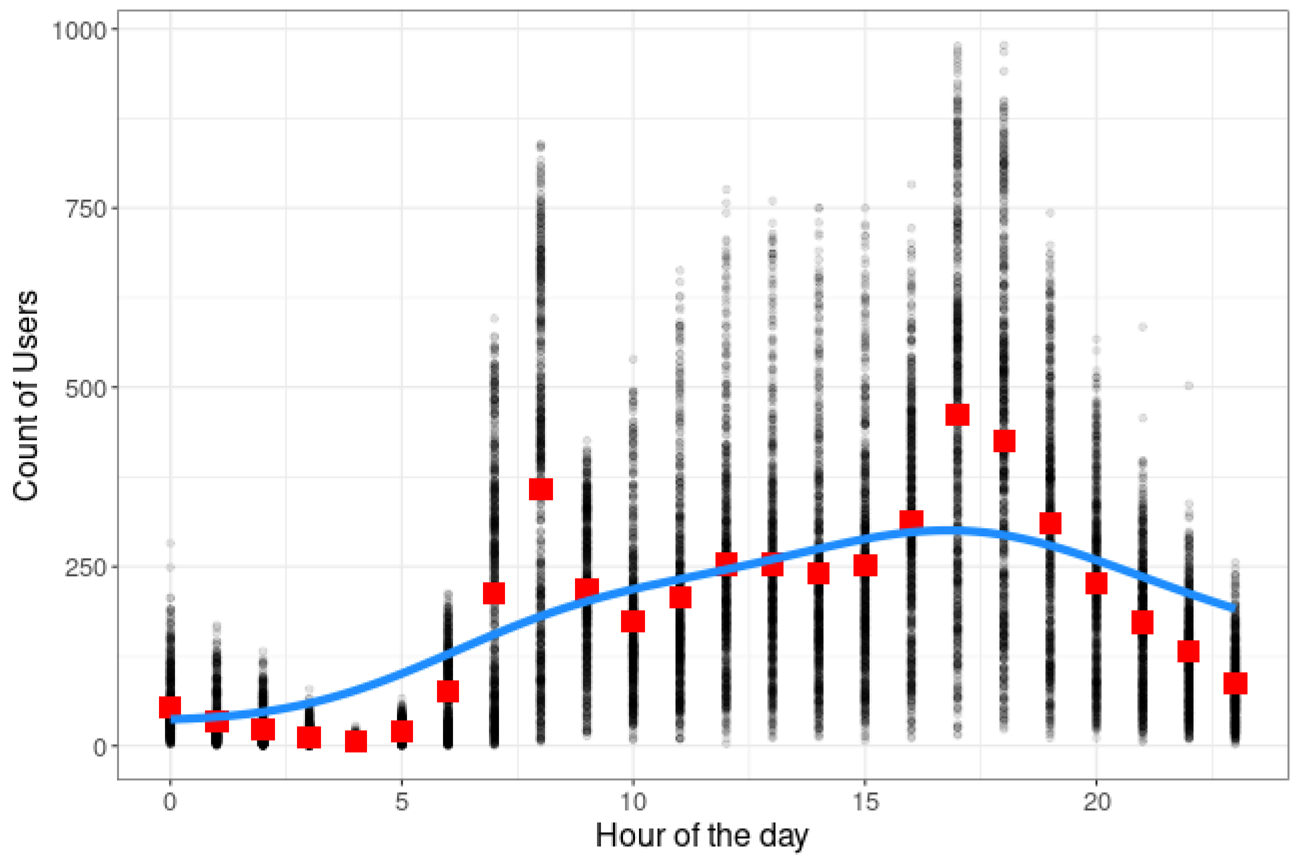
- What about if we use a gaussian kernel

```
gaussian_kernel <- npreg(cnt ~ hr, data=bike, ckertype="gaussian", bws = 3, regtype='lc')
summary(gaussian_kernel)
```

```
##
## Regression Data: 17379 training points, in 1 variable(s)
##               hr
## Bandwidth(s):  3
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 147
## R-squared: 0.355
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

```
gaussian_predict <- data.frame(hr = seq(0, 23, by=.01),
                             cnt = predict(gaussian_kernel, newdata=data.frame(hr = seq(0, 23,
```

# Example: Modeling Bike Rentals



# Example: Modeling Bike Rentals

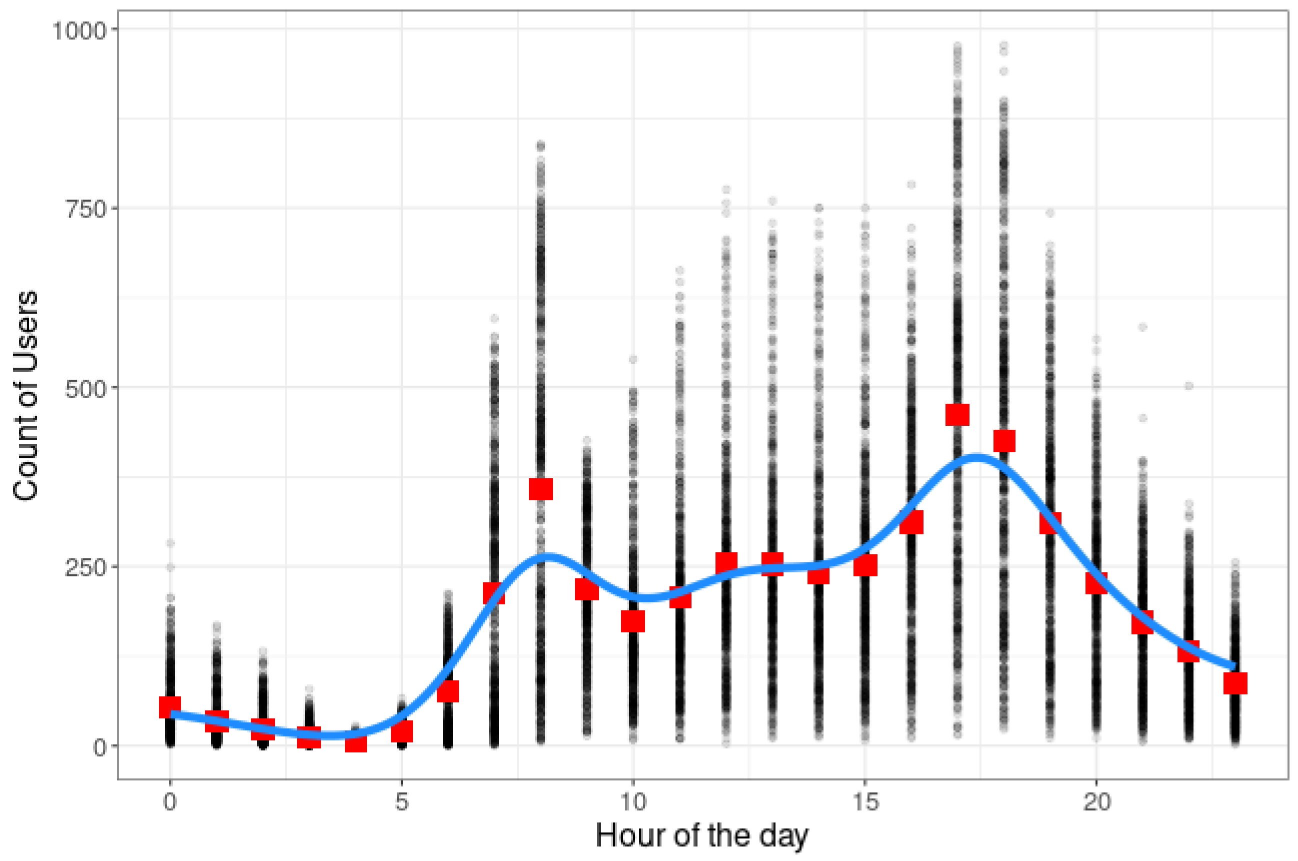
- What happens with a narrower bandwidth?

```
gaussian_kernel_2 <- npreg(cnt ~ hr, data=bike, ckertype="gaussian", bws = 1, regtype='lc')
summary(gaussian_kernel_2)
```

```
##
## Regression Data: 17379 training points, in 1 variable(s)
##               hr
## Bandwidth(s):  1
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 131
## R-squared: 0.478
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

```
gaussian_predict_2 <- data.frame(hr = seq(0, 23, by=.01),
                             cnt = predict(gaussian_kernel_2, newdata=data.frame(hr = seq(0, 23
```

# Example: Modeling Bike Rentals



# Local linear regression

- Instead of just making our prediction a weighted average, we can incorporate a least-squares adjustment
  - Fit a **line** rather than a **constant**
- Our prediction for $f(x)$ is defined as the solution to the **least-squares** problem in terms of intercept $\alpha(x)$ and slope $\beta(x)$

$$\sum_{i=1}^{n} K_h(x, x_i)[y_i - \alpha(x) - \beta(x)x_i]$$

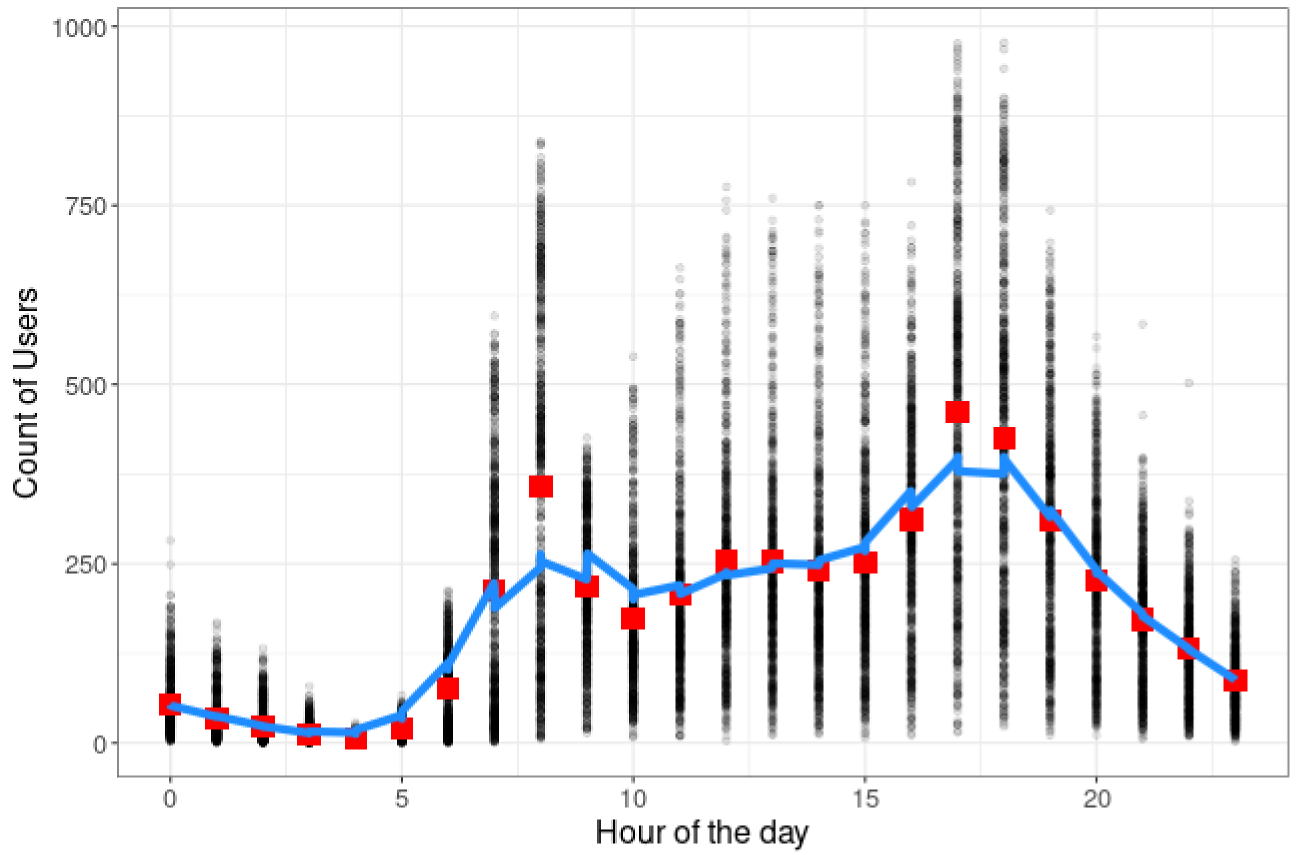
- Let's fit this with **np** using a uniform kernel and a narrow bandwidth.

```
uniform_local_linear <- npreg(cnt ~ hr, data=bike, ckertype="uniform", bws = 2, regtype='ll')
```

```
## Warning in rbandwidth(bw = tbw$bw, regtype = tbw$regtype, bwmethod =
## tbw$method, : ignoring kernel order specified with uniform kernel type
```

```
uniform_predict_ll <- data.frame(hr = seq(0, 23, by=.01),
                             cnt = predict(uniform_local_linear, newdata=data.frame(hr = seq(0,
```

## Example: Modeling Bike Rentals



## Kernel Methods

# Regression as optimization

- Throughout this course and previous courses, our goal has been to find some function $f(x)$ of inputs $x$ that acts as a "best predictor" of $Y$
    - We define some objective function $\mathcal{L}$ and attempt to minimize/maximize it
    - Least squares; maximum likelihood
- Given a set of $n$ inputs, $\{x_i, y_i\}$, we want to find a function that solves

$$\hat{f} = \arg\min_f \mathcal{L}(\{x_i, y_i\}_{i=1}^n)$$

## Space of functions

- So far, we've placed severe constraints on the space of functions over which we search.
- Linear functions

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots$$

- Basis expansion

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$$

- Can we be more flexible than this while still retaining computational feasibility?

## Space of functions defined by kernels

- "Kernel methods" consider searching over a space of functions defined implicitly by a positive-definite kernel function.
    - Example, the **Gaussian** kernel

$$K(x, x') = \exp\left( - \frac{||x - x'||^2}{2\sigma^2} \right)$$

- We'll consider a search over a set of functions defined by a linear combination of kernels evaluated at the input points in our data

$$f(x) = \sum_{i=1}^{N} c_i K(x, x_i)$$

- What is this space of functions? How "flexible" is it?

# Reproducing Kernel Hilbert Spaces

- A positive semi-definite kernel function $K(x, x')$ corresponds to a space of functions known as a "reproducing kernel Hilbert space" (RKHS).
  - A choice of kernel implies a mapping $\phi(x)$ that transforms the input vector $x$ into a higher dimensional (possibly infinite dimensional) space.
  - This space is equipped with an inner product $\langle \cdot, \cdot \rangle$ such that for any two functions, the inner product is given by an evaluation of the kernel at the inputs

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

- As a result, we can work in the space of $\phi$ without ever having to evaluate $\phi$ directly -- only its kernel.

## "The Kernel Trick"

- One important feature of optimization over functions defined over a reproducing kernel Hilbert space is that even if the functions in the space are **infinite dimensional** - solutions to optimization problems are necessarily **finite dimensional** and have the form of a linear combination of kernels.
- The **representer theorem** states that for minimization problems of the form:

$$\hat{f} = \arg\min_{f} \mathcal{L}(\{x_i, y_i\}_{i=1}^{n}) + \lambda ||f||^2$$

the solution is a finite sum of weighted kernel functions

$$\hat{f} = \sum_{i=1}^{n} \hat{c}_i K(x, x_i)$$

# Kernel Regularized Least Squares

- Intuitively, what the "kernel trick" allows us to do is design a good functional approximator by way of a conventional ridge regression estimator.
    - Relatively well-known in machine learning (particularly for classification) throughout the 2000s, but introduced to political science by **Hainmueller and Hazlett (2014)** and labeled **Kernel regularized least squares**
- Our regression matrix is constructed by creating a total of $N$ columns.
    - Each row of the "kernel regression" matrix is a linear combination of the kernel evaluated between $x_i$ and each observation in the data

$$f(x_i) = c_1 K(x_i, x_1) + c_2 K(x_i, x_2) + \dots c_n K(x_i, x_n)$$

- The regression matrix is $N \times N$
    - We need to regularize!

## Example: Interpolating spatial data

- A very common problem in spatial statistics is predicting an outcome at any given latitude/longitude coordinate given a finite set of observations at particular coordinates
    - e.g. What is the estimated average temperature given observations at a number of weather monitoring stations?
    - Finding the "best" interpolation between the observed points is known as "kriging" in the geostatistics literature
- We'll use a sample of U.S. weather stations as an example:

```
library(sf)
```

```
## Linking to GEOS 3.10.2, GDAL 3.4.1, PROJ 8.2.1; sf_use_s2() is TRUE
```

```
library(geosphere)
```

## Example: Interpolating spatial data

- Start by reading in the U.S. shapefile

```r
us_shapefile <- read_rds("data/us_shapefile.rds") %>%
  filter(!(STUSPS %in% c("AK", "HI", "AS", "GU", "MP", "PR", "VI")))
```

- Next, read in the NOAA Climate Assessment Database monthly weather station data from January, 2025.

```r
weather <- read_csv("data/monthly_summary_202501_v2.csv") %>% filter(country == "UNITED_STATES'
```

```
## Rows: 12261 Columns: 31
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (5): id, call, city, state, country
## dbl (26): date, lat, lon, elev, atmp, ntmp, nbtmp, nyrt, eyrt, amax, nbmax, ...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

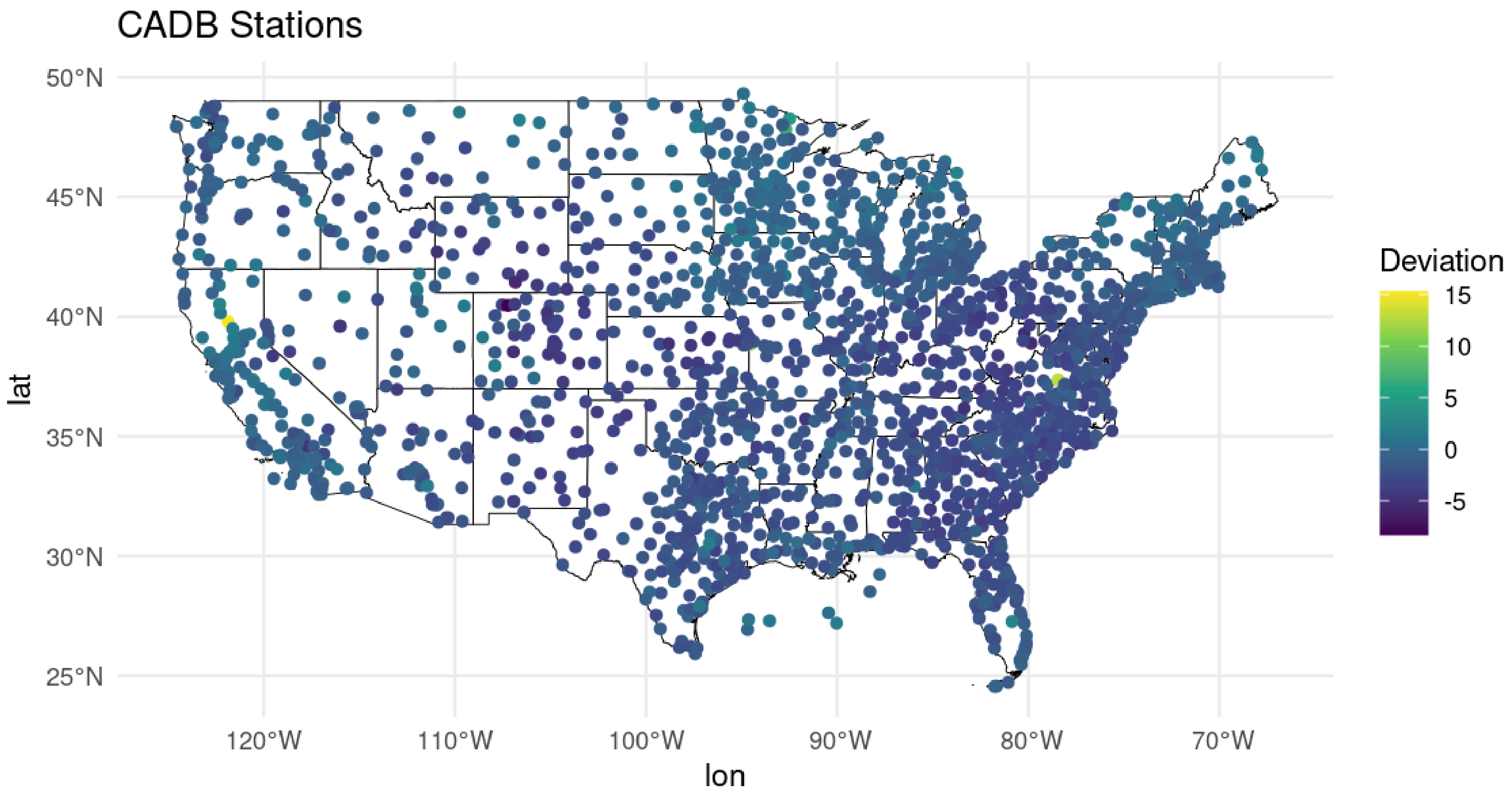
```r
weather <- weather %>% filter(atmp > -99999&ntmp > -99999) # Missing obs hard coded as -99999
weather <- weather %>% mutate(deviation = atmp - ntmp)
weather <- weather %>% filter(!(state %in% c("AK", "HI", "AS", "GU", "MP", "UM", "PR", "VI")))
```

## Example: Interpolating spatial data

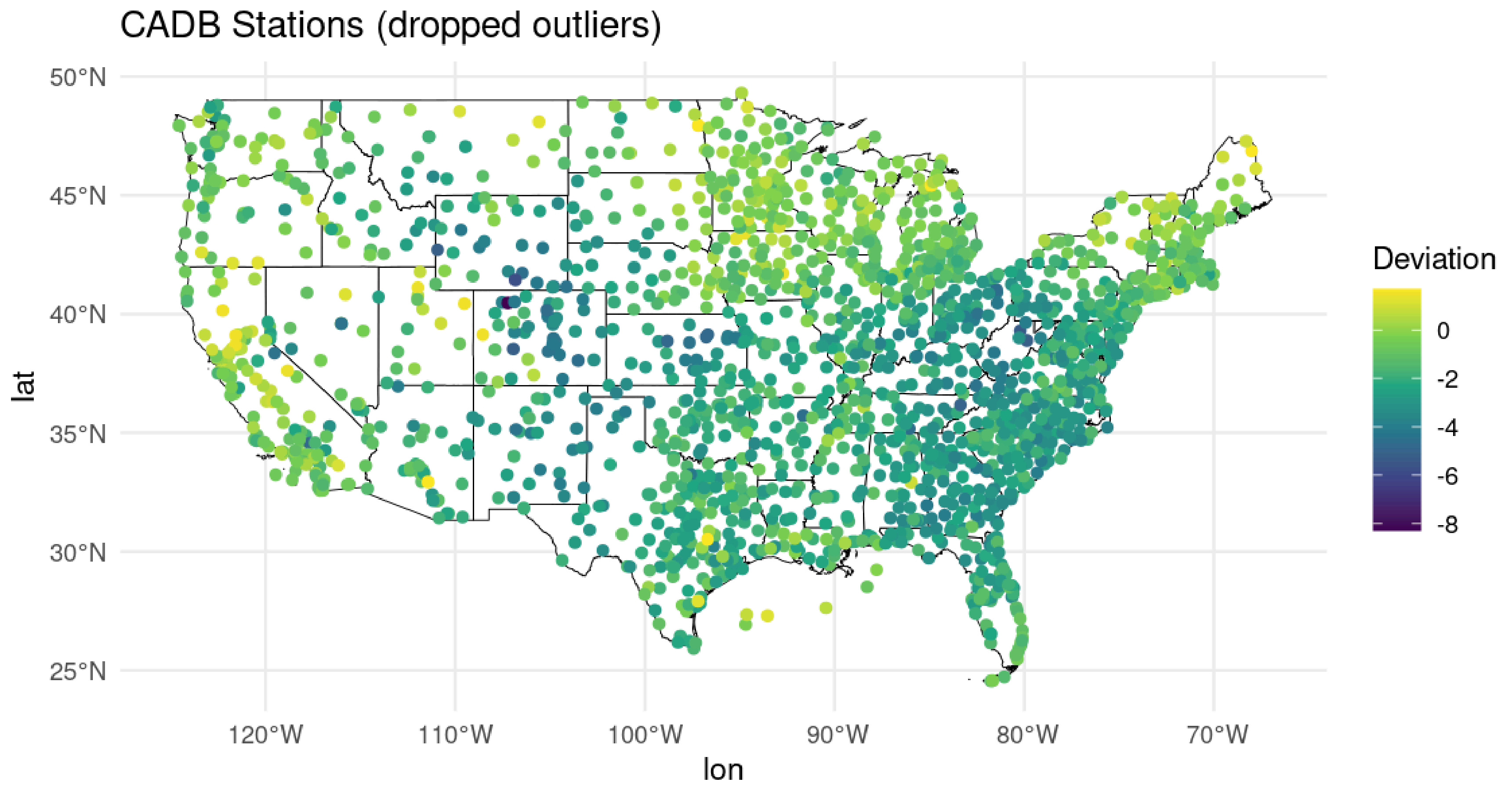
- Convert to a plotable object and overlay on the map

```r
# Convert the stations to a plottable object
weather_sf <- st_as_sf(weather, coords = c("lon", "lat"), crs = st_crs(us_shapefile), remove=F)
```

## Example: Interpolating spatial data



## Example: Interpolating spatial data



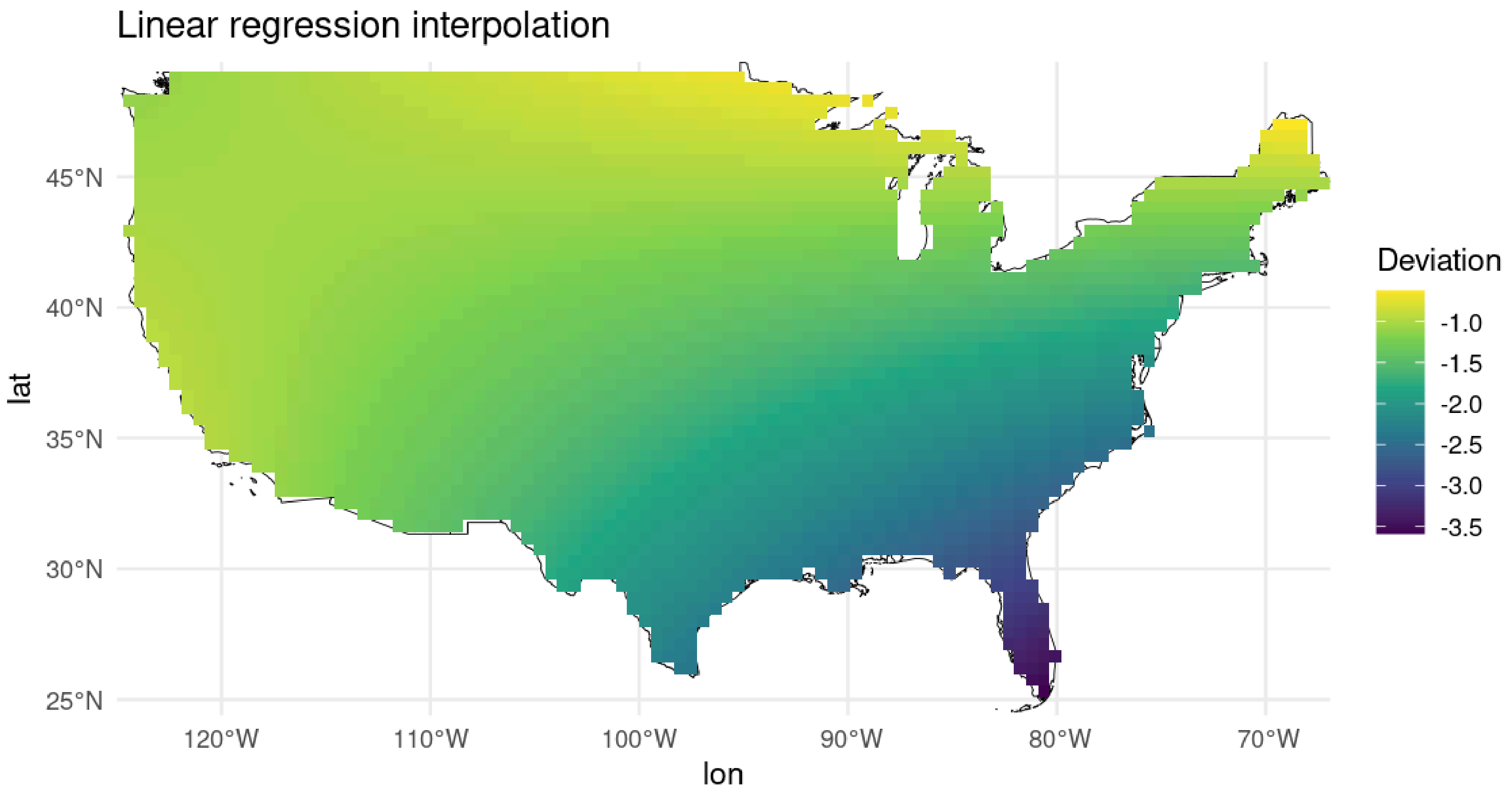
## Example: Interpolating spatial data

- We'll make a grid of 50km squares on which to to evaluate our predictions

```
# Set boundaries for U.S. (approximate)
lat_min <- 24.396308  # Southernmost point (Florida)
lat_max <- 49.384358  # Northernmost point (Canada border)
lon_min <- -125.0     # Westernmost point (California)
lon_max <- -66.93457  # Easternmost point (Maine)

# Approximate 50 km in degrees (this will vary by latitude, but we'll use an average factor)
lat_interval <- 50 / 110.574  # 1 degree latitude ≈ 110.574 km
lon_interval <- 50 / (111.320 * cos(mean(c(lat_min, lat_max)) * pi / 180))  # Longitude interva

# Create a sequence of points for latitude and longitude
latitudes <- seq(lat_min, lat_max, by = lat_interval)
longitudes <- seq(lon_min, lon_max, by = lon_interval)

# Create a grid of all combinations of latitudes and longitudes
grid_points <- expand.grid(lat = latitudes, lon = longitudes)

# Convert the grid points to an 'sf' object
grid_points_sf <- st_as_sf(grid_points, coords = c("lon", "lat"), crs = st_crs(us_shapefile), r

# Filter grid points to keep only those within the U.S. boundary
grid_points_within_us <- st_intersection(grid_points_sf, us_shapefile)
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
```

# Example: Interpolating spatial data



# Kernel Regularized Least Squares

- What happens when we fit KRLS with latitude and longitude as the covariate inputs.

```r
library(KRLS)
```

```
## ## KRLS Package for Kernel-based Regularized Least Squares.

## ## See Hainmueller and Hazlett (2014) for details.
```

- Estimate the model
  - We use the default Hainmueller and Hazlett (2014) approach to setting the bandwidth equal to the number of dimensions in the data.
  - Standard leave-one-out cross-validation of the penalty parameter

```r
krls_reg <- krls(X = model.matrix(~lat + lon, data=weather)[,-1],
                 y = weather$deviation,
                 whichkernel = "gaussian",
                 vcov=F,
                 derivative=F)
```

# Kernel Regularized Least Squares

- What do the inputs look like?

```
dim(krls_reg$K)
```

```
## [1] 1825 1825
```

```
head(krls_reg$K[,1])
```

```
##        1        2        3        4        5        6
## 1.00e+00 6.90e-07 1.38e-06 2.62e-06 8.33e-06 1.22e-05
```

```
head(krls_reg$K[,2])
```

```
##        1        2        3        4        5        6
## 6.90e-07 1.00e+00 9.69e-01 9.17e-01 8.01e-01 7.69e-01
```

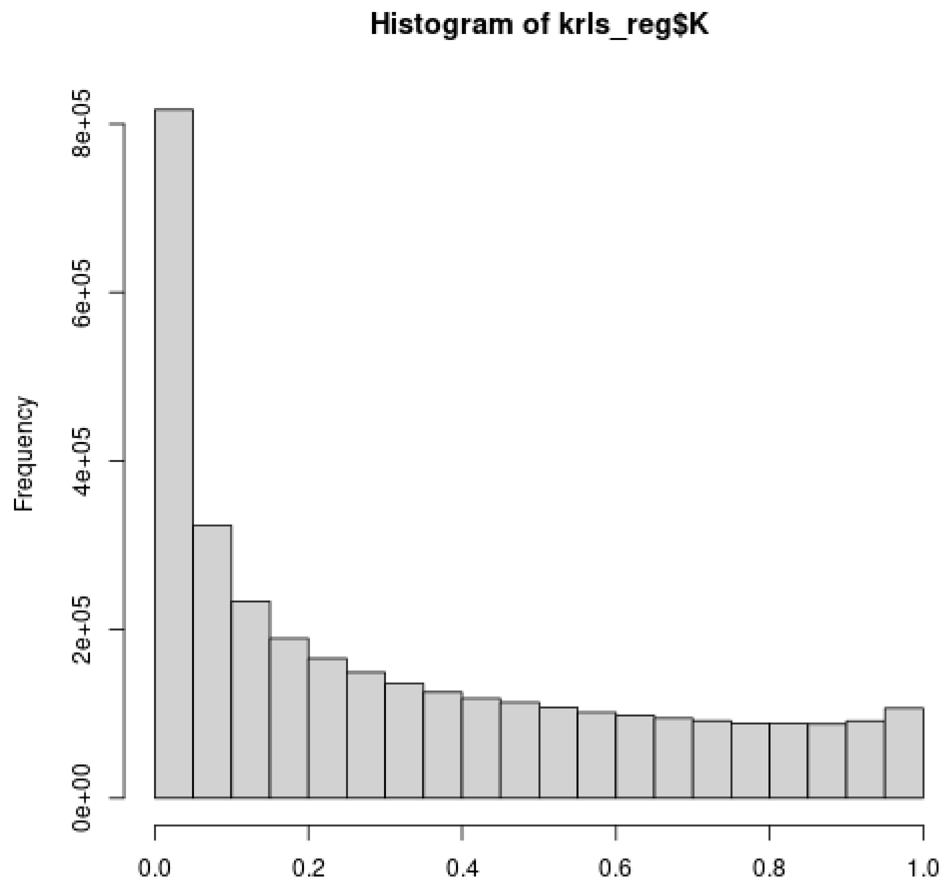
```
head(krls_reg$K[,3])
```

```
##        1        2        3        4        5        6
## 1.38e-06 9.69e-01 1.00e+00 9.86e-01 9.11e-01 8.83e-01
```

## Kernel Regularized Least Squares

- What do the inputs look like?

```
hist(krls_reg$K)
```

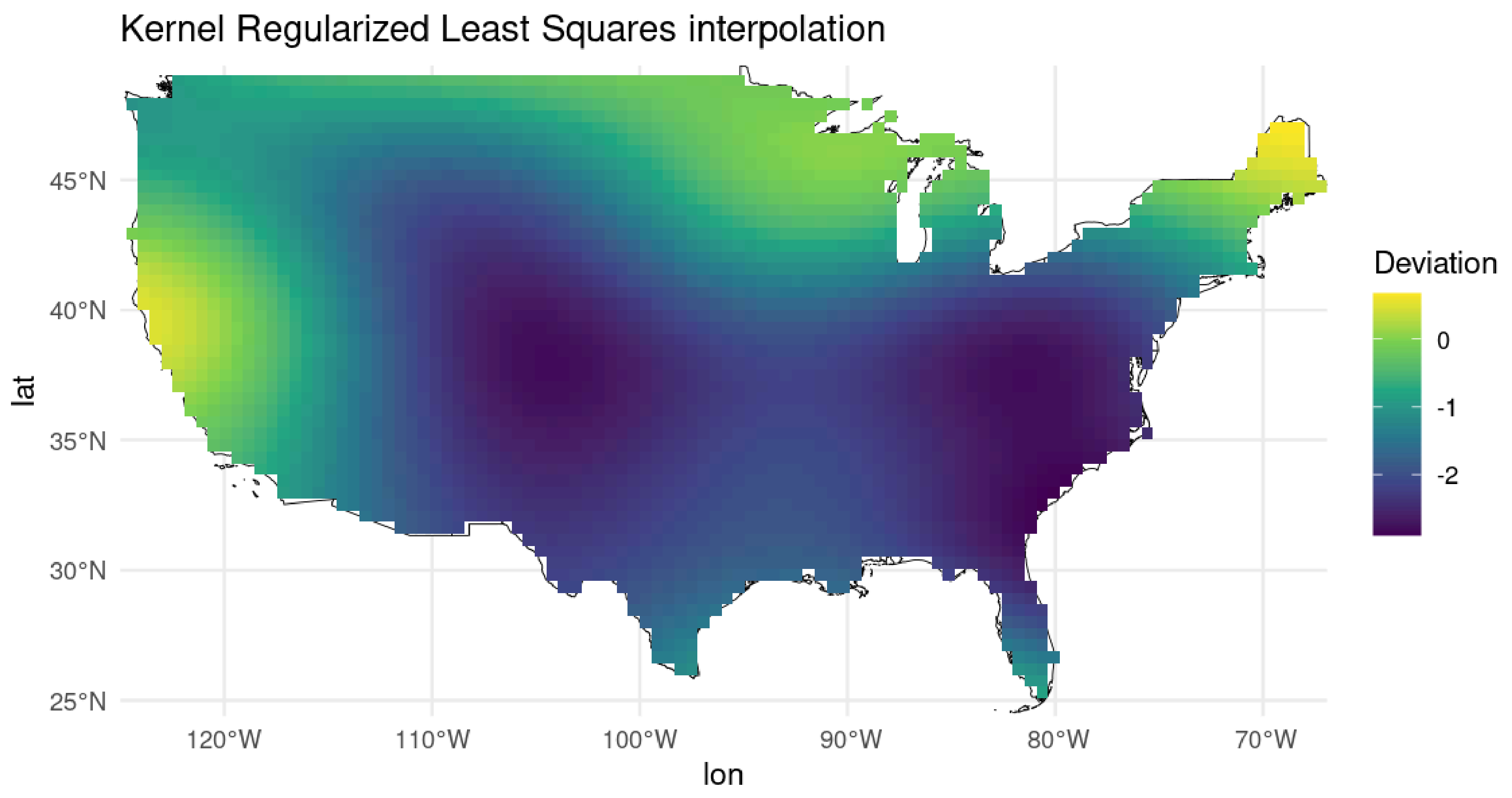


## Kernel Regularized Least Squares

- Predict using the KRLS regression for each of our grid squares.

```
grid_points_within_us$krlspredict <- predict(krls_reg, newdata= model.matrix(~lat + lon, data=g
```

# Kernel Regularized Least Squares



## Kernel Regularized Least Squares

- What if we chose a smaller bandwidth for the Gaussian kernel?

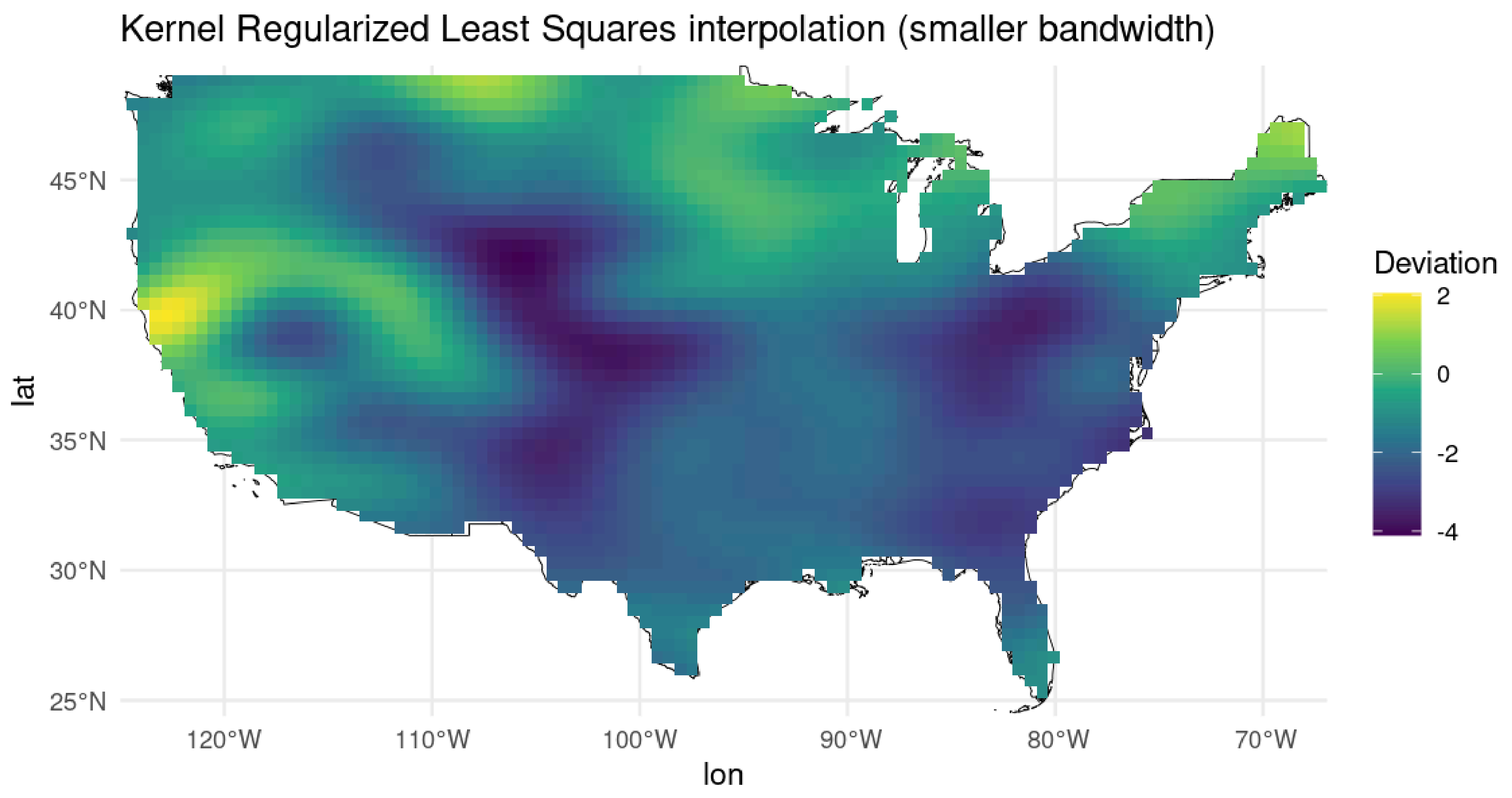
```
krls_reg_2 <- krls(X = model.matrix(~lat + lon, data=weather)[,-1],
                  y = weather$deviation,
                  sigma = 1/5,
                  whichkernel = "gaussian",
                  vcov=F,
                  derivative=F)
```

## Kernel Regularized Least Squares

- Predict using the KRLS regression for each of our grid squares.

```
grid_points_within_us$krlspredict2 <- predict(krls_reg_2, newdata= model.matrix(~lat + lon, dat
```

# Kernel Regularized Least Squares


Kernel Regularized Least Squares interpolation (smaller bandwidth)
45°N
40°N
35°N
30°N
25°N
lat
120°W
110°W
100°W
90°W
80°W
70°W
lon
Deviation
2
0
-2
-4