Fitting curves to data

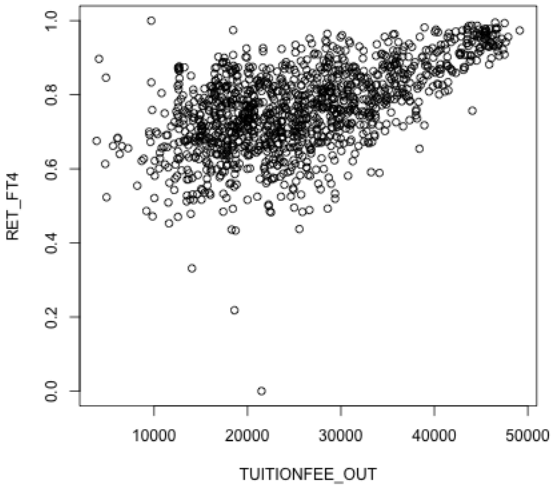
Comparing groups evaluates how a **continuous variable** (often called the response or independent variable) is related to a **categorical variable**. In our flight example, the continuous variable is the flight delay and the categorical variable is which airline carrier was responsible for the flight.

Now let us turn to relating two continuous variables. We will review the method that you learned in data 8 – linear regression – and discuss parametric models for doing inference in this scenario. The parametric model for linear regression is quite important and well known, and forms the basis for a great deal of statistical understanding about linear regression. Then we will turn to expanding these ideas for more flexible curves than just a line.

1 Linear regression with one predictor

Let's consider the following data collected by the Department of Education regarding undergraduate institutions (`https://catalog.data.gov/dataset/college-scorecard`). The department of education collects a great deal of data regarding the individual colleges/universities (including for-profit schools). Let's consider two variables, the tuition costs and the retention rate of students (percent that return after first year). We will exclude the for-profit institutes (there aren't many in this particular data set), and focus on out-of-state tuition to make the values more comparable between private and public institutions.

```
dataDir <- "../finalDataSets"
scorecard <- read.csv(file.path(dataDir, "college.csv"),
    stringsAsFactors = FALSE)
scorecard <- scorecard[-which(scorecard$CONTROL ==
    3), ]
plot(scorecard[, c("TUITIONFEE_OUT", "RET_FT4")])
```



What do you observe in these relationships?

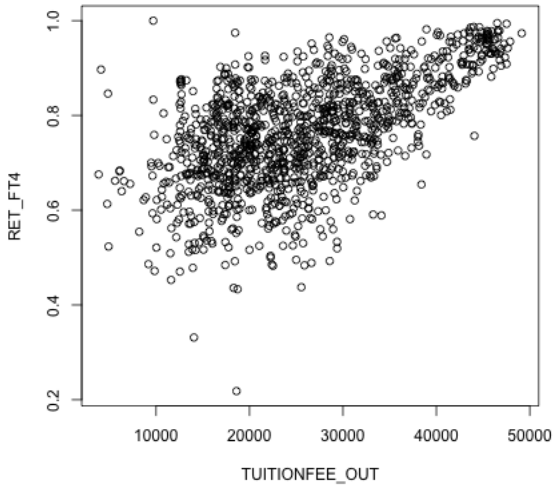
It's not clear what's going on with this observation, but a 0% return rate is an unlikely value for an accreditated institution and is highly likely to be an error. So for now we'll drop that value. This is not something we want to do lightly, and points to the importance of having some understanding of the data – knowing that 0% is a suspect number, for example. Even better would be someone to consult with subject matter expertise on this data.

```
scorecard[scorecard[, "RET_FT4"] == 0, ]
```

```
##         X                                    INSTNM STABBR ADM_RATE_ALL
## 1238 5930 Pennsylvania College of Health Sciences     PA          398
##      SATMTMID SATVRMID SAT_AVG_ALL AVGFACSAL TUITFTE TUITIONFEE_IN
## 1238      488      468         955      5728   13823         21502
##      TUITIONFEE_OUT CONTROL UGDS UGDS_WHITE UGDS_BLACK UGDS_HISP
## 1238          21502       2 1394     0.8364     0.0445    0.0509
##      UGDS_ASIAN UGDS_AIAN UGDS_NHPI UGDS_2MOR UGDS_NRA UGDS_UNKN
## 1238     0.0294     7e-04    0.0029    0.0014        0    0.0337
##       INC_PCT_LO  INC_PCT_M1  INC_PCT_M2  INC_PCT_H1  INC_PCT_H2 RET_FT4
## 1238 0.367788462 0.146634615 0.227163462 0.175480769 0.082932692       0
##      PCTFLOAN C150_4 mn_earn_wne_p10 md_earn_wne_p10 PFTFAC
## 1238   0.6735 0.6338           53500           53100 0.7564
```

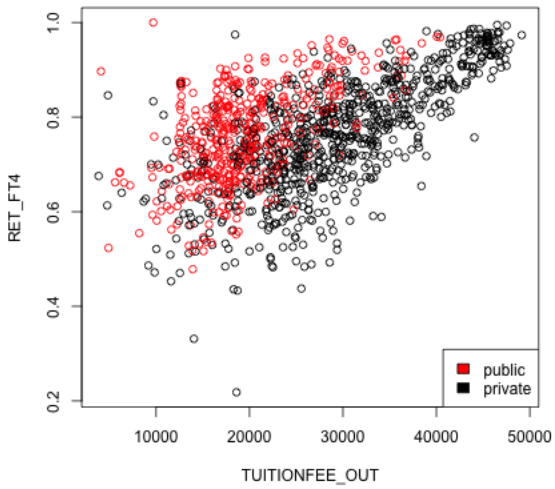
```
scorecard <- scorecard[-which(scorecard[, "RET_FT4"] ==
```

```
    0), ]
plot(scorecard[, c("TUITIONFEE_OUT", "RET_FT4")])
```



What do I see if I color the universities by whether they are private or not?

```
plot(scorecard[, c("TUITIONFEE_OUT", "RET_FT4")], col = c("red",
    "black")[scorecard[, "CONTROL"]])
legend("bottomright", c("public", "private"), fill = c("red",
    "black"))
```



This highlights why it is very important to use more than one variable in trying to understand patterns or predict, which we will spend much more time on later in the course. But for now we are going to focus on one variable analysis, so lets make this a more valid exercise by just considering private schools.

```
private <- subset(scorecard, CONTROL == 2)
public <- subset(scorecard, CONTROL == 1)
```

1.1 Linear Fit

These are convenient variables to consider the simplest relationship you can imagine for the two variables – a linear one:

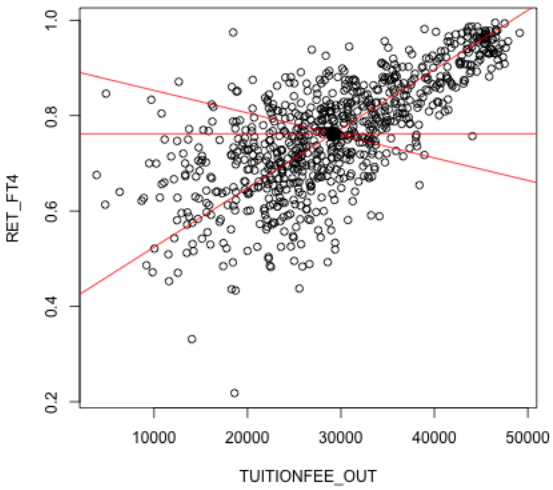
$$y = \beta_0 + \beta_1 x$$

Of course, this assumes there is no noise, so instead, we often write

$$y = \beta_0 + \beta_1 x + e$$

where e represents some noise that gets added to the $\beta_0 + \beta_1 x$; e explains why the data do not exactly fall on a line.[1]

We do not know β_0 and β_1. They are parameters of the model. We want to estimate them from the data. There are many possible lines, of course, even if we force them to go through the middle of the data (e.g. the mean of x,y)

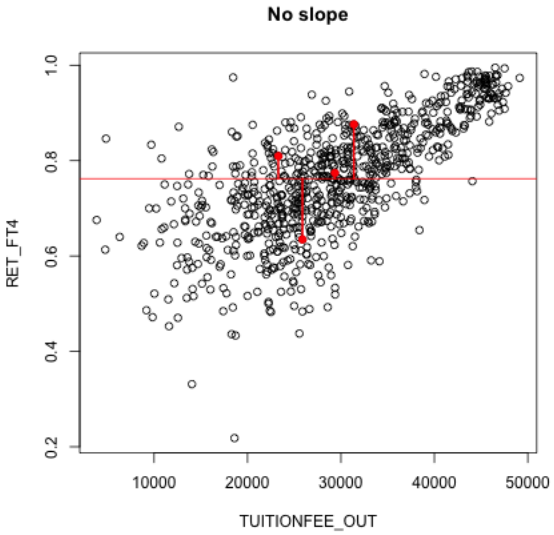


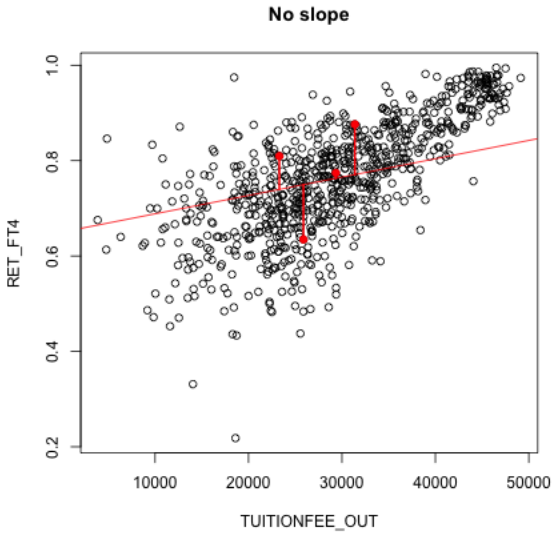
[1]It is useful to remember that *adding* noise is not the only option – this is a *choice* of a model.

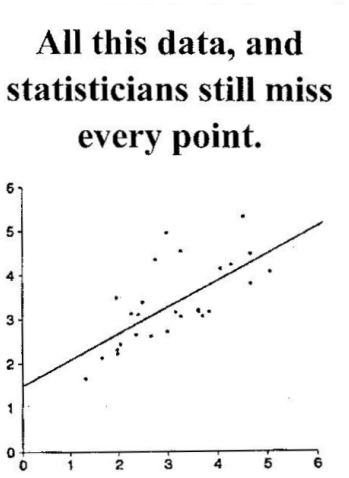
How do we decide which one is best? A reasonable choice is one that makes the smallest errors in predicting the response y. For each possible β_0, β_1 pair (i.e. each line), we can calculate the prediction from the line,

$$\hat{y}(\beta_0, \beta_1) = \beta_0 + \beta_1 x$$

and compare it to the actual observed y.





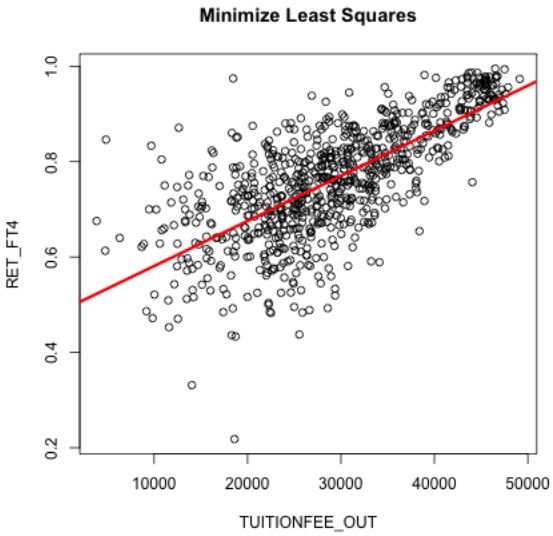


Of course, for any particular point, we can make this zero, but as the above graphic

jokes , the goal is not to exactly fit any particular point.[2] Instead, we need to find an agreement across all of the data points.

The most common one is that used by the method of **least squares**, where our measure of overall error for a β_0, β_1 is the average squared error,

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}(\beta_0, \beta_1))^2$$



What do you notice about this line?

```
lm(RET_FT4 ~ TUITIONFEE_OUT, data = private)
```

```
##
## Call:
## lm(formula = RET_FT4 ~ TUITIONFEE_OUT, data = private)
##
## Coefficients:
##    (Intercept)  TUITIONFEE_OUT
##      4.863e-01       9.458e-06
```

```
lmPrivate <- lm(RET_FT4 ~ TUITIONFEE_OUT, data = private)
names(lmPrivate)
```

[2]The above graphic comes from the 1999 winner of the annual statistics department contest for tshirt designs

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
lmPrivate$coefficients
```

```
##    (Intercept) TUITIONFEE_OUT
##   4.863443e-01   9.458235e-06
```

```
coef(lmPrivate)
```

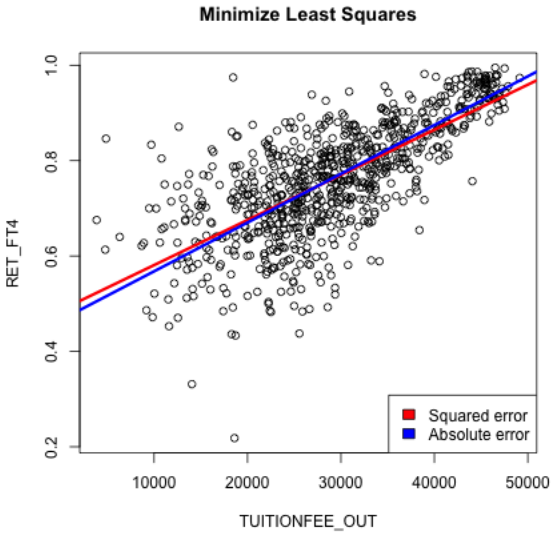
```
##    (Intercept) TUITIONFEE_OUT
##   4.863443e-01   9.458235e-06
```

How do you interpret the coefficients?

How much predicted increase in do you get for an increase of $10,000 in tuition?

Absolute Errors Least squares is quite common, particularly because it quite easily mathematically to find the solution. However, it's equally compelling to use the average absolute error, rather than squared error:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}(\beta_0, \beta_1)|$$



While least squares is more common for historical reasons, using absolute error is in many ways more compelling, just like the median can be better than the mean for summarizing the distribution of a population. With squared-error, large differences become even larger, increasing the influence of outlying points, because reducing the squared error for these outlying points will significantly reduce the overall average error.

We will continue with the traditional least squares, since we are not (right now) going to spend very long on regression before moving on to other techniques for dealing with two continuous variables.

2 Review: Inference for linear regression

One question of particular interest is determining whether $\beta_1 = 0$. Why? (Consider this data on college tuition – what does $\beta_1 = 0$ imply)?

We can use the same strategy of inference for asking this question – hypothesis testing, p-values and confidence intervals.

As a hypothesis test, we have a null hypothesis of:

$$H_0 : \beta_1 = 0$$

We can also set up the hypothesis

$$H_0 : \beta_0 = 0$$

However, this is (almost) never interesting. Consider our data: what would it mean to have $\beta_0 = 0$?

Does this mean we can just set β_0 to be anything, and not worry about it?

2.1 Bootstrap Confidence intervals

Once we get estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, how can we get bootstrap confidence intervals for the estimates?

```
bootstrapLM <- function(y, x, repetitions, confidence.level = 0.95) {
    stat.obs <- coef(lm(y ~ x))
    bootFun <- function() {
        sampled <- sample(1:length(y), size = length(y),
            replace = TRUE)
        coef(lm(y[sampled] ~ x[sampled]))
    }
    stat.boot <- replicate(repetitions, bootFun())
    nm <- deparse(substitute(x))
    row.names(stat.boot)[2] <- nm
    level <- 1 - confidence.level
    confidence.interval <- apply(stat.boot, 1, quantile,
        probs = c(level/2, 1 - level/2))
    return(list(confidence.interval = cbind(lower = confidence.interval[1,
        ], estimate = stat.obs, upper = confidence.interval[2,
        ]), bootStats = stat.boot))
}
```

```
privateBoot <- with(private, bootstrapLM(y = RET_FT4,
    x = TUITIONFEE_OUT, repetitions = 10000))
privateBoot$conf
```

```
##                         lower     estimate        upper
## (Intercept)    4.627561e-01 4.863443e-01 5.093985e-01
## TUITIONFEE_OUT 8.782905e-06 9.458235e-06 1.014645e-05
```

```
privateBoot[["confidence.interval"]]
```

```
##                         lower     estimate        upper
## (Intercept)    4.627561e-01 4.863443e-01 5.093985e-01
## TUITIONFEE_OUT 8.782905e-06 9.458235e-06 1.014645e-05
```

```
privateBoot[["conf"]]
```

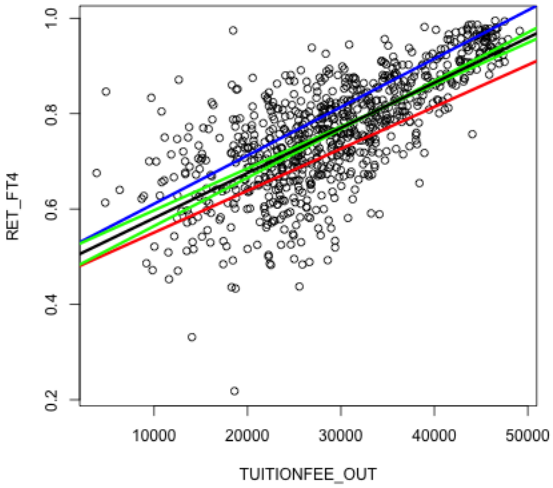
```
## NULL
```

```
privateBoot$conf[2, ] * 10000
```

```
##      lower   estimate      upper
## 0.08782905 0.09458235 0.10146450
```

How do we interpret these confidence intervals? What do they tell us about the problem?

```
plot(private[, c("TUITIONFEE_OUT", "RET_FT4")], col = "black")
abline(a = privateBoot$conf[1, 1], b = privateBoot$conf[2,
    1], col = "red", lwd = 3)
abline(a = privateBoot$conf[1, 3], b = privateBoot$conf[2,
    3], col = "blue", lwd = 3)
abline(a = privateBoot$conf[1, 1], b = privateBoot$conf[2,
    3], col = "green", lwd = 3)
abline(a = privateBoot$conf[1, 3], b = privateBoot$conf[2,
    1], col = "green", lwd = 3)
abline(lmPrivate, lwd = 3)
```



In principle, anything in this range is covered by the confidence intervals. However, that is not quite true. Our confidence in where the line is actually is narrower than what is shown, because some of the combinations of values of the two confidence intervals don't actually ever get seen together – these two statistics aren't independent from each other. Separate confidence intervals for the two values don't give you that information.[3]

2.2 Explaining output of `lm`: Parametric Models

If we look at the summary of the `lm` function that does linear regression in R, we see far more information

```
summary(lmPrivate)
```

```
##
## Call:
## lm(formula = RET_FT4 ~ TUITIONFEE_OUT, data = private)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44411 -0.04531  0.00525  0.05413  0.31388
##
## Coefficients:
```

[3]You can actually have joint confidence regions that demonstrate the dependency between these values, but that is beyond this class.

```
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.863e-01  1.020e-02   47.66   <2e-16 ***
## TUITIONFEE_OUT 9.458e-06  3.339e-07   28.32   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08538 on 783 degrees of freedom
## Multiple R-squared:  0.5061,Adjusted R-squared:  0.5055
## F-statistic: 802.3 on 1 and 783 DF,  p-value: < 2.2e-16
```

We see that it automatically spits out a table of estimated values and p-values along with a lot of other stuff. This is exceedingly common – all statistical software programs do this – so let's cover the meaning of the most important components.

Null Hypotheses Why are there 2 p-values? What are the null hypotheses that these p-values correspond to?

Test statistics Recall, for inference, we need to have a way of knowing (or estimating) the distribution of our statistic, so we need a specific test statistic. In this case, our statistic is $\hat{\beta}_1$ – it is a function of the data (Y_i and X_i), and has a distribution.

If we want we can write down the equation for $\hat{\beta}_1$ and $\hat{\beta}_0$ (you don't need to memorize these equations)

$$\hat{\beta}_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

What do you notice about the denominator of $\hat{\beta}_1$?

The numerator is also an average, only now it's an average over values that involve the relationship of x and y. Basically, the numerator is large if for the same observation i, both x_i and y_i are far away from their means.

Parametric Model for the data: `lm` uses a standard parametric method to do inference on our parameters. Let's return to our model,

$$y = \beta_0 + \beta_1 x + e$$

The standard parametric model assumes a distribution about the errors e. Specifically, we assume $e \sim N(0, \sigma^2)$, i..e normal with the *same* (unknown) variance σ^2.

Notice, that this implies each y_i is normally distributed, with mean $\beta_0 + \beta_1 x_i$; so even though the errors e_i are assumed $i.i.d$ the y_i are not i.i.d, why?

This assumption regarding the errors allows us to say what the variance of our estimate $\hat{\beta}_1$ should be (in what follows, just try to follow the logic, you don't need to memorize these equations or understand where they come from). Specifically, if we know σ^2, we have that:

$$\nu_1^2 = var(\hat{\beta}_1) = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

The important thing to note is how our assumption about the errors allows us to write down an equation for the sampline variance of our statistic, just like with the t-test. Of course, we have the same problem as the t-test – we don't know σ^2! But we can estimate σ^2 too and get an estimate of the variance (we'll talk more about how we estimate $\hat{\sigma}$ when we return to linear regression with multiple variables)

$$\hat{\nu}_1^2 = \hat{var}(\hat{\beta}_1 = \frac{\hat{\sigma}^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Our assumption regarding the normal distribution of the errors allows us to go further and say that $\hat{\beta}_1$ is normally distributed too,

$$\hat{\beta}_1 \sim N(\beta_1, \nu_1^2).$$

Notice the similarities in the broad outline of the parametric t-test for two-groups. We have an statistic, $\hat{\beta}_1$, and the parametric model gives us

- what should be the sampling variability of $\hat{\beta}_1$
- the distribution of $\hat{\beta}_1$.

Using this, we can use the same idea as the t-test for two-groups, and create a similar t-statistic for $\hat{\beta}_1$ that standardizes $\hat{\beta}_1$[4]

$$T_1 = \frac{\hat{\beta}_1}{\sqrt{\hat{var}(\hat{\beta}_1)}}$$

Just like the t-test, T_1 should be normally distributed[5]

This is exactly what `lm` gives us.

```
summary(lm(RET_FT4 ~ TUITIONFEE_OUT, data = private))
```

```
##
## Call:
## lm(formula = RET_FT4 ~ TUITIONFEE_OUT, data = private)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44411 -0.04531  0.00525  0.05413  0.31388
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.863e-01  1.020e-02   47.66   <2e-16 ***
## TUITIONFEE_OUT 9.458e-06  3.339e-07   28.32   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08538 on 783 degrees of freedom
## Multiple R-squared:  0.5061,Adjusted R-squared:  0.5055
## F-statistic: 802.3 on 1 and 783 DF,  p-value: < 2.2e-16
```

Confidence intervals We can also create parametric confidence intervals for $\hat{\beta}_1$ in the same way we did for two groups:

$$\hat{\beta}_1 \pm 2\hat{\nu}_1$$

[4]In fact, we can also do this for $\hat{\beta}_0$, with exactly the same logic, though β_0 not interesting.

[5]with the same caveat, that when you estimate the variance, you affect the distribution of T_1, which matters in small sample sizes.

```
confint(lmPrivate)
```

```
##                          2.5 %       97.5 %
## (Intercept)    4.663136e-01 5.063750e-01
## TUITIONFEE_OUT 8.802757e-06 1.011371e-05
```

Estimate σ^2 How do we estimate σ^2? Well, we don't know the true errors, but if we did, a good estimate of σ^2 would be the sample variance of the true errors:

$$\frac{1}{n-1}\sum(e_i - \bar{e})^2$$

These true errors are unknown, but we do have some idea of the errors if our estimated line is reasonably close to the truth. Namely, the error of our data from the estimated line,

$$r_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

The r_i are called the **residuals**. They are often called the errors, but they are not the actual (true) error, however. They are the error from the estimated line.

Using the residuals, we can take the sample variance of the residuals as a good estimate of σ^2,

$$\frac{1}{n-1}\sum(r_i - \bar{r})^2$$

In fact, it is an algebraic fact that $\bar{r} = 0$ – this is not a sign the line is a good fit, but just is always true, even when the line is a lousy fit to the data. Moreover, for regression, a better estimate is to divide by $n-2$ rather than $n-1$. Again, we won't go into why, but for the same reason $n-1$ is a better estimate, in the sense of getting closer to the true variance, than dividing by n; for regression dividing by $n-2$ is better than $n-1$. This gives us

$$\hat{\sigma}^2 = \frac{1}{n-2}\sum_i r_i^2.$$

2.3 Assumptions

Like the t-test, the bootstrap gives a more robust method than the parametric linear model for creating confidence intervals.

The parametric linear model makes the following assumptions:

- Errors are independent
- Errors are i.i.d, meaning they have the same variance
- Errors are normally distributed

The bootstrap makes the same kind of assumptions as with the two group comparisons:

- The i.i.d resampling of the bootstrapped data mirrors how the actual data was generated (i.e. actual data was i.i.d)
- The sample size is large enough that the sample distribution is close to the real distribution.

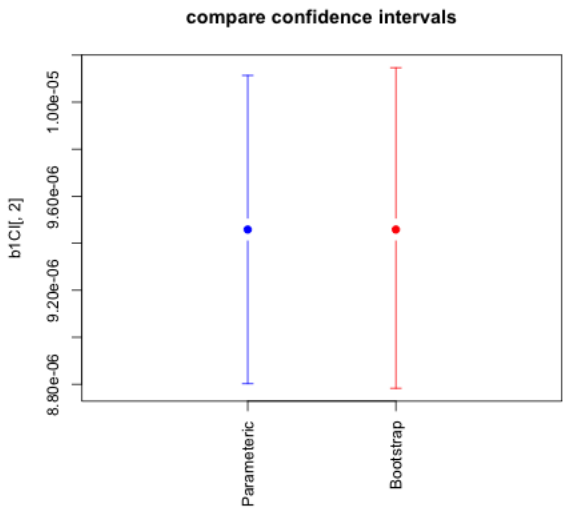
Notice, that both methods assume the data points are independent. This is the most critical assumption for both methods. Both implicitly assume that all of the observations have the same variance (i.i.d). The parametric method makes the further assumption of normality of the errors (like the t-test).

In practice, we do not see much difference in these two methods for our data:

```
b1CI <- rbind(c(lower = confint(lmPrivate)[2, 1], estimate = unname(coef(lmPrivate)[2
    upper = confint(lmPrivate)[2, 2]), privateBoot$conf[2,
    ])
print(b1CI)
```

```
##             lower     estimate        upper
## [1,] 8.802757e-06 9.458235e-06 1.011371e-05
## [2,] 8.782905e-06 9.458235e-06 1.014645e-05
```

```
library(gplots)
par(mar = c(8, 4, 4, 1))
plotCI(b1CI[, 2], ui = b1CI[, 3], li = b1CI[, 1], main = "compare confidence interval
    col = c("blue", "red"), pch = 19, xaxt = "n", xlim = c(0,
        3))
axis(1, at = c(1, 2), labels = c("Parameteric", "Bootstrap"),
    las = 2)
```



2.4 Prediction Intervals

In addition to evaluating the coefficients, we can also look at the prediction we would make. This is better way than the plots we did from the confidence intervals to get an idea of what our predictions at a particular value would actually be.

Prediction How does our model a value, say for tuition of $20,000?

```
coef(lmPrivate)[1] + coef(lmPrivate)[2] * 20000
```

```
## (Intercept)
##    0.675509
```

```
predict(lmPrivate, newdata = data.frame(TUITIONFEE_OUT = 20000))
```

```
##        1
## 0.675509
```

These predictions are themselves statistics based on the data, and the uncertainty/variability in the coefficients carries over to the predictions. So we can also

give confidence intervals for our prediction. There are two types of confidence intervals.

- Confidence intervals about the predicted average response – i.e. prediction of what is the average completion rate for all schools with tuition $20,000.
- Confidence intervals about a particular individual, i.e. prediction of a particular school that has tuition $20,000. These are actually not called confidence intervals, but **prediction intervals**.

Clearly, we predict the same value for both of these, but our estimate of the precision of these estimates varies. Which of these intervals do you think would be wider?

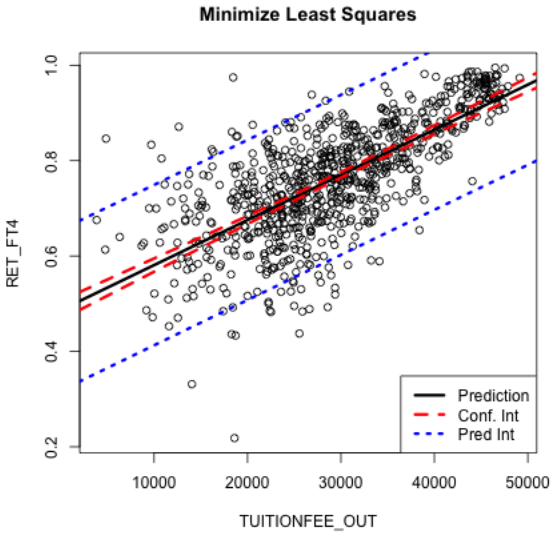
```
predict(lmPrivate, newdata = data.frame(TUITIONFEE_OUT = 20000),
    interval = "confidence")
```

```
##        fit       lwr       upr
## 1 0.675509 0.6670314 0.6839866
```

```
predict(lmPrivate, newdata = data.frame(TUITIONFEE_OUT = 20000),
    interval = "prediction")
```

```
##        fit       lwr      upr
## 1 0.675509 0.5076899 0.843328
```

```
tuit <- seq(2000, 60000, by = 1000)
cint <- predict(lmPrivate, newdata = data.frame(TUITIONFEE_OUT = tuit),
    interval = "confidence")
pint <- predict(lmPrivate, newdata = data.frame(TUITIONFEE_OUT = tuit),
    interval = "prediction")
plot(private[, c("TUITIONFEE_OUT", "RET_FT4")], col = "black",
    main = "Minimize Least Squares")
abline(lm(RET_FT4 ~ TUITIONFEE_OUT, data = private),
    col = "black", lwd = 3)
matlines(tuit, cint[, -1], lty = 2, lwd = 3, col = "red")
matlines(tuit, pint[, -1], lty = 3, lwd = 3, col = "blue")
legend("bottomright", c("Prediction", "Conf. Int",
    "Pred Int"), lty = c(1, 2, 3), col = c("black",
    "red", "blue"), lwd = 3)
```



What do you notice about the difference in the confidence lines? How does it compare to the observed data?

Parametric versus Bootstrap Notice that all of these commands use the parametric assumptions about the errors, rather than the bootstrap. We could bootstrap the confidence intervals for the prediction average. How would we do that?

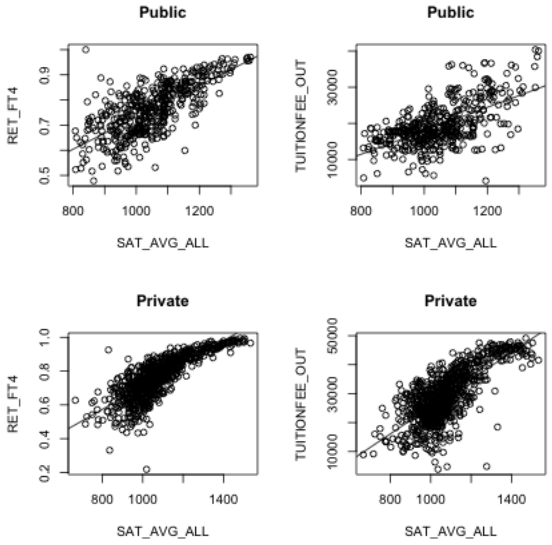
The prediction intervals, on the other hand, rely more on the parametric model for estimating how much variability and individual point will have.

3 Non-linear fits

3.1 Least Squares for Polynomial models & beyond

Least squares will spit out estimates of the coefficients and p-values to any data – the question is whether this is a good idea. For example, consider the variable `SAT_AVG_ALL` that gives the average SAT score for the school. Looking at the public institutions, what do you see as it's relationship to the other two variables?

```
par(mfrow = c(2, 2))
plot(RET_FT4 ~ SAT_AVG_ALL, data = public, main = "Public")
abline(lm(RET_FT4 ~ SAT_AVG_ALL, data = public))
plot(TUITIONFEE_OUT ~ SAT_AVG_ALL, data = public, main = "Public")
abline(lm(TUITIONFEE_OUT ~ SAT_AVG_ALL, data = public))
plot(RET_FT4 ~ SAT_AVG_ALL, data = private, main = "Private")
abline(lm(RET_FT4 ~ SAT_AVG_ALL, data = private))
plot(TUITIONFEE_OUT ~ SAT_AVG_ALL, data = private,
    main = "Private")
abline(lm(TUITIONFEE_OUT ~ SAT_AVG_ALL, data = private))
```



We might imagine that other functions would be a better fit to the data. What might be some reasonable choices of functions?

We can fit other functions in the same way. Take quadratic, for example. What does that look like for a model?

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + e$$

We can, again, find the best choices of those co-efficients by calculating the error for each one

$$\hat{y}_i(\beta_0, \beta_1, \beta_2) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2,$$

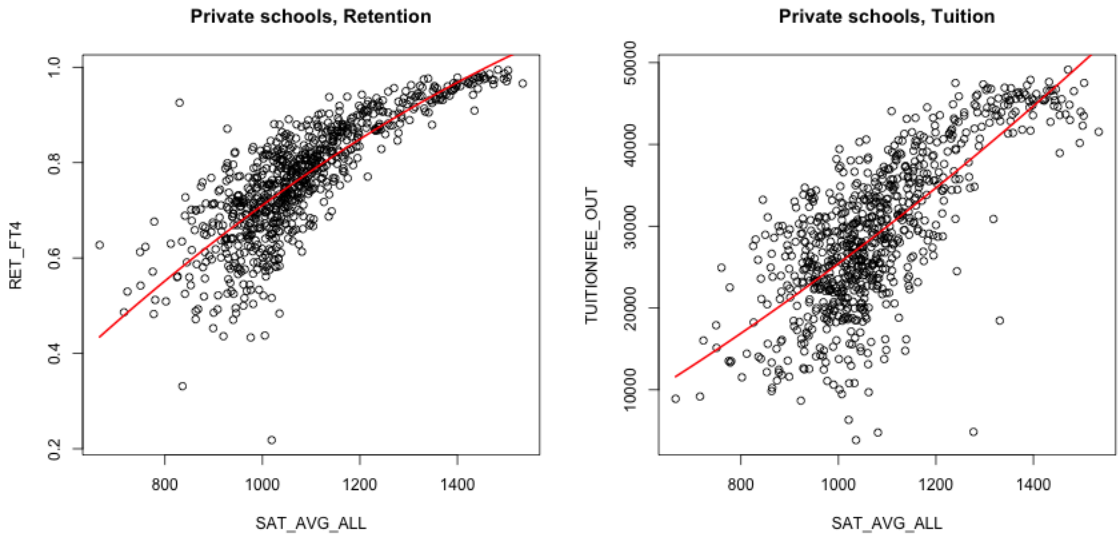
find the error

$$y_i - \hat{y}_i(\beta_0, \beta_1, \beta_2)$$

and trying to find the choices that minimizes the measure of overall error.

If we do least squares for this quadratic model, we get

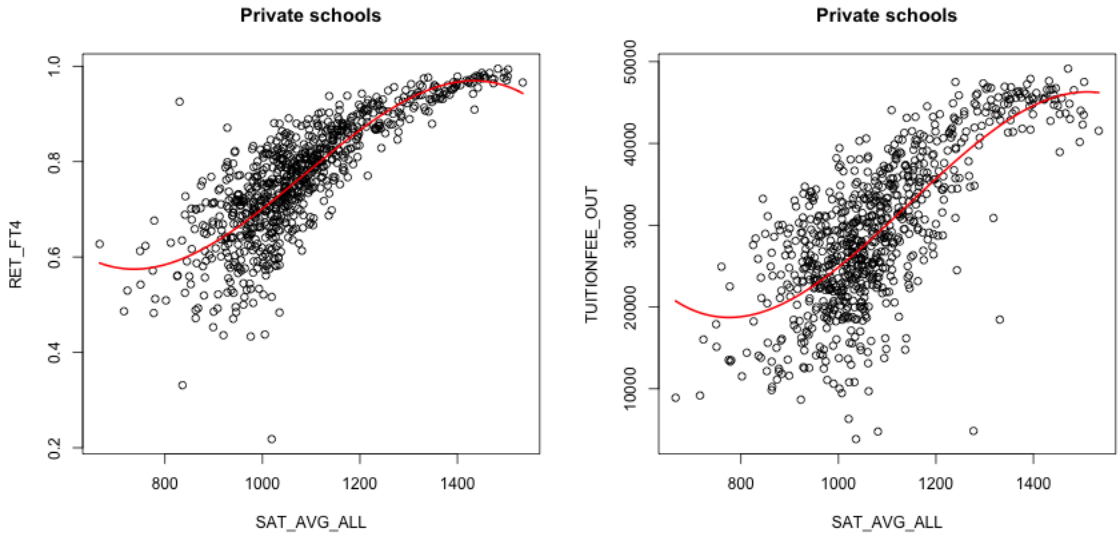
```
modelRET2 <- lm(RET_FT4 ~ SAT_AVG_ALL + I(SAT_AVG_ALL^2),
    data = private)
modelTUT2 <- lm(TUITIONFEE_OUT ~ SAT_AVG_ALL + I(SAT_AVG_ALL^2),
    data = private)
quadCurve <- function(x, cf) {
    cf[1] + cf[2] * x + cf[3] * x^2
}
par(mfrow = c(1, 2))
plot(RET_FT4 ~ SAT_AVG_ALL, data = private, main = "Private schools, Retention")
f <- function(x) {
    quadCurve(x, cf = coef(modelRET2))
}
curve(f, add = TRUE, col = "red", lwd = 2)
plot(TUITIONFEE_OUT ~ SAT_AVG_ALL, data = private,
    main = "Private schools, Tuition")
f <- function(x) {
    quadCurve(x, cf = coef(modelTUT2))
}
curve(f, add = TRUE, col = "red", lwd = 2)
```



It's a little better, but not much. We could try other functions. A cubic function, for example, is exactly the same idea.

$$\hat{y}_i(\beta_0, \beta_1, \beta_2) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3.$$

```
modelRET3 <- lm(RET_FT4 ~ SAT_AVG_ALL + I(SAT_AVG_ALL^2) +
    I(SAT_AVG_ALL^3), data = private)
modelTUT3 <- lm(TUITIONFEE_OUT ~ SAT_AVG_ALL + I(SAT_AVG_ALL^2) +
    I(SAT_AVG_ALL^3), data = private)
cubicCurve <- function(x, cf) {
    cf[1] + cf[2] * x + cf[3] * x^2 + cf[4] * x^3
}
par(mfrow = c(1, 2))
plot(RET_FT4 ~ SAT_AVG_ALL, data = private, main = "Private schools")
f <- function(x) {
    cubicCurve(x, cf = coef(modelRET3))
}
curve(f, add = TRUE, col = "red", lwd = 2)
plot(TUITIONFEE_OUT ~ SAT_AVG_ALL, data = private,
    main = "Private schools")
f <- function(x) {
    cubicCurve(x, cf = coef(modelTUT3))
}
curve(f, add = TRUE, col = "red", lwd = 2)
```



What do you think about the cubic fit?

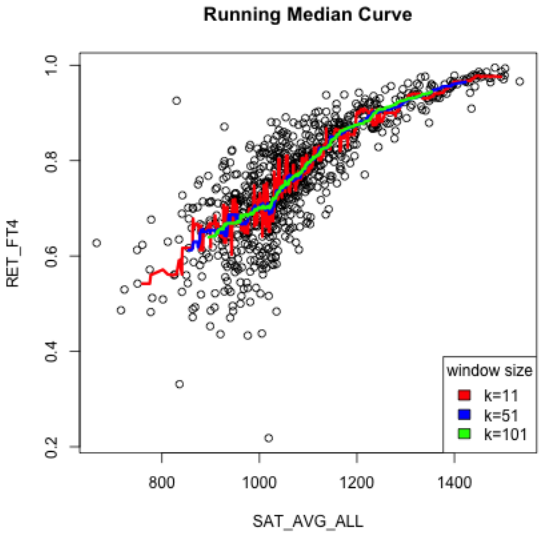
3.2 Local fitting

Defining a particular function to match the entire scope of the data might be difficult. Instead we might want something that is more flexible.

Brainstorm with a partner: what ideas can you imagine for how you might get a descriptive curve/line/etc to describe this data?

Running Mean or Median One simple idea is to take a running mean or median over the data. In otherwords, take a window of points, and as you slide this window across the x-axis, take the mean. There are a lot of varieties on this same idea. For example, you could make the window not fixed width, but a fixed number of points, etc. While it's conceptually easy to code from scratch, there are a lot of nitpicky details, so we'll use a built in implementation that does a fixed number of points.

```
library(zoo)
xorder <- private$SAT_AVG_ALL[order(private$SAT_AVG_ALL)]
yorder <- private$RET_FT4[order(private$SAT_AVG_ALL)]
ypred11 <- rollmedian(yorder, k = 11)
xpred11 <- rollmedian(xorder, k = 11)
ypred51 <- rollmedian(yorder, k = 51)
xpred51 <- rollmedian(xorder, k = 51)
ypred101 <- rollmedian(yorder, k = 101)
xpred101 <- rollmedian(xorder, k = 101)
plot(xorder, yorder, main = "Running Median Curve",
    xlab = "SAT_AVG_ALL", ylab = "RET_FT4")
lines(xpred11, ypred11, col = "red", lwd = 3)
lines(xpred51, ypred51, col = "blue", lwd = 3)
lines(xpred101, ypred101, col = "green", lwd = 3)
legend("bottomright", c("k=11", "k=51", "k=101"), fill = c("red",
    "blue", "green"), title = "window size")
```



What do you notice when I change the number of points in each window? Which seems more reasonable here?

Loess: Local Mean One disadvantage to a running median is that it can create a curve that is rather jerky as you add in one point/take away a point. Alternatively, if you have a wide window, then your curve at any point x will average the points that can be quite far away, and treat them equally as points that are nearby.

We've already seen a similar concept when we talked about kernel density estimation, instead of histograms. There we saw that we could *weight* our points, so that closer points count more than further away points. We can do the same idea for our running mean.

In otherwords, for each x we would calculate the mean of all of the data, only every point would be weighted. So our estimate of $\hat{f}(x)$ would be

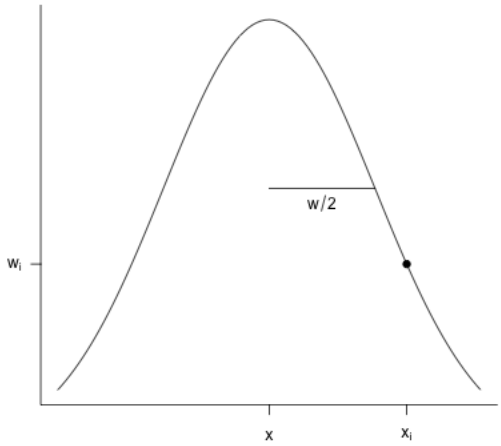
$$\sum_{i=1}^{n} w_i y_i$$

This is called a weighted mean, where $\sum_i w_i = 1$. A regular mean, in contrast, has equal weights for every obeservation ($w_i = 1/n$). So those with larger weights w_i will count more. How do we determine w_i? It will be similar to that of kernel density estimation. Specifically, if we are trying to estimate $f(x)$, we want to get the average of points near x, and those far away shouldn't matter as much.

When we did a running mean it was this same idea, only $w_i = 0$ if you aren't in

the window around x, and otherwise, the weight is equal among points in the window. This is like our rectangle kernel in density estimation.

Just as in density estimation, you tend to get smoother results if our weights aren't abruptly changing from 0 once a point moves in the window. So we will use the same idea, where we weight our point i based on how close x_i is to the x for which we are trying to estimate $f(x)$. And just like in density estimation, a gaussian kernel is the common choice for how to decide the weight:



So we could write our estimate at a point x as a weighted mean of all our observations, where the weight of each data point depends smoothly on the distance of x_i from the point x we are trying to estimate:

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

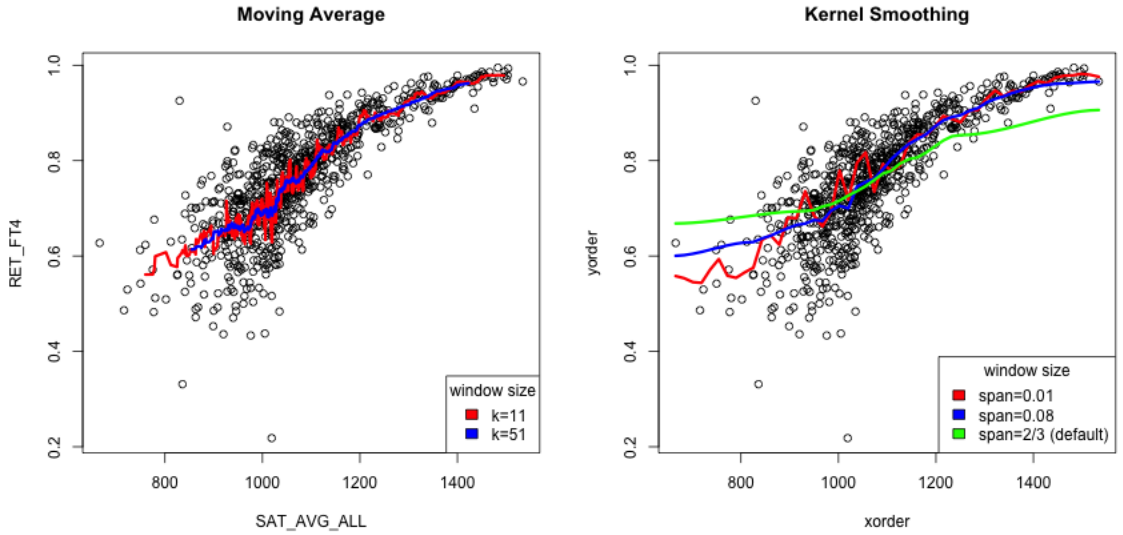
Indeed there are other weighting schemes that can be used as well.[6]

Here's how the default smoothing weights used in R to a rolling mean (i.e. based on fixed windows)

```
ypred11_mean <- rollmean(yorder, k = 11)
ypred51_mean <- rollmean(yorder, k = 51)
par(mfrow = c(1, 2))
plot(xorder, yorder, main = "Moving Average", xlab = "SAT_AVG_ALL",
    ylab = "RET_FT4")
lines(xpred11, ypred11_mean, col = "red", lwd = 3)
```

[6]The implementation in R actually uses a polynomial decay function.

```
lines(xpred51, ypred51_mean, col = "blue", lwd = 3)
legend("bottomright", c("k=11", "k=51"), fill = c("red",
    "blue"), title = "window size")
plot(xorder, yorder, main = "Kernel Smoothing")
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    span = 0.01, degree = 0), col = "red", lwd = 3)
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    span = 0.1, degree = 0), col = "blue", lwd = 3)
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    degree = 0), col = "green", lwd = 3)
legend("bottomright", c("span=0.01", "span=0.08", "span=2/3 (default)"),
    fill = c("red", "blue", "green"), title = "window size")
```



The `span` argument tells you what percentage of points are used in predicting x (like bandwidth in density estimation)[7]. So there's still an idea of a window size; it's just that within the window, you are giving more emphasis to points near your x value.

Notice that one advantage is that you can define an estimate for any x in the range of your data – the estimated curve doesn't have to jump as you add new points. Instead it transitions smoothly.

What other comparisons might you make here?

[7]There's a lot of details about span and what points are used, but we are not going to worry about them. What I've described here gets at the idea

Loess: Local Polynomial Regression Fitting In fact, very few people actually use this particular strategy, because taking a mean is not the greatest approach nor is it particularly robust, particularly for the tail ends where you have less data to work with. Instead, the most common approach is to use this basic idea of a window around x with points weighted by their distance from x, but to not calculate the mean of the data, but a line through those points, or a quadratic.

So with local mean, we weighted the estimation of the mean based on weights. For a linear fit instead, for each point x, we can calculate a linear regression with coefficients $\hat{\beta}_0(x)$ and $\hat{\beta}_1(x)$, where these coefficients are calculated by weighting the points (x_i, y_i) in calculating the best line. In otherwords, when we penalize for errors, errors in prediction y_i where x_i are near x count for more than when x_i is far from x.

To imagine what this means, consider if our weights were like a moving window, where $w_i(x) = 0$ if x_i is not in the window around x. Then we would only fit a line for those points in the window, and exclude the other points. With smoother weights, all the points could be used, but errors in points far away will not make a difference.

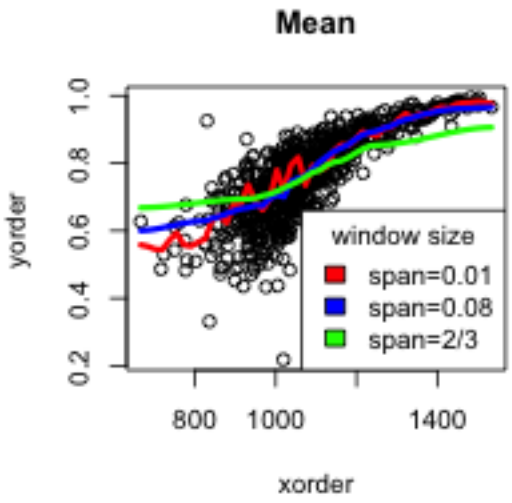
Then our estimate of the curve at x is given by

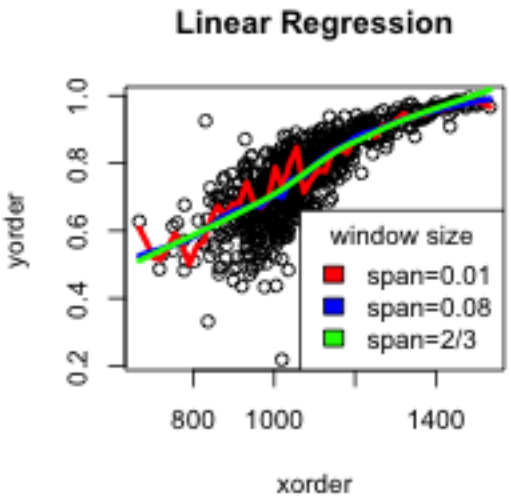
$$\hat{f}(x) = \hat{\beta}_0(x) + \hat{\beta}_1(x)x$$

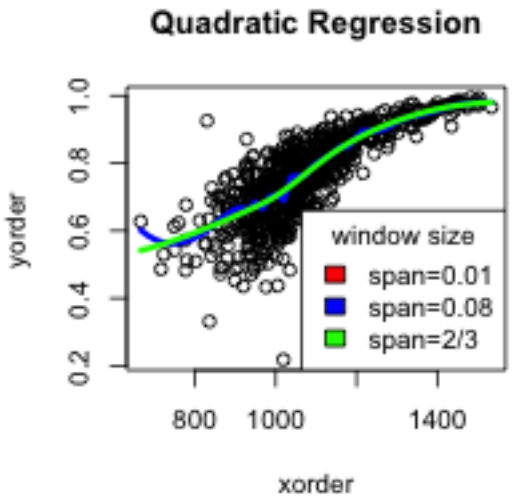
Because our weights for each point (x_i, y_i) will change as x changes, this will give different estimates $\hat{\beta}_0(x)$ and $\hat{\beta}_1(x)$ for every single x. But $\hat{f}(x)$ will change smoothly as we move x since our weights won't change very much.

```
par(mfrow = c(2, 2))
plot(xorder, yorder, main = "Mean")
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    span = 0.01, degree = 0), col = "red", lwd = 3)
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    span = 0.1, degree = 0), col = "blue", lwd = 3)
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    degree = 0), col = "green", lwd = 3)
legend("bottomright", c("span=0.01", "span=0.08", "span=2/3"),
    fill = c("red", "blue", "green"), title = "window size")
plot(xorder, yorder, main = "Linear Regression")
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    span = 0.01, degree = 1), col = "red", lwd = 3)
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    span = 0.1, degree = 1), col = "blue", lwd = 3)
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    degree = 1), col = "green", lwd = 3)
legend("bottomright", c("span=0.01", "span=0.08", "span=2/3"),
    fill = c("red", "blue", "green"), title = "window size")
```

```
plot(xorder, yorder, main = "Quadratic Regression")
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    span = 0.1, degree = 2), col = "blue", lwd = 3)
lines(loess.smooth(x = private$SAT_AVG_ALL, y = private$RET_FT4,
    degree = 2), col = "green", lwd = 3)
legend("bottomright", c("span=0.01", "span=0.08", "span=2/3"),
    fill = c("red", "blue", "green"), title = "window size")
```







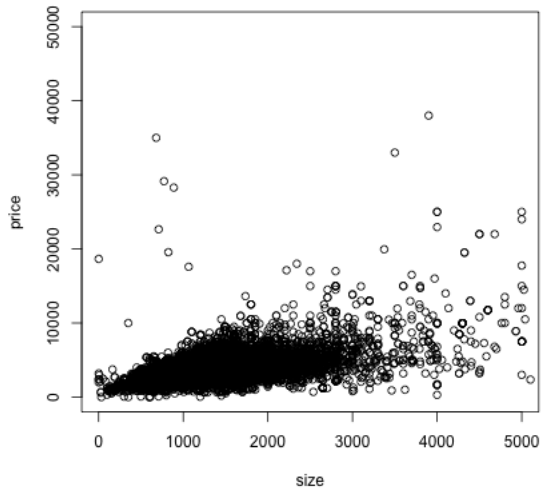
What conclusions would you draw about the difference between choosing the degree of the fit (mean/linear/quadratic)?

4 Other Examples

4.1 Big Data clouds

It can be particularly helpful to have a smooth scatter for visualization when you have a lot of data points. Consider the following data on craigs list rentals that you saw in lab. We would suspect that size would be highly predictive of price, and indeed if we plot price against size that's pretty clear.

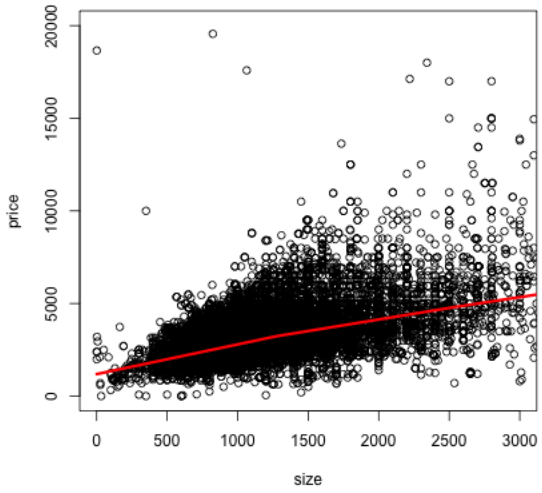
```
craigs <- read.csv(file.path(dataDir, "craigslist.csv"),
    header = TRUE)
plot(price ~ size, data = craigs, xlim = c(0, 5000))
```



But, because of the number of points, we can't really see much of what's going on. In fact our eye is drawn to outlying (and less representative) points, while the rest is just a black smear where the plots are on top of each other.

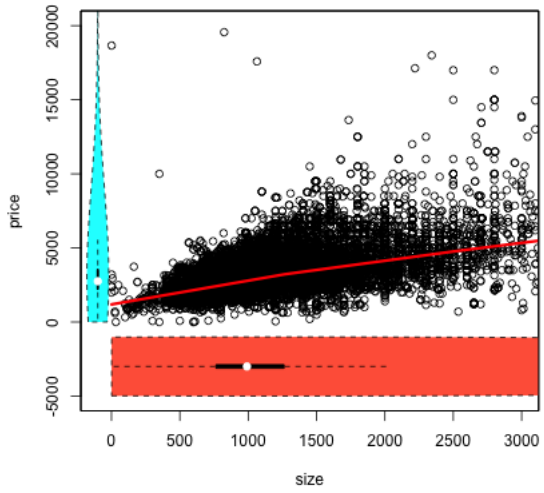
We can add a loess smooth curve to get an idea of where the bulk of the data lie. We'll zoom in a bit closer as well by changing the x and y limits of the axes.

```
loessCraigs <- loess.smooth(y = craigs$price, x = craigs$size)
plot(price ~ size, data = craigs, xlim = c(0, 3000),
    ylim = c(0, 20000))
lines(loessCraigs, col = "red", lwd = 3)
```



What does this tell you about the data?

```
library(vioplot)
par(mfrow = c(1, 1))
plot(price ~ size, data = craigs, xlim = c(-100, 3000),
    ylim = c(-5000, 20000))
lines(loessCraigs, col = "red", lwd = 3)
vioplot(craigs$size[!is.na(craigs$size)], col = "tomato",
    horizontal = TRUE, at = -3000, add = TRUE, lty = 2,
    wex = 5000)
vioplot(craigs$price[!is.na(craigs$size)], col = "cyan",
    horizontal = FALSE, at = -100, add = TRUE, lty = 2,
    wex = 200)
```



Add Violin plot

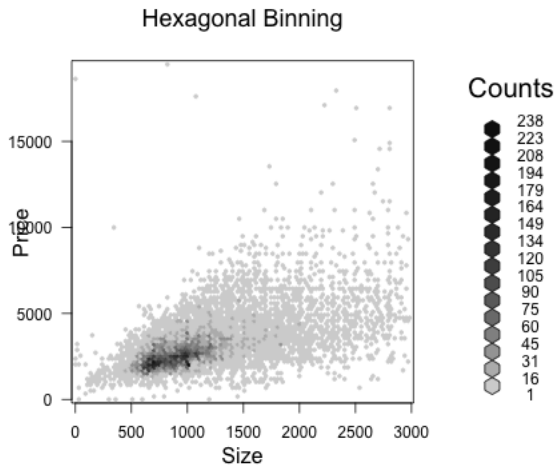
2D density smoothing plots If we really want to get a better idea of what's going on under that smear of black, we can use 2D density smoothing plots. This is the same idea as density smoothing plots for probability densities, only for 2D. Imagine that instead of a histogram along the line, a 2D histogram. This would involve griding the 2D plane into rectangles (instead of intervals) and counting the number of points within each rectangle. The high of the bars (now in the 3rd dimension) would give a visualization of how many points there are in different places in the plot.

Then just like with histograms, we can smooth this, so that we get a smooth curve over the 2 dimensions.

A 3D picture of this would be cool, but difficult to actually see information, axes, etc. So its common to instead smash this information into 2D, by representing the 3rd dimension (the density of the points) by a color scale instead.

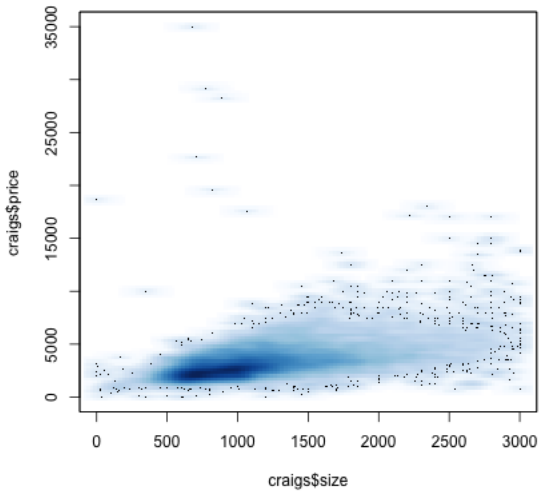
Here is an example of such a visualization of a 2D histogram

```
library(hexbin)
wh <- with(craigs, which(size < 3000 & size > 0 & price <
    20000))
bin <- hexbin(x = craigs$size[wh], y = craigs$price[wh],
    xbins = 100)
plot(bin, main = "Hexagonal Binning", xlab = "Size",
    ylab = "Price")
```

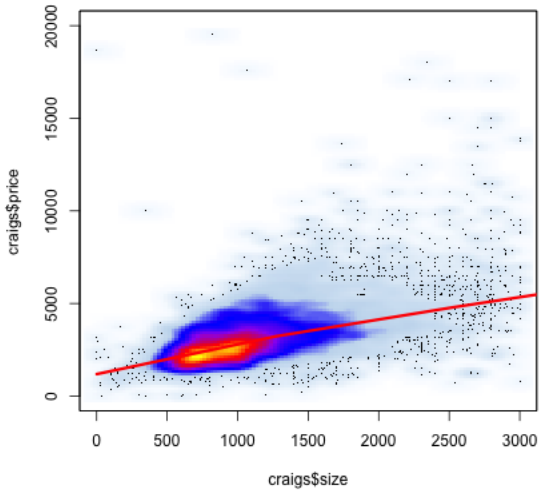


We can use a smoother version of this and get more gradual changes (and a less finicky function)

```
smoothScatter(y = craigs$price, x = craigs$size, xlim = c(0,
    3000), nrpoints = 500)
```




```
mycolramp <- colorRampPalette(c("white", head(blues9,
    3), "blue", "purple", "red", "yellow"))
smoothScatter(y = craigs$price, x = craigs$size, xlim = c(0,
    3000), nrpoints = 1200, ylim = c(0, 20000), colramp = mycolramp)
lines(loessCraigs, col = "red", lwd = 3)
```

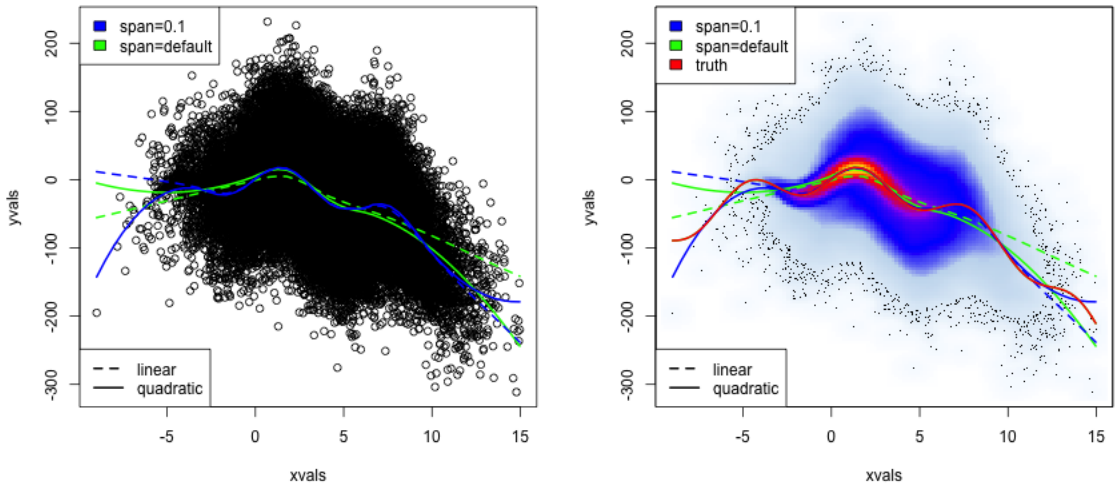


What do these colors tell you? How does this compare to the smooth line? What do you see about those points that grabbed our eye before (and which the loess line ignored)?

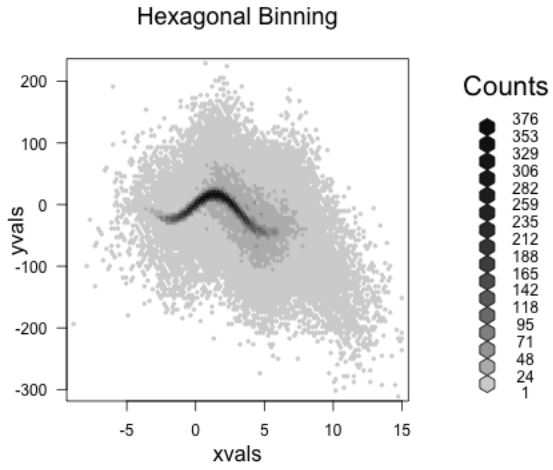
Simulated Example For this data, it turned out that the truth was pretty linear. But many times, the cloud of data can significantly impair our ability to see the data. We can simulate a more complicated function with many points.

```
trueF <- function(x) {
    exp(-x^2) - x^2 + 20 * sin(x)
}
n1 <- 20000
n2 <- 40000
xvals <- c(rnorm(n1, mean = 1.2, sd = 2), rnorm(n2,
    mean = 4, sd = 3))
xlim <- c(1, 2.4)
yvals <- trueF(xvals) + c(rnorm(n1, mean = 0, sd = 5),
    rnorm(n2, mean = 0, sd = 55))
par(mfrow = c(1, 2))
plot(xvals, yvals)
lines(loess.smooth(y = yvals, x = xvals, degree = 1),
    col = "green", lty = 2, lwd = 2)
lines(loess.smooth(y = yvals, x = xvals, degree = 2),
    col = "green", lty = 1, lwd = 2)
lines(loess.smooth(y = yvals, x = xvals, span = 0.1,
```

```
    degree = 2), col = "blue", lty = 1, lwd = 2)
lines(loess.smooth(y = yvals, x = xvals, span = 0.1,
    degree = 1), col = "blue", lty = 2, lwd = 2)
legend("topleft", c("span=0.1", "span=default"), fill = c("blue",
    "green"))
legend("bottomleft", c("linear", "quadratic"), lty = c(2,
    1), lwd = 2)
smoothScatter(y = yvals, x = xvals, nrpoints = 1000,
    colramp = mycolramp)
curve(trueF, add = TRUE, col = "green", lwd = 2)
lines(loess.smooth(y = yvals, x = xvals, degree = 1),
    col = "green", lty = 2, lwd = 2)
lines(loess.smooth(y = yvals, x = xvals, degree = 2),
    col = "green", lty = 1, lwd = 2)
lines(loess.smooth(y = yvals, x = xvals, span = 0.1,
    degree = 2), col = "blue", lty = 1, lwd = 2)
lines(loess.smooth(y = yvals, x = xvals, span = 0.1,
    degree = 1), col = "blue", lty = 2, lwd = 2)
legend("topleft", c("span=0.1", "span=default", "truth"),
    fill = c("blue", "green", "red"))
legend("bottomleft", c("linear", "quadratic"), lty = c(2,
    1), lwd = 2)
curve(trueF, add = TRUE, col = "red", lwd = 2)
```




```
bin <- hexbin(x = xvals, y = yvals, xbins = 100)
plot(bin, main = "Hexagonal Binning")
```



4.2 Time trends

Let's look at another common example of fitting a trend – time data. In the following dataset, we have the average temperatures (in celecius) by city per month since 1743.

```
temp <- read.csv(file.path(dataDir, "GlobalLandTemperaturesByMajorCity.csv"),
    header = TRUE)
head(temp)
```

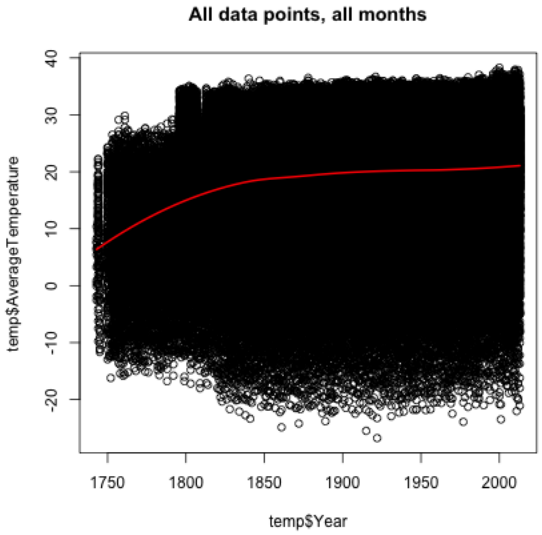
```
##           dt AverageTemperature AverageTemperatureUncertainty    City
## 1 1849-01-01              26.704                         1.435 Abidjan
## 2 1849-02-01              27.434                         1.362 Abidjan
## 3 1849-03-01              28.101                         1.612 Abidjan
## 4 1849-04-01              26.140                         1.387 Abidjan
## 5 1849-05-01              25.427                         1.200 Abidjan
## 6 1849-06-01              24.844                         1.402 Abidjan
##         Country Latitude Longitude
## 1 Cte D'Ivoire    5.63N     3.23W
## 2 Cte D'Ivoire    5.63N     3.23W
## 3 Cte D'Ivoire    5.63N     3.23W
## 4 Cte D'Ivoire    5.63N     3.23W
## 5 Cte D'Ivoire    5.63N     3.23W
## 6 Cte D'Ivoire    5.63N     3.23W
```

```
temp$Year <- as.numeric(sapply(strsplit(as.character(temp$dt),
```

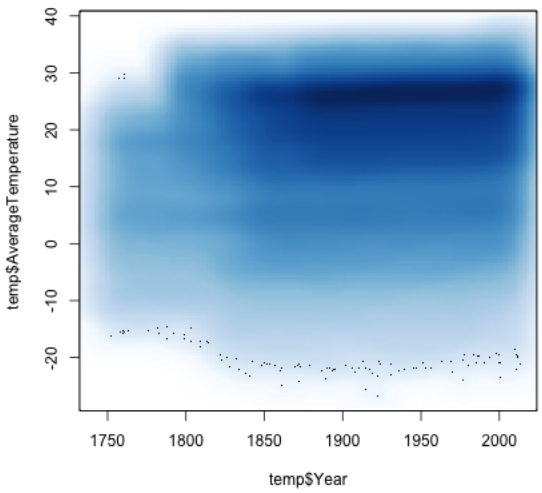
```
    "-"), .subset2, 1))
temp$Month <- as.numeric(sapply(strsplit(as.character(temp$dt),
    "-"), .subset2, 2))
```

Given the scientific consensus that the planet is warming, it is interesting to look at this data, limited though it is, to see how different cities are affected.

```
plot(temp$Year, temp$AverageTemperature, main = "All data points, all months")
lines(loess.smooth(x = temp$Year, y = temp$AverageTemperature,
    degree = 2), col = "red", lwd = 2)
```

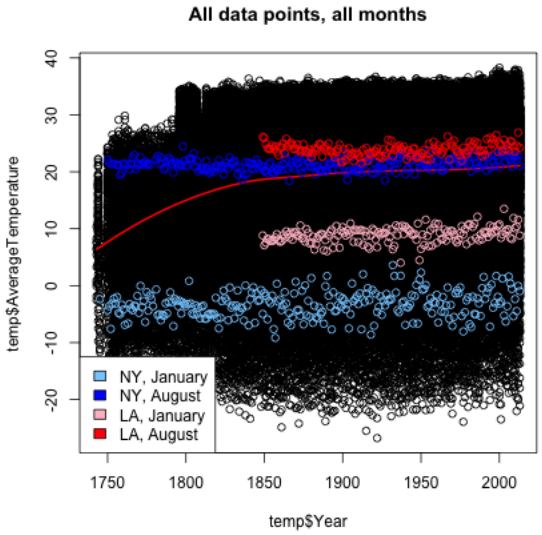



```
smoothScatter(x = temp$Year, y = temp$AverageTemperature)
```



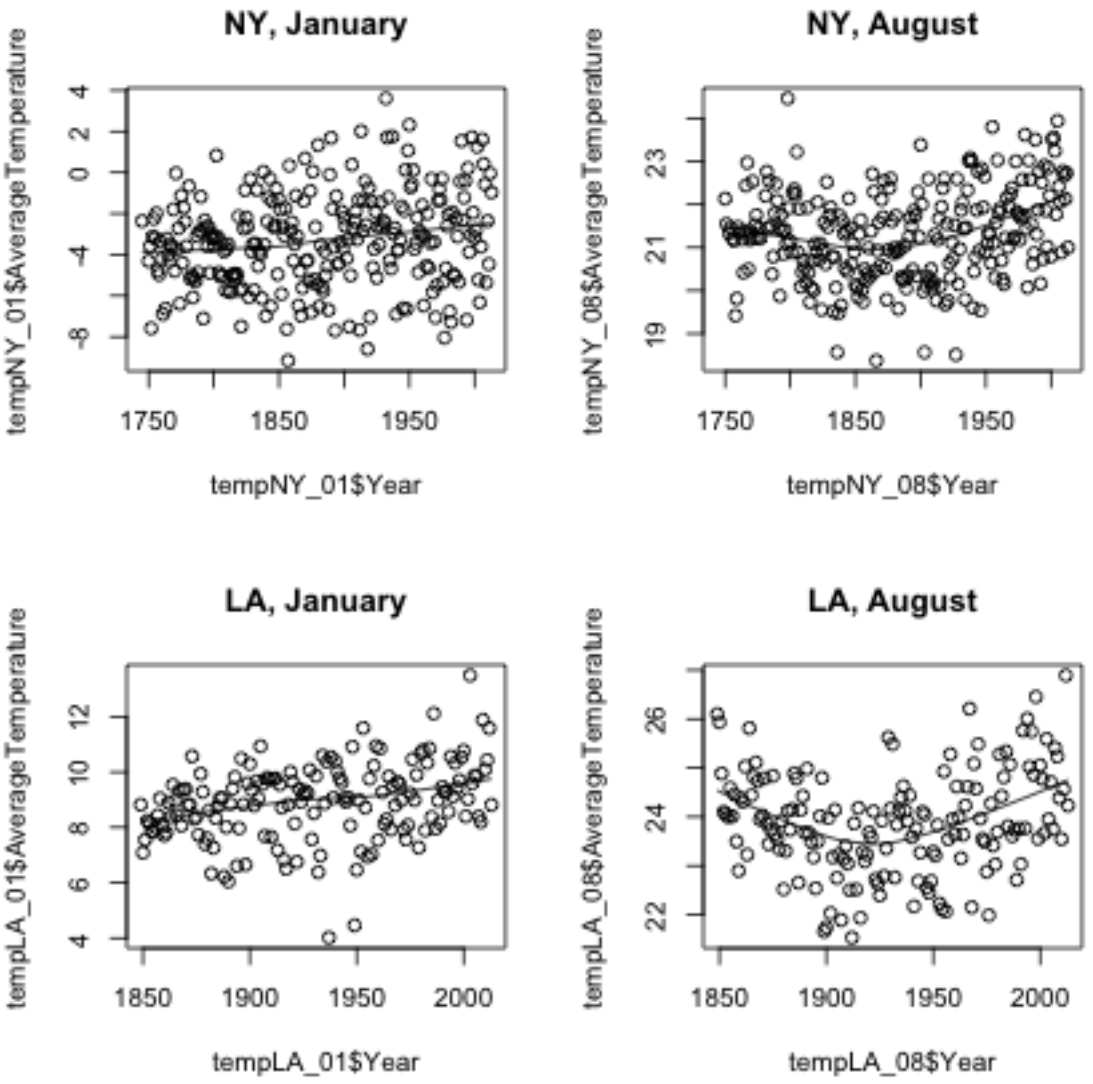
This is a very uninformative plot, despite our best efforts. Why?

```
plot(temp$Year, temp$AverageTemperature, main = "All data points, all months")
lines(loess.smooth(x = temp$Year, y = temp$AverageTemperature,
    degree = 2), col = "red", lwd = 2)
tempNY_01 <- subset(temp, City == "New York" & Month ==
    1)
tempNY_08 <- subset(temp, City == "New York" & Month ==
    8)
tempLA_01 <- subset(temp, City == "Los Angeles" & Month ==
    1)
tempLA_08 <- subset(temp, City == "Los Angeles" & Month ==
    8)
with(tempNY_01, points(x = Year, y = AverageTemperature,
    col = "skyblue1"))
with(tempNY_08, points(x = Year, y = AverageTemperature,
    col = "blue"))
with(tempLA_01, points(x = Year, y = AverageTemperature,
    col = "pink"))
with(tempLA_08, points(x = Year, y = AverageTemperature,
    col = "red"))
legend("bottomleft", c("NY, January", "NY, August",
    "LA, January", "LA, August"), fill = c("skyblue1",
    "blue", "pink", "red"), bg = "white")
```



We can consider for different cities or different months how average temperatures have changed. We use the function `scatter.smooth` that both plots the points and places a loess curve on top.

```
par(mfrow = c(2, 2))
tempNY_01 <- subset(temp, City == "New York" & Month ==
    1)
tempNY_08 <- subset(temp, City == "New York" & Month ==
    8)
tempLA_01 <- subset(temp, City == "Los Angeles" & Month ==
    1)
tempLA_08 <- subset(temp, City == "Los Angeles" & Month ==
    8)
scatter.smooth(tempNY_01$Year, tempNY_01$AverageTemperature,
    main = "NY, January")
scatter.smooth(tempNY_08$Year, tempNY_08$AverageTemperature,
    main = "NY, August")
scatter.smooth(tempLA_01$Year, tempLA_01$AverageTemperature,
    main = "LA, January")
scatter.smooth(tempLA_08$Year, tempLA_08$AverageTemperature,
    main = "LA, August")
```

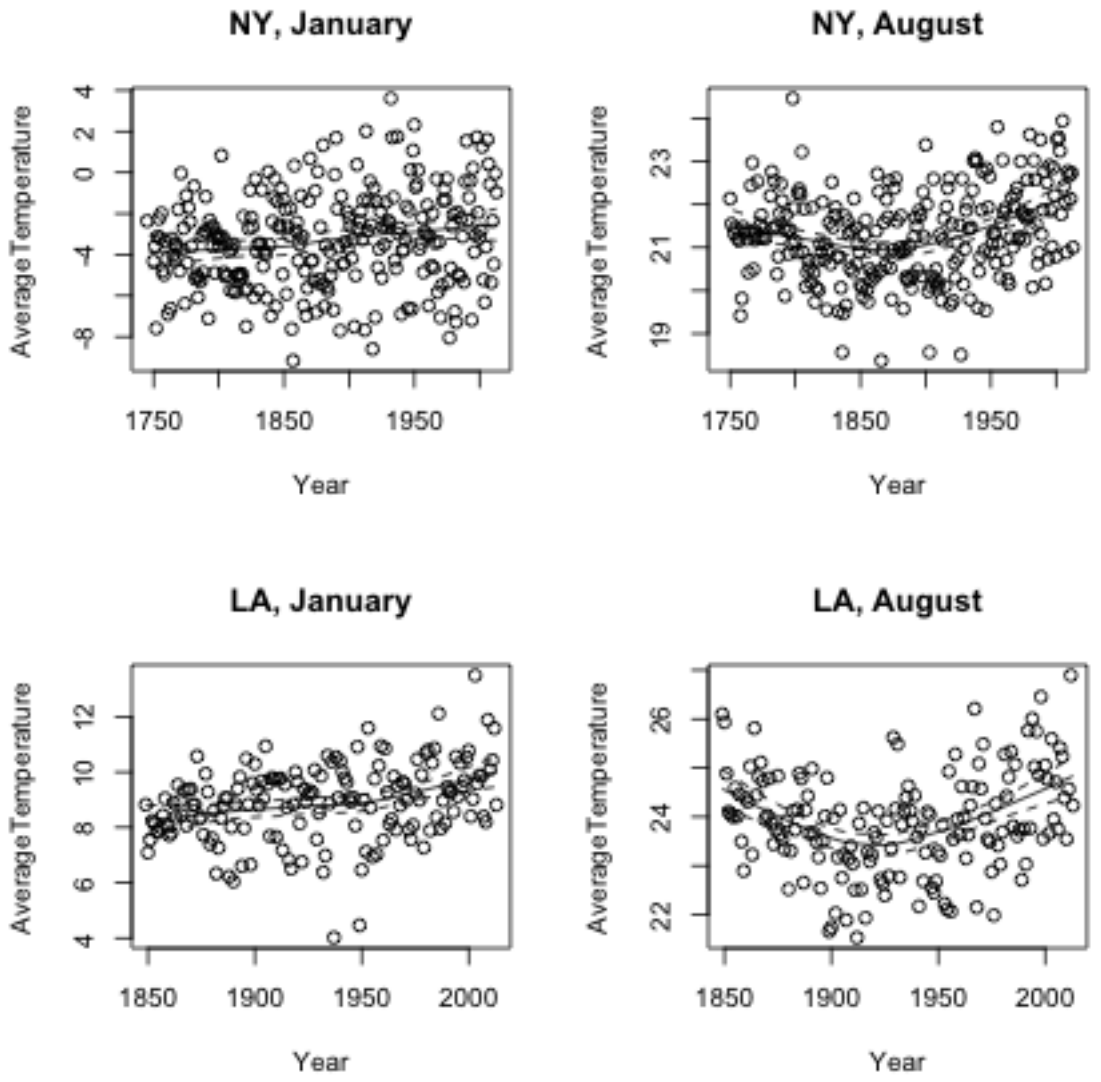


Loess Prediction Intervals We can even calculate (parametric) confidence intervals around these curves (based on a type of t-statistic for kernel smoothers), with a bit more lines of code. They are called prediction intervals, because they are confidence intervals for the prediction at each point.

In fact, since it's a bit annoying, I'm going to write a little function to do it.

```
loessWithCI <- function(dataset, ...) {
    xseq <- seq(min(dataset$Year), max(dataset$Year),
        length = 100)
    loessPred <- predict(loess(AverageTemperature ~
        Year, data = dataset, span = 2/3, degree = 1),
```

```
        newdata = data.frame(Year = xseq), se = TRUE)
    plot(AverageTemperature ~ Year, data = dataset,
        ...)
    lines(xseq, loessPred$fit)
    lines(xseq, loessPred$fit - qt(0.975, loessPred$df) *
        loessPred$se, lty = 2)
    lines(xseq, loessPred$fit + qt(0.975, loessPred$df) *
        loessPred$se, lty = 2)
}
par(mfrow = c(2, 2))
loessWithCI(tempNY_01, main = "NY, January")
loessWithCI(tempNY_08, main = "NY, August")
loessWithCI(tempLA_01, main = "LA, January")
loessWithCI(tempLA_08, main = "LA, August")
```



Look at the code. In what way do they look like t-statistic intervals?

Comparing Many Cities Smooth scatter plots can be useful to compare the time trends of many groups. It's difficult to plot each city, but we can plot their loess curve. I will write a function to automate this. For ease of comparison, I will pick just a few cities in the northern hemisphere.

```
temp08 <- subset(temp, Month == 8 & City %in% c("Peking",
    "Los Angeles", "Toronto", "Riyadh", "Kabul", "Mexico",
    "Rome", "New York"))
```

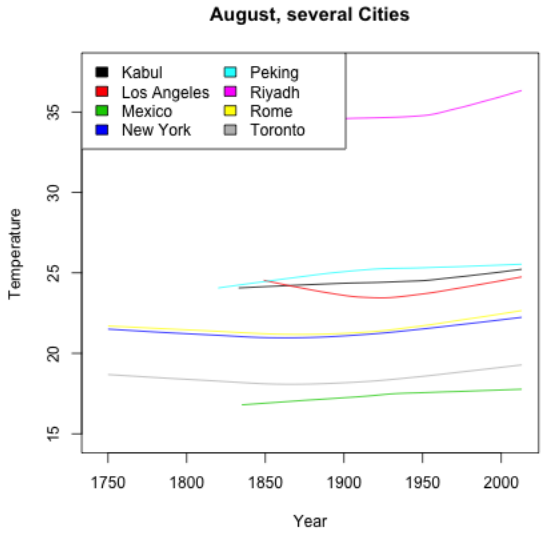
```
nlevels(temp08$City)
```

```
## [1] 100
```

```
temp08 <- droplevels(temp08)
cityColors <- palette()[1:nlevels(temp08$City)]
names(cityColors) <- levels(temp08$City)
plot(temp08$Year, temp08$AverageTemperature, type = "n",
    main = "August, several Cities", xlab = "Year",
    ylab = "Temperature")
by(temp08, temp08$City, function(x) {
    lines(loess.smooth(x = x$Year, y = x$AverageTemperature),
        col = cityColors[unique(x$City)])
})
```

```
## temp08$City: Kabul
## NULL
## ------------------------------------------------------------
## temp08$City: Los Angeles
## NULL
## ------------------------------------------------------------
## temp08$City: Mexico
## NULL
## ------------------------------------------------------------
## temp08$City: New York
## NULL
## ------------------------------------------------------------
## temp08$City: Peking
## NULL
## ------------------------------------------------------------
## temp08$City: Riyadh
## NULL
## ------------------------------------------------------------
## temp08$City: Rome
## NULL
## ------------------------------------------------------------
## temp08$City: Toronto
## NULL
```

```
legend("topleft", levels(temp08$City), ncol = 2, fill = cityColors)
```

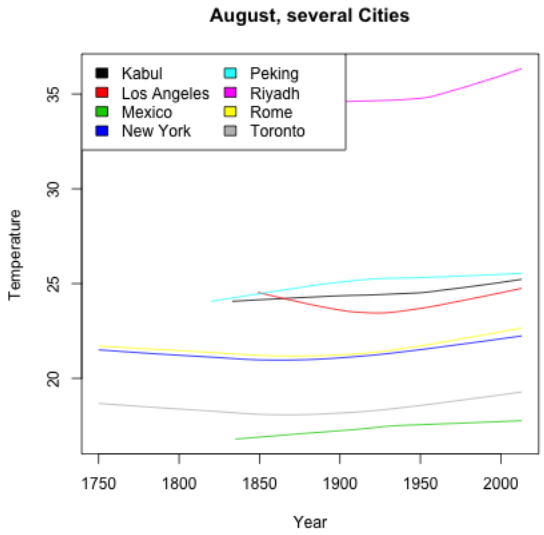



```
loessByCity08 <- by(temp08, temp08$City, function(x) {
    loess.smooth(x = x$Year, y = x$AverageTemperature)
})
xlim <- range(sapply(loessByCity08, function(x) {
    range(x$x)
}))
ylim <- range(sapply(loessByCity08, function(x) {
    range(x$y)
}))
plot(temp08$Year, temp08$AverageTemperature, type = "n",
    xlim = xlim, ylim = ylim, main = "August, several Cities",
    xlab = "Year", ylab = "Temperature")
by(temp08, temp08$City, function(x) {
    lines(loess.smooth(x = x$Year, y = x$AverageTemperature),
        col = cityColors[unique(x$City)])
})
```

```
## temp08$City: Kabul
## NULL
## ------------------------------------------------------------
## temp08$City: Los Angeles
## NULL
## ------------------------------------------------------------
## temp08$City: Mexico
## NULL
## ------------------------------------------------------------
## temp08$City: New York
## NULL
```

```
## ------------------------------------------------------------
## temp08$City: Peking
## NULL
## ------------------------------------------------------------
## temp08$City: Riyadh
## NULL
## ------------------------------------------------------------
## temp08$City: Rome
## NULL
## ------------------------------------------------------------
## temp08$City: Toronto
## NULL
```

```
legend("topleft", names(loessByCity08), ncol = 2, fill = cityColors)
```

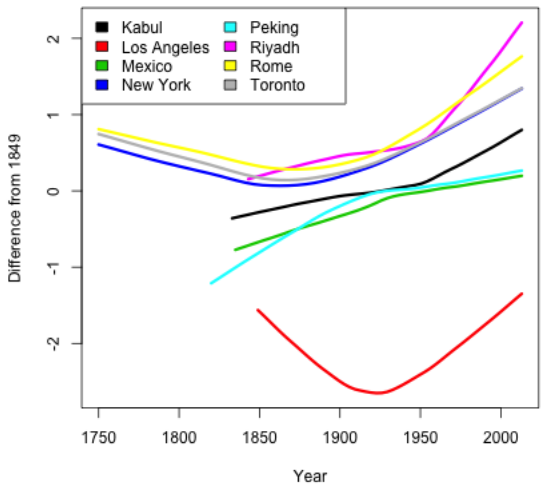


What makes these curves so difficult to compare?

Instead, I'm going to subtract off their temperature in 1849, so that we plot everything relative to that temperature.

```
yearCenter <- 1849
loessByCity08_center1 <- by(temp08, temp08$City, function(x) {
    tempCenter <- x$AverageTemperature[x$Year == yearCenter]
    loess.smooth(x = x$Year, y = x$AverageTemperature -
```

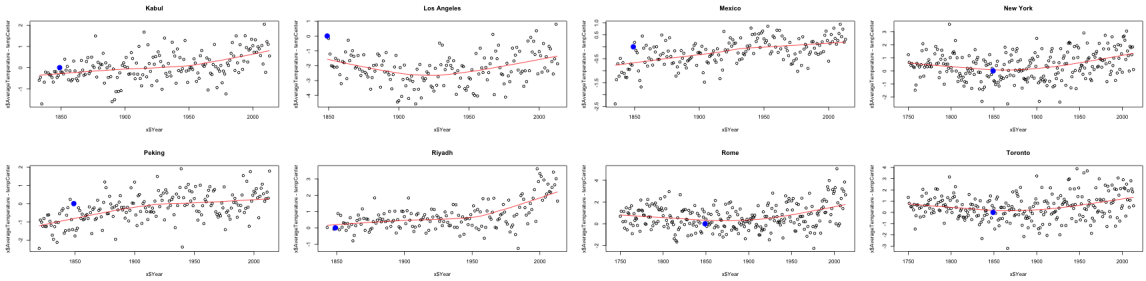
```
        tempCenter)
})
xlim <- range(sapply(loessByCity08_center1, function(x) {
    range(x$x)
}))
ylim <- range(sapply(loessByCity08_center1, function(x) {
    range(x$y)
}))
plot(0, 0, type = "n", xlim = xlim, ylim = ylim, xlab = "Year",
    ylab = paste("Difference from", yearCenter))
trash <- mapply(loessByCity08_center1, cityColors,
    FUN = function(x, col) {
        lines(x, col = col, lwd = 3)
    })
legend("topleft", names(loessByCity08_center1), ncol = 2,
    fill = cityColors)
```



Why don't the curves all go through the same point at 1849? Consider the following plots of the 8 cities, with the 1849 point highlighted in blue.

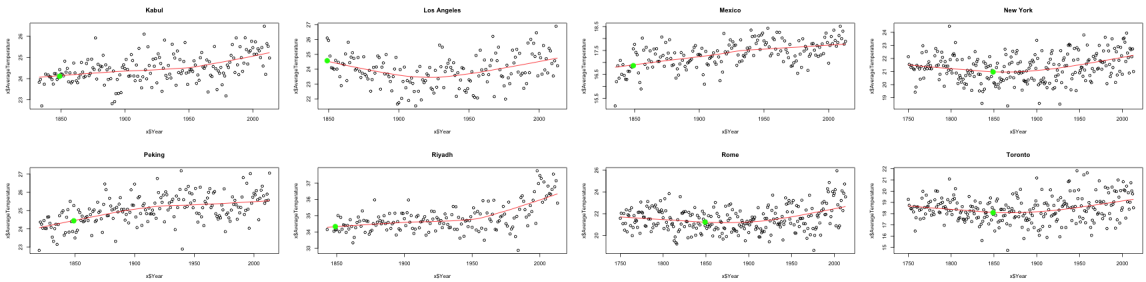
```
par(mfrow = c(2, 4))
test <- by(temp08, temp08$City, function(x) {
    whYear <- which(x$Year == yearCenter)
    tempCenter <- x$AverageTemperature[whYear]
    plot(x$Year, x$AverageTemperature - tempCenter,
```

```
        main = as.character(unique(x$City)))
    lines(loess.smooth(x = x$Year, y = x$AverageTemperature -
        tempCenter), col = "red")
    points(x$Year[whYear], (x$AverageTemperature -
        tempCenter)[whYear], col = "blue", pch = 19,
        cex = 2)
})
```



It would be better to center based on the loess prediction at that point.

```
loessCenterValue <- by(temp08, temp08$City, function(x) {
    predict(loess(AverageTemperature ~ Year, data = x,
        span = 2/3, degree = 1), newdata = data.frame(Year = yearCenter))
})
par(mfrow = c(2, 4))
test <- by(temp08, temp08$City, function(x) {
    whYear <- which(x$Year == yearCenter)
    plot(x$Year, x$AverageTemperature, main = as.character(unique(x$City)))
    lines(loess.smooth(x = x$Year, y = x$AverageTemperature),
        col = "red")
    predValue <- predict(loess(AverageTemperature ~
        Year, data = x, span = 2/3, degree = 1), newdata = data.frame(Year = yearCent
    points(x$Year[whYear], predValue, col = "green",
        pch = 19, cex = 2)
})
```



Now we can subtract off that value instead.

```
ylim <- range(mapply(loessByCity08, loessCenterValue,
    FUN = function(x, val) {
        range(x$y - val)
    }))
plot(0, 0, type = "n", xlim = xlim, ylim = ylim, xlab = "Year",
    ylab = paste("Difference from", yearCenter))
trash <- mapply(loessByCity08, cityColors, loessCenterValue,
    FUN = function(x, col, center) {
        x$y <- x$y - center
        lines(x, col = col, lwd = 3)
    })
legend("topleft", names(loessByCity08_center1), ncol = 2,
    fill = cityColors)
```