Density Estimation

DS 6410 | Spring 2024

density.pdf

Contents

1 Density Estimation Intro

1.1 Required R Packages

We will be using the R packages of:

- `tidyverse` for data manipulation and visualization
- `fitdistrplus` for parametric estimation
- `ks` for non-parametric kernel density estimation

```
library(fitdistrplus) # install.packages("fitdistrplus")
library(ks)           # install.packages("ks")
library(tidyverse)
```

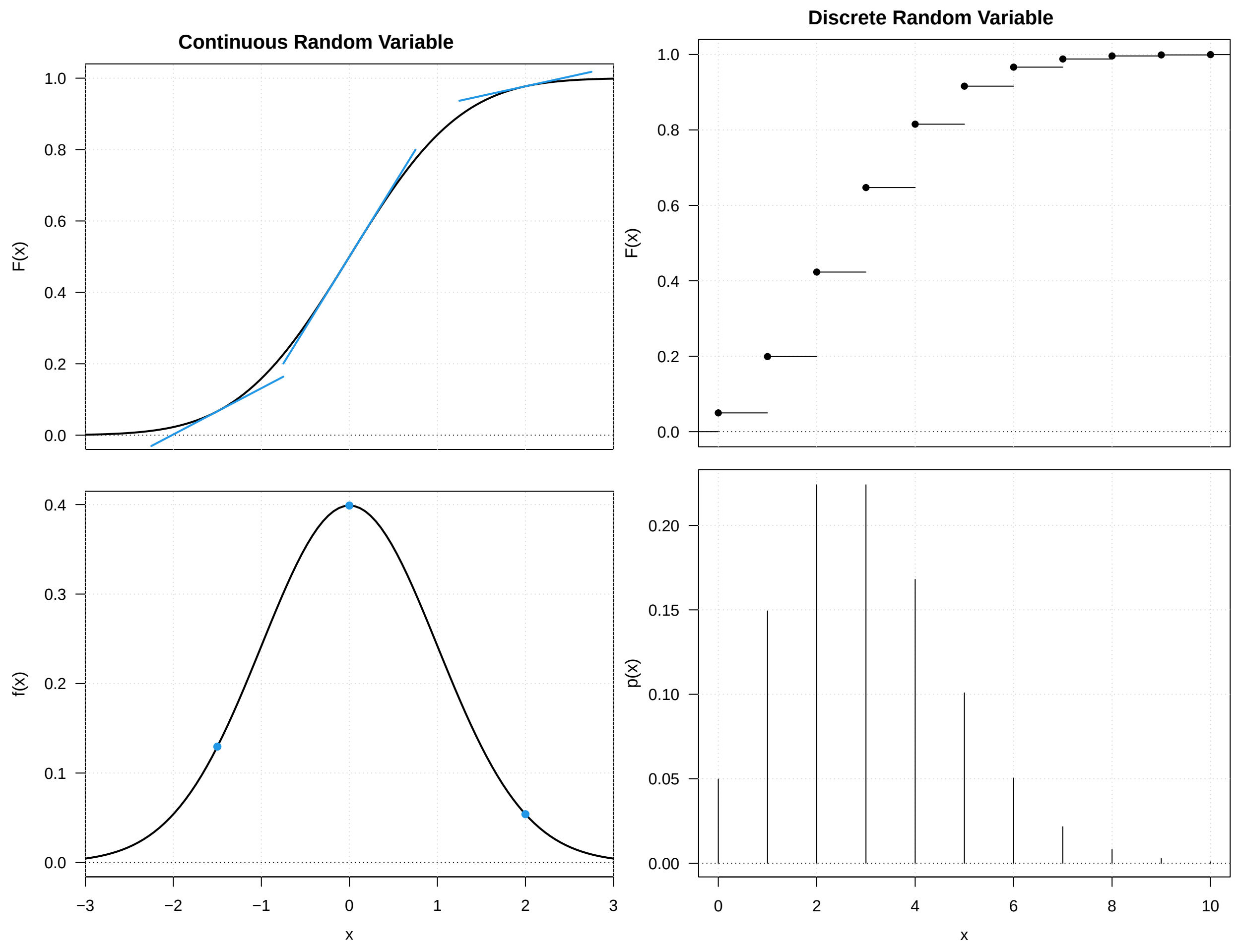
1.2 Distributions

- For many problems, an optimal decision can be formulated if we know the **distribution** of the relevant random variable(s) .
  - The random variable(s) are the unknown or unobserved values.
- Often, only certain properties of the distribution (expected value, variance, quantiles) are needed to make decisions.
- Much of statistics is involved with estimation of the distributions or their properties.

1.2.1 Random Variables

Let X be a **random variable** of interest.

- The **cumulative distribution function (cdf)** is $F(x) = \Pr(X \leq x)$.
  - $F(x)$ is the probability that the random variable X (“big X”) will take a value less than or equal to x (“little x”).
- For *discrete* random variables, the **probability mass function (pmf)** is $f(k) = \Pr(X = k)$.
  - $f(k) \geq 0, \ \sum_k f(k) = 1$
  - $f(k) = F(k) - F(k-1)$
- For *continuous* random variables, the **probability density function (pdf)** is $f(x) = \frac{d}{dx}F(x)$.
  - $f(x) \geq 0, \ \int_{\infty}^{\infty} f(x) = 1$



1.2.2 Parametric Distributions

A **parametric** distribution, $f(x; \boldsymbol{\theta})$ is one that is fully characterized by a set of parameters, $\boldsymbol{\theta}$. Examples include:

- Normal/Gaussian
  - parameters: mean μ, standard deviation σ
- Poisson
  - parameter: rate λ
- Binomial
  - parameters: size n, probability p
- There are also multivariate versions: Gaussian $N(\mu, \Sigma)$.

If we model (assume) the random variable as following a specific parametric distribution, then we only need to estimate the parameter(s) to have the entire distribution characterized. The parameters are often of direct interest themselves (mean, standard deviation).

Note

More details about some common parametric distributions can be found in the Distribution Reference Sheet

1.2.3 Non-Parametric Distributions

A distribution can also be estimated using **non-parametric** methods (e.g., histograms, kernel methods, splines). These approaches do not enforce a parametric family (which is essentially a type of prior knowledge), but let the data determine the shape of the density/pmf/cdf. As you might imagine more data is required for these methods to work well. Non-parametric approaches are excellent for exploratory data analysis, but can also be very useful for other types of modeling (e.g., classification, anomaly detection).

1.3 Example: Default Classification

Density estimation can be useful in *classification problems*, where the goal is to determine which class a new observation belongs to.

Below are two *histogram* density estimates; one for customers of a German bank that have good credit (blue) and the other for customers who defaulted (green). If a new customer is observed to have $X = 5$, then the evidence favors them having good credit because $X = 5$ is more likely for customers with good credit (this needs to be coupled with the proportion of population with good/bad credit).

The bottom plot shows the corresponding log density ratio, which can help the bank make a decision on the customer's credit-worthiness.

```
#> Error: '../data/german-credit.csv' does not exist in current working directory ('/home/mporter/Dro
#> Error in data[, -21]: object of type 'closure' is not subsettable
#> Error in data[, 21]: object of type 'closure' is not subsettable
#> Error in eval(expr, envir, enclos): object 'Y' not found
#> Error in eval(expr, envir, enclos): object 'G' not found
```

```
#> Error in eval(expr, envir, enclos): object 'germanCredit' not found
#> Error in eval(expr, envir, enclos): object 'x' not found
#> Error in eval(expr, envir, enclos): object 'xrng' not found
#> Error in eval(expr, envir, enclos): object 'x' not found
#> Error in eval(expr, envir, enclos): object 'x' not found
#> Error in eval(expr, envir, enclos): object 'h1' not found
#> Error in eval(expr, envir, enclos): object 'x' not found
#> Error in eval(expr, envir, enclos): object 'x' not found
#> Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): plot.new has not been called ye
#> Error in box(): plot.new has not been called yet
#> Error in eval(expr, envir, enclos): object 'h1' not found
#> Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): plot.new has not been called ye
#> Error in eval(expr, envir, enclos): object 'h1' not found
#> Error in eval(expr, envir, enclos): object 'h0' not found
#> Error in eval(expr, envir, enclos): object 'h1.shrink' not found
#> Error in eval(expr, envir, enclos): object 'bins' not found
#> Error in (function (s, units = "user", cex = NULL, font = NULL, vfont = NULL, : plot.new has not b
```

1.4 Example: Association Analysis

Density estimation can be useful in *association analysis*, where the goal is to find the regions with unusually high density (bump-hunting).

1.5 Example: Disease Outbreak Detection

Density estimation can be useful in *anomaly detection systems*, where the goal is to (often quickly) determine the time point when observations starting coming from a new or different distribution.

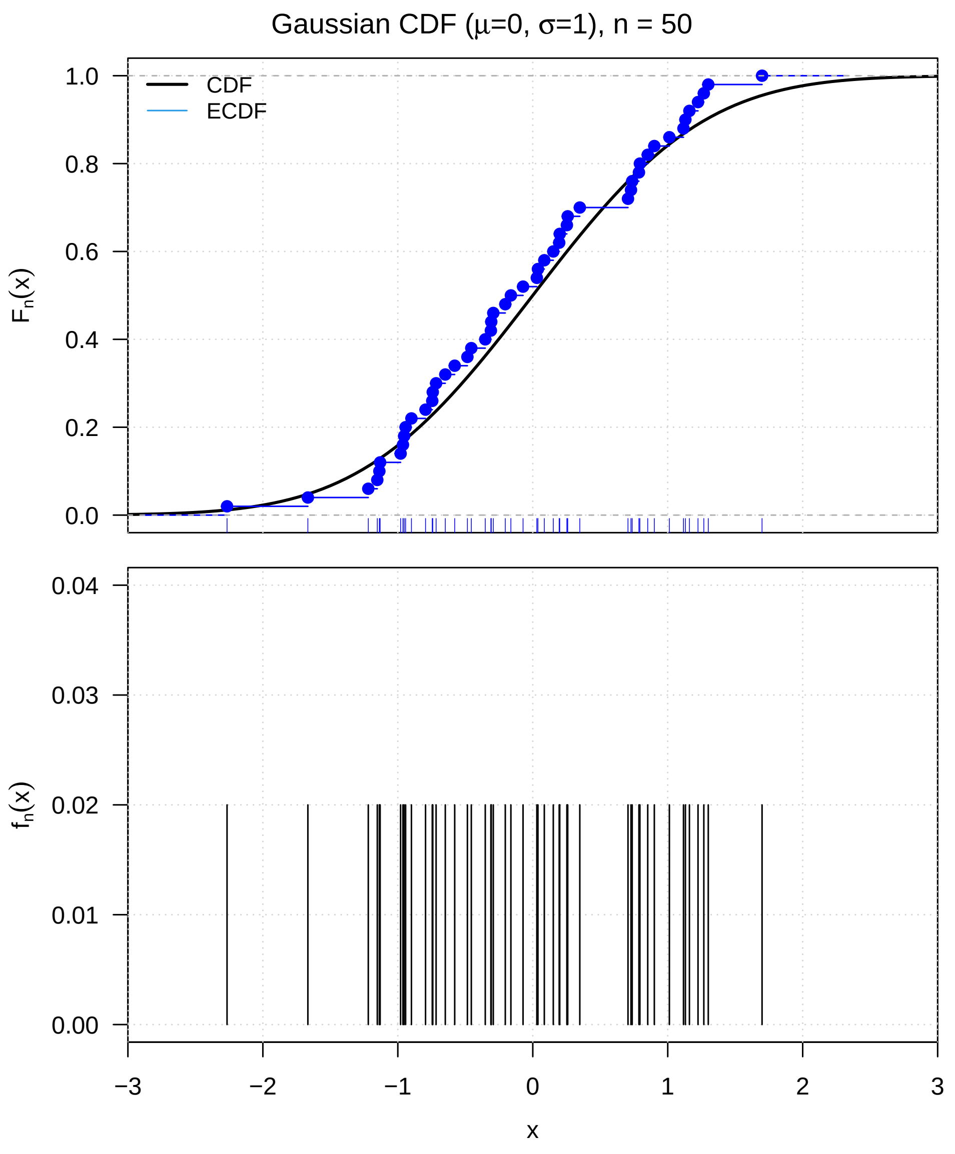
Below is simulated disease outbreak data representing the number of cases of some reported symptoms in an Emergency Department. If we can estimate the distribution of the baseline, or normal counts, on each day, then we will be able to flag an anomaly whenever the observations become *unlikely*.

```
#> Error: '../data/ED-train-01.csv' does not exist in current working directory ('/home/mporter/Dropb
#> Error in eval(expr, envir, enclos): object 'ER' not found
```

1.6 Estimation

- These problems would be relatively easy to solve if we knew the exact distributions of the random variables of interest.
- Unfortunately, this is usually never the case (but of course: flipping coins, drawing cards, and playing with urns is different).
- We must use data to estimate the aspects/parameters of a distribution necessary to make good decisions.
  - And it is important to be mindful of the resulting uncertainty (bias, variance) in our estimation.

1.6.1 Empirical CDF and PDF



Data: $n = 50$ from $X \sim N(0,1)$

2 Parametric Density Estimation

2.1 Method of Moments Estimation (MOM)

- Let X be a random variable with *pdf/pmf* $f(x;\theta)$ parameterized by $\theta \in \Theta$.
- Let $D = \{X_1, X_2, \dots, X_n\}$ be the observed data.
- *Method of Moments (MOM)* estimators match the sample moments to the theoretical moments
  - This works when the parameter(s) can be written as functions of the moments.
  - To estimate p parameters, use p moments.
- 1st moments

  - The 1st *theoretical* moment $E[X]$ is the mean.
  - The 1st *sample* moment is the sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} X_i$
  - If $g_1(\theta) = E[X]$, then set $g_1(\theta_{\text{MM}}) = \bar{x}$ and solve for θ_{MM}.

- 2nd (central) moments
  - The 2nd *theoretical* central moment $V(X) = E[(X - \mu)^2]$ is the variance.
  - The 2nd *sample* central moment is the sample variance $\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$.
  - If $g_2(\theta) = V(X)$, then set $g_2(\theta_{\text{MM}}) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$ and solve for θ_{MM}.

- Maximum Likelihood estimation usually produces a *better* estimate, so we will focus on the MLE approach.

2.2 Maximum Likelihood Estimation (MLE)

- Let X be a random variable with *pdf/pmf* $f(x;\theta)$ parameterized by $\theta \in \Theta$.
- Let $D = \{X_1, X_2, \dots, X_n\}$ be the observed data.
- *Maximum Likelihood Estimation (MLE)* uses the value of θ that maximizes the *likelihood*:

$$L(\theta) = P(X_1, X_2, \dots, X_n; \theta)$$

- The likelihood is written as a function of θ and treating the observed data as known
- When the observations are *independent*, this becomes:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{n} P(X_i;\theta) \\ &= \prod_{i=1}^{n} f(x_i;\theta) \end{aligned}$$

- The log-likelihood (under independence) becomes:

$$\log L(\theta) = \sum_{i=1}^{n} \log f(x_i;\theta)$$

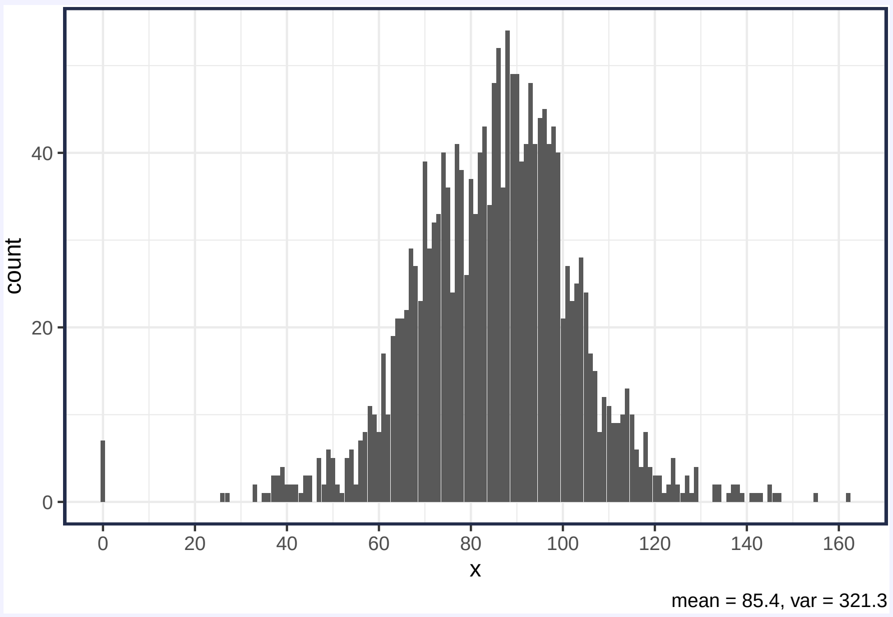
- And the MLE is:

$$\begin{aligned} \theta_{\text{MLE}} &= \arg\max_{\theta\in\Theta} L(\theta) \\ &= \arg\max_{\theta\in\Theta} \log L(\theta) \end{aligned}$$

Your Turn #1

In the disease outbreak example, we needed to estimate the distribution of reported symptoms of some disease *on a normal day*. Then *unusual or rare* counts could be considered anomalous and a potential indication of a disease outbreak or bio-attack.

Estimate the baseline density of ER counts.



1. Use a Poisson model.
2. Use a Negative Binomial model.
3. Use a Gaussian model.
4. What is the probability that we would get more than > 150 or < 50 counts on a *regular* day?

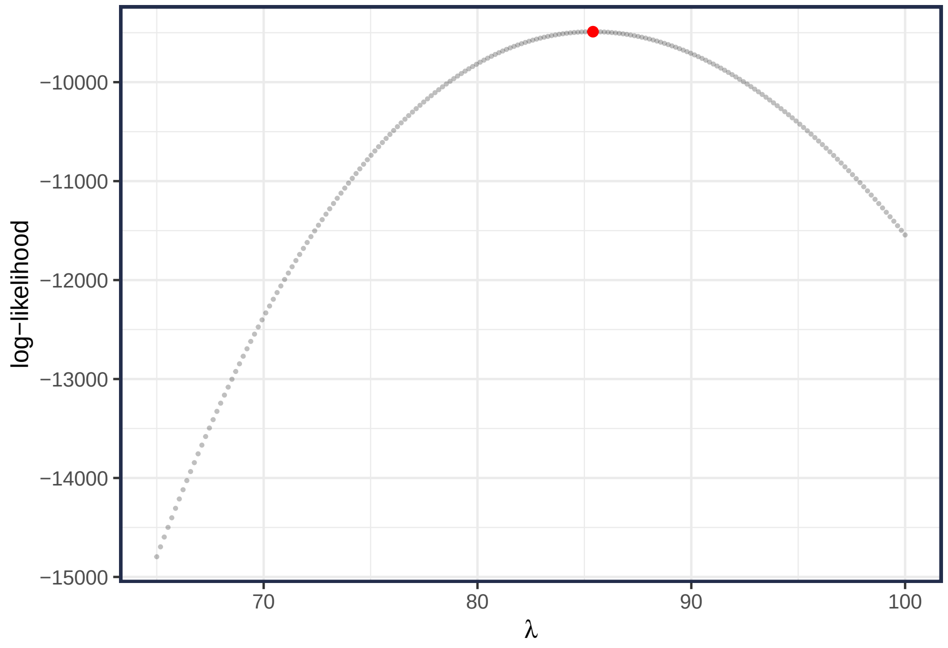
Note: Distribution Reference Sheet

2.2.1 Poisson MLE: Grid Search

- Notation:
  - $X \sim Pois(\lambda)$
  - $\Pr(X = x) = f(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$ where $f(x)$ is a pmf and $x = \{0, 1, \ldots\}$.
  - $E[X] = V[X] = \lambda$
- Calculate the log-likelihood over a range of λ values and choose the one that gives the maximum.

```
## Grid Search

#: The Poisson log-likelihood function
loglike_poisson <- function(lambda){
  # Note that the data, x,  is treated as known and put inside function
  #  to help clarify that the likelihood is a function of the parameters
  sum(dpois(x, lambda=lambda, log=TRUE))
}

#: Get sequence of lambda values
lam_seq = seq(65, 100, length=200)
nlam = length(lam_seq)

#: Calculate the log-likelihood for those lambda values
loglike = numeric(nlam)
for(i in 1:nlam){
  loglike[i] = loglike_poisson(lam_seq[i])
}

#: Alternative to loop using purrr::map
```

```r
loglike2 = map_dbl(lam_seq, loglike_poisson)
all.equal(loglike, loglike2)
#> [1] TRUE

#- best lambda via grid search:
lam_seq[which.max(loglike)]
#> [1] 85.4
```



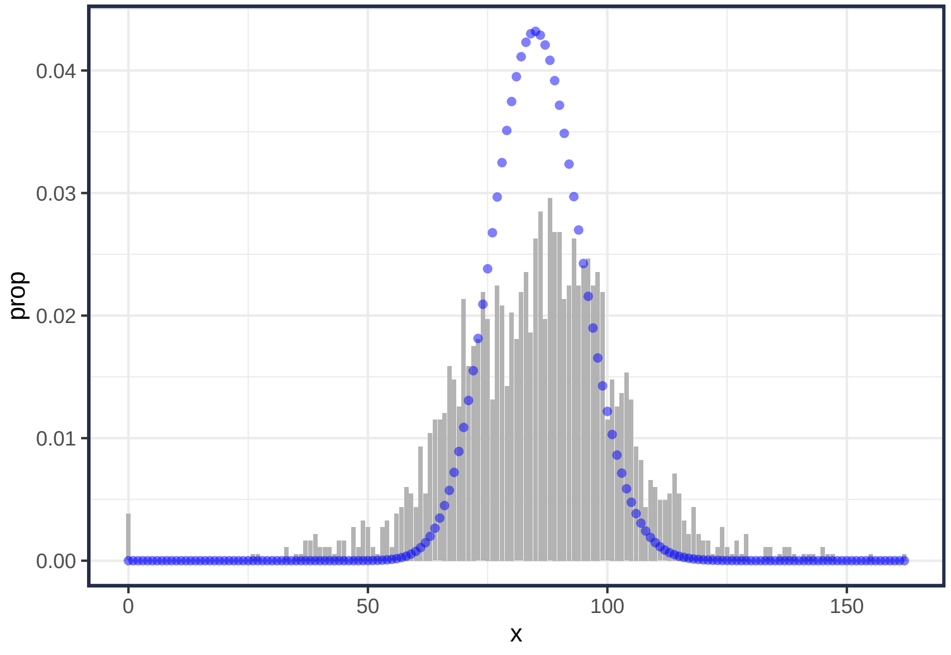
- The grid search gives $\hat{\lambda} = 85.402$
  - Searched 200 values between 65 and 100.

2.2.2 Poisson MLE: Calculus

Your Turn #2

Derive the MLE using Poisson model using calculus.

Estimated pmf using Poisson MLE



Note

- Does the Poisson model look like a good one?
- Where do you see lack of fit?

2.2.3 Poisson MLE: using `fitdistrplus` R package

```
library(fitdistrplus)
poisson_fit = fitdist(data=x, distr="pois", method="mle")
coef(poisson_fit)
#> lambda
#>  85.38
```

2.2.4 Negative Binomial: MLE

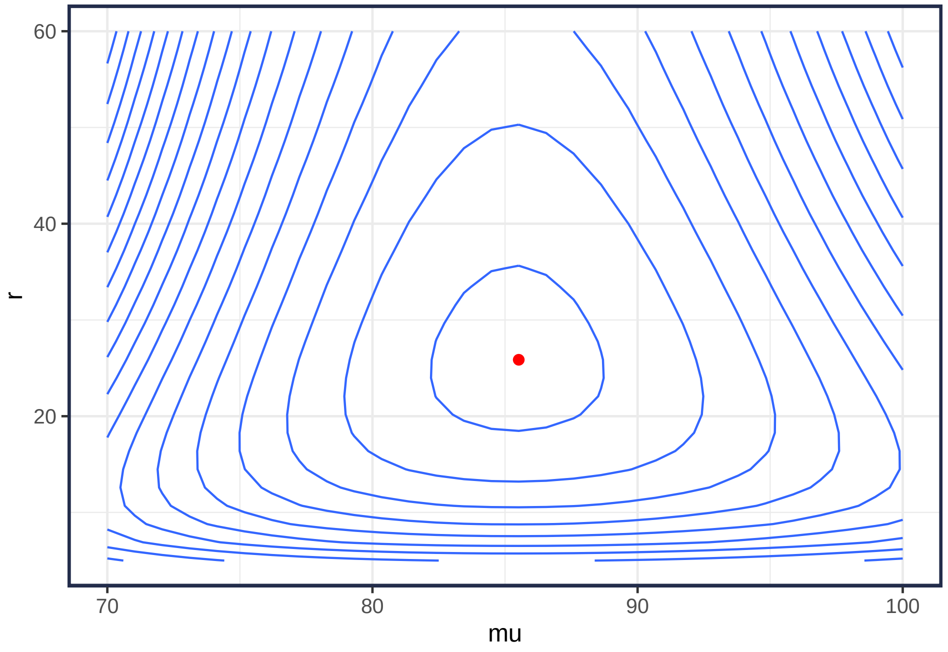
- The Negative Binomial distribution can help for modeling count data with overdispersion
- Notation:
  - $X \sim NBin(\mu, r)$ (using *mean* parameterization)
  - $\mu > 0, r > 0$
  - $E[X] = \mu, V[X] = \mu + \mu^2/r$
  - Mean representation: https://en.wikipedia.org/wiki/Negative_binomial_distribution

$$\Pr(X = x; r, \mu) = \frac{\Gamma(r+x)}{x!\Gamma(r)} \left(\frac{r}{r+\mu}\right)^r \left(\frac{\mu}{r+\mu}\right)^x$$

  - $n! = \Gamma(n-1)$
  - $\Gamma(n) = \int_0^\infty e^{-x} x^{n-1} dx$

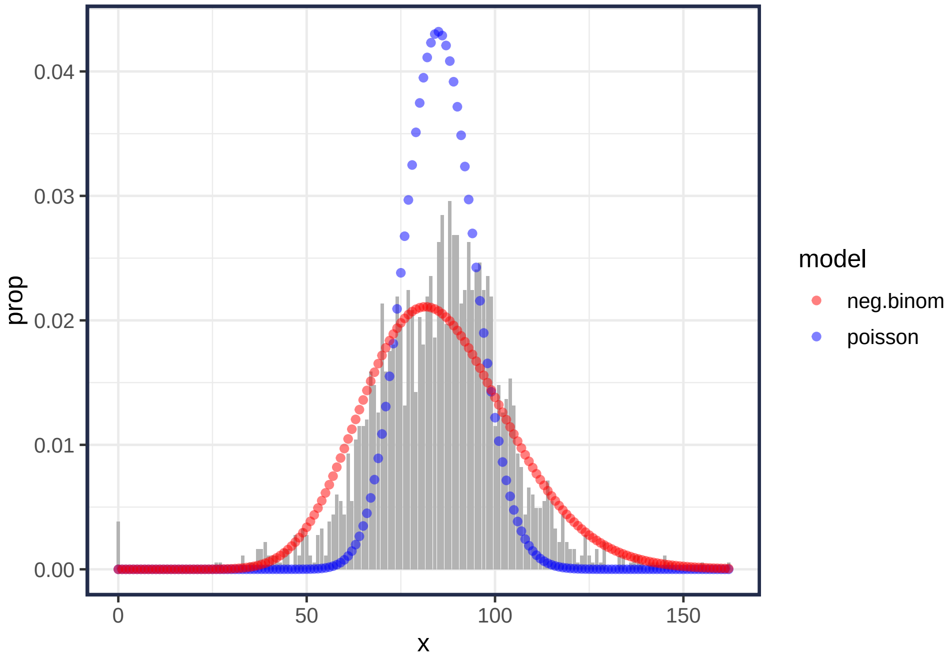
Your Turn #3

Use maximum likelihood to estimate $\theta = (r, \mu)$.



- Use R `fitdistrextra` package.

```
library(fitdistrplus)
nbinom_fit = fitdist(data=x, distr="nbinom", method="mle")
(nb_pars = coef(nbinom_fit))
#>  size    mu
#> 25.63 85.39
```



Note

- Does the Negative Binomial model look better than Poisson?
- Are there any remaining concerns?

2.2.5 Example: Gaussian/Normal

- Data are non-negative integers, not continuous, so Gaussian is clearly "wrong". But as the famous saying goes:

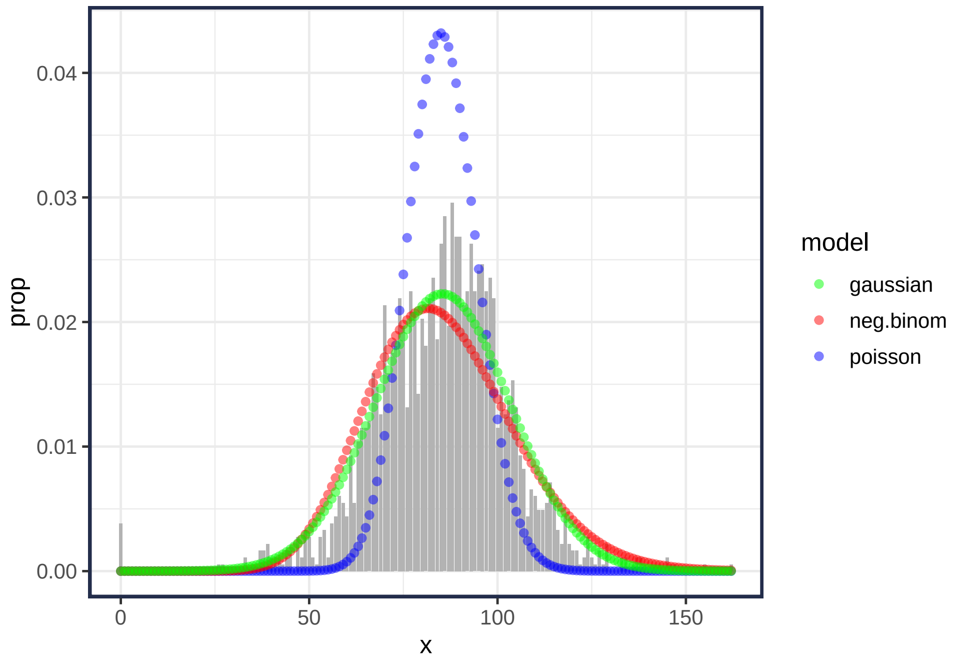
  "All models are wrong, but some are useful". - George E. P. Box

- Notation:
  - $X \sim N(\mu, \sigma)$
  - $\mu \in \mathbf{R}, \sigma > 0$
  - $E[X] = \mu, V[X] = \sigma^2$

Your Turn #4

Find the MLE for (μ, σ).

2.2.6 Comparison of Models



- Models:
  1. Poisson: $\lambda = 85.382$
  2. Neg.Binom: $\mu = 85.386,\ r = 25.627$
  3. Gaussian: $\mu = 85.382, \sigma = 17.919$

Your Turn #5

Which model do you choose? Why?

2.2.7 Density Evaluation

1. AIC/BIC
2. Hold out/test data (e.g., Cross-validation)
   - need a loss function (e.g., negative log-likelihood)
   - can consider other Goodness-of-Fit (GOF) metrics
   - bootstrap may not be best for non-parametric density estimators

2.3 Bayesian Estimation

In Bayesian analysis, the parameter(s) themselves are treated as *random variables*.

- In MLE, the parameters are assumed fixed, but unknown.

Prior knowledge, which is any information known about the parameter(s) *before the data are seen*, is captured in the *prior distribution*.

- Let $g(\theta)$ be the (possibly multivariate) prior pmf/pdf

Bayes theory gives us the *posterior distribution*,

$$f(\theta \mid D) = \frac{P(D \mid \theta) g(\theta)}{\int_{\theta \in \Theta} P(D \mid \theta) g(\theta) \, \mathrm{d}\theta} \propto \text{Likelihood x Prior}$$

- $P(D \mid \theta) = P(X_1, X_2, \ldots, X_n \mid \theta) =$ *likelihood*
- $\int_{\theta \in \Theta} P(D \mid \theta) g(\theta) \, \mathrm{d}\theta = P(D)$ is the *normalizing constant* (not function of θ).
- $f(\theta \mid D)$ is the *posterior distribution*, which contains the updated knowledge about the parameter(s).

2.3.1 Bayesian Point Estimation of parameter(s)

1. Posterior Mean

$$\hat{\theta}_{\mathrm{PM}} = E[\theta \mid D] = \int_{\theta \in \Theta} \theta f(\theta \mid D) \, \mathrm{d}\theta$$

2. MAP (Maximum a posteriori)

$$\begin{aligned}
\hat{\theta}_{\mathrm{MAP}} &= \arg\max_{\theta \in \Theta} f(\theta \mid D) \\
&= \arg\max_{\theta \in \Theta} P(D \mid \theta) g(\theta) \\
&= \arg\max_{\theta \in \Theta} \log P(D \mid \theta) + \log g(\theta)
\end{aligned}$$

3 Non-Parametric Density Estimation

3.1 Example: Old Faithful

The old faithful geyser in Yellowstone National Park is one of the most regular geysers in the park. The waiting time between eruptions is between 35 and 120 mins.

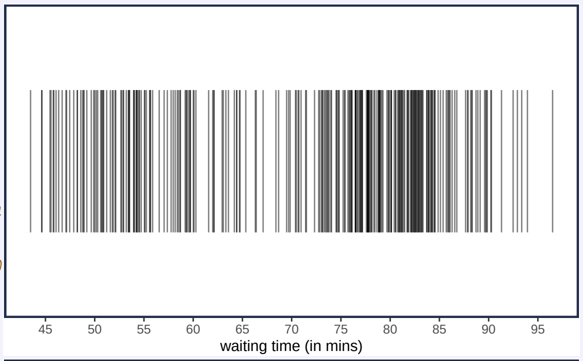
Live Streaming Webcam with eruption predictions

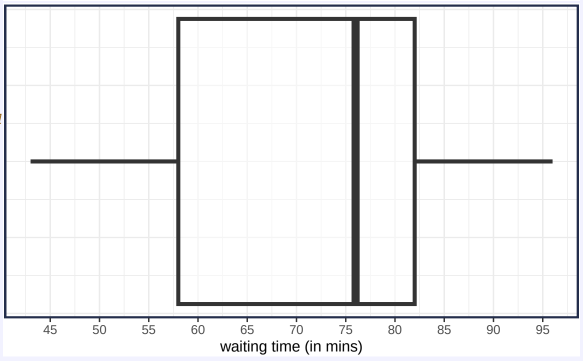
Because the nearby Yellowstone Lodge is nice and warm in the winter, and serves good ice cream in the summer, you may be distracted from stepping outside to watch the eruption. Let's see if we can determine the best time to leave the cozy lodge and go outside to watch the next eruption.

Your Turn #6 : Old Faithful

The data, summary statistics, and plots below represent a sample of *waiting times*, the time (in min) between Old Faithful eruptions.

```r
#: Load the Old Faithful data
wait = datasets::faithful$waiting

#: Calculate summary stats
length(wait)              # sample size
#> [1] 272
summary(wait)             # six number summa
#>    Min. 1st Qu.  Median    Mean 3rd Qu.
#>    43.0    58.0    76.0    70.9    82.0
mean(wait)                # mean
#> [1] 70.9
sd(wait)
#> [1] 13.59
median(wait)
#> [1] 76
quantile(wait, probs=c(.25,.50,.75))   # qu
#> 25% 50% 75%
#>  58  76  82
```



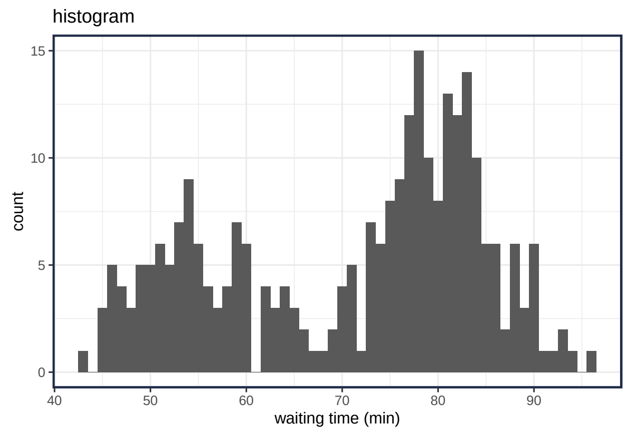


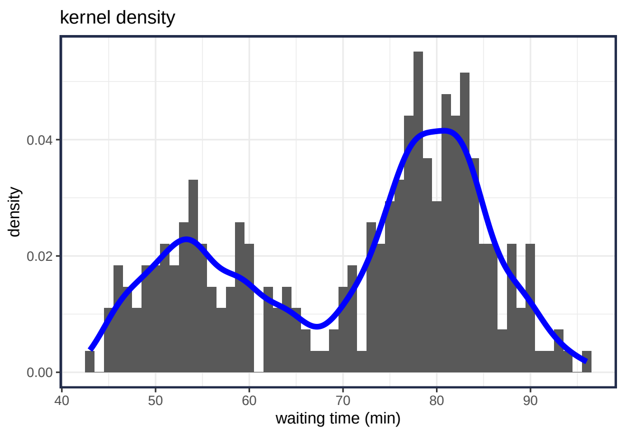
1. What can you say about the shape of the distribution?
2. Would a Gaussian (i.e., Normal) Distribution be a good choice for modeling the distribution of these data?
3. What would you recommend?

More detail can be obtained by examining the histograms:

```r
#: Put data into a data.frame/tibble for use with ggplot
wait_df = tibble(wait)

#: Make a ggplot object
pp = ggplot(wait_df, aes(x=wait)) + xlab("waiting time (min)")
```

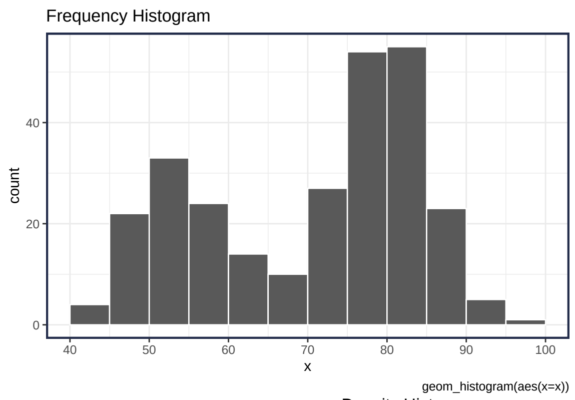
```r
#: Histogram
pp + geom_histogram(binwidth = 1) + ggtitle("histogram")

#: overlay kernel density plot
pp +
  geom_histogram(binwidth = 1, aes(y=after_stat(density))) +  # *density* histogram
  geom_density(bw=2, linewidth=2, color="blue") + # kernel density estimate (KDE)
  ggtitle("kernel density")
```

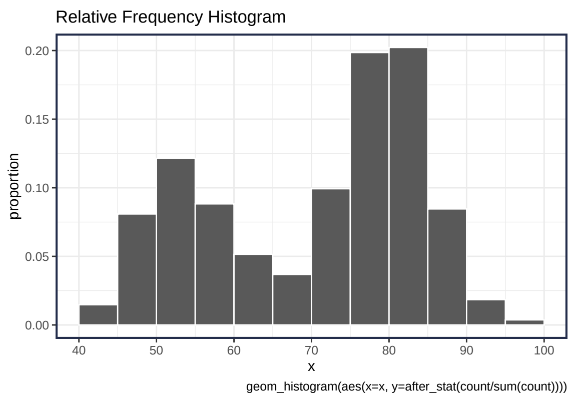


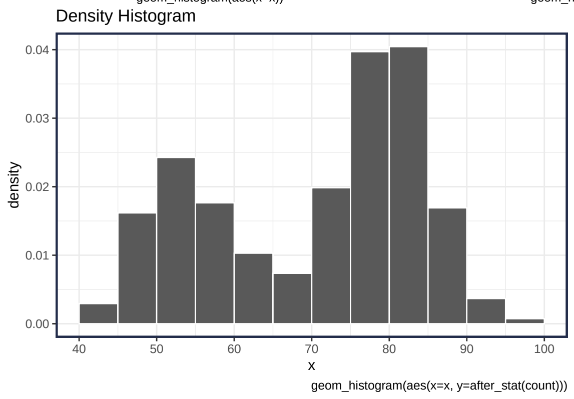


3.2 Histograms

- A histogram is a visual representation of a *piece-wise constant* function.
- There are three primary types of histograms: **frequency**, **relative frequency**, and **density**.







3.3 Density Histograms

- A *density histogram* is a special type of histogram that has the property of being a proper pdf: non-negative and integrate to 1.
  - $f(x) \geq 0 \quad \forall x$ and $\int f(x)dx = 1$

Histograms estimate the density as a piecewise constant function.

$$\hat{f}(x) = \sum_{j=1}^{J} b_j(x)\,\hat{\theta}_j$$

where $b_j(x) = \mathbb{1}(x \in \text{bin}_j)/h_j$ and

- $\text{bin}_j = [t_j, t_{j+1})$
- $t_1 < t_2 < \ldots < t_J$ are the break points for the bins
- $h_j = [t_j, t_{j+1}) = t_{j+1} - t_j$ is the **bin width** of bin j
  - bin widths do *not* have to be equal
- $\text{bin}_j \cap \text{bin}_k = \emptyset$

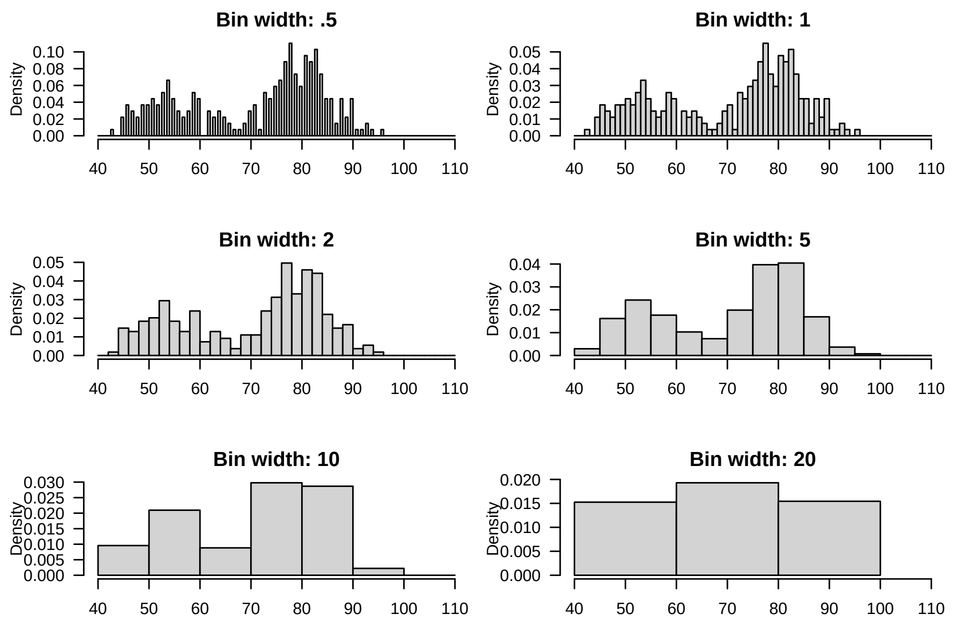
3.3.1 Estimating Density Histograms

- Observe data $D = \{X_1, X_2, \ldots, X_N\}$
- specify bins
- Denote n_j as the number of observations in bin j
- $\hat{\theta}_j = \hat{p}_j = n_j/N$ is the usual (MLE) estimate
  - Given binning, *multinomial distribution*

3.4 Local Properties of Histograms

- Histograms can be thought of as a *local* method of density estimation.
  - Points are local to each other if they fall in the same bin
  - Local is determined by bin break points
- But this has some issues:
  - Some observations are “closer” to observations in a neighboring bin
  - Estimate is not *smooth* (but many true density functions can often be assumed smooth)
  - **Bin shifts can have a big influence on the resulting density**

3.4.1 Old Faithful: Sensitivity to bin width



3.4.2 Histogram Neighborhood

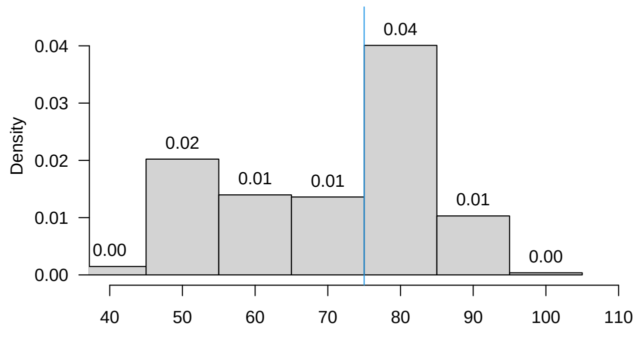
Consider estimating the density at a location x

- For a regular histogram (with bin width h), the MLE density is

$$\hat{f}(x) = \frac{n_j}{nh} \qquad \text{for } x \in \text{bin}_j$$

  which is a function of the number of observations in bin j.

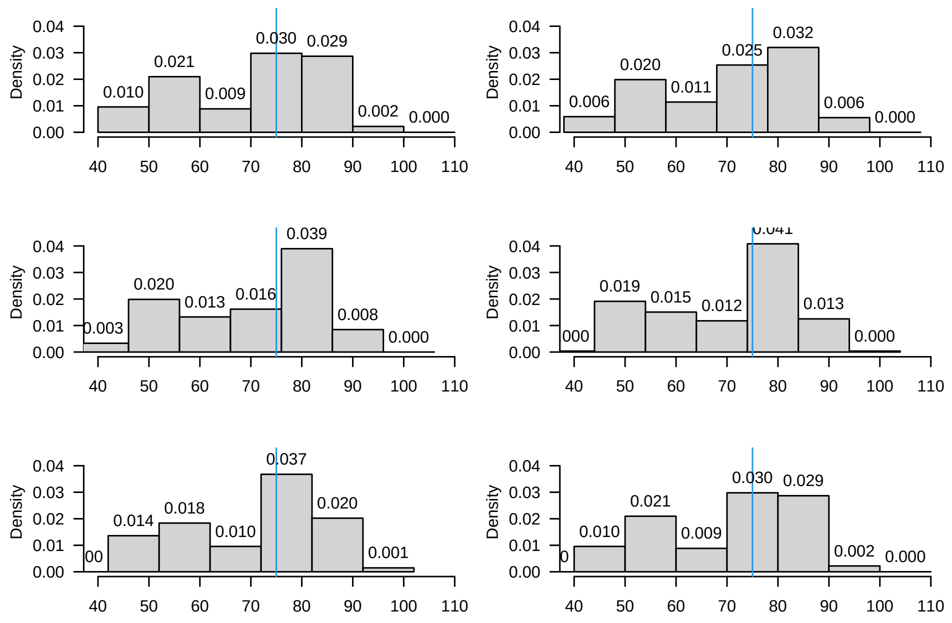
- But how do you feel if x is close to the boundary (e.g., $x = 75$)?



3.4.3 Old Faithful: Sensitivity to bin shifts / anchor points

Note

What density should we use for at $x = 75$?


Density
0.00
0.01
0.02
0.03
0.04
40
50
60
70
80
90
100
110
0.010
0.021
0.009
0.030
0.029
0.002
0.000
0.006
0.020
0.011
0.025
0.032
0.006
0.000
0.003
0.020
0.013
0.016
0.039
0.008
0.000
000
0.019
0.015
0.012
0.041
0.013
0.000
00
0.014
0.018
0.010
0.037
0.020
0.001
0
0.010
0.021
0.009
0.030
0.029
0.002
0.000

4 Kernel Density Estimation (KDE)

Kernel Density Estimation is an improvement on density histograms:

- Removes the bin anchor point parameter
- Allows more flexible definition of "neighborhood"

4.1 Local Density Estimation - Moving Window

Consider again estimating the density at a location x

- Regular Histogram (with midpoints m_j and bin width h)

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{\mathbb{1}\left(|x_i - m_j| \le \frac{h}{2}\right)}{h} \qquad \text{for } x \in \text{bin } j$$

- Consider a **moving window** approach

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{\mathbb{1}\left(|x_i - x| \le \frac{h}{2}\right)}{h}$$

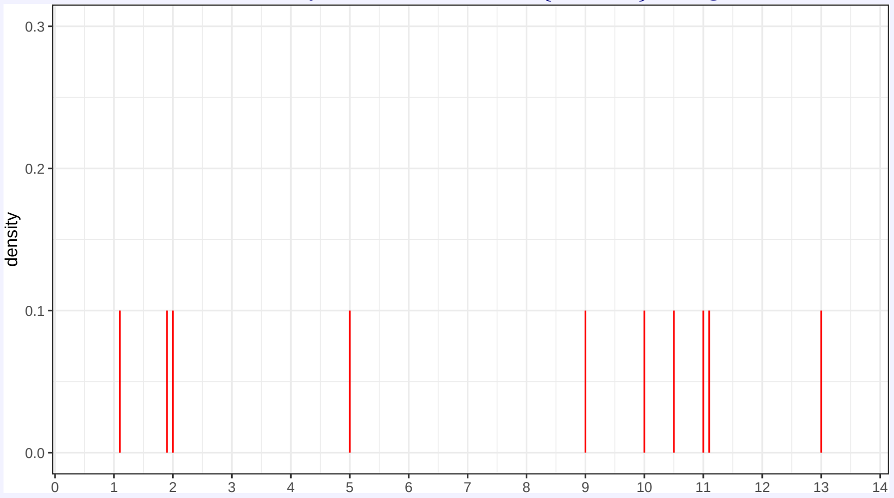
  This gives a more pleasing definition of local by centering a bin at x, to calculate the density at x.

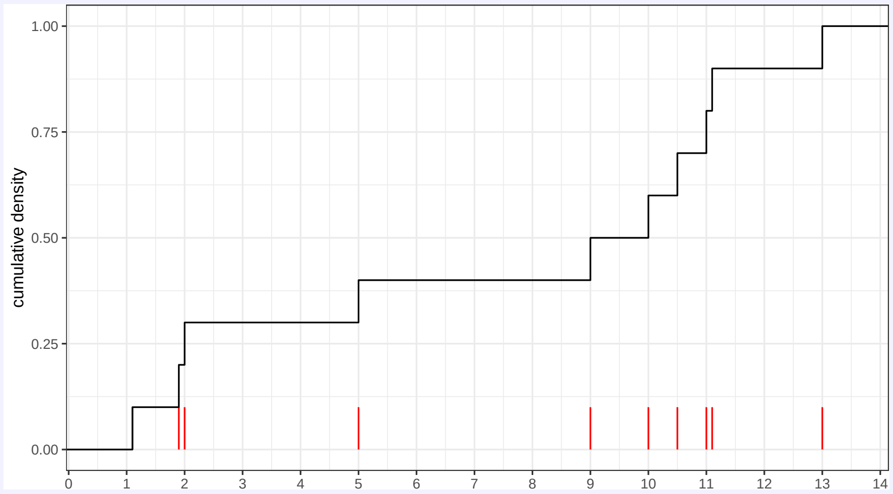
- Equivalently, this estimates the derivative of ECDF

$$\hat{f}(x) = \frac{F_n(x + h/2) - F_n(x - h/2)}{h}$$

Your Turn #7

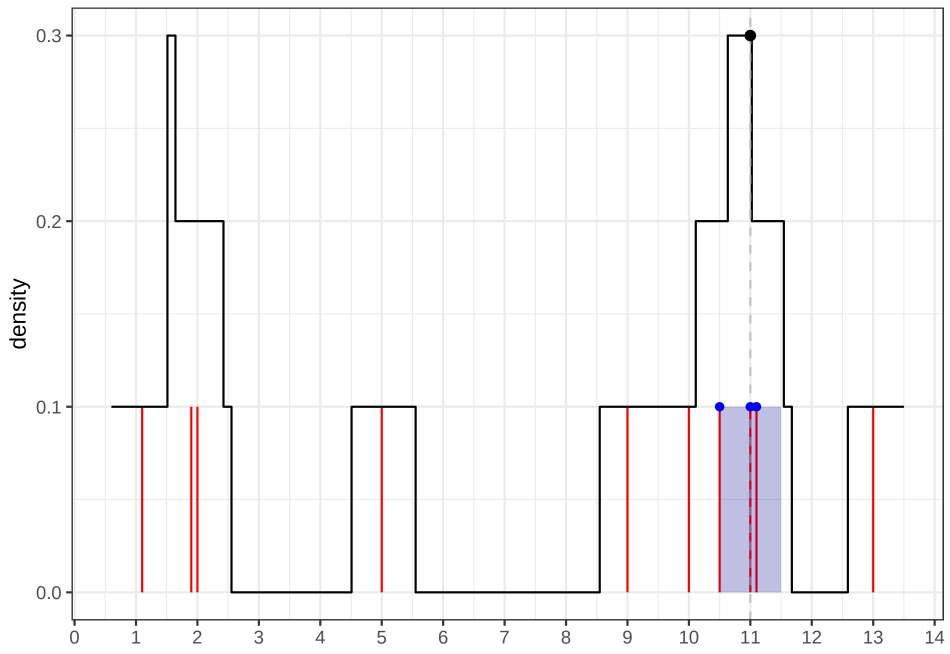
Consider a dataset $D = \{1.1, 1.9, 2, 5, 9, 10, 10.5, 11, 11.1, 13\}$. Use a *moving window* (*uniform kernel*) estimator to estimate the density at locations $x_0 = \{2, 7, 11\}$ using $h = 1$.





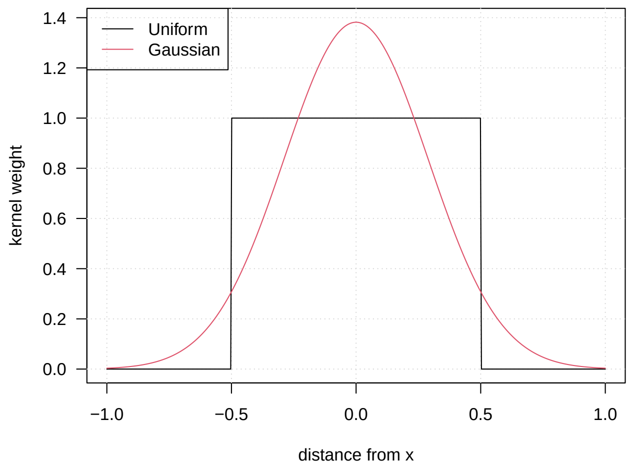
Note

Demo of moving windows (gif)
Demo of moving windows (movie)



4.2 Kernels

- The moving window approach looks better than a histogram with the same bin width, but it is still not smooth
- Instead of giving every observation in the window the same weight, we can assign a weight according to its distance from x



- More generally, the weights $K_h(u) = h^{-1}K\left(\frac{u}{h}\right)$ are called kernel functions
- Thus, a kernel density estimator is of the form

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

where the smoothing parameter h is called the bandwidth and controls how fast the weights decay as a function of the distance from x

Kernels

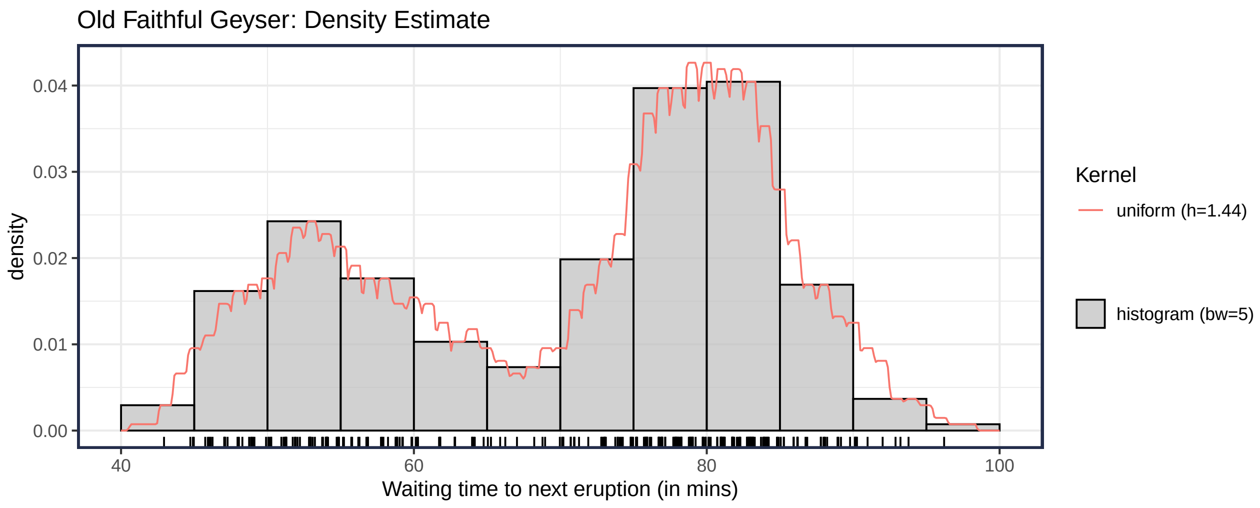
There are several uses of *kernels* in this course. Generally speaking a kernel $K(x, y) \in \mathbf{R}$ is a measure of *similarity* between the two points/vectors x and y.

- For KDE its common to set the kernel to be a function of distance $K(|x-y|)$ or $K(\|x-y\|)$

4.2.1 Uniform/Rectangular kernel

- The moving window uses a *uniform* (or rectangular) kernel

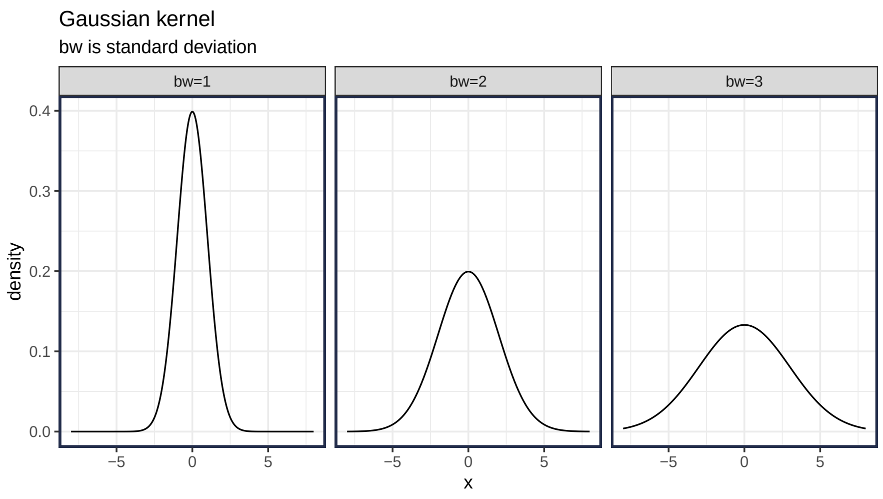
$$K_h^{\text{unif}} = \frac{\mathbb{1}\left(|x_i - x| \le \frac{h}{2}\right)}{h}$$

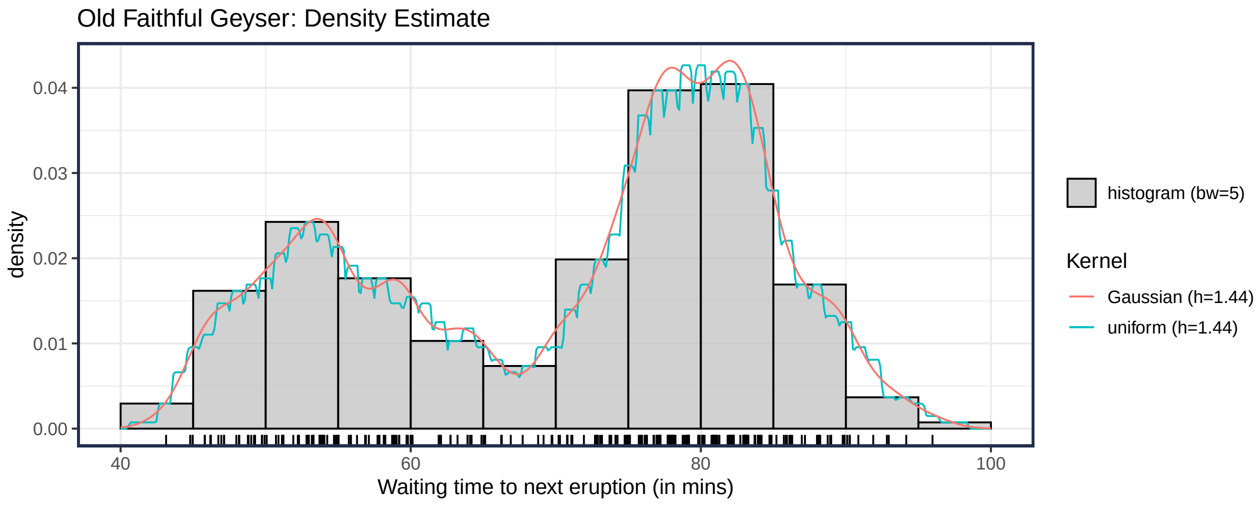


4.2.2 Gaussian/Normal kernel

- The most popular kernel is the **Gaussian Kernel**

$$\begin{aligned} K_h^{\text{gauss}}(u) &= \frac{1}{h\sqrt{2\pi}} \exp\left(-\frac{u^2}{2h^2}\right) \\ &= h^{-1}K(u/h) (\text{where } K(\cdot) \text{ is standard normal pdf}) \end{aligned}$$



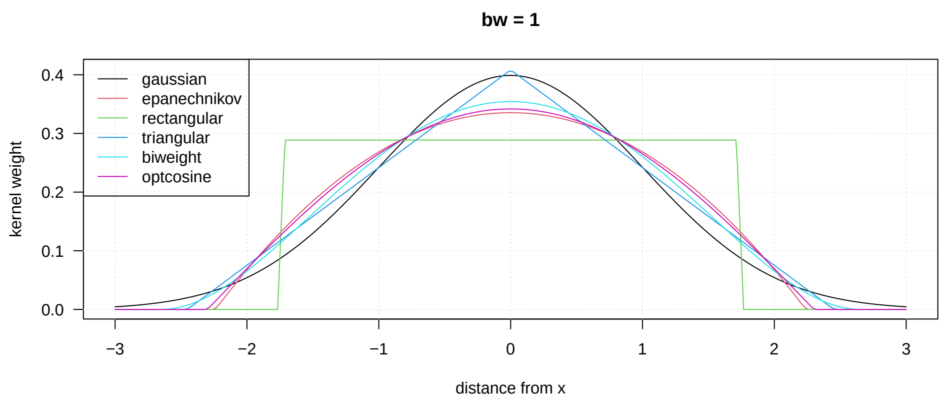


Note

Demo of Gaussian KDE (gif)
Demo of Gaussian KDE (movie)

4.2.3 Other kernels

- There are many choices for kernels



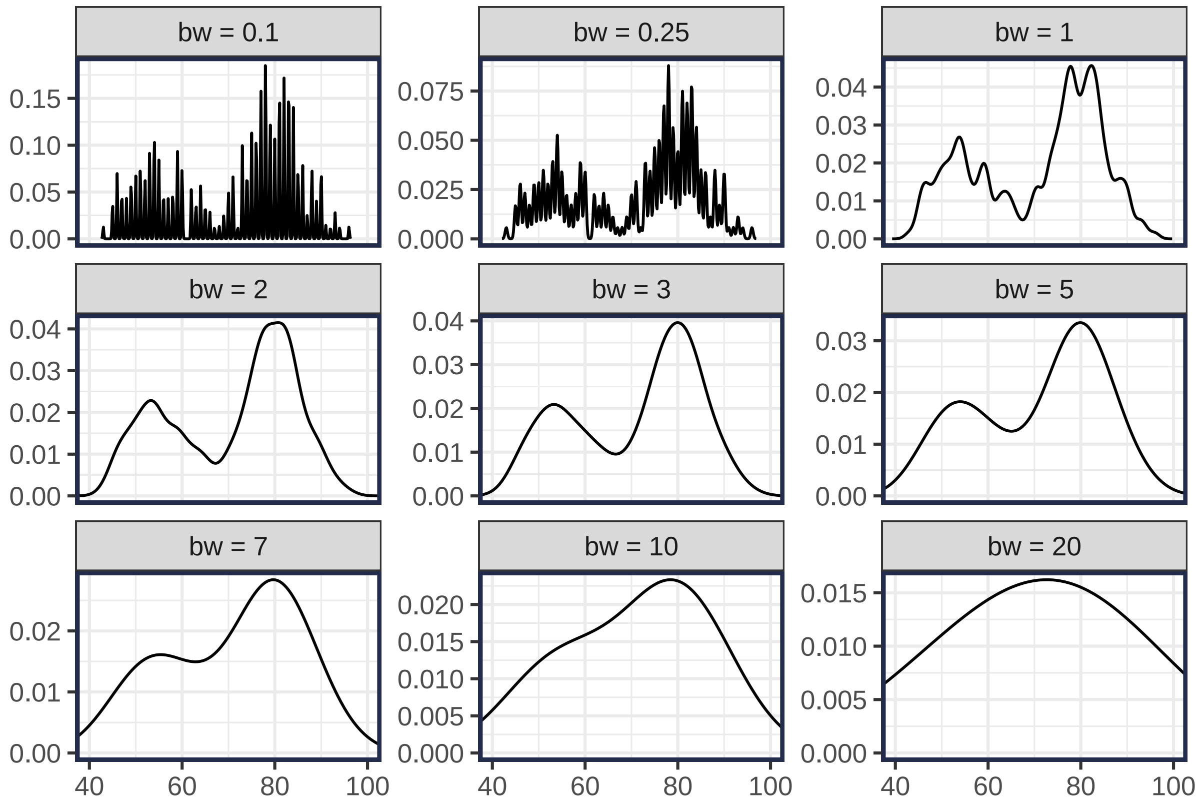
4.2.4 Kernel Properties

A kernel is usually considered to be a symmetric probability density function (pdf):

- $K_h(u) \geq 0$ (non-negative)
- $\int K_h(u)\, du = 1$ (integrates to one)
- $K_h(u) = K_h(-u)$ (symmetric about 0)
- Notice that if the kernel has compact support, so does the resulting density estimate
- The Gaussian kernel is the most popular, but has infinite support
  - This is good when the true density has infinite support
  - However, it can require more computation than kernels with finite support
  - But it is familiar and properties are well understood

4.3 Bandwidth

- The bandwidth parameter, h controls the amount of smoothing
  - It is equivalent to the bin size parameter for histograms
- What happens to the density estimate when $h \uparrow \infty$?
- What happens to the density estimate when $h \downarrow 0$?
- **The choice of bandwidth is much more important than the choice of kernel**
- Bandwidth is usually defined as the standard deviation of the kernel
  - but not always, so check the implementation.



4.4 KDE Demo

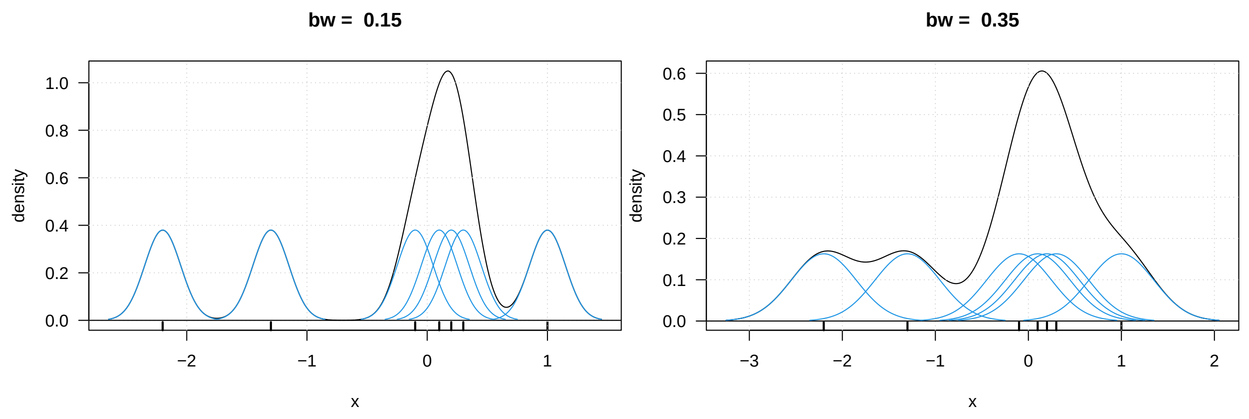
Note

Check out the shiny app for visualizing the effects of the smoothing parameters on density histograms and kernel density estimation.

4.5 Another Perspective

- We have described KDE as taking a local density around location x
- An alternative perspective is to view KDE as an n component *mixture model* with mixture weights of $1/n$

$$\begin{aligned}\hat{f}(x) &= \frac{1}{n}\sum_{i=1}^{n} K_h(x_i - x) \\ &= \sum_{j=1}^{n} \frac{1}{n} f_i(x) && (f_i(x) = K_h(x_i - x))\end{aligned}$$



4.6 Bandwidth Selection

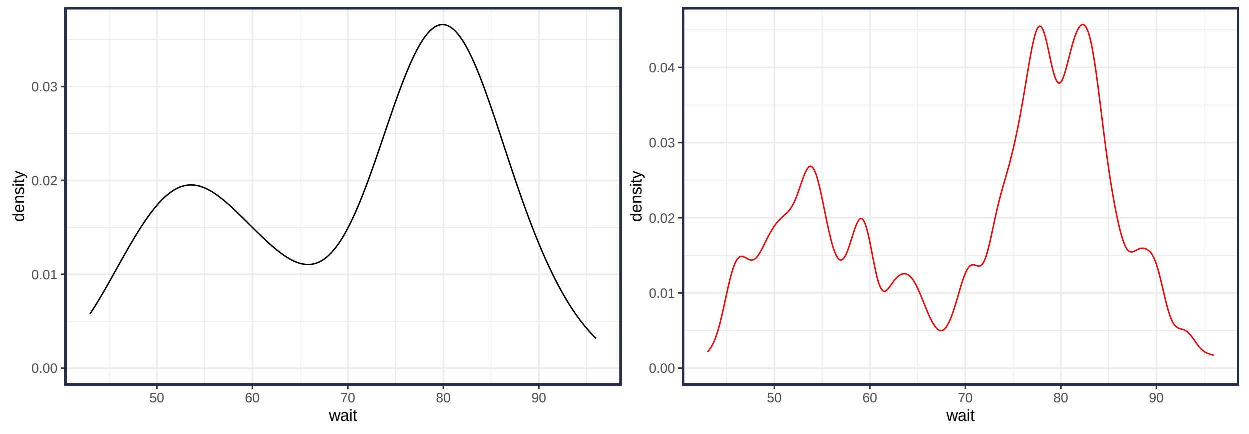
- The best bandwidth is the one that is best for your problem
  - E.g., *The best bandwidth for density estimation may not be best for classification*
  - Choosing a bandwidth that is visually appealing may be suitable
- For density estimation, we want a "Goldilocks" bandwidth; one that produces a density that is not *too wiggly* nor *too smooth*.
- Or to state it plainly, when the goal is density estimation, we want to find the bandwidth that gives a density estimate closest to the true density. But,
  a. We don't know the true density
  b. There are many ways to define "close"
- For this class, we will not go into the details other than to say most bandwidth selection methods are based on *plug-in* or *cross-validation*.

4.6.1 Bandwidth Selection in R

There are a few KDE functions in R.

1. `density()` in base R is used for ggplot's `geom_density()`

```
ggplot() + geom_density(aes(x=wait))   # runs bw.nrd0()
ggplot() + geom_density(aes(x=wait), bw=1, color="red")
```



See the R help: `?bw.nrd0` to get a description of several bandwidth selection methods.

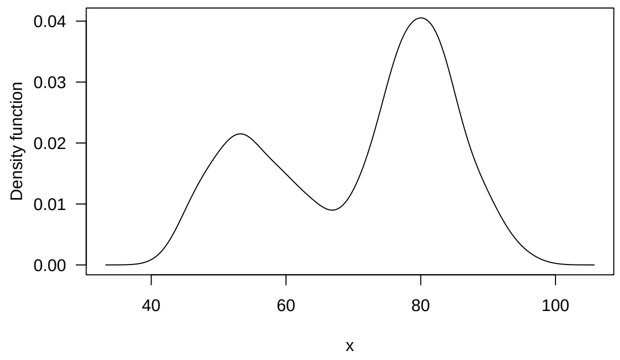
```
bw.nrd0(wait)     # get default bandwidth
#> [1] 3.988
c(bw.nrd0 = bw.nrd0(wait), bw.nrd=bw.nrd(wait), bw.bcv=bw.bcv(wait),
  bw.SJ = bw.SJ(wait), bw.ucv=bw.ucv(wait))
#> bw.nrd0  bw.nrd  bw.bcv   bw.SJ  bw.ucv
#>   3.988   4.696   2.598   2.504   2.658
```

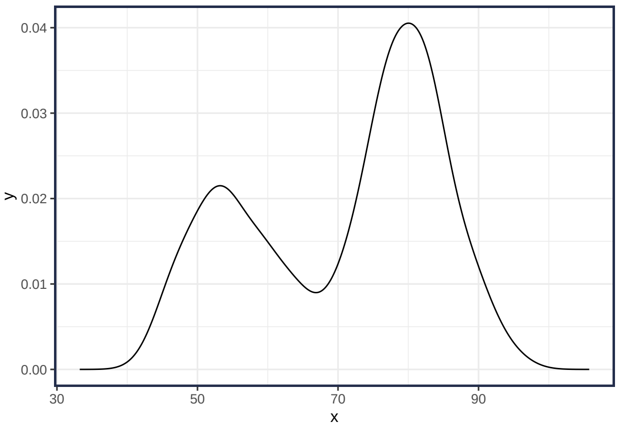
Note

In the R function `density()`, the bandwidth refers to the standard deviation of the kernel. Thus the `rectangular()` (i.e., uniform) kernel has a standard deviation of width$/\sqrt(12)$.

2. I recommend using the function `kde()` in the `ks` package.
   - It allows multivariate kernel estimation and bandwidth selection

```
library(ks)
f.kde = kde(wait)
f.kde$h                       # The bandwidth parameter is denoted by h
#> [1] 2.636
plot(f.kde, las=1)            # plots the "kde" object

#: Using ggplot2
tibble(x = f.kde$eval.points, y=f.kde$estimate) %>%
  ggplot(aes(x,y)) + geom_line()
```

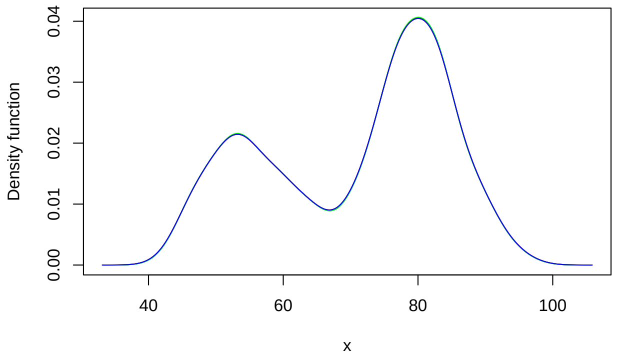




- There are several bandwidth selection methods
  - `hpi()`: plug-in
  - `hlscv()`: least squares cross-validation
  - `hscv()`: smoothed cross-validation
  - `hucv()`: unbiased cross-validation
  - Details
- See Chapter 3 of the textbook (pdf link)

```
h1 = hpi(wait)
h2 = hlscv(wait)
#> Warning in hlscv(wait): Data contain duplicated values: LSCV is not
#> well-behaved in this case
h3 = hscv(wait)
h4 = hucv(wait)
#> Warning in hlscv(...): Data contain duplicated values: LSCV is not well-behaved
#> in this case
```

```r
c(hpi=h1, hlscv=h2, hscv=h3, hucv=h4)
#>    hpi hlscv  hscv  hucv
#> 2.636 2.665 2.581 2.665

plot(kde(wait, h=h1))
plot(kde(wait, h=h2), add=TRUE, col="red")
plot(kde(wait, h=h3), add=TRUE, col="green")
plot(kde(wait, h=h4), add=TRUE, col="blue")
```



Note

Note: these are functions for selecting bandwidths in univariate data. The multivariate equivalents use uppercase H, e.g. `Hpi()`.

5 Bivariate Gaussian/Normal Distribution

A bivariate normal random variable ($\mathbf{X} = X_1, X_2$) has the joint pdf:

$$f(\mathbf{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^{\mathsf{T}}\Sigma^{-1}(\mathbf{x}-\mu)\right]$$

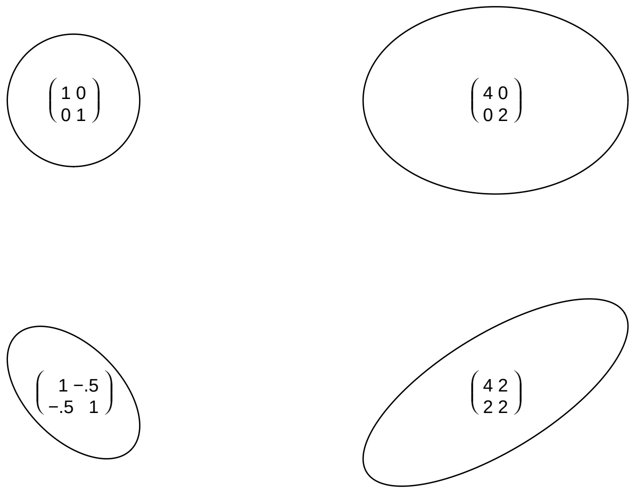
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

$$\begin{aligned}\Sigma &= \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) \\ Cov(X_2, X_1) & Var(X_2) \end{bmatrix} \\ &= \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}\end{aligned}$$

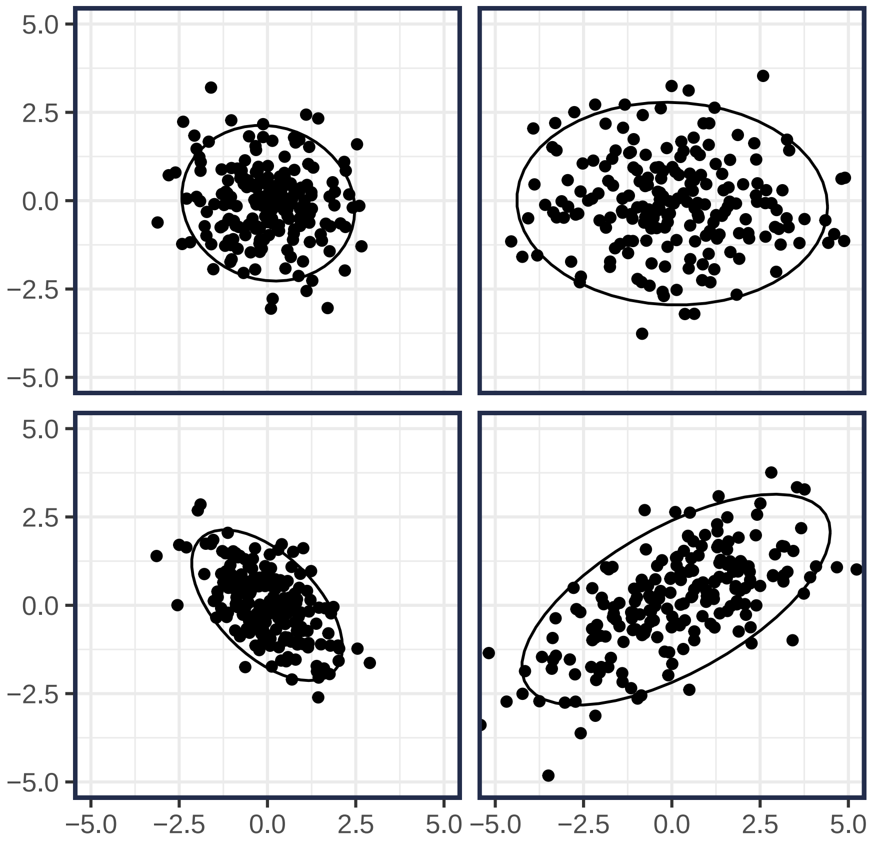
- $|\Sigma|$ is the *determinant* of the variance-covariance matrix Σ
  - $\det(\Sigma) = \prod_{j=1}^{2} \lambda_i$ where λ are eigenvalues of Σ

$$\left|\begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}\right| = \sigma_1^2\sigma_2^2 - \sigma_{12}^2$$

- Σ controls the orientation, shape, and volume of the density contours



$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & -.5 \\ -.5 & 1 \end{pmatrix}$
$\begin{pmatrix} 4 & 2 \\ 2 & 2 \end{pmatrix}$



5.1 Independence and EDF

Your Turn #8 : Number of Parameters

How many parameters needs to be estimated in a bivariate Gaussian model?

- It is a special property of the multi-variate Gaussian distribution that if the random variables are uncorrelated, then they are also independent.
  - So if the off-diagonal terms are zero ($\sigma_{12} = 0$), then the elements of the random vector are *independent*
- For bivariate data, the random vector $\mathbf{X} = [\mathbf{X_1}, \mathbf{X_2}]$ has two elements.
  - Marginal pdf $f_1(X_1)$, $f_2(X_2)$
  - Joint pdf $f_{12}(\mathbf{X}) = \mathbf{f_{2|1}(X_2 \mid X_1)f_1(X_1) = f_{1|2}(X_1 \mid X_2)f_2(X_2)}$
  - If X_1 and X_2 are *independent*, then $f_{12}(\mathbf{X}) = \mathbf{f_1(X_1)f_2(X_2)}$

- If the data are independent and follow a Gaussian distribution then

$$\mathbf{X} \sim \mathcal{N}(\mu, \mathbf{\Sigma}) \qquad \mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

$$\begin{aligned} f_{12}(\mathbf{X}) &= f_1(X_1) f_2(X_2) \\ &= \left[\frac{1}{\sigma_1 \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{(X_1-\mu_1)}{\sigma_1}\right)^2} \right] \left[\frac{1}{\sigma_2 \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{(X_2-\mu_2)}{\sigma_2}\right)^2} \right] \\ &= \frac{1}{2\pi\sigma_1\sigma_2} e^{-\frac{1}{2}\left[\left(\frac{(X_1-\mu_1)}{\sigma_1}\right)^2 + \left(\frac{(X_2-\mu_2)}{\sigma_2}\right)^2\right]} \end{aligned}$$

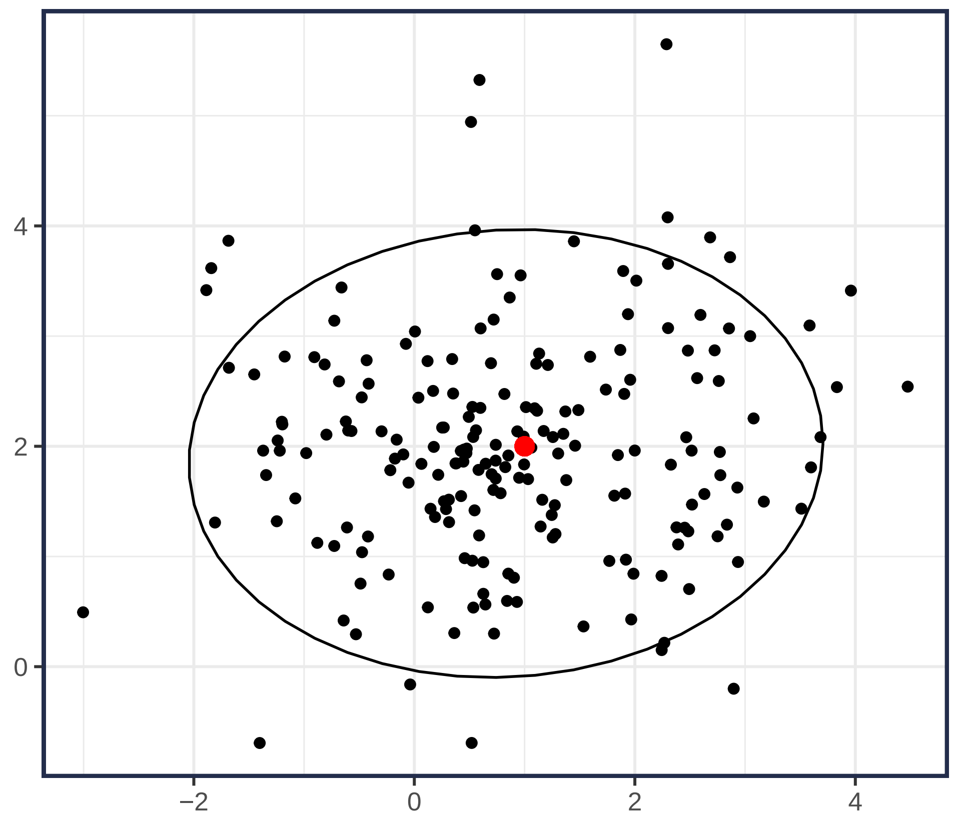
- So, if you *choose* to use a Gaussian and *choose* independence, then you can estimate the parameters for each marginal density independently.
  - Will be important knowledge for multivariate KDE and Gaussian Mixture Models (GMMs)
  - And you only have four parameters to etimate: $(\mu_1, \sigma_1), (\mu_2, \sigma_2)$

5.2 Parameter Estimation

Just like a one-dimensional Gaussian, we need to estimate the equivalent to the *mean* and *variance*

- For bivariate data, the mean has two elements (i.e., two element vector)
- For bivariate data, the variance has four elements (i.e., two by two symmetric matrix)
  - Note the are only 3 *unique* values since the off-diagonal are equal

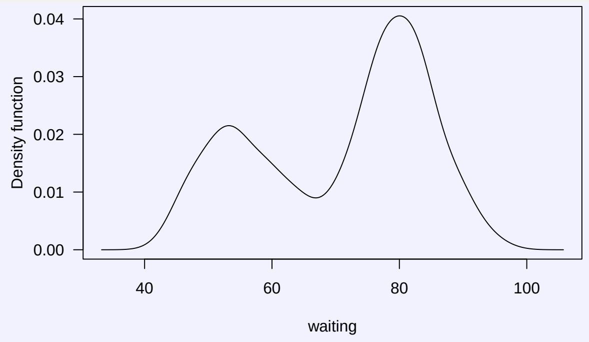
```r
library(mvtnorm)
n = 200
Sigma = matrix(c(2, 0, 0, 1), nrow=2)
mu = c(1,2)
X = rmvnorm(n = 200, mean= mu, sigma = Sigma) %>% as_tibble()

ggplot(X, aes(V1, V2)) +
  geom_point() +
  geom_point(aes(x=mu[1], y=mu[2]), color="red", size=3) +
  stat_ellipse(level=0.95) +
  coord_equal() +
  theme(axis.title=element_blank())
```


4
2
0
−2
0
2
4

6 Multivariate Kernel Density Estimation

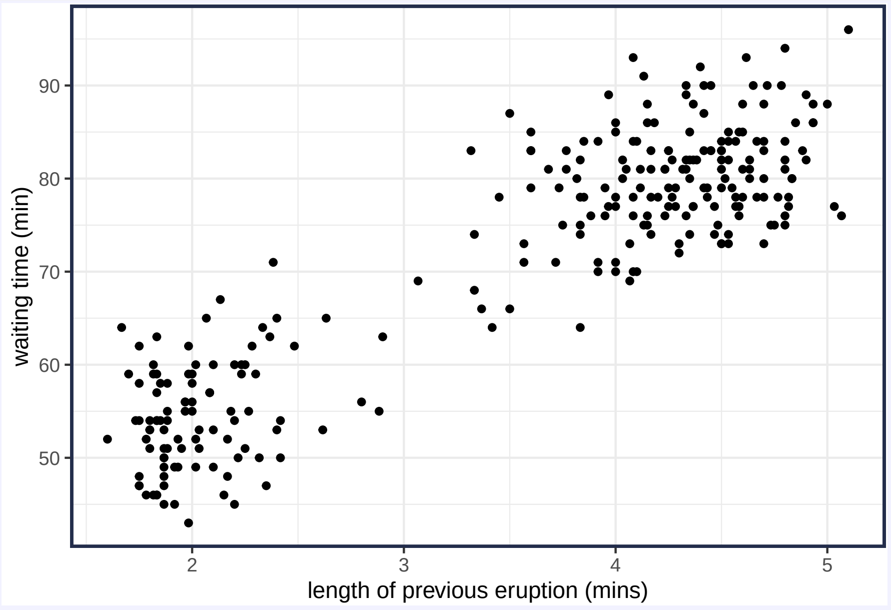
Your Turn #9 : The Return to Old Faithful

```r
#: Univariate density
f_wait = kde(wait)
plot(f_wait, las=1, xlab="waiting")
```



Did I forget to mention what we have additional information about old faithful eruptions? It turns out that we also have information on the *duration of the previous eruption*.

```r
#: Load the Old Faithful data
X = datasets::faithful

#: Scatterplot
ggplot(X) + geom_point(aes(eruptions, waiting)) +
  labs(x = "length of previous eruption (mins)", y = "waiting time (min)")
```



1. What patterns do you see?
2. Think about how to estimate this *bivariate* density.
3. Would a 2D Gaussian be appropriate?

There are three primary approaches to multivariate (d dimensional) KDE:

1. Multivariate kernel

- (e.g., $K(u) = N(\mathbf{0}, \Sigma)$)

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K(x, x_i)$$
$$= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}n}\sum_{i=1}^{n} \exp\left(-\frac{1}{2}(x-x_i)^{\mathsf{T}}\Sigma^{-1}(x-x_i)\right)$$

- Let $\Sigma = h^2 A$ where $|A| = 1$, thus $|\Sigma| = h^{2d}$

$$\hat{f}(x) = \frac{1}{(2\pi)^{d/2}h^d n}\sum_{i=1}^{n} \exp\left(-\frac{1}{2}(x-x_i)^{\mathsf{T}}A^{-1}(x-x_i)\right)$$

2. Product Kernel
   - Specifies $\Sigma = \mathbf{h}^2 I_d$.
   - The kernel is independent Gaussian.
   - Estimate bandwidth for each dimension.

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K(x, x_i)$$
$$= \frac{1}{n}\sum_{i=1}^{n}\left(\prod_{j=1}^{d} K_{h_j}(x_j - x_{ij})\right)$$

3. Independence

- Estimate a KDE separately for each dimension.
  - Multiply the resulting density estimates.
  - This has strongest constraints (least complex).

$$\hat{f}(x) = \prod_{j=1}^{d}\hat{f}_j(x)$$
$$= \prod_{j=1}^{d}\left(\frac{1}{n}\sum_{i=1}^{n} K_{h_j}(x_j - x_{ij})\right)$$

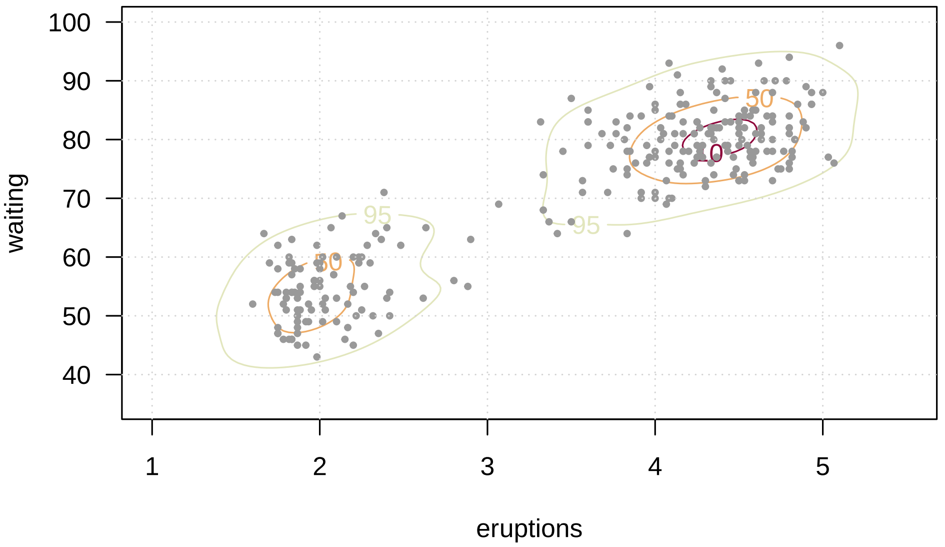
6.1 Multivariate KDE with `kde`

```
head(X)      # first 6 rows of the full old faithful data
#> # A tibble: 6 x 2
#>   eruptions waiting
#>       <dbl>   <dbl>
#> 1       3.6      79
#> 2       1.8      54
#> 3       3.33     74
#> 4       2.28     62
#> 5       4.53     85
#> 6       2.88     55
```

```
(H1 = Hscv(X))       # smoothed cross-validation bw estimator
```

```
#>         [,1]   [,2]
#> [1,] 0.0601  0.511
#> [2,] 0.5110 12.408
```

```r
f1 = kde(X, H=H1)    # use H for multivariate data
plot(f1,
     cont = c(10, 50, 95),                          # set contour levels
     # display = "filled.contour",                  # use filled contour
     las=1, xlim = c(1.0, 5.5), ylim=c(35, 100))  # set asthetics
points(X, pch=19, cex=.5, col='grey60')            # add points
grid()                                              # add grid lines
```



Above is the *unconstrained* Gaussian kernel.

- The Kernel is a MV Normal, $K(\mathbf{X}; H)$, with variance-covariance matrix

$$\Sigma = H = \begin{bmatrix} H_{11} & H_{12} \\ H_{12} & H_{22} \end{bmatrix}$$

- The diagonal terms correspond to the **variances** in each dimension
  - Note: take square root to compare with univariate bandwidth
- The off-diagonal term corresponds to the *correlation*: $H_{12} = \rho h_1 h_2$, where $h_i = \sqrt{H_{ii}}$

6.1.1 Product Kernel

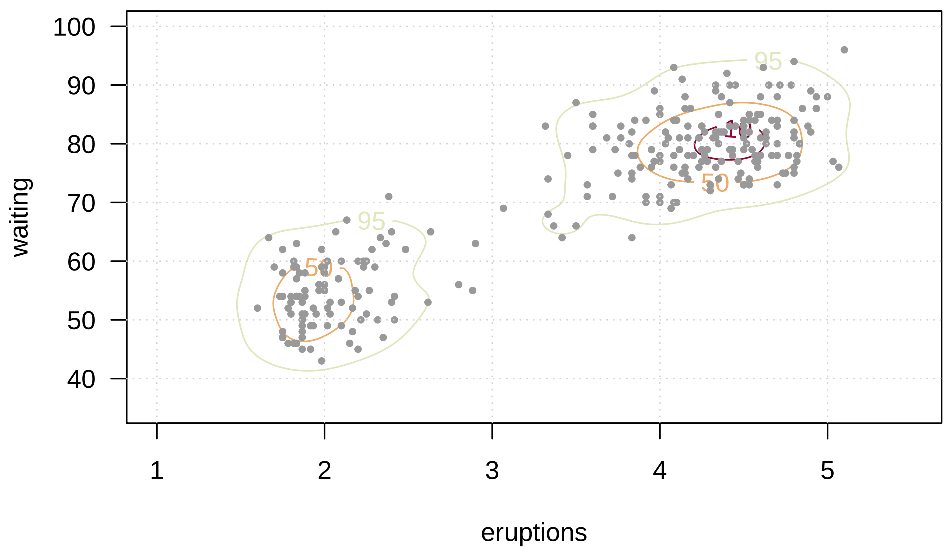
If the off-diagonal/correlation is zero, then kernel reduces to the product of two univariate kernels:

$$K((x_1, x_2); H) = K_1(x_1; h_1) K_2(x_2; h_2)$$

where $h_1 = \sqrt{H_{11}}$.

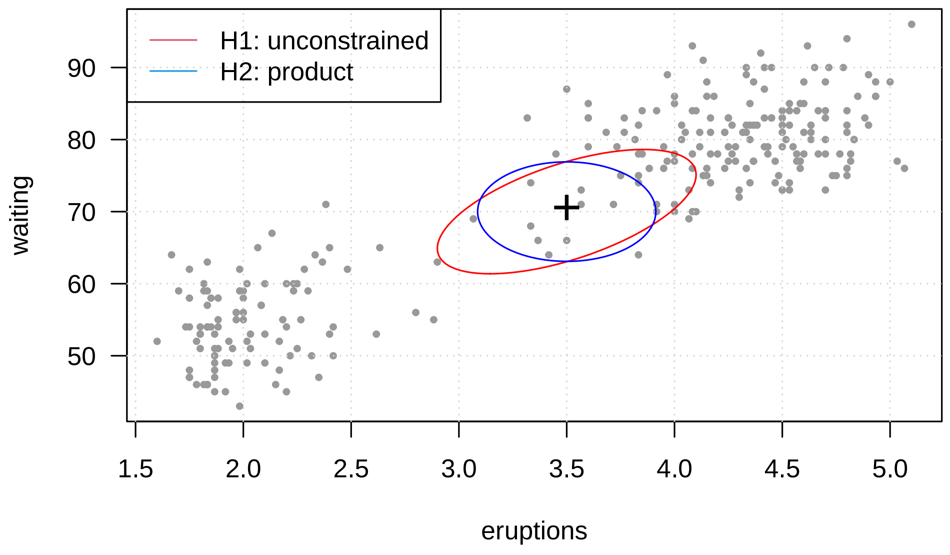
```r
(H2 = Hscv.diag(X))                                 # product kernel
#>         [,1]  [,2]
#> [1,] 0.0285 0.000
#> [2,] 0.0000 7.949
f2 = kde(X, H=H2)

plot(f2,
     cont = c(10, 50, 95),                          # set contour levels
     las=1, xlim = c(1.0, 5.5), ylim=c(35, 100))  # set asthetics
```

```r
points(X, pch=19, cex=.5, col='grey60')          # add points
grid()
```



We can plot the kernels (using `mixtools::ellipse()` to see their shape. Here is the kernel at location (3.5, 70).

```r
library(mixtools)
plot(X, pch=19, cex=.5, col='grey60', las=1); grid()
points(3.5, 70, pch="+", cex=2)         # use (3.5, 70) as center

#: Unconstrained Kernel
mixtools::ellipse(mu=c(3.5, 70),        # center of ellipse
                  sigma=H1,             # bandwidth matrix
                  alpha = .05,          # 1-alpha confidence
                  col="red")            # color


#: Product kernel
mixtools::ellipse(mu=c(3.5, 70),
                  sigma=H2,
                  alpha = .05, col="blue")

#: Legend
legend("topleft", c("H1: unconstrained", "H2: product"), col=c(2,4), lty=1)
```



6.1.2 Independence

The least complex model is one based on independence. Just like Naive Bayes!

- Estimate the KDE for each dimension separately.
- The joint density estimate is the product of the p KDEs

6.1.3 R Code

- The vignette [ks: Kernel density estimation for bivariate data] https://cran.r-project.org/web/packages/ks/vignettes/kde.pdf has some more information on using `ks::kde()` for bivariate data.

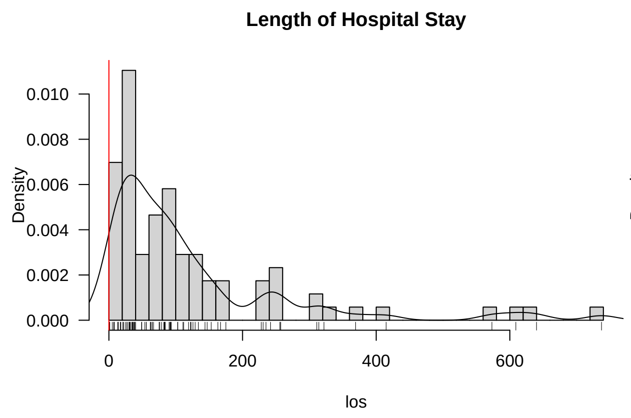
7 Extra Details

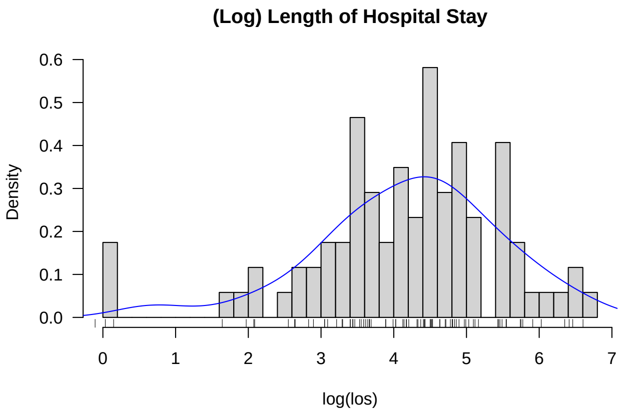
7.1 Edge Effects

Sometimes there are known boundaries in the data (e.g., the amount of rainfall cannot be negative). Here are some options:

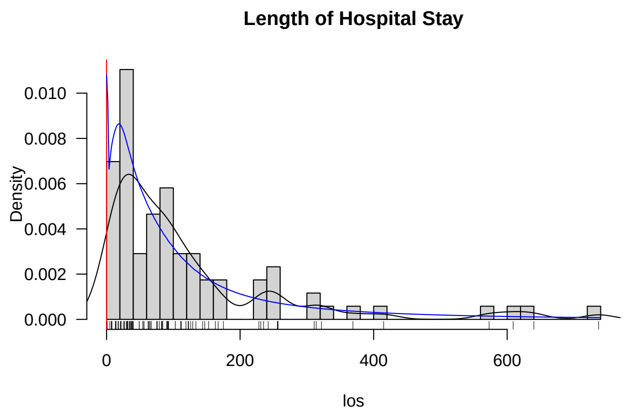
1. Do nothing - as long as not many events are near the boundary and the bandwidth is small, this may not be too problematic. However, it will lead to an increased bias around the boundaries.
2. Transform the data (e.g., $x' = \log(x)$), estimate the density is the transformed space, then transform back
3. Use an edge correction technique

7.1.1 Log-Transformations





- Let $Y = \ln(X)$ be the transformed RV.
- $F_X(x) = \Pr(X \leq x) = \Pr(e^Y \leq x) = \Pr(Y \leq \ln(x)) = F_Y(\ln(x))$
- $f_X(x) = \frac{d}{dx}F_X(x) = \frac{d}{dx}F_Y(\ln(x)) = \frac{1}{x}f_Y(\ln(x))$
- The argument `positive=TRUE` in `ks::kde()` will perform this transformation.



7.1.2 Edge Correction

The simplest approach requires a modification of the kernels near the boundary. Let $\mathcal{S} = [a, b]$.

- Recall that $\int_a^b K_h(x_i - x)dx$ should be 1 for every i.
- But near a boundary $\int_a^b K_h(x_i - x)dx < 1$
- Denote $w_h(x_i) = \int_a^b K_h(x_i - x)dx$
- The resulting edge corrected KDE equation becomes

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

Another approach corrects the kernel for each particular x

- Denote $w_h(x) = \int_a^b K_h(u - x)\,\mathrm{d}u$

- The resulting edge corrected KDE equation becomes

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

- This approach is not guaranteed to integrate to 1, but for some problems (e.g., density estimation for classification) this is not a major concern

7.2 Adaptive Kernels

Up to this point, we have considered fixed bandwidths. But what if we let the bandwidth vary? There are two main approaches:

- Balloon Estimator

$$\begin{aligned}\hat{f}(x) &= \frac{1}{n}\sum_{i=1}^{n} K_{h(x)}(x_i - x)\\ &= \frac{1}{nh(x)}\sum_{i=1}^{n} K\left(\frac{x_i - x}{h(x)}\right)\end{aligned}$$

- Sample Point Estimator

$$\begin{aligned}\hat{f}(x) &= \frac{1}{n}\sum_{i=1}^{n} K_{h(x_i)}(x_i - x)\\ &= \frac{1}{n}\sum_{i=1}^{n} h(x_i)^{-1} K\left(\frac{x_i - x}{h(x_i)}\right)\end{aligned}$$

7.3 k-Nearest Neighbor Density Approach

Like what we discussed for percentile binning, we can estimate the density from the size of the window containing the k nearest observations.

$$\hat{f}_k(x) = \frac{k}{nV_k(x)}$$

where $V_k(x)$ is the volume of a neighborhood that contains the k-nearest neighbors.

- This is an adaptive version of the moving window (uniform kernel) approach
- Probably won't integrate to 1
- It is also possible to use the k-NN distance as a way to select an adaptive bandwidth $h(x)$.

7.4 Mixture Models

Mixture models offer a flexible compromise between kernel density and parametric methods.

- A mixture model is a mixture of densities

$$f(x) = \sum_{j=1}^{p} \pi_j g_j(x \mid \xi_j)$$

where

  - $0 \leq \pi_j \leq 1$ and $\sum_{j=1}^{p} \pi_j$ are the mixing proportions
  - $g_j(x \mid \xi_j)$ are the component densities
- This idea is behind model-based clustering, radial basis functions (ESL 6.7), etc.

- Usually the parameters θ_j and weights π_j have to be estimated (EM algorithm shows up here)

7.5 Kernels, Mixtures, and Splines

All of these methods can be written:

$$f(x) = \sum_{j=1}^{J} b_j(x)\theta_j$$

- For KDE:
  - $J = n, b_j(x) = K_h(x - x_j), \theta_j = 1/n$ (bw h estimated)
- For Mixture Models:
  - $b_j(x) = g_j(x \mid \xi_j), \theta_j = \pi_j$ (J, ξ_j, π_j estimated)
- B-splines
  - $b_j(x)$ is a B-spline, (θ_j, and maybe J and knots, estimated)
  - Note: For density estimation, $\log f(x) = \sum_{j=1}^{J} b_j(x)\theta_j$ may be easier to estimate

7.6 Other Issues

- One major problem with KDE and kernel regression (and k-NN) is that all of training data must be stored in memory.
  - For large data, this is unreasonable
  - Binning (i.e., histograms) are used to reduce data storage and improve computation
- Multi-dimensional kernels are not very good for high dimensions (unless simplified by using product kernels)
- But temporal kernels good for adaptive procedures (e.g., only remembers most recent observations)
  - Similar to EWMA

8 Appendix: R Code

Interactive Shiny App

8.1 Parametric Density Models

```r
#: Load Required Packages
library(fitdistrplus) # install.packages("fitdistrplus")
library(ks)           # for KDE
library(tidyverse)    # for data manipulation
```

8.1.1 Emergency Room Data

```r
#--------------------------------------------------------------------------#
# Load ER counts Data
#--------------------------------------------------------------------------#
#: Load Data
url = 'https://mdporter.github.io/DS6030/data/ED-counts.csv'
x = readr::read_csv(url)$count  # extract as a vector
```

```r
#: empirical pmf
ggplot() + geom_bar(aes(x=x))

#: Summary
summary(x)
mean(x)
var(x)
```

8.1.2 Poisson Model: ER counts Data

1. `fitdistrplus`

```r
#: MLE
library(fitdistrplus)
opt = fitdist(data=x, distr="pois", method="mle")
pois_pars = opt$estimate
```

2. Grid search

```r
## Grid Search
#: Get sequence of lambda values
lam_seq = seq(65, 100, length=200)
nlam = length(lam_seq)

#: Calculate the log-likelihood for those lambda values
loglike = numeric(nlam)
for(i in 1:nlam){
  loglike[i] = sum(dpois(x, lambda=lam_seq[i], log=TRUE))
}

#: Alternative to loop using purrr::map
loglike2 = map_dbl(lam_seq, ~sum(dpois(x, lambda=., log=TRUE)))
all.equal(loglike, loglike2)

#- best lambda via grid search:
lam.opt = lam_seq[which.max(loglike)]
```

```r
#: Make log-likelihood plot
lam.data = tibble(lam_seq, loglike)
```

```r
ggplot(lam.data, aes(lam_seq, loglike)) +
  geom_line() +
  geom_point(data=filter(lam.data, loglike == max(loglike)),
             color="red", size=2) +
  labs(x = expression(lambda), y="log-likelihood")
```

3. Calculus

See notes.

Make plot of estimated pmf

```r
#: Plot counts with pmf overlaid
fit_pois = tibble(x=seq(min(x), max(x), by=1), y=dpois(x, lambda=lam.opt))

ggplot() +
  geom_bar(aes(x, y=after_stat(prop)), fill="grey70") +
  geom_point(aes(x,y), data=fit_pois, color="blue", alpha=.5) +
  labs(y = "pmf")
```

8.1.3 Negative Binomial Model: ER counts Data

```r
#: MLE
library(fitdistrplus)
opt = fitdist(data=x, distr="nbinom", method="mle")
nb_pars = opt$estimate
```

```r
#: Plots
#: Make Data
x.seq = 0:162       # sequence of x values
f.nb = dnbinom(x.seq, size=nb_pars[1], mu=nb_pars[2])   # pmf values at x.seq
f.pois = dpois(x.seq, lambda=mean(x))

fit.data = tibble(x.seq, poisson=f.pois, neg.binom=f.nb) %>%
  gather(model, pmf, -x.seq)

#: Make PMF curve; overlay histogram/barplot
ggplot() +
  geom_bar(aes(x, y=after_stat(prop)), fill="grey70") +
  geom_point(data=fit.data,
             aes(x=x.seq, y=pmf, color=model),  alpha=.5) +
  scale_color_manual(values= c("red", "blue"))
```

8.1.4 Normal Model: ER counts Data

```r
#: MLE/MOM
mean(x)
sd(x)
n = length(x)
sd(x)*sqrt((n-1)/n)    # correction for n-1

library(fitdistrplus)
opt = fitdist(data=x, distr="norm", method="mle")
gauss_pars = opt$estimate
```

```
#: Plots
f.gauss = dnorm(x.seq, mean=gauss_pars[1], sd=gauss_pars[2])  # pmf values at x.seq

# if we wanted to be particular, we could discretize the gaussian
# f.gauss = pnorm(x.seq+.5, mean=gauss_pars[1], sd=gauss_pars[2]) - pnorm(x.seq-.5, mean=gauss_pars[

fit.data = tibble(x.seq, poisson=f.pois, neg.binom=f.nb, gaussian=f.gauss) %>%
  gather(model, pmf, -x.seq)

#: Make PMF curve; overlay histogram/barplot
ggplot() +
  geom_bar(aes(x, y=after_stat(prop)), fill="grey70") +
  geom_point(data=fit.data,
             aes(x=x.seq, y=pmf, color=model),  alpha=.5) +
  scale_color_manual(values= c("green", "red", "blue"))


#- base R version of plot
h = hist(x, breaks=seq(-.5, 163, by=.5), plot=FALSE)
h$counts = h$counts/sum(h$counts)   # make relative frequency
plot(h, ylim=range(f.pois), ylab='prop', col='lightgrey', border=NA)
lines(x.seq, f.pois, col="blue")
lines(x.seq, f.nb, col="red")
lines(x.seq, f.gauss, col="green")
```

8.2 Histograms

```
#: Load the Old Faithful data
wait = datasets::faithful$waiting


#: Histogram settings
bw = 10                         # binwidth parameter
bks = seq(40, 110, by=bw)    # create a sequence of numbers
```

```
#: Frequency Histogram
ggplot() + geom_histogram(aes(x=wait), breaks = bks, color="white") +
  labs(title="Frequency Histogram")

hist(wait, breaks=bks, las=1, main="Frequency Histogram")
```

```
#: Relative Frequency Histogram
ggplot() +
  geom_histogram(aes(x=wait, y=after_stat(count/sum(count))), breaks = bks, color="white") +
  labs(title="Relative Frequency Histogram", y="proportion")

h.rf = hist(wait, breaks=bks, plot=FALSE)
h.rf$counts = h.rf$counts/sum(h.rf$counts)    # make relative frequency
plot(h.rf, las=1, main="Relative Frequency Histogram")
```

```
#: Density Histogram
ggplot() +
  geom_histogram(aes(x=wait, y=after_stat(density)), breaks = bks, color="white") +
  labs(title="Density Histogram")

hist(wait, freq=FALSE, breaks=bks, las=1, main="Density Histogram")
```

```
#: Manual histogram calculations
hist.data = tibble(wait) %>%
  mutate(bin = cut_width(wait, width=bw, boundary=40)) %>%
  count(bin) %>%
  mutate(rel.freq = n/sum(n), density=rel.freq/bw)

ggplot(hist.data) + geom_col(aes(bin, rel.freq), width=1)
```

8.3 Kernel Density Estimation (KDE)

```
library(ks)

#: Histogram
bw = 5                          # binwidth parameter
bks = seq(40, 100, by=bw)       # create a sequence of numbers
hh = hist(wait,  breaks=bks)# histogram object

#: KDE
library(ks)
f = kde(wait, h=bw/3)
plot(f)
```

```
#: Plot hist and kde
plot(hh, freq=FALSE, ylim=c(0, max(c(hh$density, f$estimate))),
     las=1, main='', border='white', col='grey75')
rug(jitter(wait))
lines(f$eval.points, f$estimate, col=2, lwd=1.25)
# OR: plot(f, add=TRUE, col=2, lwd=1.25)

#: ggplot version
est = tibble(wait = f$eval.points, fhat = f$estimate)
ggplot(tibble(wait), aes(wait)) +
  geom_histogram(aes(y=after_stat(density)), breaks=bks, fill="grey75", color="white") +
  geom_rug(aes(x=jitter(wait))) +
  geom_line(data=est, aes(y=fhat), color="red")
```

8.3.1 Multivariate KDE

```
#: Load the Old Faithful data
X = datasets::faithful

#: Plot: Base R
plot(X, las=1); grid()
#: Plot: ggplot
ggplot(X, aes(eruptions, waiting)) + geom_point() +
  geom_density_2d() # geom_density2d_filled(), geom_contour_filled(), ...
```

```
#: MV KDE: Unconstrained
H1 = Hscv(X)                    # smoothed cross-validation bw estimator
f1 = kde(X, H=H1)               # use H for multivariate data

plot(f1,
     cont = c(10, 50, 95),                      # set contour levels
     # display = "filled.contour",              # use filled contour
     las=1, xlim = c(1.0, 5.5), ylim=c(35, 100)) # set asthetics
points(X, pch=19, cex=.5, col='grey60')         # add points
grid()                                          # add grid lines
```

```r
## Note: you can inspect the output from kde() to see what goodies are returned
str(f1)
## notice that f1$eval.points is a list of the X1, X2 values where estimates were made
##  and f1$estimate is a matrix of the density estimates
```

```r
#: Predict function
# the ks package includes a predict() function to help make predictions
X.eval = expand.grid(eruptions = seq(min(X$eruptions), max(X$eruptions), length=100),
                     waiting = seq(min(X$waiting), max(X$waiting), length=100))
X.eval %>% mutate(fhat = predict(f1, x = .)) %>%
  ggplot(aes(eruptions, waiting)) +
  geom_contour_filled(aes(z=fhat)) + # or unfilled with geom_contour()
  geom_point(data=X, color="white")
```

```r
#: Tidy function
# the broom package includes a tidy() function to help extract predictions
# Note: use kde(..., eval.points=) to control the eval points in this approach
library(broom)
broom::tidy(f1) %>% pivot_wider(names_from=variable, values_from=value) %>%
  rename(eruptions=x1, waiting=x2) %>%
  ggplot(aes(eruptions, waiting)) +
  geom_contour(aes(z=estimate)) + # or unfilled with geom_contour()
  geom_point(data=X, color="black")
```

```r
#: Product kernel
H2 = Hscv.diag(X)     # forces off-diagonal var-cov terms to be 0
f2 = kde(X, H=H2)

plot(f2,
     cont = c(10, 50, 95),                            # set contour levels
     las=1, xlim = c(1.0, 5.5), ylim=c(35, 100))      # set asthetics
points(X, pch=19, cex=.5, col='grey60')               # add points
grid()                                                # add grid lines
```

```r
#: Independence (not the best model for this problem)
he = hscv(X$eruptions)
fe = kde(X$eruptions, h=he)
hw = hscv(X$waiting)
fw = kde(X$waiting, h=hw)

plot(fe, xlab="eruptions")
plot(fw, xlab="waiting")

X.eval %>%
  mutate(fhat.e = predict(fe, x=eruptions),
         fhat.w = predict(fw, x=waiting),
         fhat = fhat.e*fhat.w) %>%
  ggplot(aes(eruptions, waiting)) +
  geom_contour_filled(aes(z=fhat)) +
  geom_point(data=X, color="white")
```

```r
#--------------------------------------------------------------------------#
# Visualize kernel shapes
#--------------------------------------------------------------------------#
plot(X, las=1); grid()

#: Add 95% confidence ellipse for *unconstrained* at location (2, 60)
library(mixtools)
points(2, 60, pch="+", col="red", cex=1.5)
mixtools::ellipse(mu=c(2, 60),
                    sigma=H1,
                    alpha = .05, col="red")

#: Add 95% confidence ellipse for *product kernel* at location (4, 80)
library(mixtools)
points(4, 80, pch="+", col="blue", cex=1.5)
mixtools::ellipse(mu=c(4, 80),
                    sigma=H2,
                    alpha = .05, col="blue")

legend(
  "topleft",
  legend = c("unconstrained kernel", "product kernel"),
  lty = 1,
  col = c("red", "blue")
)
```