Week 07 - Regression

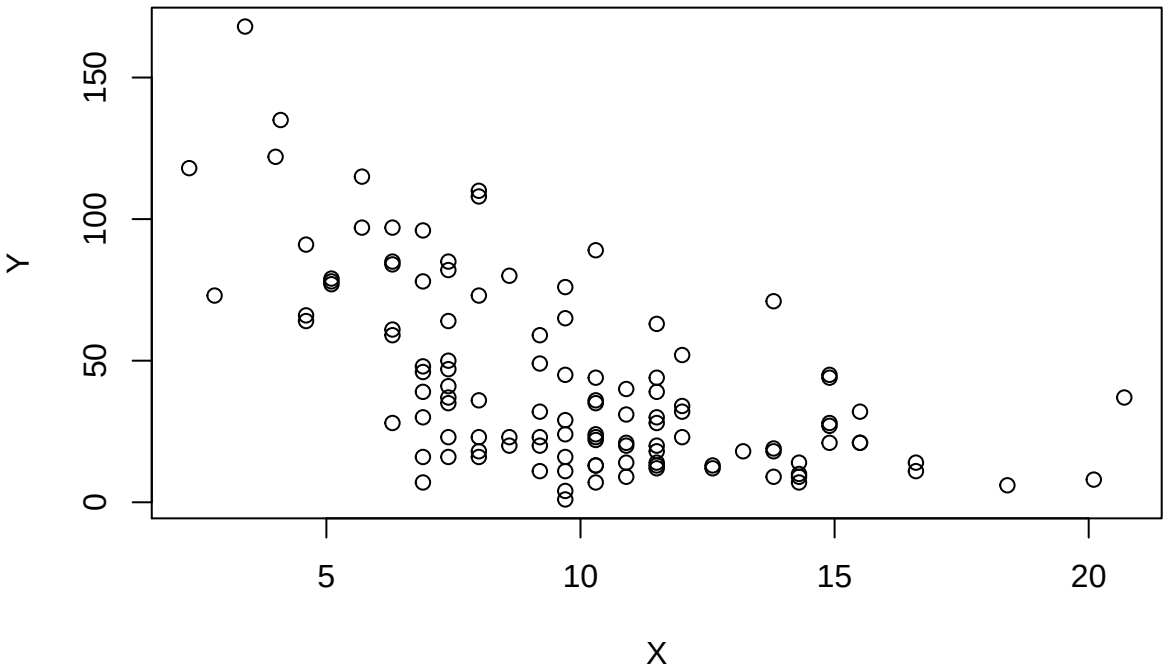
Linear Regression

Last week we looked at correlation, which gives us a normalized representation of how well to datasets covary, but what if we wanted to model that relationship, or even use it for prediction.

Ordinary linear regression

OLR is a simple way to create a linear model of that relates one or more vectors (X) to a target data series (Y). Let's use the airquality dataset to explore this.

```
X=airquality$Wind
Y=airquality$Ozone
toRemove=which(is.na(X) | is.na(Y))
X=X[-toRemove]
Y=Y[-toRemove]
plot(X,Y)
```



In regression, we will refer to X as the design matrix. This is the matrix of predictors. It must have the same number of rows as Y, but can have any number of columns. The number of columns will correspond to the number of coefficients in the model. We represent these coefficients in a column vector that has as many rows as there are columns in X.

Lets make our design matrix. It will be Wind, plus a column of ones so we can get a Y intercept in our model.

```
ones=matrix(1,nrow=length(X))
length(X)
```

```
## [1] 116
```

```
Xd=cbind(ones,X) #this is our design matrix
```

Linear algebra doesn't love NAs, let's make sure we've removed them all.

```
any(is.na(Xd))
```

```
## [1] FALSE
```

Remember that in OLS, we're trying to minimize the sum of squares of the residuals, or the error, which we call E.

$Y = Xb + E$

Where Y is a column of predictands, X is our design matrix, b is our column of coefficients, and E is the error, the difference between Xb and Y. We want to find b that makes the sum of squares in E as small as possible.

We refered to the derivation of this on the board, and will now use the result of this, called the "Normal equation" to find b, that minimize the sum of squares if E.

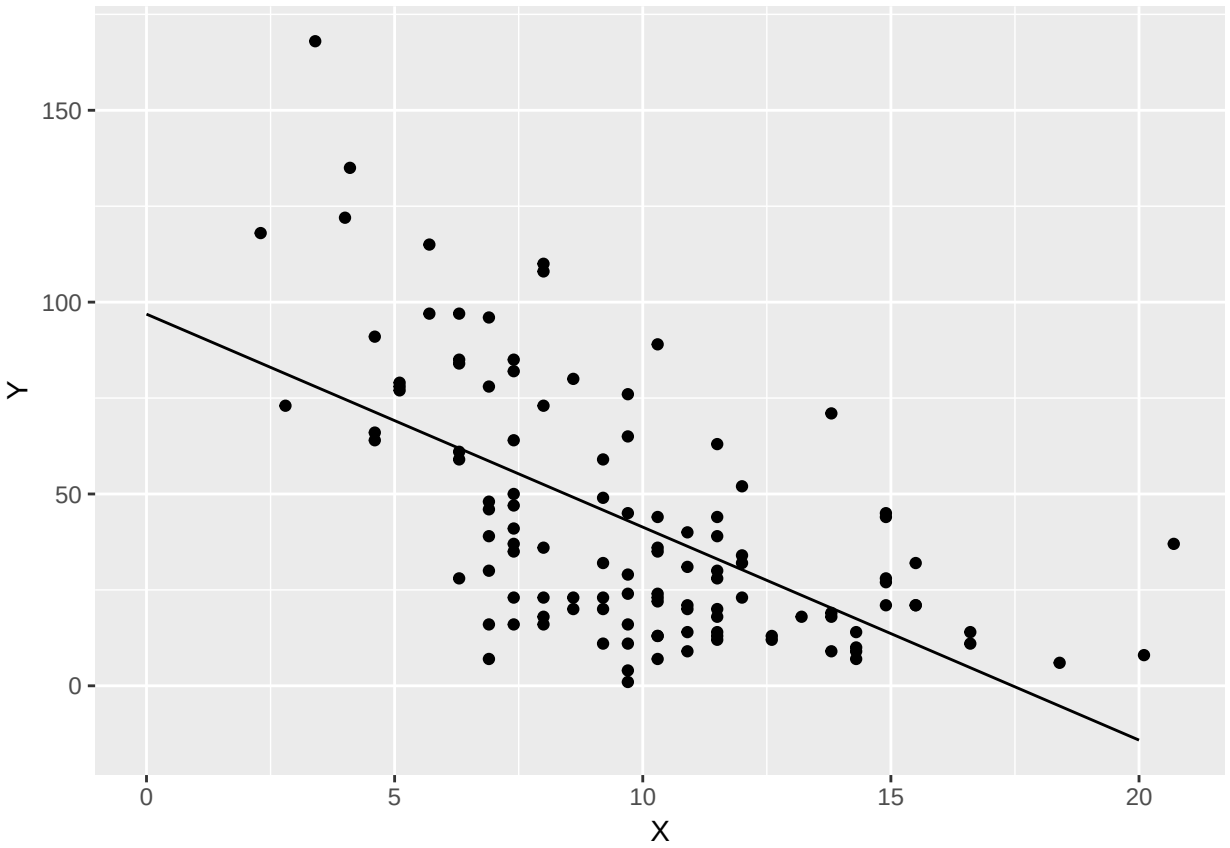
```
#Use normal equation (X'X)-1 * (X'Y)
XX=t(Xd)%*%Xd
XY=t(Xd)%*%Y
B=solve(XX)%*%XY
print(B)
```

```
##        [,1]
##   96.872895
## X -5.550923
```

OK. That was easy, now we have our model. Let's see how well it works.

Let's use the model to calculate predicted Y over the interval 0 to 20.

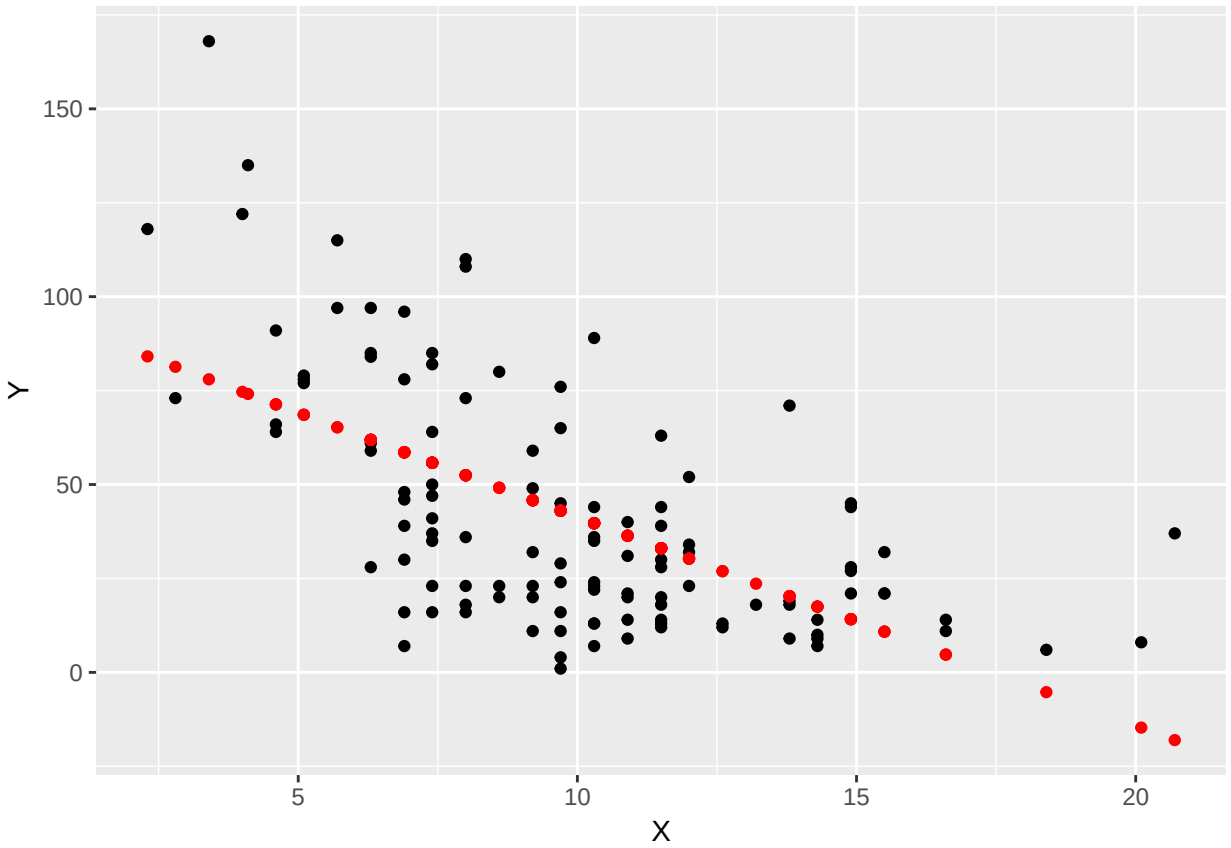
```
xseq=seq(0,20)
yhat=xseq*B[2]+B[1] #just calculating it as a simple y=mx+b type equation.
#now lets plot the original data, and add a line for our model.
library(ggplot2)
regPlot = ggplot()+geom_point(aes(X,Y))+geom_line(aes(xseq,yhat))
print(regPlot)
```



Rather than calculating Yhat the slow way by spelling out the linear equation, we can do it simply using the design matrix, to model the values present in X. Or creating a new design matrix to model any range in X.

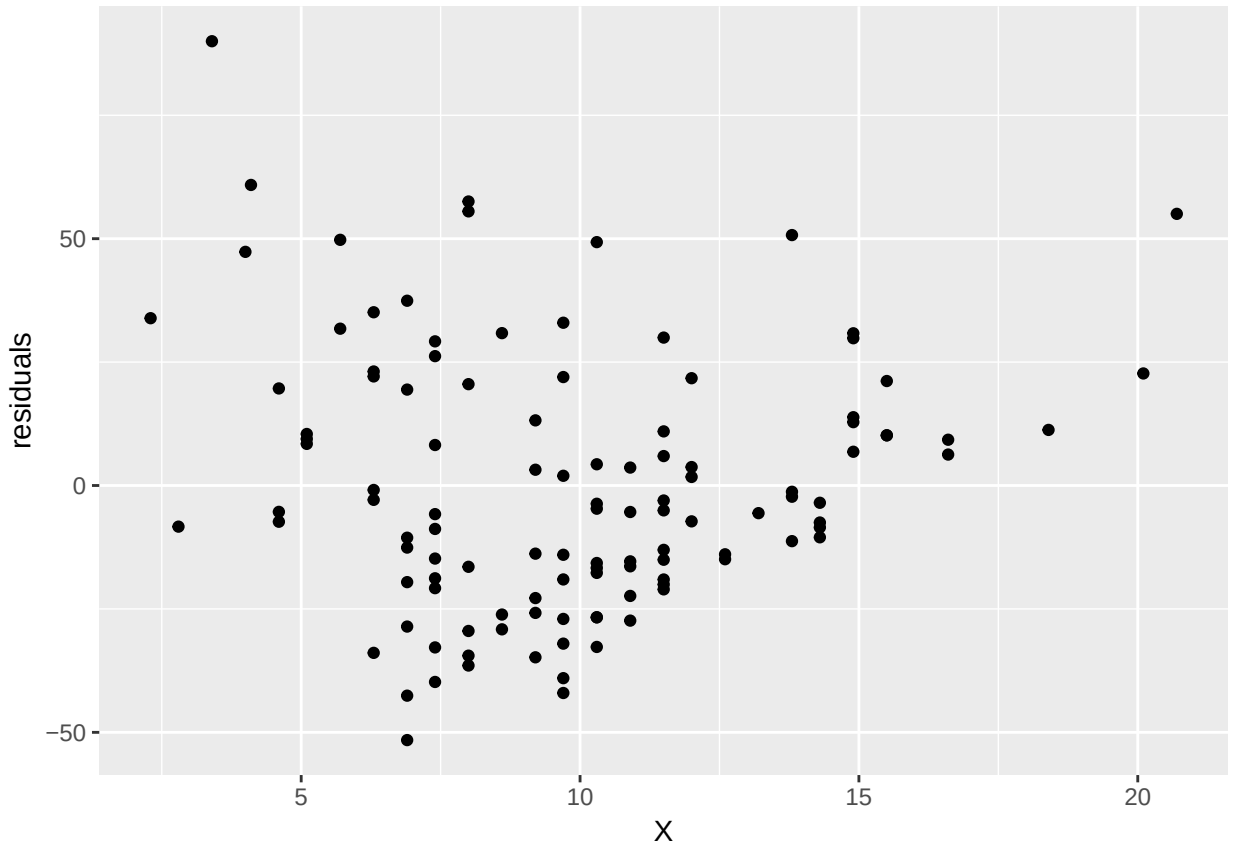
Remember, our equation was $Y = Xb + E$ so $\hat{Y} = Xb$

```
yhat2 = Xd%*%B
ggplot()+geom_point(aes(X,Y))+geom_point(aes(X,yhat2),colour = "red")
```



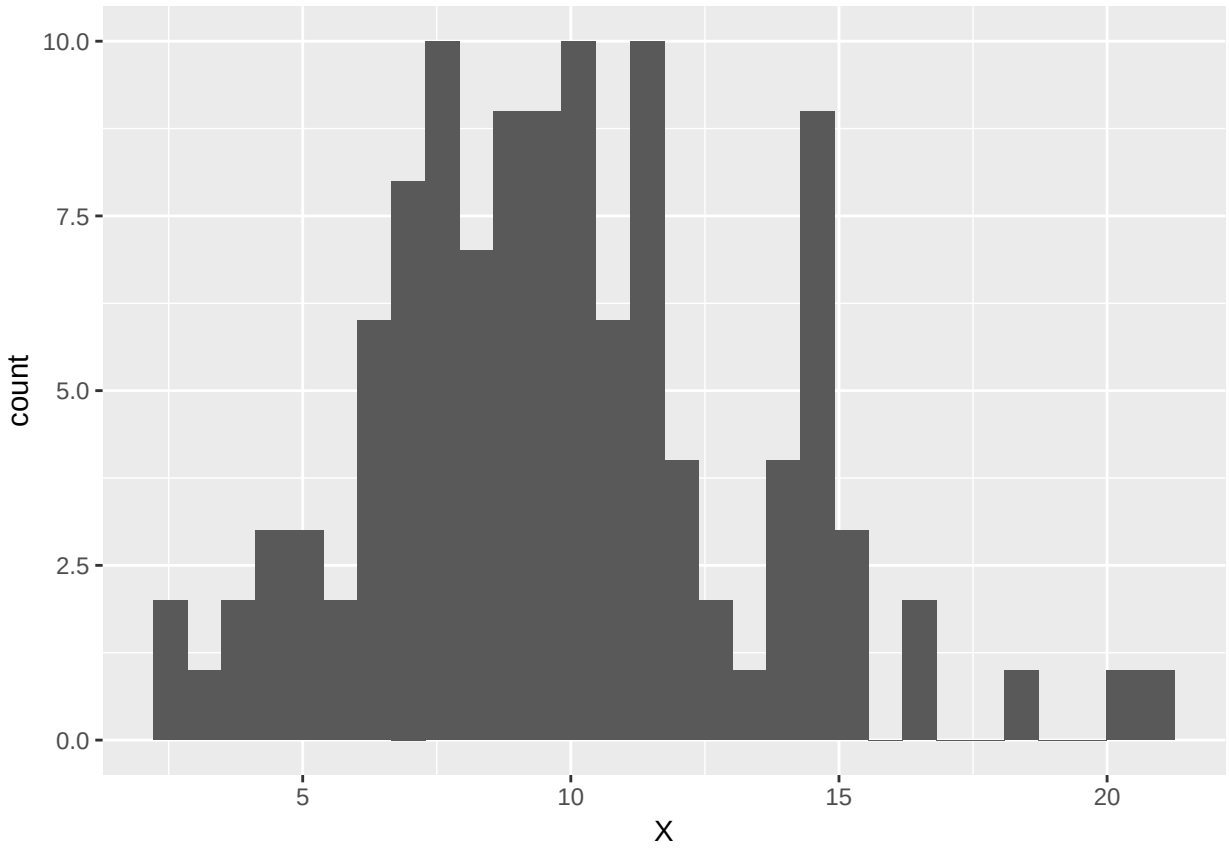
The difference between Y and Yhat is our E. It's also called the residuals. Let's take a look at our residuals.

```
residuals = Y-yhat2
ggplot()+geom_point(aes(X,residuals)) #as a line plot
```




```
ggplot()+geom_histogram(aes(X)) #as a histogram
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Uncertainty in our regression model

Our model is not a perfect representation of reality, and we'd like to estimate the uncertainty on our parameters in B. We're going to base this on the residuals.

First we need to calculate the root mean squared error (RMSE) of our residuals:

```
SSE = t(residuals)%*%residuals
MSE = SSE / (length(Y)-length(B)) #the degrees of freedom here is the number of observations minus the
#Now moving forward we want the root mean square error (RMSE) or s .
s = sqrt(MSE)
print(s)
```

```
##          [,1]
## [1,] 26.46729
```

OK, now that we have an estimate of σ for our residuals, we can use that to find the covariance matrix of B, like this:

```
#Now, let's use s to calculate uncertainty on B
#Find the covariance of B. Using this equation
covB = solve(t(Xd)%*%Xd)*as.vector(s^2)
print(covB)
```

```
##                      X
##   52.398615 -4.7008056
## X -4.700806  0.4766551
```

This is the covariance matrix of B, and the standard error of B is the square root of the diagonal:

```
stdB = diag(sqrt(covB))
```

```
## Warning in sqrt(covB): NaNs produced
```

```
print(stdB)
```

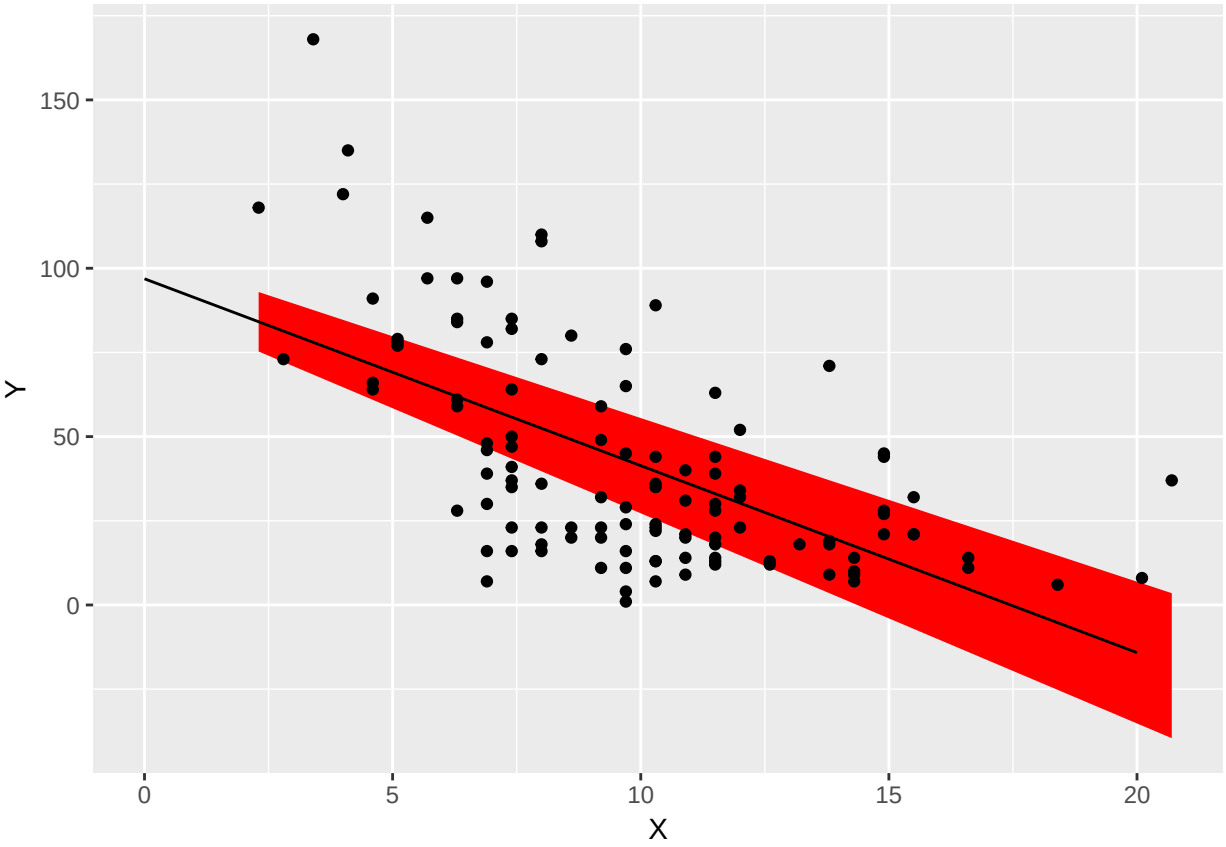
```
##                   X
## 7.2386887 0.6904021
```

OK, now let's calculate some line that show the uncertainty in our model. We'll show one standard error. This means that our regression parameters could be higher or lower than we calculated. So we'll calculate some new $\hat{Y}$ values for hi and low values.

```
#OK now calculate lines that correspond to the uncertainty
yhatHi = Xd%*%(B+stdB)
yhatLo = Xd%*%(B-stdB)
```

Now let's add those to our plot.

```
ggplot()+geom_ribbon(aes(x=X,ymin = yhatLo,ymax = yhatHi),fill = "red")+geom_point(aes(X,Y))+geom_line(a
```



Lastly, what if we want to estimate the uncertainty on certain prediction?

Just create a desing matrix for that prediction!

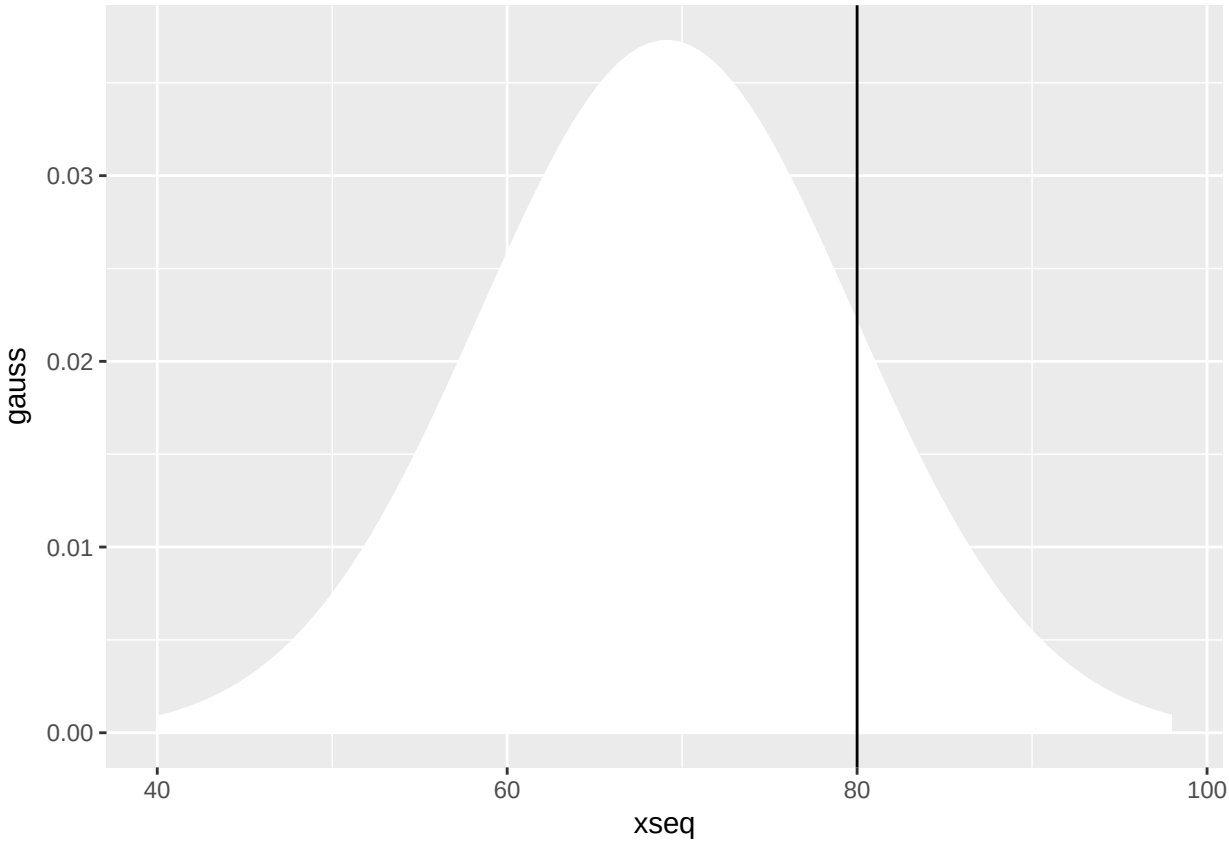
```
X5 = c(1, 5)#Create a single value design matrix to make a prediction
pred = X5%*%B#multiply that matrix by B to get you modeled prediction
pred.unc = X5%*%stdB#multiply that matrix by std error of B to get uncertainty on prediction
```

If we assume that prediction is normal, with a mean of the prediction and a standard deviation of the probability, what's probability in our prediction that if the wind is blowing 5 mph, the ozone concentration

will be above 80?

Let's make a plot to visualize what we're testing:

```
xseq = seq(40,98,length.out = 100)
gauss = dnorm(xseq,mean = pred,sd=pred.unc)
ggplot()+geom_area(aes(xseq,gauss),fill = "white")+geom_vline(xintercept = 80)
```



And then we can use pnorm to calculate the area right of 80...

```
1-pnorm(80,mean = pred,sd = pred.unc)
```

```
## [1] 0.1543704
```

Side note - using geom text and hacking legends.

add geom text

ggplot()+geom_area(aes(xseq,gauss,colour = "Distribution"),fill = "white")+geom_vline(aes(xintercept = 80,colour = "Critical Value"))+geom_text(aes(x = 60,y=0.02,label="this is text"))

Multiple Regression

OK, so what if we decide we want to use more to make our prediction? Maybe Solar radiation could help predict as well. And why not throw in temperature too?

We want to set up our design matrix such that

$Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3$

so what would our design matrix look like?

```
airqualityGood = na.omit(airquality)

X1 = airqualityGood$Wind
X2 = airqualityGood$Solar.R
X3 = airqualityGood$Temp
Y = airqualityGood$Ozone
ones = matrix(1,nrow = length(Y))
#so our design matrix is
Xd2 = cbind(ones,X1,X2,X3)
```

OK - now easy peasy, use your normal equations.

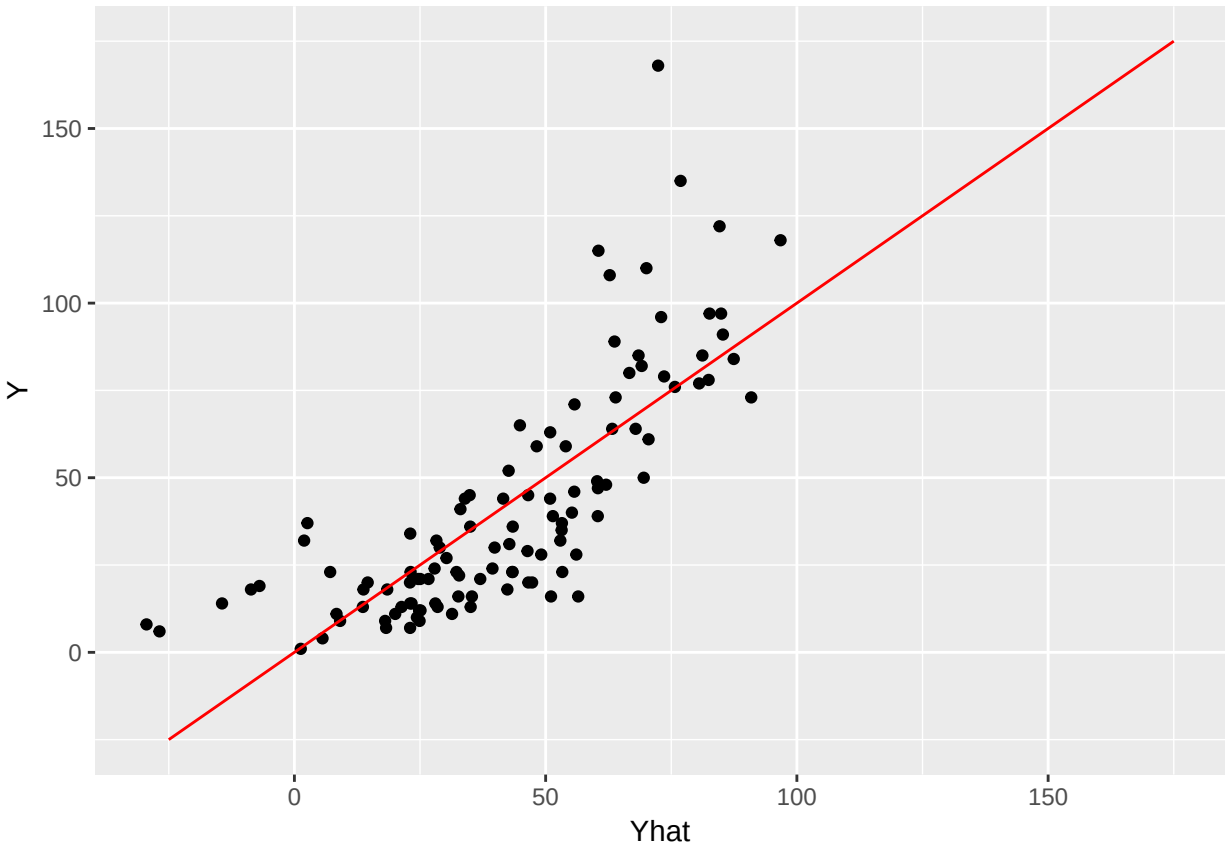
```
#Use normal equation (X'X)-1 * (X'Y)
XX=t(Xd2)%*%Xd2
XY=t(Xd2)%*%Y
B=solve(XX)%*%XY
print(B)
```

```
##            [,1]
##    -64.34207893
## X1  -3.33359131
## X2   0.05982059
## X3   1.65209291
```

Great - how can we plot this?

Just calculate Yhat, and plot Y vs Yhat

```
Yhat = Xd2%*%B
ggplot()+geom_point(aes(Yhat,Y))+geom_line(aes(c(-25,175),c(-25,175)),colour = "red")
```



How about uncertainty? It's also the same.

```
#UNCERTAINTY IS DONE THE SAME WAY
residuals = Y-Yhat
SSE = t(residuals)%*%residuals
MSE = SSE / (length(Y)-length(B))
#Now moving forward we want the root mean square error (RMSE) or s .
s = sqrt(MSE)
covB = solve(t(Xd2)%*%Xd2)*as.vector(s^2)
stdB = diag(sqrt(covB))
```

```
## Warning in sqrt(covB): NaNs produced
```

```
stdB
```

```
##                      X1          X2          X3
## 23.05472435  0.65440710  0.02318647  0.25352979
```

Which of our variables is most important? Depends how you define it...

We could look at the ratio of the coefficients to their standard deviations...

```
B/apply(Xd2,2,sd)
```

```
##             [,1]
##             -Inf
## X1 -0.9370039347
## X2  0.0006562708
## X3  0.1733576355
```

```
#or we could look at the ratio of the standard errors to the coefficients - this tells us something dif
stdB/B
```

```
##           [,1]
##    -0.3583149
## X1 -0.1963069
## X2  0.3876001
## X3  0.1534598
```

```
#Or we could model each individually, and see which has the smallest standard errors...
```

There's still a curve in those data that we're not capturing...

Perhaps we think that we could model Ozone better with a 2nd-order polynomial of windspeed than a line.

$Y = b_0 + b_1 x + b_2 x^2$

Easy now, we just set up a new design matrix:

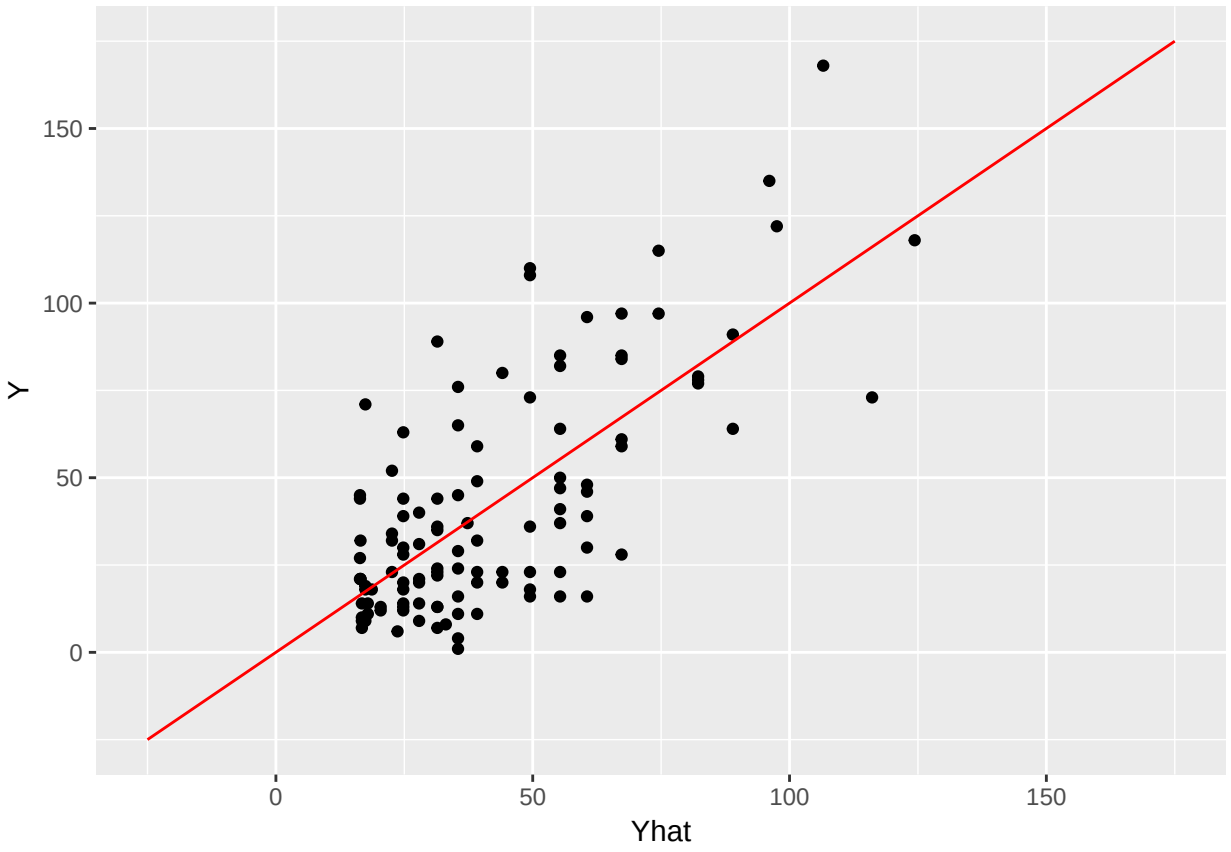
```
Xd3 = cbind(ones,X1,X1^2)

#Use normal equation (X'X)-1 * (X'Y)
XX=t(Xd3)%*%Xd3
XY=t(Xd3)%*%Y
B=solve(XX)%*%XY
print(B)
```

```
##            [,1]
##    166.7662893
## X1 -19.9586495
##       0.6620691
```

And calculate our predictions and plot..

```
Yhat = Xd3%*%B
ggplot()+geom_point(aes(Yhat,Y))+geom_line(aes(c(-25,175),c(-25,175)),colour = "red")
```




```
#UNCERTAINTY IS DONE THE SAME WAY
residuals = Y-Yhat
SSE = t(residuals)%*%residuals
MSE = SSE / (length(Y)-length(B))
#Now moving forward we want the root mean square error (RMSE) or s .
s = sqrt(MSE)
covB = solve(t(Xd3)%*%Xd3)*as.vector(s^2)
stdB = diag(sqrt(covB))
```

```
## Warning in sqrt(covB): NaNs produced
```

```
stdB
```

```
##                    X1
## 14.3437613  2.7412814  0.1240971
```