Part 7: Smoothing

Summary

Smoothing refers generally to procedures that attempt to extract underlying signal from noise in a nonparametric manner.

The term “nonparametric” means the use of a statistical model that does not make restrictive assumptions. The data are allowed to “speak for themselves” as much as possible.

Here we consider such smoothing procedures as tools for summarizing noisy data.

Basics of Density Estimation

Nonparametric density estimation refers to techniques that estimate the distribution for a variable, without assuming any parametric form. Examples of parametric forms include the normal distribution, the exponential distribution, etc.

Histograms are a simple example of a nonparametric density estimator. While these are useful, they suffer from limitations, namely the arbitrariness of the binning, and discontinuous (“jagged”) nature of the estimate of a distribution that is typically assumed to be smooth. Smooth estimators also have statistical efficiency advantages over histograms.

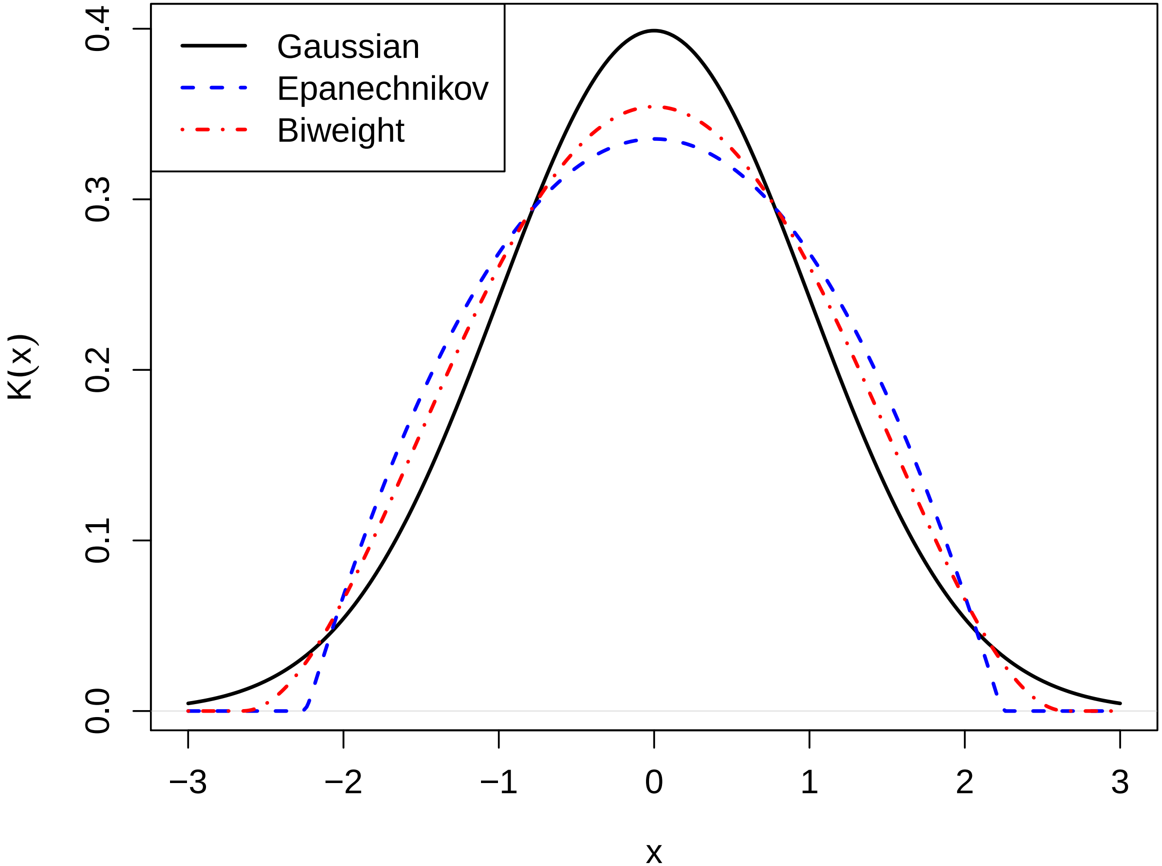
The standard nonparametric approach to density estimation is the kernel density estimator. This estimate is constructed by summing a smooth kernel function centered at each of the observed data points.

Formally, we write:

$$\widehat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where $K(\cdot)$ is the kernel function and h is the smoothing parameter or bandwidth. The sample is $x_1, x_2, \ldots, x_n$.

The kernel function is itself a density, almost always taken to be a smooth density peaked at zero, and symetric around zero. Three standard examples are shown on the next page.



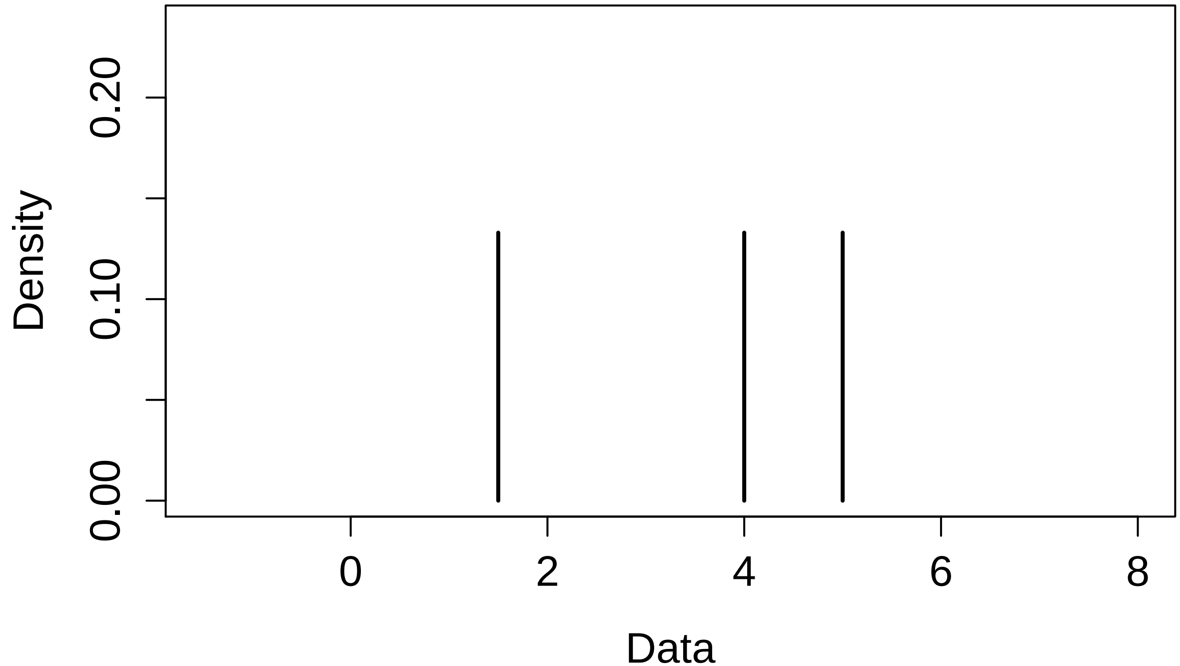
1

The choice of the kernel is not too influential on the shape of the density estimate. The bandwidth h is influential, however.

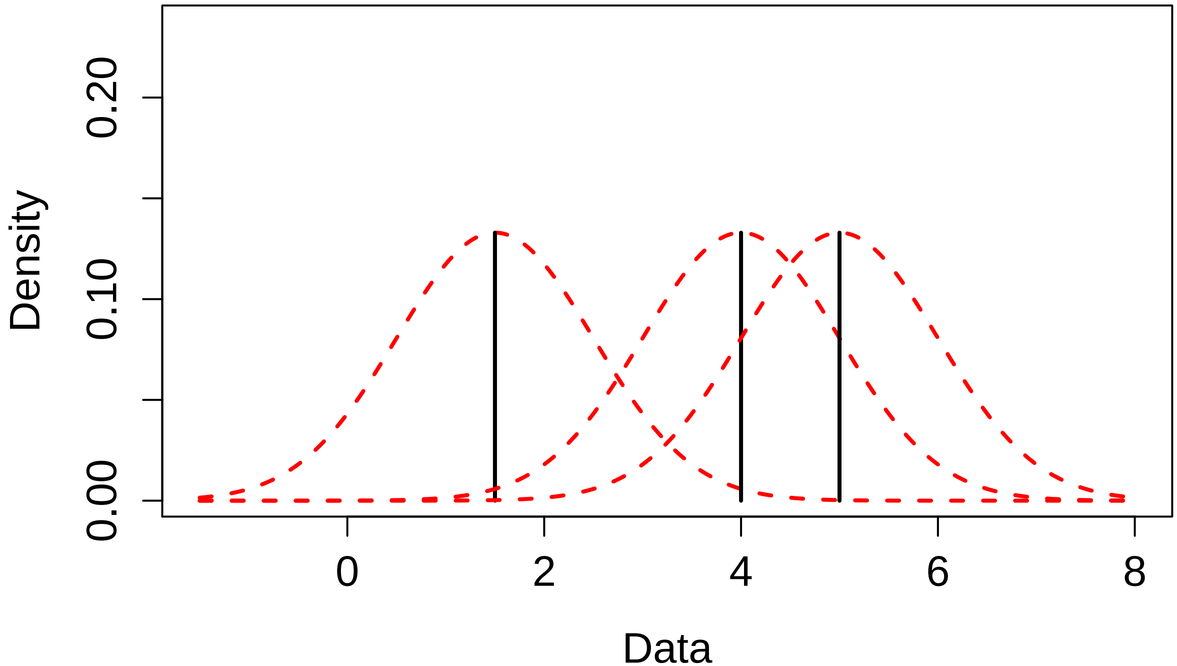
Larger values of h result in a density estimate that is smoother. Smaller values of h produce an estimate that is "rough" and "wiggly."

As a very simple example to illustrate the concept, imagine my data set consisted of three observations: 1.5, 4, and 5. The next three slides illustrate the steps going from the raw data to the final estimate. The subsequent slides show the effect of varying the bandwidth.

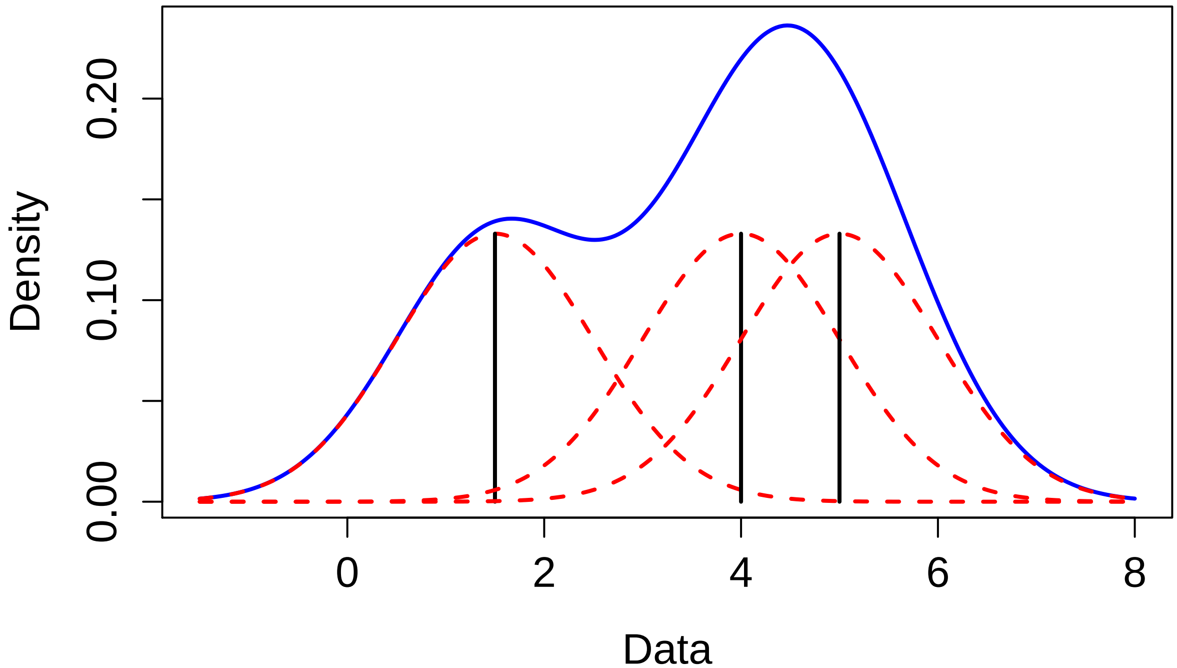
The positions of the three observations:



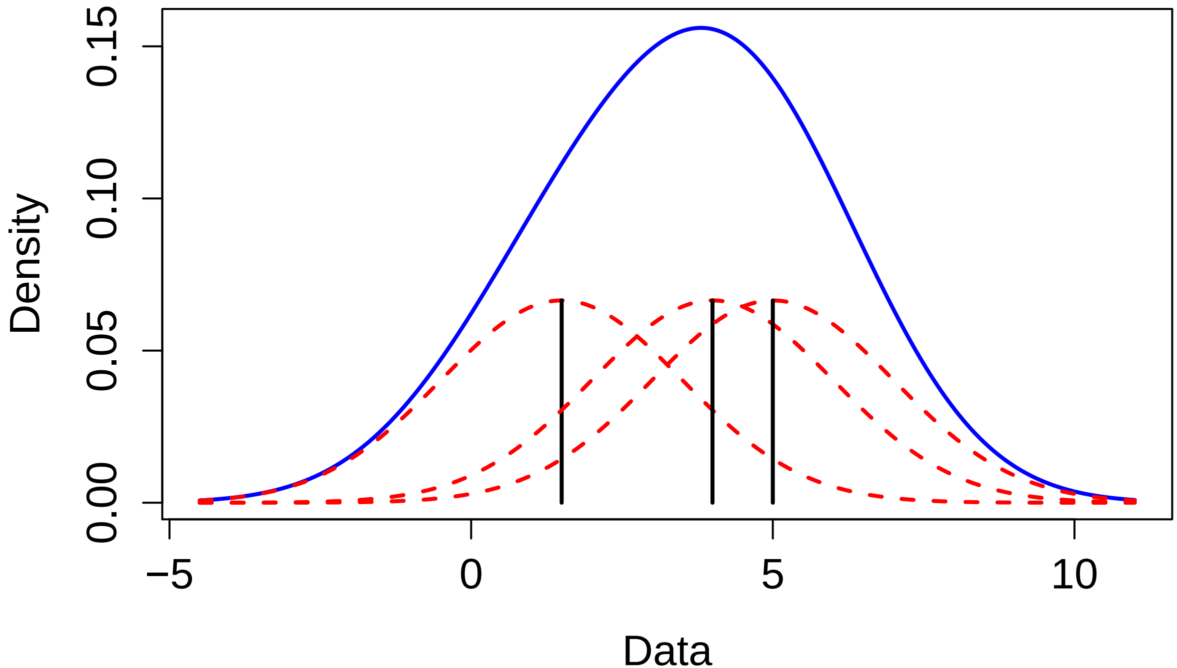
The gaussian kernel with $h = 1$ shown centered at each observation:



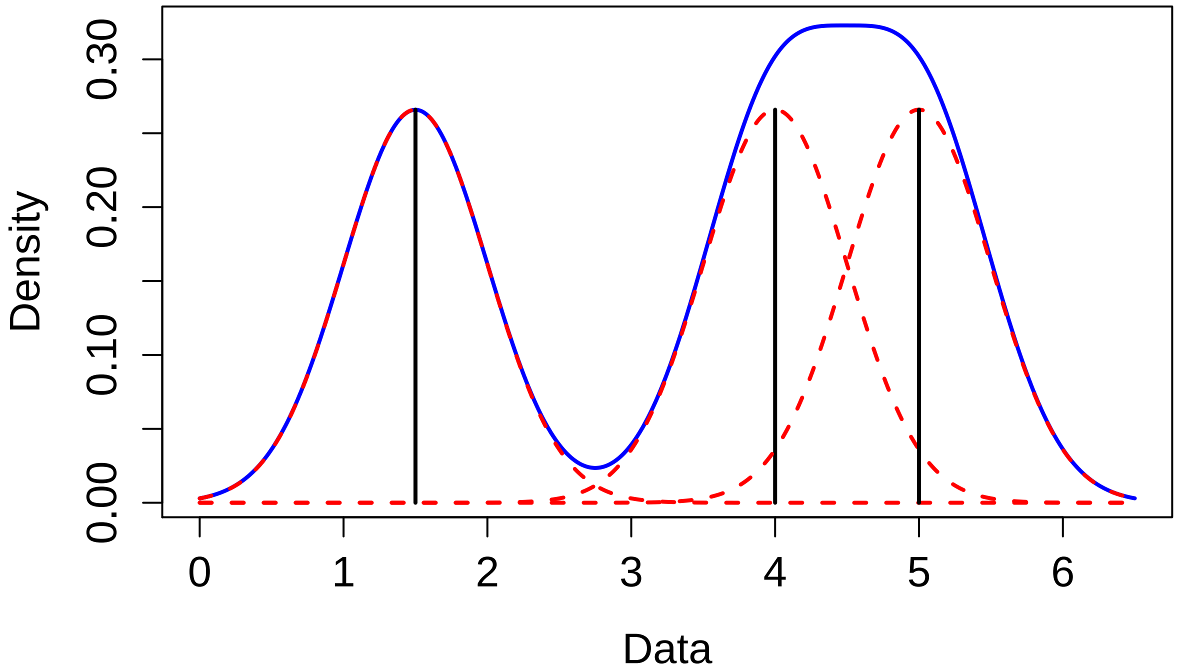
1 These kernels are summed to create the final estimate:

2 

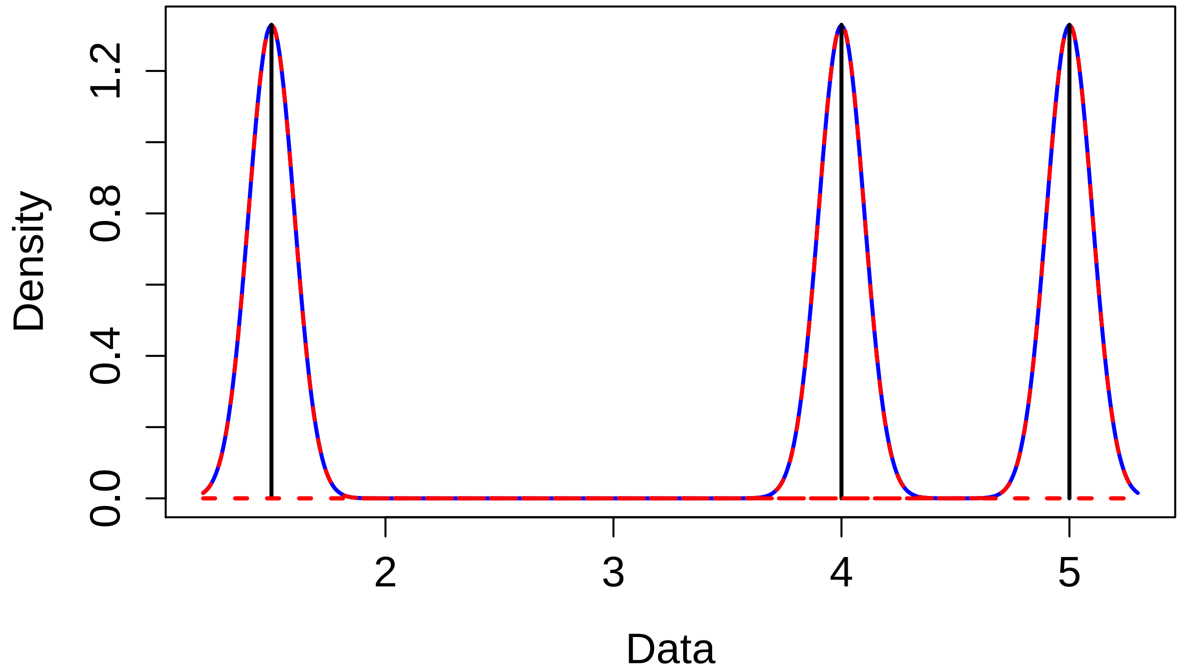
Compare this with the case where $h = 2$:



And when $h = 0.5$:



Or when $h = 0.1$:



Exercise: What is the relationship between using a Gaussian kernel and making an assumption that a sample is drawn from a Gaussian (normal) distribution?

Rigorous ways of choosing h do exist. See Jones, Marron, and Sheather (1996) for an overview.

Briefly, the theory is built on the assumption that the available sample $x_1, x_2, \ldots, x_n$ are drawn i.i.d. from some population with density $f(\cdot)$. The criterion that one seeks to minimize is the mean integrated squared error:

$$\text{MISE} = E\left[\int \left(\widehat{f}_h(x) - f(x)\right)^2 dx\right].$$

As is often the case, consideration is made of the asymptotic behavior of the criterion as n increases. One can show that the the asymptotic mean integrated squared error is

$$\text{AMISE} = \frac{R(K)}{nh} + h^4 R(f'')\left(\frac{1}{2}\int x^2 K(x)\,dx\right)^2$$

where

$$R(\phi) = \int \phi^2(x)\,dx$$

The optimal choice for h can then be shown to be

$$h = \left[\frac{R(K)}{nR(f'')\left(x^2K(x)\right)^2}\right]^{1/5}$$

The Sheather-Jones (S-J) approach to finding the bandwidth is to find the h which solves

$$h = \left[\frac{R(K)}{nR(\widehat{f}''_{g(h)})\left(x^2K(x)\right)^2}\right]^{1/5}$$

where $g(h)$ is some function of h which accounts for the fact that the optimal choice of bandwidth for estimating f'' is not the same as the optimal choice for estimating f.

The choice of h can also be dictated by less theoretical considerations.

In particular, if the motivation is to achieve a summary of a distribution via smoothing the distribution to a particular scale.

One could, for example, smooth a distribution using multiple bandwidth choices, and then using the resulting smooth estimates as input into a model. Formal procedures could then be used to determine how useful each "scale" is to the problem at hand, i.e., a prediction challenge.

This idea will be explored through examples later.

Kernel Density Estimation in R

The basic function for kernel density estimation in R is `density()`.

The argument `kernel` specifies the kernel function; the default is `"gaussian"`.

The bandwidth is provided using the argument `bw`. The user can specify either a number, or specify a method for determining the bandwidth. For example, the function `bw="SJ"` calculates the S-J bandwidth.

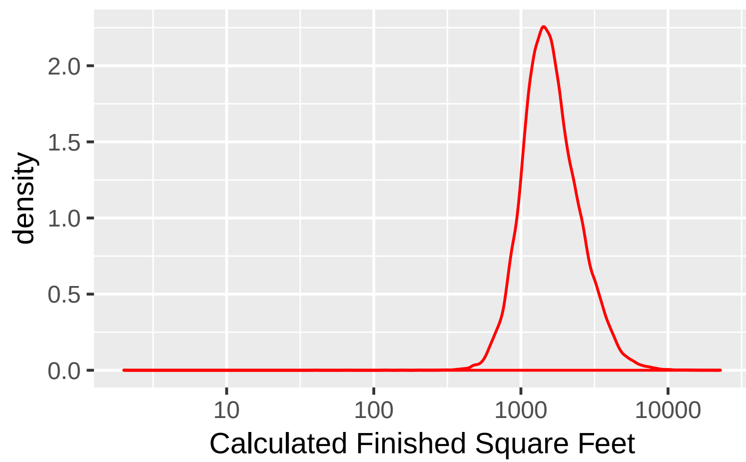
The argument `adjust` can be useful when attempting to try multiple scales. The bandwidth utilized is actually equal to `adjust` times the value of `bw`. This makes it easy to fit with, e.g., double the S-J bandwidth, half the S-J bandwidth, etc.

Exercise: Try the following code, and discuss the structure of the output of `density()`. What happens as the sample size is varied?

```
> x = rnorm(1000)
> densout = density(x, bw="SJ")
> plot(densout)
> curve(dnorm(x), add=T, col="red", lty=2)
```

There is a function `geom_density()` as part of `ggplot`. It accepts the same arguments as does `density()`.

```
> ggplot(trainmerged, aes(x=calculatedfinishedsquarefeet)) +
+     geom_density(bw="SJ", color="red") +
+     labs(x="Calculated Finished Square Feet") +
+     scale_x_log10()
```



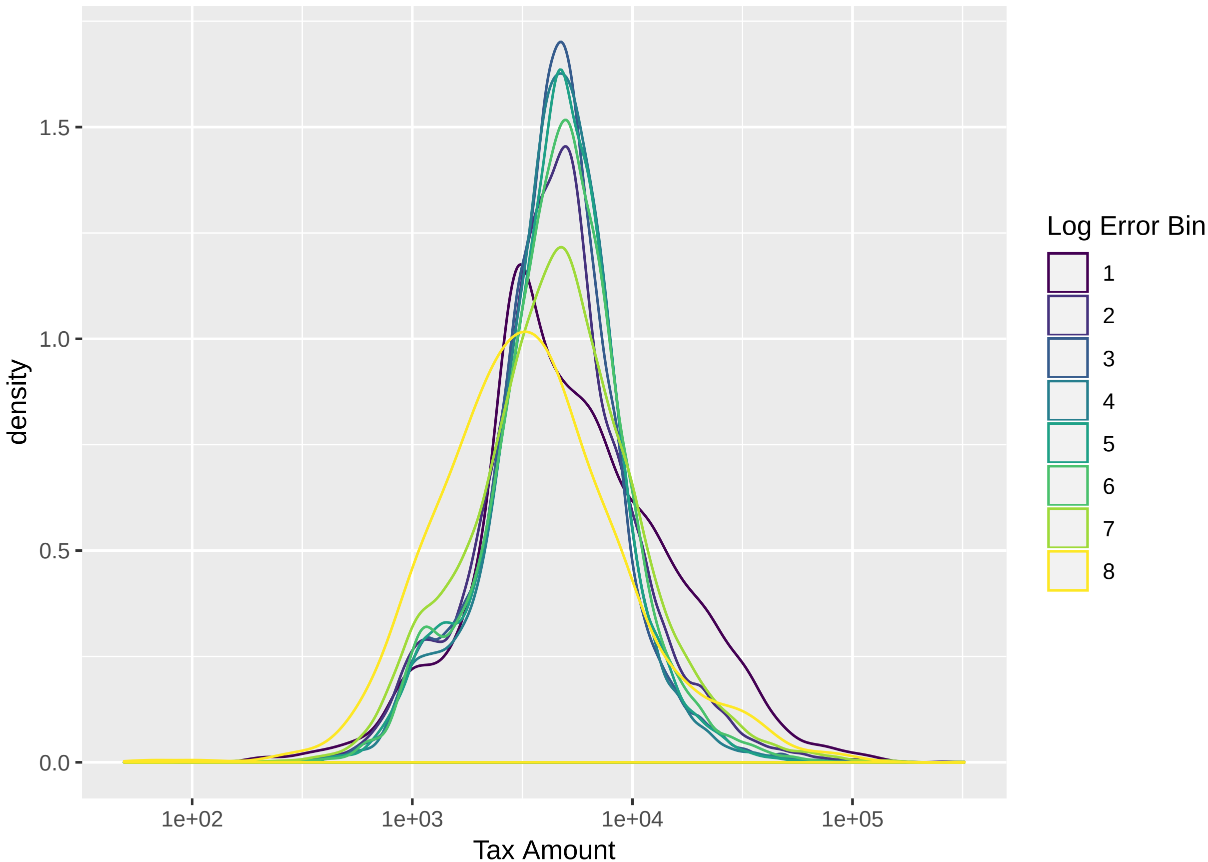
Exercise: How does one interpret the vertical scale "Density" on the preceding plot?

Insights from Density Estimates

As stated above, a primary motivation for smoothing in general, and non-parametric density estimation in particular, is to work to separate real features (the "signal") from the uninformative randomness (the "noise").

Consider how the next plot presents changes in the distribution of the amount of property tax over the different log error bins. This builds on our previous Zillow example.

```
> ggplot(trainmerged, aes(x=taxamount, color=logerrorbin)) +
+     geom_density(bw="SJ") +
+     labs(x="Tax Amount", color="Log Error Bin") +
+     scale_x_log10()
```



density
1.5
1.0
0.5
0.0
1e+02
1e+03
1e+04
1e+05
Tax Amount
Log Error Bin
1
2
3
4
5
6
7
8

1

Exercise: What conclusion(s) could you draw from the previous plot?

A kernel density estimate is often utilized to determine an appropriate "named" parametric distribution, if it exists.

As an example, use package `quantmod` to obtain the daily data for Kellogg's (K) from 2010 through 2016:

```
> Kellogg = getSymbols("K",
+   from="2010-1-1",to="2016-12-31", auto.assign=F)
```

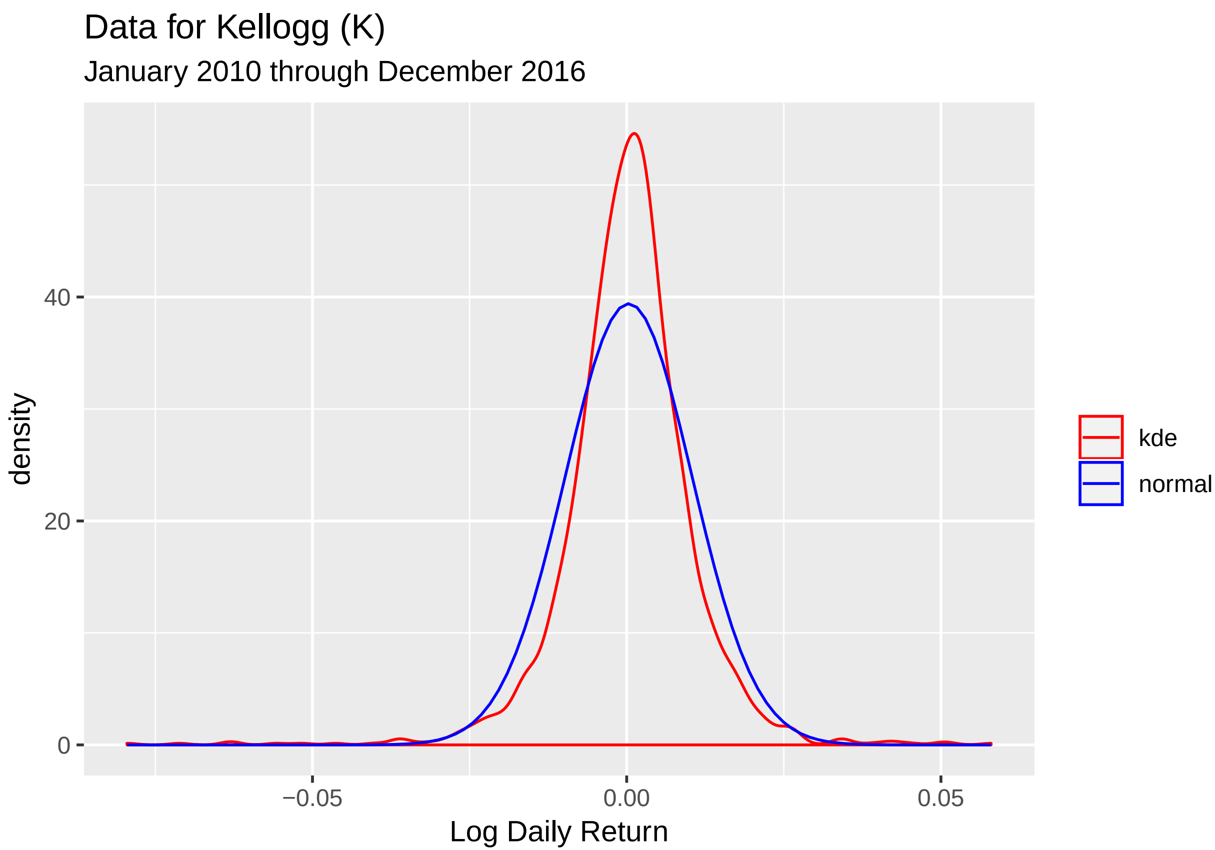
Then calculate the log daily returns:

```
> ldrK = data.frame(dailyReturn(Ad(Kellogg),
+                                            type="log"))
```

The function `Ad()` extracts the "adjusted closing price" column from the data set.

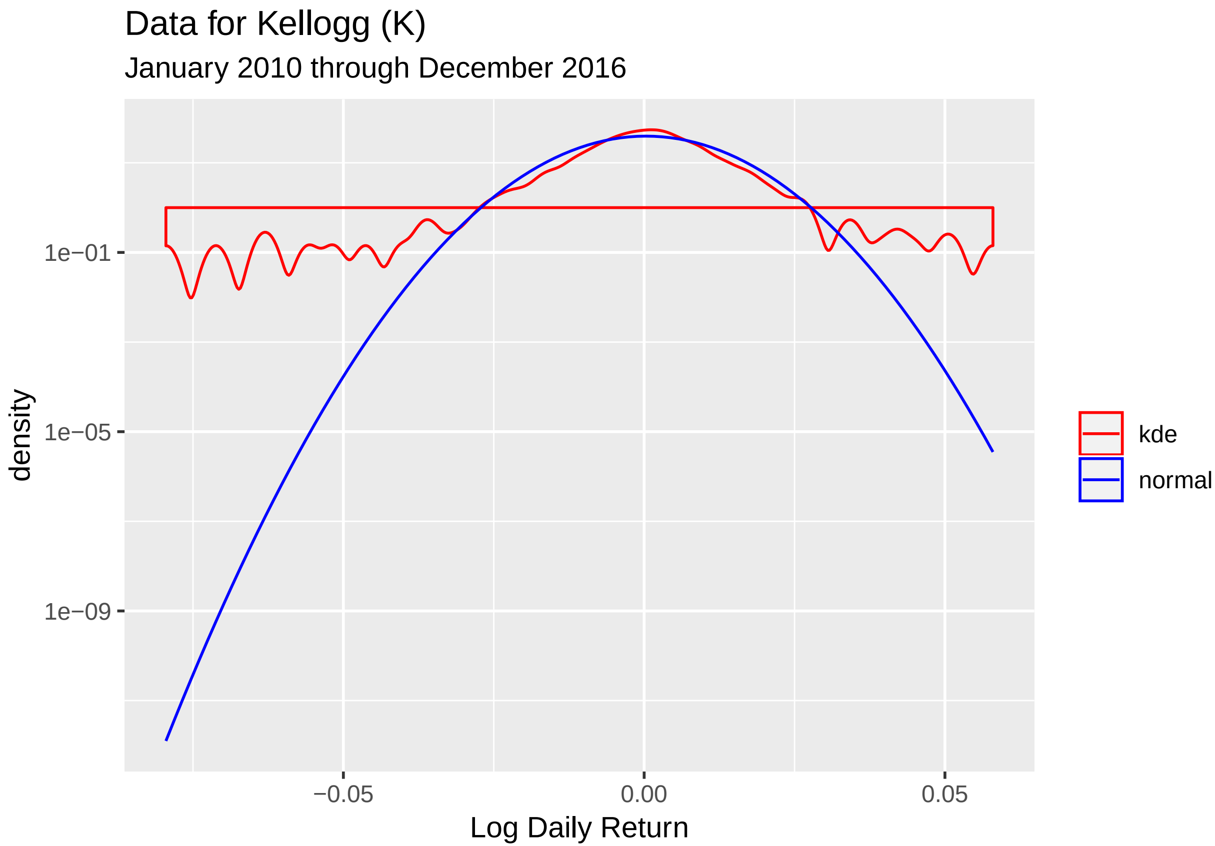
1 Consider the following plot which compares the kernel density estimate
2 with the normal distribution:

```
> ggplot(ldrK,aes(x=daily.returns)) +
+   geom_density(bw="SJ",aes(color="kde")) +
+   stat_function(fun=dnorm, aes(color="normal"),
+       args=list(mean=mean(ldrK$daily.returns),
+                 sd=sd(ldrK$daily.returns))) +
+   scale_color_manual(name="",
+               values=c("kde"="red","normal"="blue")) +
+   labs(x="Log Daily Return",title="Data for Kellogg (K)",
+        subtitle="January 2010 through December 2016")
```


Data for Kellogg (K)
January 2010 through December 2016
density
40
20
0
−0.05
0.00
0.05
Log Daily Return
kde
normal

1 The plot is not as informative as one on the log scale:

```
> ggplot(ldrK,aes(x=daily.returns)) +
+   geom_density(bw="SJ",aes(color="kde")) +
+   stat_function(fun=dnorm, aes(color="normal"),
+       args=list(mean=mean(ldrK$daily.returns),
+                 sd=sd(ldrK$daily.returns))) +
+   scale_color_manual(name="",
+                 values=c("kde"="red","normal"="blue")) +
+   labs(x="Log Daily Return",title="Data for Kellogg (K)",
+        subtitle="January 2010 through December 2016") +
+   scale_y_log10()
```


Data for Kellogg (K)
January 2010 through December 2016
density
1e−01
1e−05
1e−09
−0.05
0.00
0.05
Log Daily Return
kde
normal

1

Exercise: Comment on the appropriateness of the normal approximation to log returns in this case.

Converting to the CDF and Percentiles

Smooth density estimates can be transformed into an estimate of the cumulative distribution function (CDF) and its inverse (i.e., percentiles).

A useful function in R to achieve this is `approxfun()`. It takes `x` and `y` vectors and returns a function which linearly interpolates the given values. Note that the output of `approxfun()` is itself a function.

Also, there is a function `integrate()` that will perform numerical integration.

1 Consider the following function `kCDF()`:

```
> kCDF = function(x, res=100, ...)
+ {
+    holddens = density(x,...)
+    interpdens = approxfun(holddens$x, holddens$y,
+                                 yleft=0, yright=0)
+    xseq = seq(min(holddens$x),max(holddens$x),length=res)
+    holdout = numeric(res)
+    for(i in 1:res)
+    {
+       holdout[i] = integrate(interpdens, lower=min(holddens$x),
+                    upper=xseq[i], stop.on.error=FALSE)$value
+    }
+    CDF = approxfun(xseq, holdout, yleft=0, yright=1)
+    invCDF = approxfun(holdout, xseq, yleft=NA, yright=NA)
+    list(CDF=CDF, invCDF=invCDF)
+ }
```

Exercise: Discuss what the function `kCDF()` will do.

For this example, let's return to the market structure data that we considered in Parts 4 and 5:

```
> fulldata = read.table("q1_2017_all.csv", sep=",", header=T, quote="")
> fulldata$Date = as.Date(as.character(fulldata$Date), format="%Y%m%d")
> fulldata = fulldata[!duplicated(fulldata[,c(1,3)], fromLast=TRUE),]
> fulldata$McapRank = factor(fulldata$McapRank, ordered=TRUE)
> fulldata$TurnRank = factor(fulldata$TurnRank, ordered=TRUE)
> fulldata$VolatilityRank = factor(fulldata$VolatilityRank, ordered=TRUE)
> fulldata$PriceRank = factor(fulldata$PriceRank, ordered=TRUE)
```

We will consider the variable `Cancels`. Split this variable up by the ticker symbol, and exclude those with fewer than 5 observations:

```
> cancelbyticker = split(fulldata$Cancels,fulldata$Ticker)
> cancelbyticker =
+   cancelbyticker[sapply(cancelbyticker,length)>=5]
```

Now, we can construct the estimate of the CDF for the symbols using the following:

```
> CDFsbyTicker = lapply(cancelbyticker, kCDF, bw="SJ")
```

I want to evaluate each of the inverse CDFs at a regular grid of probabilities ranging from 0.1 to 0.9:

```
> pseq = seq(0.1,0.9, by=0.1)
>
> holdquantiles = matrix(ncol=length(pseq),
+                        nrow=length(CDFsbyTicker))
> for(i in 1:length(CDFsbyTicker))
+ {
+    holdquantiles[i,] = CDFsbyTicker[[i]]$invCDF(pseq)
+ }
```

Exercise: What just happened in this previous code? What was the point of this?

Smoothing Relationships

Smoothing techniques are also very commonly applied to situations where one variable is naturally thought of as a response, i.e., a quantity to be predicted, and other variables are considered as predictors.

The discussion that follows presents an overview of nonparametric regression. The focus is placed on constructing useful summaries that will lead to better understanding of the relationship between the response and the predictor(s).

Example: Pricing Options

A European call option on a stock is a contract that gives the holder the right to purchase that stock at the named strike price on the expiration date.

A American call option is the same, but allows the holder to purchase the stock at that price **on or before** the expiration date.

The classic Black-Scholes Theory for option pricing establishes a price for a European option as a function of the following: the strike price, the time to expiration, the current asset price, the volatility of the price, and the current risk-free rate of return.

We will consider a data set consisting of 1,000 call options sampled on September 29, 2017.

To generate this sample, NYSE stock symbols were randomly chosen, and then one currently-listed call option was randomly chosen for each equity, weighted by the volume of trading.

Information recorded for each were as follows: The strike price, the time to expiration, the current asset price, the historical volatility (based on daily returns of 30 most recent trading days), the implied volatility and the Black-Scholes price for the option (with a risk-free rate of zero assumed).

The full R code to generate such a sample can be found on Canvas. There are some important components worth pointing out:

The package `quantmod` includes a function `getOptionChain()` which allows one to easily obtain the current options information for a ticker symbol.

The package `fOptions` includes `GBSOption()` and `GBSVolatility()`, which can be used to calculate the Black-Scholes price and the implied volatility for an option, respectively.

The package `TTR` includes a function `stockSymbols()` which will return all of the ticker symbols (along with other useful information) for the AMEX, NASDAQ, or NYSE exchange.

Exercise: The following appears as part of the provided code.

```
> holdout = try(getOptionChain(cand,NULL,auto.assign=F),TRUE)
>
> if("try-error" %in% class(holdout) || length(holdout) == 0)
+ {
+    next
+ }
```

What is the purpose of this syntax?

This sample can be found in the Data Sets area on Canvas, with the name `optionssample09302017.txt`.

```
> optionsdata = read.table("optionssample09302017.txt",
+                          sep=",", header=T)
```

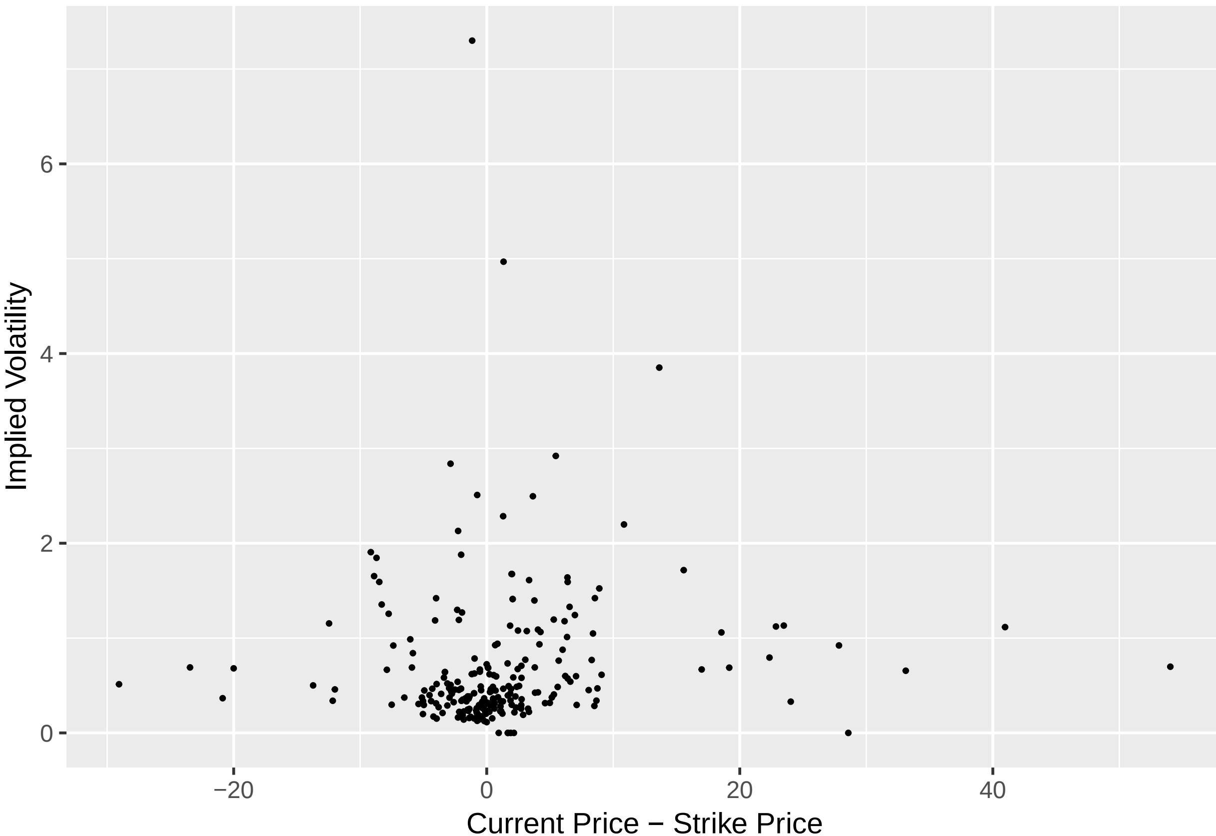
Our objective is to explore the relationships between the current ask price for the option and the available variables.

This is an "empirical" alternative to Black-Scholes theory, whose limitations are well-known, mainly due to the failure of the underlying normality assumption.

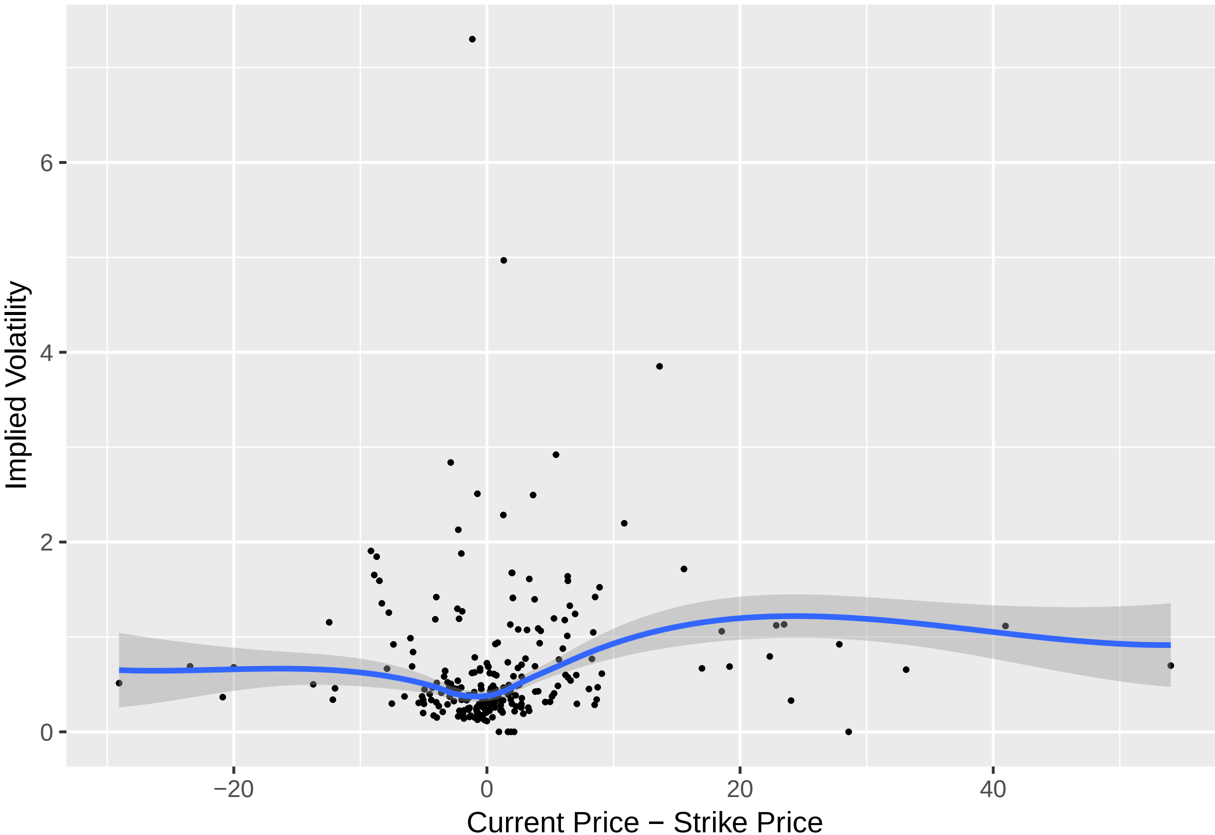
A classic manifestation of the failure of Black-Scholes is the so-called volatility smile. Under the theory, a plot of implied volatility versus how far “in the money” the option currently is should be flat for options with the same expiration date.

Let’s create this plot in our case. Is there evidence of the smile here?

```
> optionsdata$inmoney = optionsdata$curprice -
+                       optionsdata$strike
> optionsdata20 = filter(optionsdata, timetoexpiry==20)
> ggplot(optionsdata20, aes(x=inmoney,y=implvol)) +
+   geom_point(size=0.5) +
+   labs(x="Current Price - Strike Price",
+        y="Implied Volatility")
```


Implied Volatility
0
2
4
6
−20
0
20
40
Current Price − Strike Price

1



Implied Volatility
0
2
4
6
−20
0
20
40
Current Price − Strike Price

1

Local Linear Regression

The figure on the previous page shows a nonparametric regression or nonparametric smooth of the scatter plot.

There are many variants of nonparametric regression, but here we focus on local linear regression, a version of local polynomial regression. This approach has proven to be very powerful, and possesses many strong theoretical properties to justify its use.

Briefly stated, this procedure works by fitting a sequence of linear models: Each is fit not to the entire data set, but to only data within a neighborhood of a target point. The size of this neighborhood is a smoothing parameter: Large neighborhoods yield a large degree of smoothing, while small neighborhoods result in minimal smoothing.

Our model here is that we observe (x_i, Y_i) for $i = 1, 2, \ldots, n$ and that

$$Y_i = f(x_i) + \epsilon_i$$

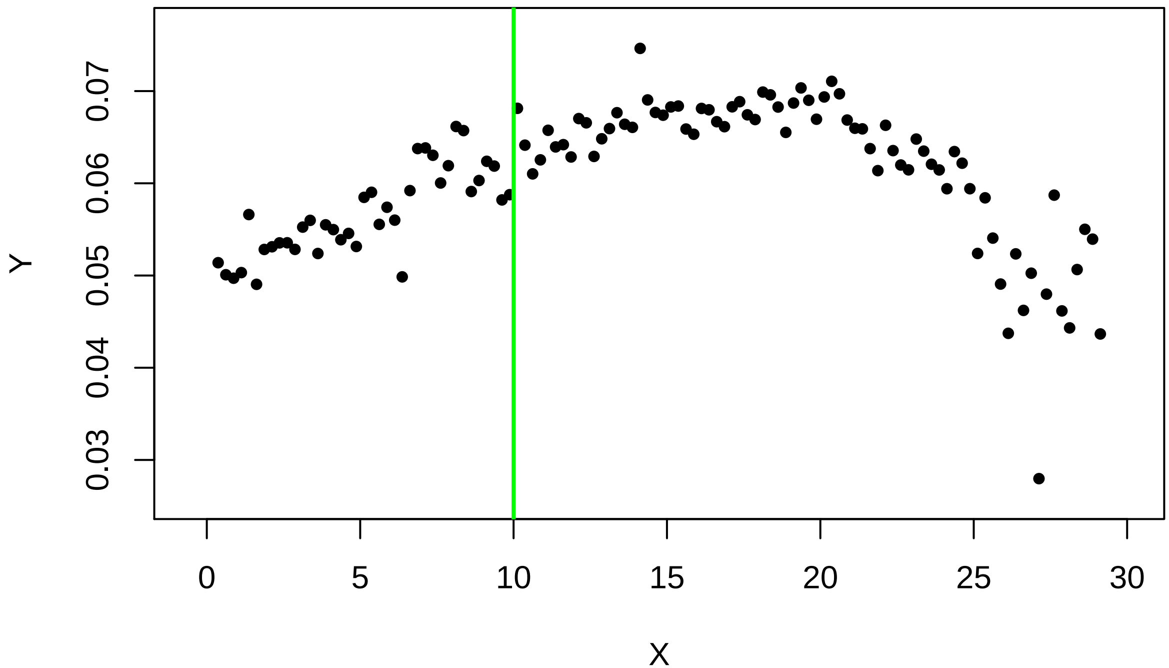
where the ϵ_i are iid with mean zero and variance σ^2. Assuming that the ϵ_i are normal will lead to further nice properties, but this development does not require that assumption.

In order to construct the local linear regression estimate of $f(\cdot)$, it is best to consider a sequence of steps for each fixed x_0 at which $f(\cdot)$ will be estimated.

(For context on these data, see Ruppert and Matteson, Section 11.3.)

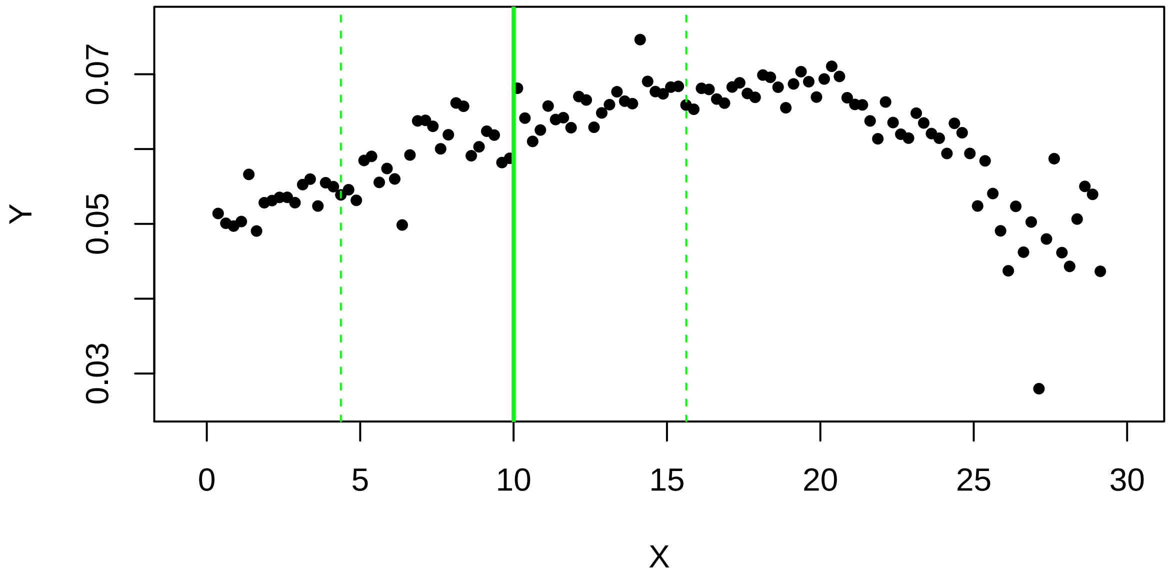
Step One: Fix the target point x_0.

Our objective is to estimate the regression function at x_0.



Step Two: Create the neighborhood around x_0.

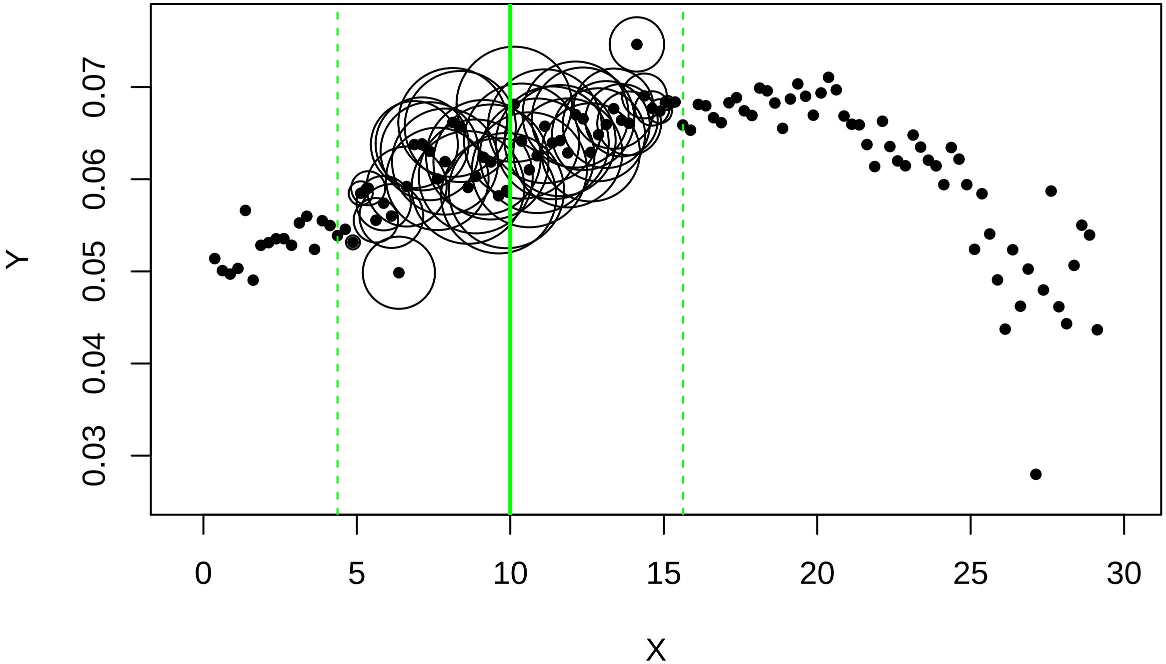
A common way to choose the neighborhood size is to choose is large enough to capture proportion α of the data. This parameter α is often called the span. A typical choice is $\alpha \approx 0.5$.



Step Three: Weight the data in the neighborhood.

Values of x which are close x_0 will receive a larger weight than those far from x_0. Denote by w_i the weight placed on observation i. The default choice is the tri-cube weight function:

$$w_i = \begin{cases} \left(1 - \left|\frac{x_i - x_0}{\text{max dist}}\right|^3\right)^3, & \text{if } x_i \text{ in the neighborhood of } x_0 \\ 0, & \text{if } x_i \text{ is not in neighborhood of } x_0 \end{cases}$$

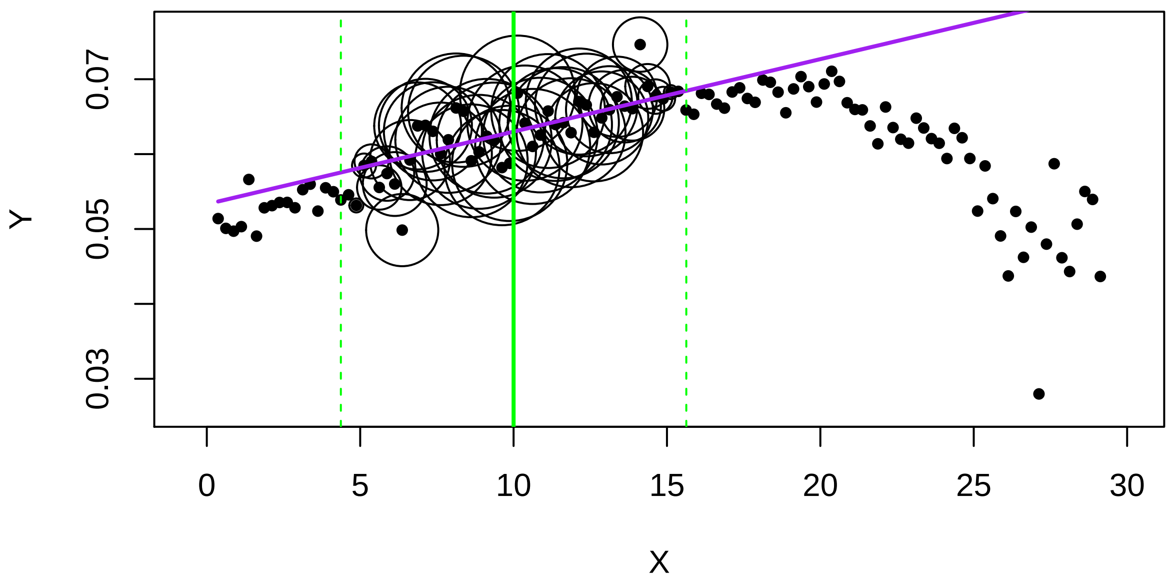


1

Step Four: Fit the local regression line.

This is done by finding β_0 and β_1 to minimize the weighted sum of squares

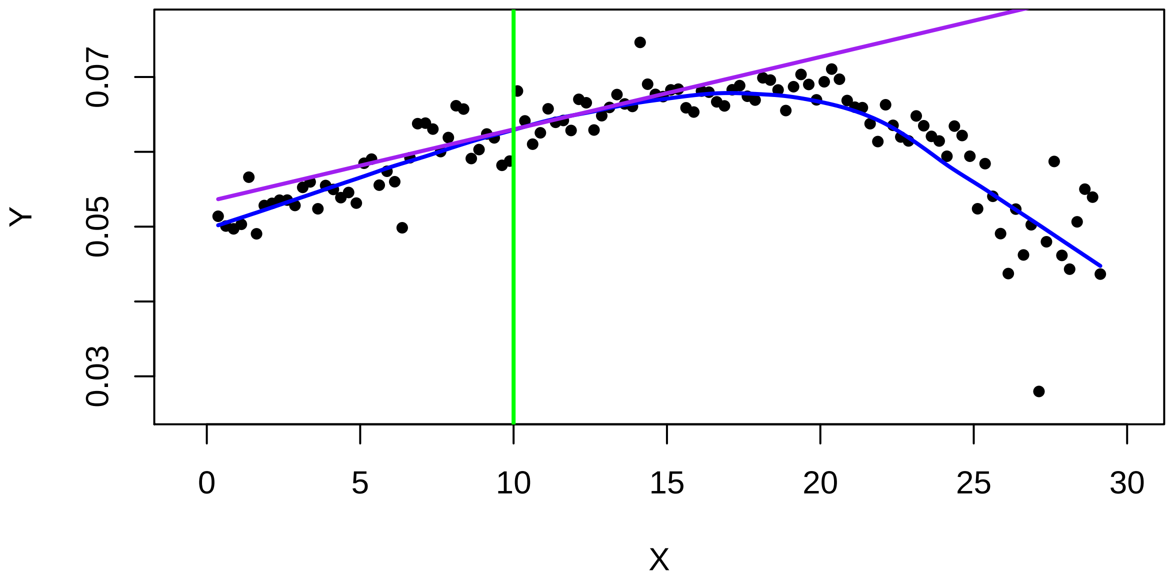
$$\sum_{i=1}^{n} w_i \left(y_i - (\beta_0 + \beta_1 x_i)\right)^2$$



Step Five: Estimate $f(x_0)$.

This is done using the fitted regression line to estimate the regression function at x_0:

$$\widehat{f}(x_0) = \widehat{\beta}_0 + \widehat{\beta}_1 x_0$$



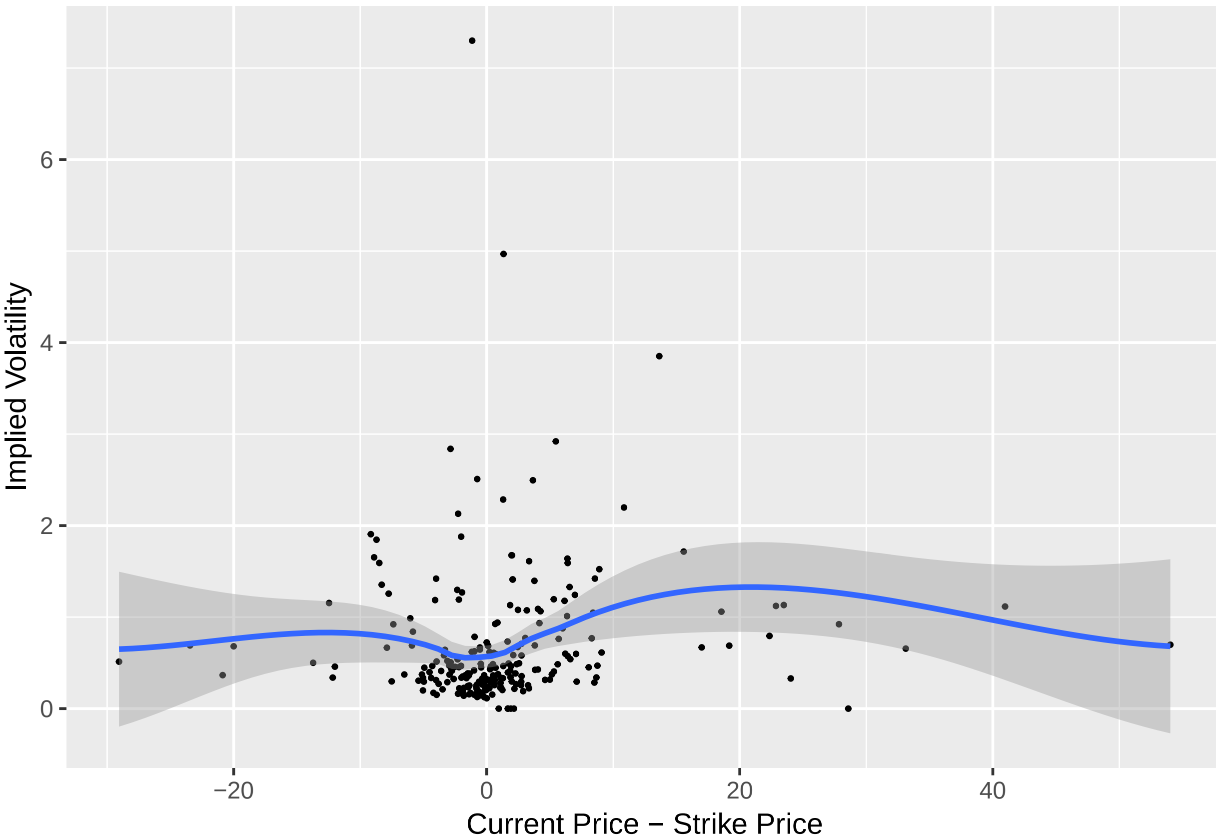
This is implemented in R using `loess()`:

```
> holdlo = loess(implvol ~ inmoney,
+                     data=optionsdata20, degree=1)
```

Setting `degree = 1` is necessary in order to get local linear models. You can use the default `degree = 2` to get local quadratic fits, but this is usually unnecessary.

`ggplot` makes it easy to place the local linear regression fit onto the plot. Note how `degree=1` is passed on to `loess()`:

```
> ggplot(optionsdata20, aes(x=inmoney,y=implvol)) +
+   geom_point(size=0.5) + geom_smooth(method="loess",
+               method.args=list(degree=1)) +
+   labs(x="Current Price - Strike Price",
+         y="Implied Volatility")
```


Implied Volatility
0
2
4
6
−20
0
20
40
Current Price − Strike Price

1

Smoothing parameter selection in Regression

Just like the choice of the bandwidth in kernel density estimation, the smoothing parameter in nonparametric regression controls how "smooth" the resulting estimate is: A smaller value leads to a "rougher" estimate, a larger value gives a "smoother" estimate.

The argument `span` to `loess()` controls the span, as defined above. The default value is 0.75.

Exercise: Try refitting the model above with smaller and larger values of `span`.

Automated approaches to smoothing parameter selection do exist. Different ideas are available, but a very popular approach is based on cross validation.

The motivation behind cross validation is that we seek a regression function that makes accurate predictions for **new** observations, not for those that are used to fit the model. A regression function that makes accurate predictions only for training data is said to be overfit. Overfitting is a serious problem, and is a leading drawback to the use of flexible nonparametric and machine learning methods.

Exercise: Discuss how nonparametric regression, and the choice of the smoothing parameter, leads to increased risk of overfitting. Does choosing the smoothing parameter to minimize residual sum of squares seem like a good idea?

The residual sum of squares in this case is

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(Y_i - \widehat{f}_h(x_i)\right)^2$$

where $\widehat{f}_h(x_i)$ is the estimate of the regression function $f()$ when smoothing parameter h is used.

Instead of choosing h to minimize RSS, we minimize the leave-one-out cross validation score:

$$\mathrm{LOOCV} = \sum_{i=1}^{n} \left(Y_i - \widehat{f}_h^{(-i)}(x_i)\right)^2$$

where $f_h^{(-i)}(x_i)$ is the estimate of the regression function when the i^{th} observation is excluded from the training set.

The idea here is that, by leaving out observation i, we are treating it as if it were a "new" observation, and testing to see how well this choice of h does in predicting the response.

There is another R function `loess.as()`, which is part of the package `fANCOVA`, which has built in selection of the span.

The syntax is

```
> holdlo2 = loess.as(optionsdata20$inmoney,
+         optionsdata20$implvol,criterion = "gcv")
```

With this function the default is `degree = 1`. Setting `criterion="gcv"` uses generalized cross validation, which is an approximation to leave-one-out cross validation.

The optimal span can be found via

```
> holdlo2$pars$span
[1] 0.7038925
```

Robust Fits

Exercise: Reconsider the smooth shown in the Figure on Slide 53 above. Does it appear to be a good fit?

The problem in the fit is that the extreme values are overly influential. Least squares is particularly sensitive to these extremes; the large response values are "pulling up" the estimate.

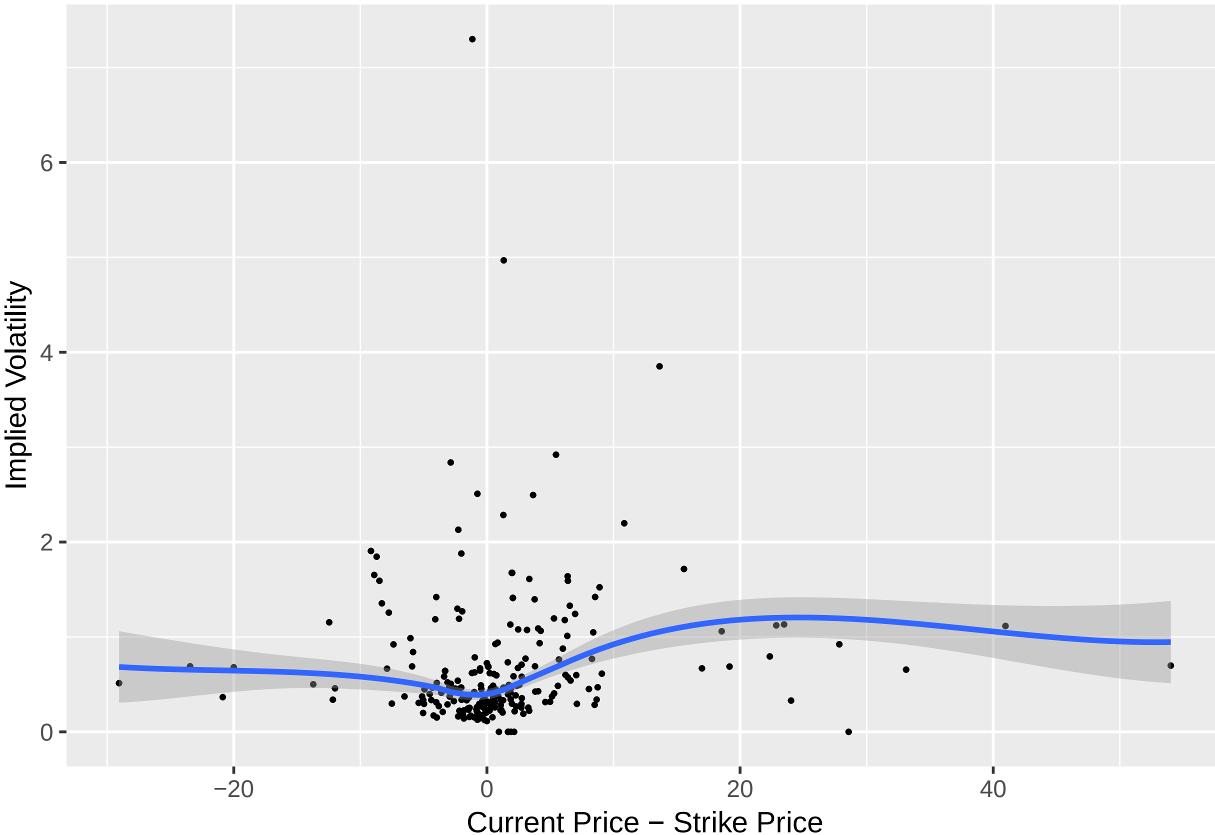
Fortunately, `loess()` and `loess.as()` allow one to move away from least squares, by setting the argument `family="symmetric"` instead of the default `family="gaussian"`.

```
> holdlo3 = loess.as(optionsdata20$inmoney,
+         optionsdata20$implvol, family="symmetric",
+         criterion = "gcv")
```

Exercise: What is the reasoning behind using the name `family`, with value `symmetric`, to specify a robust fit?

Now, we refit with this span, using `family="symmetric"`:

```
> ggplot(optionsdata20, aes(x=inmoney,y=implvol)) +
+   geom_point(size=0.5) +
+   geom_smooth(method="loess", span=holdlo3$pars$span,
+       method.args=list(degree=1,
+        family="symmetric")) +
+   labs(x="Current Price - Strike Price",
+        y="Implied Volatility")
```


Implied Volatility
0
2
4
6
−20
0
20
40
Current Price − Strike Price

1

Confidence Interval

The depicted gray region around the regression function estimate is a 95% pointwise confidence band for the regression function.

The **proper** interpretation is as follows: For each predictor value x, the gray region displays a 95% confidence interval for $f(x)$.

Here are two crucial things to understand regarding the band:

1. The shaded area does **not** show a **simultaneous** 95% confidence band for the **entire** regression function.
2. The shaded area does **not** show 95% prediction intervals for new observations.

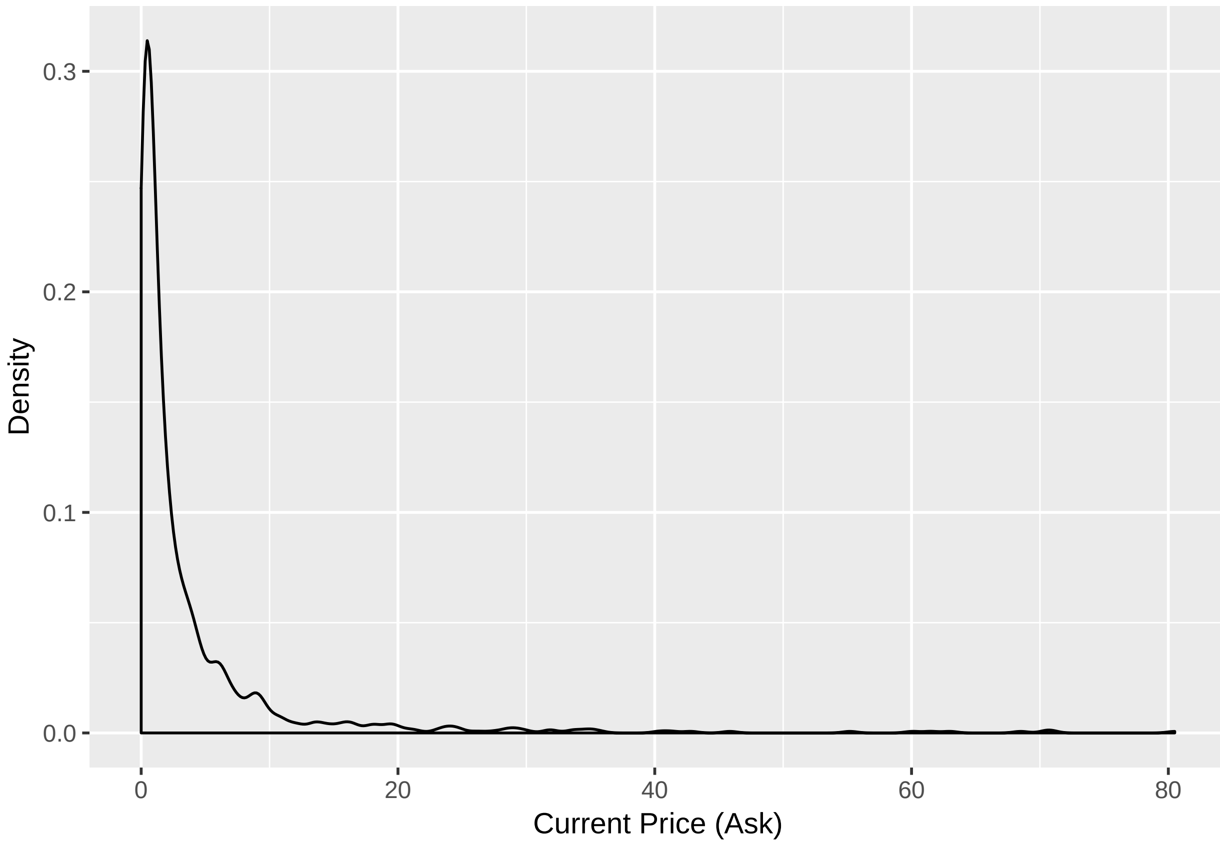
Exercise: Discuss the two misconceptions given above, and how they differ from the real interpretation.

Exploring Relationships

Now we will dig a little deeper into the options sample, and explore relationships between the relevant properties of an option and its current ask price.

First, consider the distribution of the prices. There is a notable skew. In what follows we will work with the logarithm of the ask price.

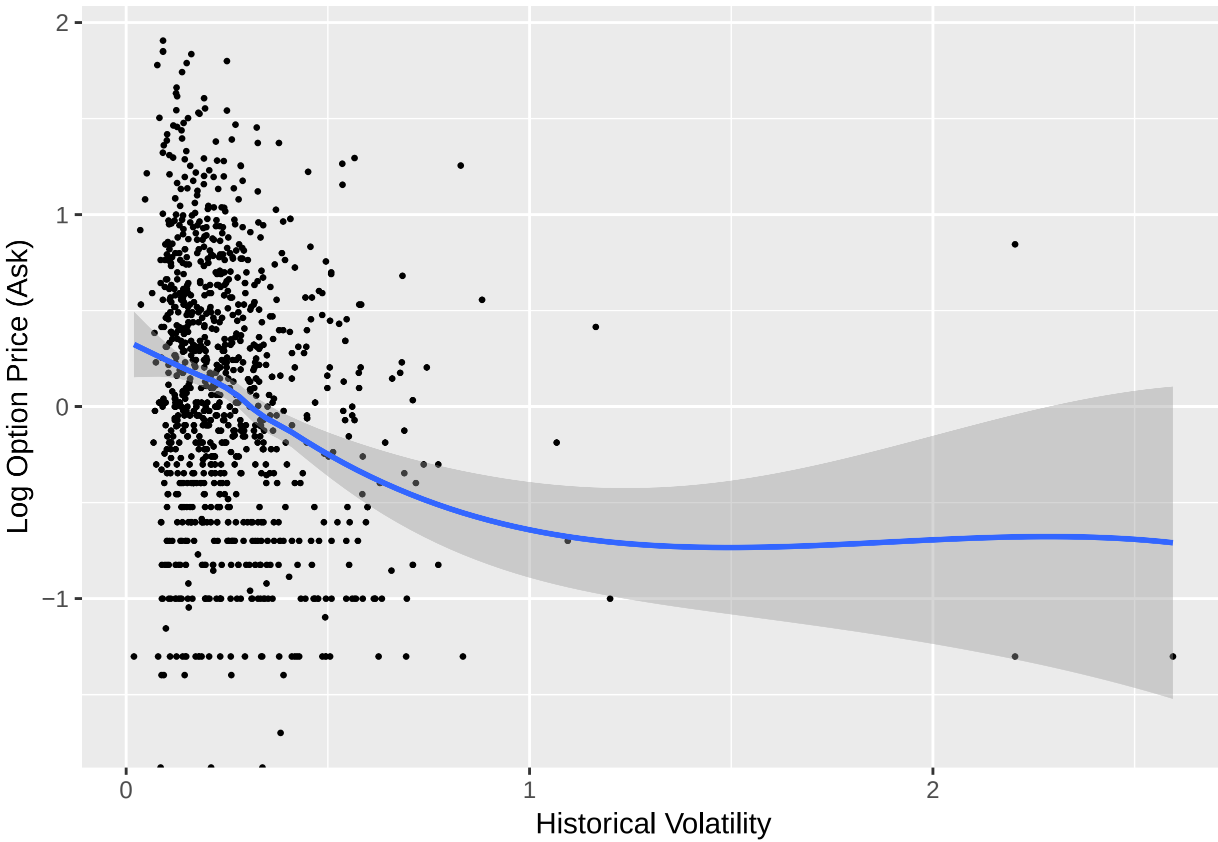
```
> ggplot(optionsdata, aes(x=ask)) + geom_density() +
+   labs(x="Current Price (Ask)", y="Density")
```



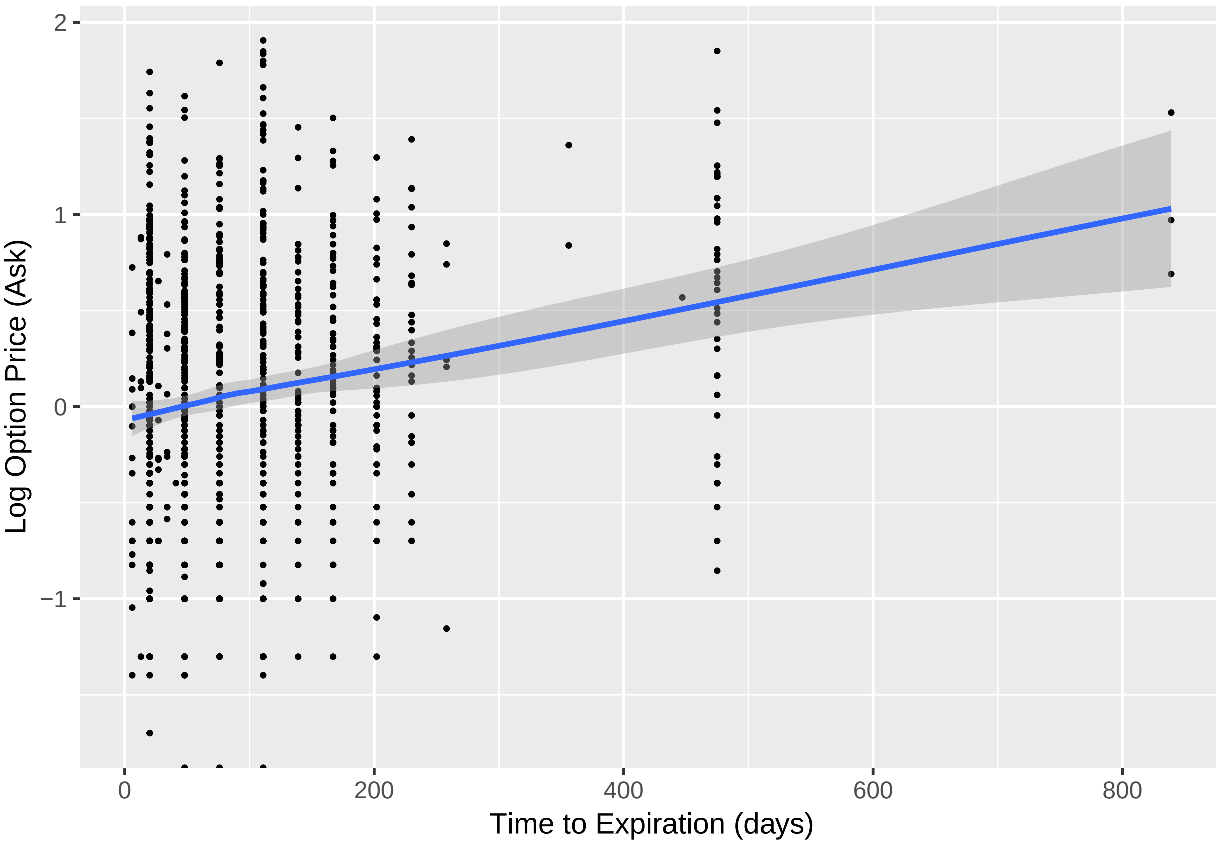
1

The next plots compare historical volatility, time to expiration, and strike price to the price. The code is only shown here for the first plot:

```
> ggplot(optionsdata, aes(x=histvol,y=log10(ask))) +
+   geom_point(size=0.5) +
+   geom_smooth(method="loess",
+               method.args=list(degree=1)) +
+   labs(x="Historical Volatility",
+        y="Log Option Price (Ask)")
```

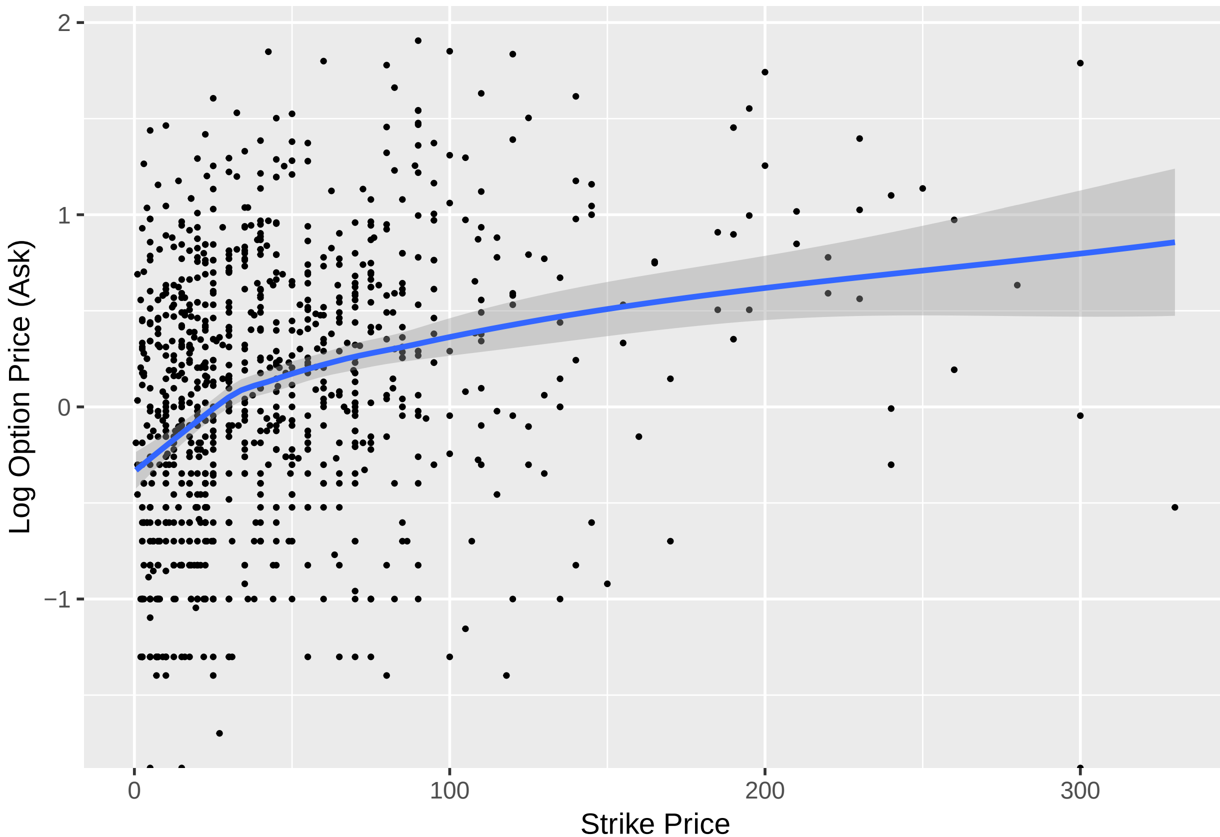


1



Log Option Price (Ask)
Time to Expiration (days)
2
1
0
−1
0
200
400
600
800

1


2
1
0
−1
Log Option Price (Ask)
0
100
200
300
Strike Price

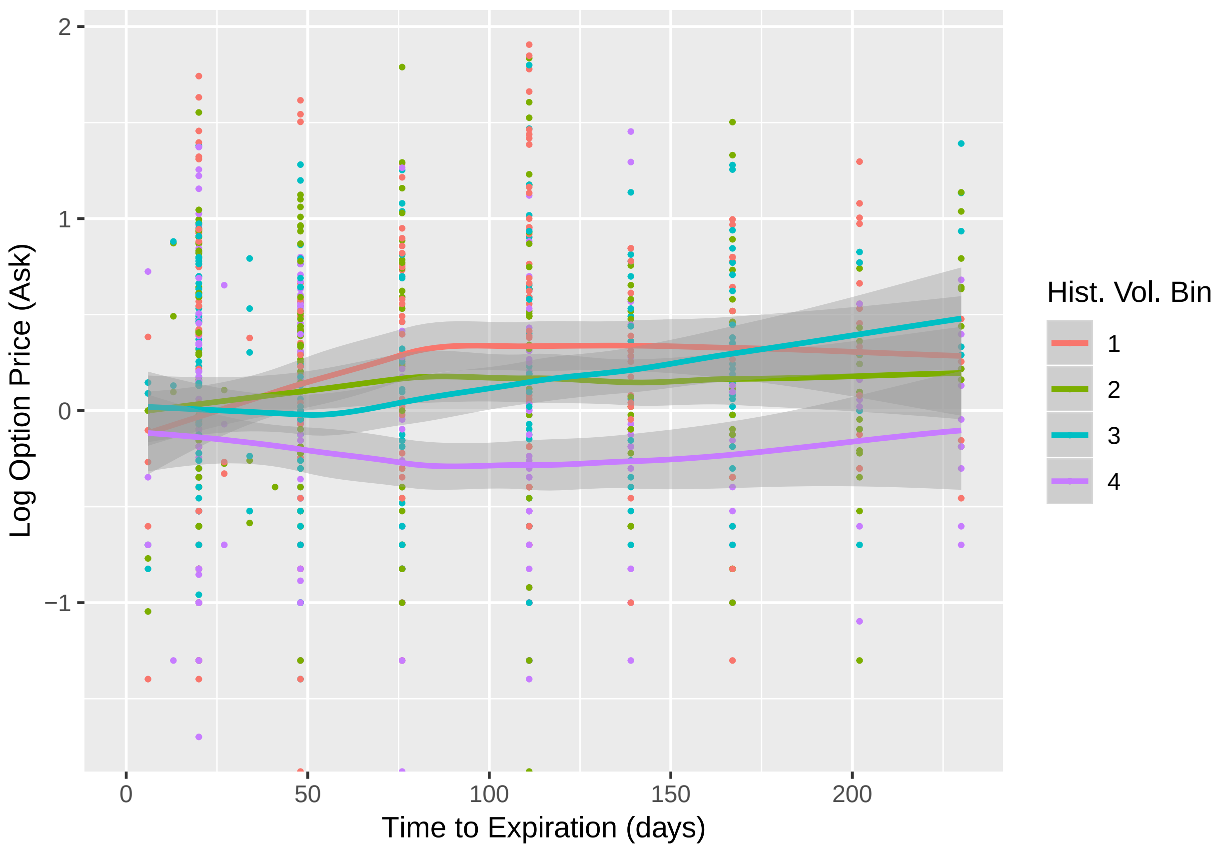
Exercise: One might guess that there is a strong relationship between the price of the option and how far "in the money" it currently is, i.e., the difference between the current price of the stock and its strike price. Construct a plot to explore this.

Next we will construct a plot attempting to explore how a pair of variables relate to the price.

Let's bin the historical volatility into four groups:

```
> breaks = c(-Inf, quantile(optionsdata$histvol,
+                            c(0.25,0.5,0.75)), Inf)
> optionsdata$histvolbin = cut(optionsdata$histvol,
+                       breaks, labels=c(1,2,3,4))
```

```
> ggplot(optionsdata, aes(x=timetoexpiry,y=log10(ask),
+                          color=histvolbin)) +
+   geom_point(size=0.5) +
+   geom_smooth(method="loess",
+               method.args=list(degree=1)) +
+   labs(x="Time to Expiration (days)",
+        y="Log Option Price (Ask)",
+        color="Hist. Vol. Bin") +
+   xlim(0,230)
```



Log Option Price (Ask)
2
1
0
−1
0
50
100
150
200
Time to Expiration (days)
Hist. Vol. Bin
1
2
3
4

Exercise: Interpret the graph on the previous page.