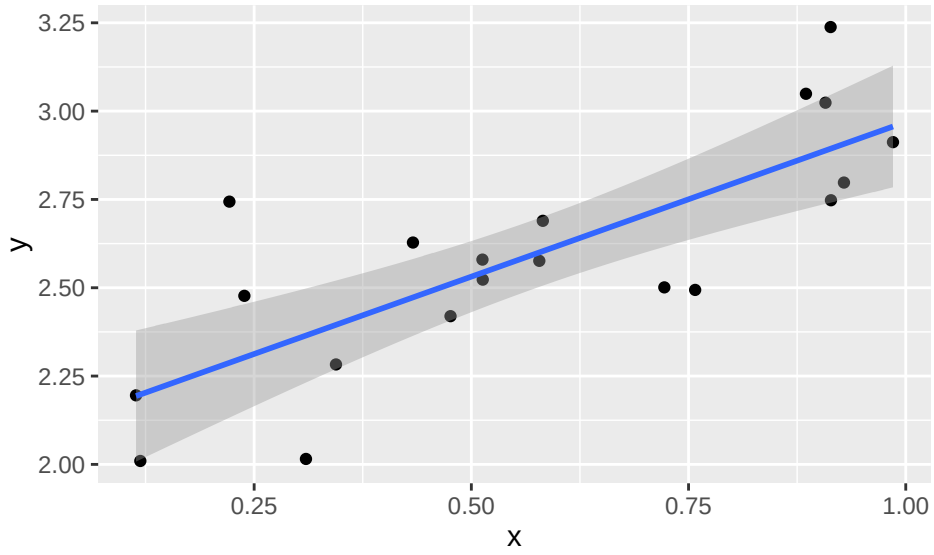
Chapter 1

Regression

```
library(ggplot2)
library(dplyr)
library(ggfortify)  # for diagnostic plots in ggplot2 via autoplot()
```

We continue to want to examine the relationship between a predictor variable and a response but now we consider the case that the predictor is continuous and the response is also continuous. In general we are going to be interested in finding the line that best fits the observed data and determining if we should include the predictor variable in the model.

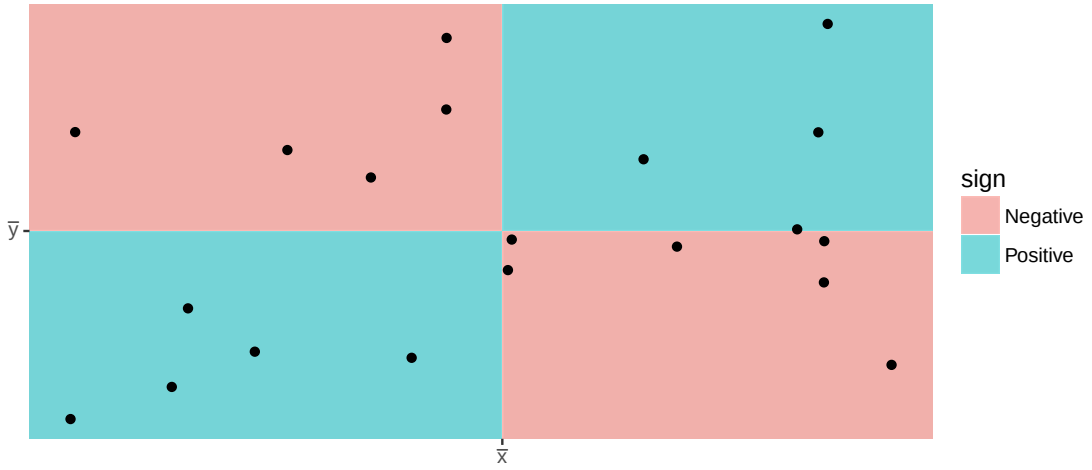


1.1 Pearson's Correlation Coefficient

We first consider Pearson's correlation coefficient, which is a statistics that measures the strength of the linear relationship between the predictor and response. Consider the following Pearson's correlation statistic

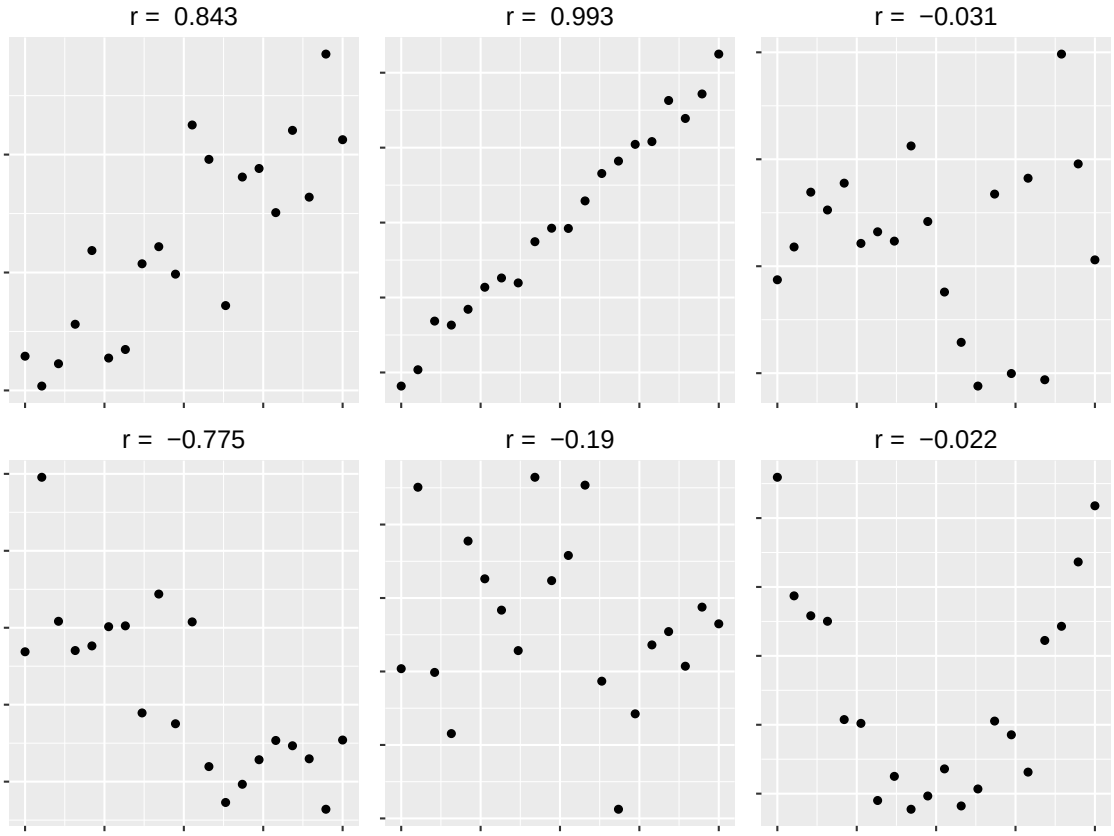
$$r = \frac{\sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s_x}\right)\left(\frac{y_i - \bar{y}}{s_y}\right)}{n-1}$$

where x_i and y_i are the x and y coordinate of the ith observation. Notice that each parenthesis value is the standardized value of each observation. If the x-value is big (greater than $\bar{x}$) and the y-value is large (greater than $\bar{y}$), then after multiplication, the result is positive. Likewise if the x-value is small and the y-value is small, both standardized values are negative and therefore after multiplication the result is positive. If a large x-value is paired with a small y-value, then the first value is positive, but the second is negative and so the multiplication result is negative.



The following are true about Pearson's correlation coefficient:

1. r is unit-less because we have standardized the x and y values.
2. $-1 \leq r \leq 1$ because of the scaling by $n - 1$
3. A negative r denotes a negative relationship between x and y, while a positive value of r represents a positive relationship.
4. r measures the strength of the *linear* relationship between the predictor and response.



1.2 Model Theory

To scatterplot data that looks linear we often want to fit the model

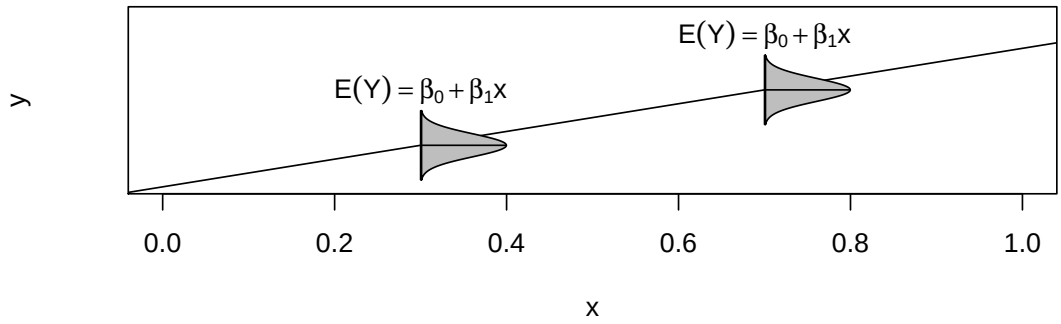
$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad \text{where } \epsilon_i \overset{iid}{\sim} N\left(0, \sigma^2\right)$$

where β_0 is the y-intercept term[1] and β_1 is the slope term[2]. The assumptions of this model are:

1. *The relationship between the predictor and response is actually linear*
2. *The error terms come from a normal distribution*
3. *The variance of the errors is the same for every value of x (homoscedasticity)*
4. *The error terms are independent*

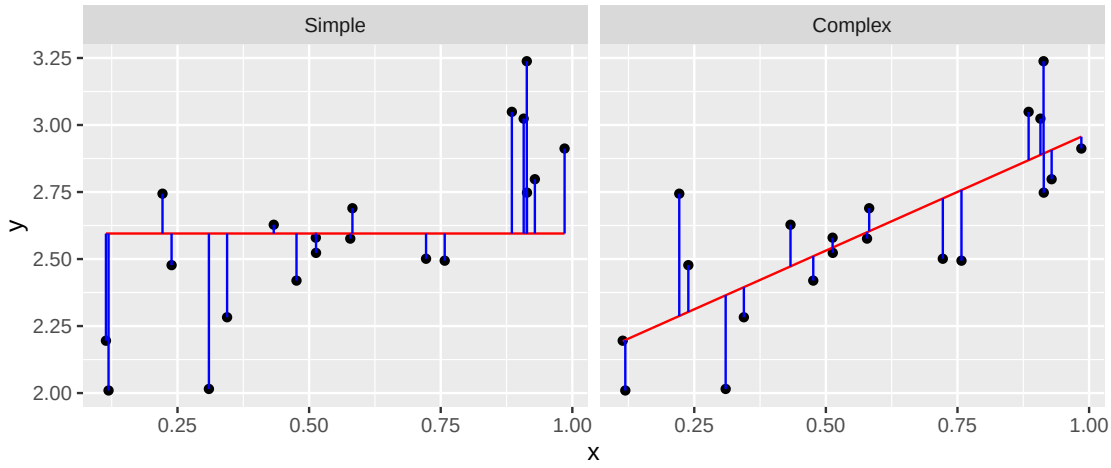
Under this model, the expected value of an observation with covariate $X = x$ is $E\left(Y \mid X = x\right) = \beta_0 + \beta_1 x$ and a new observation has a standard deviation of σ about the line.

[1] The y-intercept is the height of the line when $x = 0$.
[2] The slope is the change in y for every one-unit change in x



Given this model, how do we find estimates of β_0 and β_1? In the past we have always relied on using some sort of sample mean, but it is not obvious what we can use here. Instead of a mean, we will use the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the sum of squared error (SSE) where

$$\begin{aligned} \hat{y}_i &= \hat{\beta}_0 + \hat{\beta}_1 x_i \\ e_i &= y_i - \hat{y}_i \\ SSE &= \sum_{i=1}^{n} e_i^2 \end{aligned}$$



Fortunately there are simple closed form solutions for $\hat{\beta}_0$ and $\hat{\beta}_1$

$$\begin{aligned} \hat{\beta}_1 &= r\left(\frac{s_y}{s_x}\right) \\ \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x} \end{aligned}$$

and using these estimates several properties can be shown

1. $\hat{\beta}_0$ and $\hat{\beta}_1$ are the intercept and slope values that minimize SSE.
2. The regression line goes through the center of mass of the data $(\bar{x}, \bar{y})$.
3. The sum of the residuals is 0. That is: $\sum e_i = 0$

4. $\hat{\beta}_0$ and $\hat{\beta}_1$ are unbiased estimators of β_0 and β_1

We are also interested in an estimate of σ^2 and we will use our usual estimation scheme of

$$\begin{aligned}\hat{\sigma}^2 &= \frac{1}{n-2}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2 \\ &= \frac{\sum_{i=1}^{n} e_i^2}{n-2} \\ &= \frac{SSE}{n-2} \\ &= MSE\end{aligned}$$

where the -2 comes from having to estimate β_0 and β_1 before we can estimate σ^2. As in the ANOVA case, we can interpret σ as the typical distance an observation is from its predicted value.

As always we are also interested in knowing the estimated standard deviation (which we will now call *Standard Error)* of the model parameters β_0 and β_1 and it can be shown that

$$StdErr\left(\hat{\beta}_0\right) = \hat{\sigma}\sqrt{\frac{1}{n}+\frac{\bar{x}^2}{S_{xx}}}$$

and

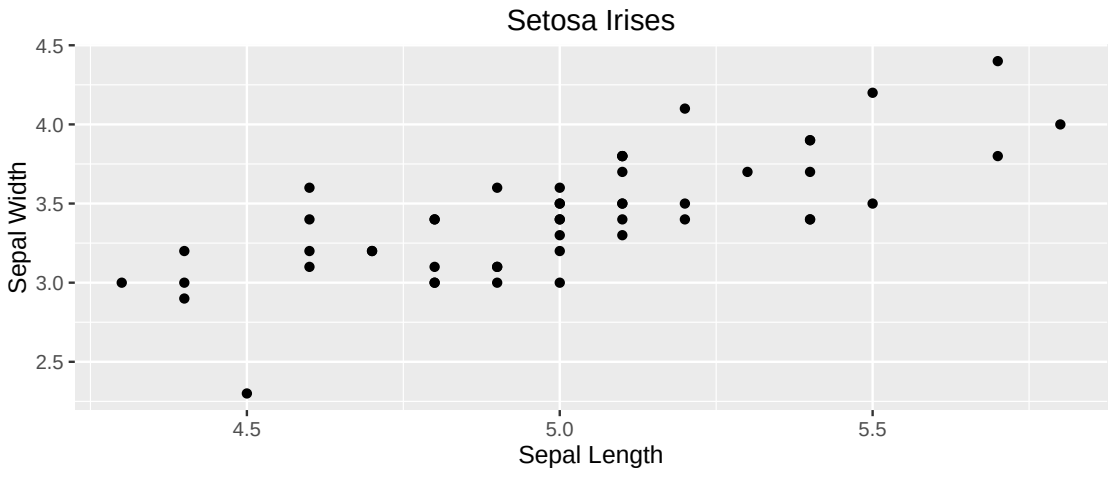
$$StdErr\left(\hat{\beta}_1\right) = \hat{\sigma}\sqrt{\frac{1}{S_{xx}}}$$

where $S_{xx} = \sum(x_i-\bar{x})^2$. These intervals can be used to calculate confidence intervals for β_0 and β_1 using the formulas:

$$\hat{\beta}_i \pm t_{n-2}^{1-\alpha/2} StdErr\left(\hat{\beta}_i\right)$$

Again we consider the `iris` dataset that is available in R. I wish to examine the relationship between sepal length and sepal width in the species *setosa.*

```
library(ggplot2)
library(dplyr)
setosa <- iris %>% filter( Species == 'setosa' )
ggplot(setosa, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point() +
  labs(x="Sepal Length", y="Sepal Width", title='Setosa Irises')
```



```
x <- setosa$Sepal.Length
y <- setosa$Sepal.Width
n <- length(x)
r <- sum( (x-mean(x))/sd(x) * (y-mean(y))/sd(y) ) / (n-1)
b1 <- r*sd(y)/sd(x)
b0 <- mean(y) - b1*mean(x)
cbind(r, b0, b1)

##              r         b0        b1
## [1,] 0.7425467 -0.5694327 0.7985283

yhat <- b0 + b1*x
resid <- y - yhat
SSE <- sum( resid^2 )
s2 <- SSE/(n-2)
s2

## [1] 0.06580573

Sxx <- sum( (x-mean(x))^2 )
stderr.b0 <- sqrt(s2) * sqrt( 1/n + mean(x)^2 / Sxx)
stderr.b1 <- sqrt(s2) * sqrt(1 / Sxx )
cbind(stderr.b0, stderr.b1)

##      stderr.b0 stderr.b1
## [1,] 0.5217119 0.1039651

t.star <- qt(.975, df=n-2)
c(b0-t.star*stderr.b0, b0+t.star*stderr.b0)

## [1] -1.6184048  0.4795395

c(b1-t.star*stderr.b1, b1+t.star*stderr.b1)

## [1] 0.5894925 1.0075641
```

Of course, we don't want to have to do these calculations by hand. Fortunately statistics packages will do all of the above calculations. In R, we will use `lm()` to fit a linear regression model and then call various accessor functions to give me the regression output I want.

```
cor( setosa$Sepal.Width,  setosa$Sepal.Length )

## [1] 0.7425467
```

```
model <- lm(Sepal.Width ~ Sepal.Length, data=setosa)
coef(model)

##  (Intercept) Sepal.Length
##   -0.5694327    0.7985283
```

```
confint(model)

##                   2.5 %    97.5 %
## (Intercept)  -1.6184048 0.4795395
## Sepal.Length  0.5894925 1.0075641
```

In general, most statistics programs will give a table of output summarizing a regression and the table is usually set up as follows:

Coefficient	Estimate	Standard Error	t-stat	p-value
Intercept	$\hat{\beta}_0$	$StdErr\left(\hat{\beta}_0\right)$	$t_0 = \frac{\hat{\beta}_0}{StdErr\left(\hat{\beta}_0\right)}$	$2 * P\left(T_{n-2} > \lvert t_0 \rvert\right)$
Slope	$\hat{\beta}_1$	$StdErr\left(\hat{\beta}_1\right)$	$t_1 = \frac{\hat{\beta}_1}{StdErr\left(\hat{\beta}_1\right)}$	$2 * P\left(T_{n-2} > \lvert t_1 \rvert\right)$

This table is printed by R by using the `summary()` function:

```
model <- lm(Sepal.Width ~ Sepal.Length, data=setosa)
summary(model)

##
## Call:
## lm(formula = Sepal.Width ~ Sepal.Length, data = setosa)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72394 -0.18273 -0.00306  0.15738  0.51709
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -0.5694     0.5217  -1.091    0.281
## Sepal.Length   0.7985     0.1040   7.681 6.71e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2565 on 48 degrees of freedom
## Multiple R-squared:  0.5514,Adjusted R-squared:  0.542
## F-statistic: 58.99 on 1 and 48 DF,  p-value: 6.71e-10
```

The first row is giving information about the y-intercept. In this case the estimate is -0.5694 and the standard error of the estimate is 0.5217. The t-statistic and associated p-value is testing the hypotheses: $H_0 : \beta_0 = 0$ vs $H_a : \beta_0 \neq 0$. This test is not usually of much interest. However because the equivalent test in the slope row testing $\beta_1 = 0$ vs $\beta_1 \neq 0$, the p-value of the slope row is *very* interesting because it tells me if I should include the slope variable in the model. If β_1 could be zero, then we should drop the predictor from our model and use the simple model $y_i = \beta_0 + \epsilon_i$ instead.

There are a bunch of other statistics that are returned by `summary()`. The `Residual standard error` is just $\hat{\sigma} = \sqrt{MSE}$ and the degrees of freedom for that error is also given. The rest are involved with the ANOVA interpretation of a linear model.

1.2.1 Anova Interpretation

Just as in the ANOVA analysis, we really have a competition between two models. The full model

$$y_i = \beta_0 + \beta_1 x + \epsilon_i$$

vs the simple model where x does not help predict y

$$y_i = \mu + \epsilon_i$$

where I've rewritten $\beta_0 = \mu$ to try to keep our notation straight. If I were to look at the simple model I would use $\bar{y} = \hat{\mu}$ as an estimate of μ and my Sum of Squared Error in the simple model will be

$$\begin{aligned} SSE_{simple} &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n} (y_i - \hat{\mu})^2 \end{aligned}$$

and the appropriate Mean Squared Error is

$$MSE_{simple} = \frac{1}{n-1} \sum (y_i - \hat{\mu})^2$$

We can go through the same sort of calculations for the full complex model and get

$$\begin{aligned} SSE_{complex} &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n} \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) \right)^2 \end{aligned}$$

and

$$MSE_{complex} = \frac{1}{n-2} \sum_{i=1}^{n} \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) \right)^2$$

Just as in the AVOVA analysis, if we often like to look at the difference between $SSE_{simple} - SSE_{comples} = SSE_{diff}$ and think of this quantity as the amount of variability that is explained by adding the slope parameter to the model. Just as in the AVOVA case we'll calculate

$$MSE_{diff} = SSE_{diff}/df_{diff}$$

where df_{diff} is the number of parameters that we added to the simple model to create the complex one. In the simple linear regression case, $df_{diff} = 1$.

Just as in the ANOVA case, we will calculate an f-statistic to test the null hypothesis that the simple model suffices vs the alternative that the complex model is necessary. The calculation is

$$f = \frac{MSE_{diff}}{MSE_{complex}}$$

and the associated p-value is $P(F_{1,n-2} > f)$. Notice that this test is *exactly* testing if $\beta_1 = 0$ and therefore the p-value for the F-test and the t-test for β_1 are the same. It can easily be shown that $t_1^2 = f$.

The Analysis of Variance table looks the same as what we have seen, but now we recognize that the rows actually represent the complex and simple models and the difference between them.

Source	df	Sum of Squares	Mean Squared	F-value	P-value
Difference	1	SSE_{diff}	$MSE_{diff} = \frac{SSE_{diff}}{1}$	$f = \frac{MSE_{diff}}{MSE_{complex}}$	$P(F_{1,n-2} > f)$
Complex	$n-2$	$SSE_{complex}$	$MSE_{complex} = \frac{SSE_{complex}}{n-2}$		
Simple	$n-1$	SSE_{simple}			

As usual, the ANOVA table for the regression is available in R using the `anova()` command.

```
model <- lm(Sepal.Width ~ Sepal.Length, data=setosa)
anova(model)

## Analysis of Variance Table
##
## Response: Sepal.Width
##              Df Sum Sq Mean Sq F value   Pr(>F)
## Sepal.Length  1 3.8821  3.8821  58.994 6.71e-10 ***
## Residuals    48 3.1587  0.0658
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

But we notice that R chooses not to display the row corresponding to the simple model.

I could consider SSE_{simple} as a baseline measure of the amount of variability in the data. It is interesting to look at how much of that baseline variability has been explained by adding the additional parameter to the model. Therefore we'll define the ratio R^2 as:

$$R^2 = \frac{SSE_{diff}}{SSE_{simple}} = \frac{SSE_{simple} - SSE_{complex}}{SSE_{simple}} = r^2$$

where r is Pearson's Correlation Coefficient. R^2 has the wonderful interpretation of the percent of variability in the response variable that can be explained by the predictor variable x.

1.2.2 Confidence Intervals vs Prediction Intervals

There are two different types of questions that we might ask about predicting the value for some x-value x_{new}.

We might be interested in a confidence interval for regression line. For this question we want to know how much would we expect the sample regression line move if we were to collect a new set of data. In particular, for some value of x, say x_{new}, how variable would the regression line be? To answer that we have to ask what is the estimated variance of $\hat{\beta}_0 + \hat{\beta}_1 x_{new}$? The variance of the regression line will be a function of the variances of $\hat{\beta}_0$ and $\hat{\beta}_1$ and thus the standard error looks somewhat reminiscent of the standard errors of $\hat{\beta}_0$ and $\hat{\beta}_1$. Recalling that $S_{xx} = \sum (x_i - \bar{x})^2$, we have:

$$\hat{Var}\left(\hat{\beta}_0 + \hat{\beta}_1 x_{new}\right) = \hat{\sigma}^2 \left(\frac{1}{n} + \frac{(x_{new} - \bar{x})^2}{S_{xx}}\right)$$

and therefore its $StdErr(\hat{\beta}_0 + \hat{\beta}_1 x_{new})$ is

$$StdErr\left(\hat{\beta}_0 + \hat{\beta}_1 x_{new}\right) = \hat{\sigma}\sqrt{\frac{1}{n} + \frac{(x_{new} - \bar{x})^2}{S_{xx}}}$$

We can use this value to produce a confidence interval for the regression line for any value of x_{new}.

$$\begin{aligned} Estimate &\pm t\ StdErr\left(Estimate\right) \\ \left(\hat{\beta}_0 + \hat{\beta}_1 x_{new}\right) &\pm t_{n-2}^{1-\alpha/2}\ \hat{\sigma}\sqrt{\frac{1}{n} + \frac{\left(x_{new} - \bar{x}\right)^2}{S_{xx}}} \end{aligned}$$

the expected value of new observation $\hat{E}\left(Y \mid X = x_{new}\right)$. This expectation is regression line but since the estimated regression line is a function of the data, then the line isn't the exactly the same as the true regression line. To reflect that, I want to calculate a confidence interval for where the true regression line should be.

I might instead be interested calculating a confidence interval for y_{new}, which I will call a *prediction interval* in an attempt to keep from being confused with the confidence interval of the regression line. Because we have

$$y_{new} = \beta_0 + \beta_1 x_{new} + \epsilon_{new}$$

Then my prediction interval will still be centered at $\hat{\beta}_0 + \hat{\beta}_1 x_{new}$ but the the uncertainty should be the sum of the uncertainty associated with the estimates of β_0 and β_1 and the additional variability associated with ϵ_{new}. In short,

$$\begin{aligned} \hat{Var}\left(\hat{\beta}_0 + \hat{\beta}_1 x_{new} + \epsilon\right) &= \hat{Var}\left(\hat{\beta}_0 + \hat{\beta}_1 x_{new}\right) + \hat{Var}\left(\epsilon\right) \\ &= \hat{\sigma}^2\left(\frac{1}{n} + \frac{\left(x_{new} - \bar{x}\right)^2}{S_{xx}}\right) + \hat{\sigma}^2 \end{aligned}$$

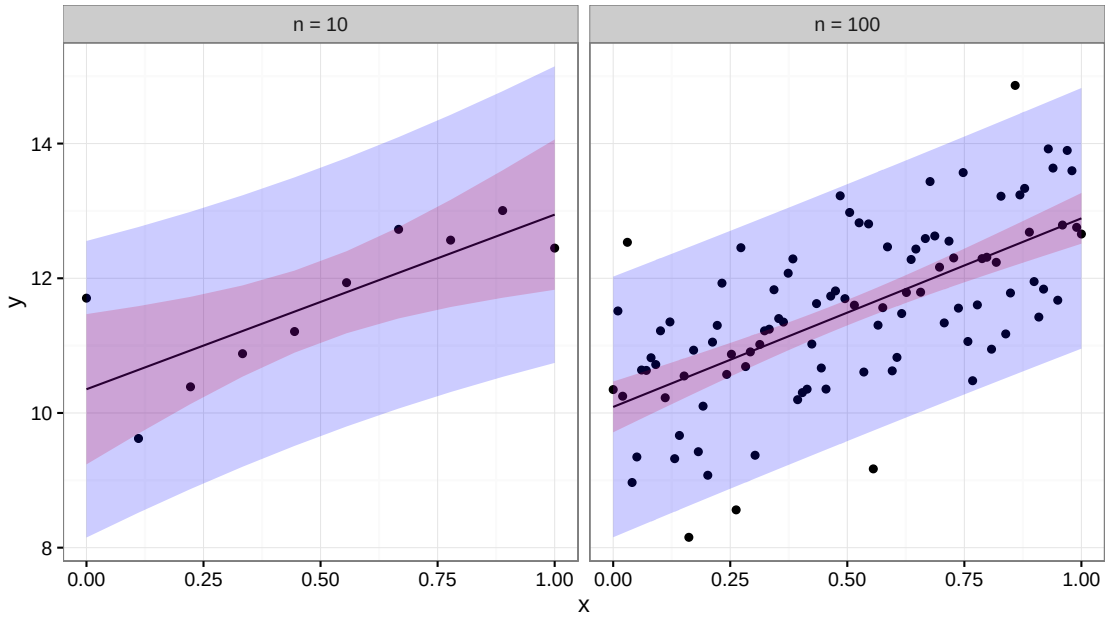
and the $StdErr\left(\right)$ of a new observation will be

$$StdErr\left(\hat{y}_{new}\right) = \hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{\left(x_{new} - \bar{x}\right)^2}{S_{xx}}}$$

So the prediction interval for a new observation will be:

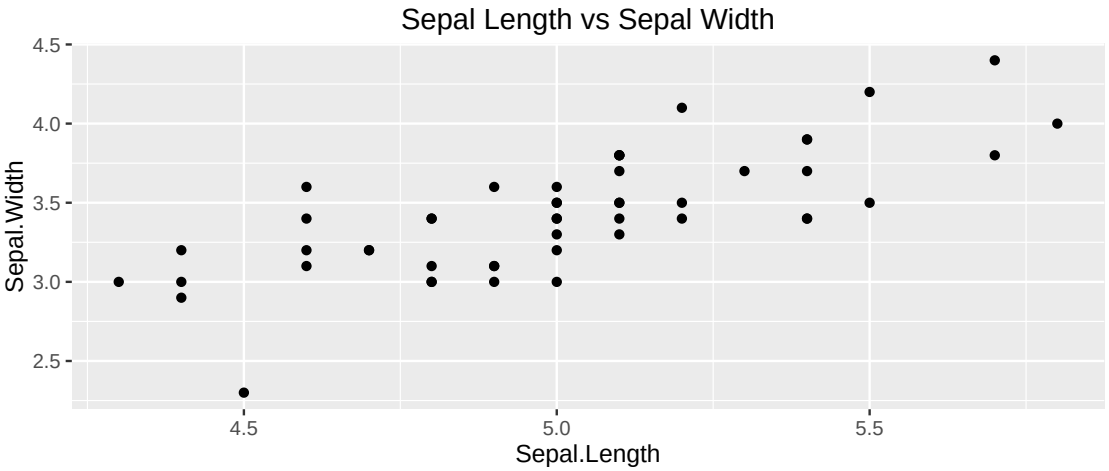
$$\left(\hat{\beta}_0 + \hat{\beta}_1 x_{new}\right) \pm t_{n-2}^{1-\alpha/2}\ \hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{\left(x_{new} - \bar{x}\right)^2}{S_{xx}}}$$

To emphasize the difference between confidence regions (capturing where we believe the regression line to lay) versus prediction regions (where new data observations will lay) we note that as the sample size increases, the uncertainty as to where the regression line lays decreases, but the prediction intervals will always contain a minimum width due to the error associated with an individual observation. Below are confidence (red) and prediction (blue) regions for two different sample sizes.



In general, you will not want to calculate the confidence intervals and prediction intervals by hand. Fortunately R makes it easy to calculate the intervals. The function `predict()` will calculate the point estimates along with confidence and prediction intervals. The function requires the `lm()` output along with an optional data frame (if you want to predict values not in the original data).

```
ggplot(setosa, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point() +
  ggtitle('Sepal Length vs Sepal Width')
```



```r
# fit the regression
model <- lm(Sepal.Width ~ Sepal.Length, data=setosa)

# display the first few predictions
head( predict(model, interval="confidence") )
```

```
##        fit      lwr      upr
## 1 3.503062 3.427519 3.578604
## 2 3.343356 3.267122 3.419590
## 3 3.183650 3.086634 3.280666
## 4 3.103798 2.991890 3.215705
## 5 3.423209 3.350256 3.496162
## 6 3.742620 3.632603 3.852637
```

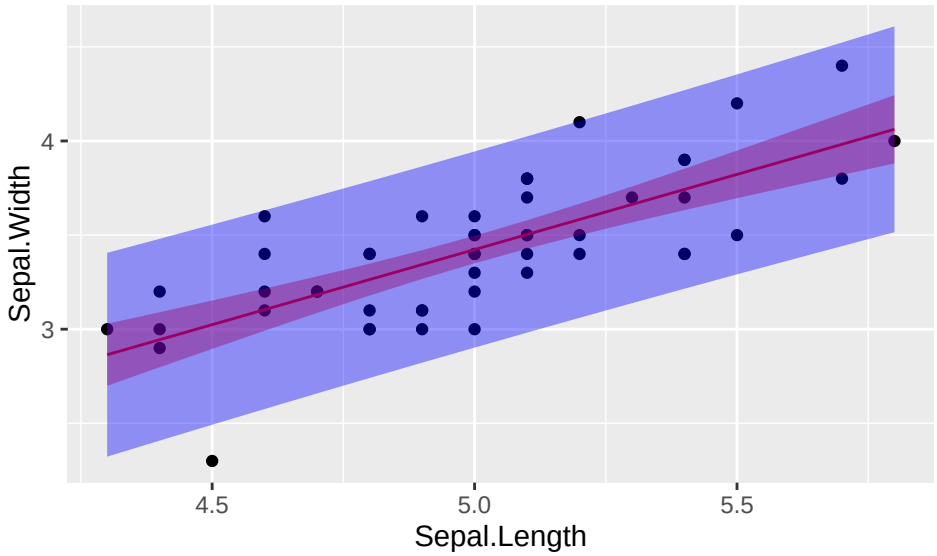
```r
# predict at x = 5.0
predict(model, interval="prediction", newdata=data.frame(Sepal.Length = 5.0))
```

```
##        fit      lwr      upr
## 1 3.423209 2.902294 3.944123
```

We can create a nice graph of the regression line and associated confidence and prediction regions using the following code in R:

```r
# ask for the confidence and prediction intervals
conf.region <- predict(model, interval='confidence')
pred.region <- predict(model, interval='prediction')

# add them to my original data frame
setosa <- setosa %>%
  mutate( fit = fitted(model),
          conf.lwr = conf.region[,2],
          conf.upr = conf.region[,3],
          pred.lwr = pred.region[,2],
          pred.upr = pred.region[,3])
```

```
# make a nice plot
ggplot(setosa) +
  geom_point(  aes(x=Sepal.Length, y=Sepal.Width) ) +
  geom_line(   aes(x=Sepal.Length, y=fit), col='red' ) +
  geom_ribbon( aes(x=Sepal.Length, ymin=conf.lwr, ymax=conf.upr), fill='red',  alpha=.4) +
  geom_ribbon( aes(x=Sepal.Length, ymin=pred.lwr, ymax=pred.upr), fill='blue', alpha=.4)
```

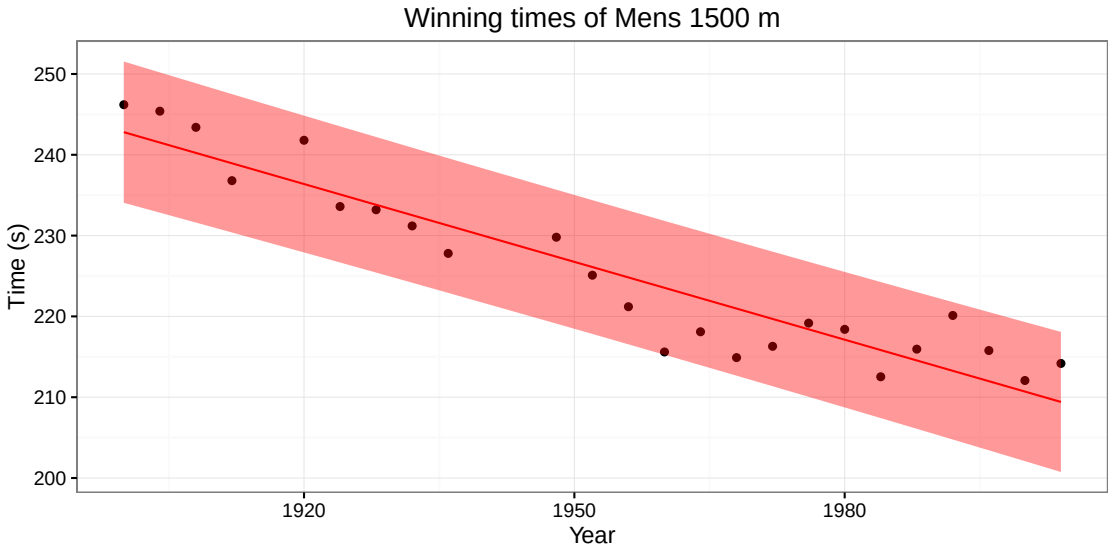


It is worth noting that these confidence intervals are all *point-wise* confidence intervals. If I want to calculate confidence or prediction intervals for a large number of x_{new} values, then I have to deal with the multiple comparisons issue. Fortunately this is easy to do in the simple linear regression case. Instead of using the $t_{n-2}^{1-\alpha/2}$ quantile in the interval formulas, we should use $W = \sqrt{2 * F_{1-\alpha,\,2,\,n-2}}$. Many books ignore this issue as does the `predict()` function in R.

1.3 Extrapolation

The data observed will inform a researcher about the relationship between the x and y variables, but *only in the range for which you have data!* Below are the winning times of the men's 1500 meter Olympic race.

```
library(HSAUR2)
data(men1500m)
small <- men1500m %>% filter( year != 1896 )  # Remove the 1896 Olympics

# fit the model and get the prediction interval
model <- lm( time ~ year, data=small )
small <- cbind(small, predict(model, interval='prediction') )

ggplot(small, aes(x=year, y=time, ymin=lwr, ymax=upr)) +
  geom_point() +
  geom_line( aes(y=fit), col='red' ) +
  geom_ribbon( fill='red',  alpha=.4) +
  labs( x='Year', y='Time (s)', title='Winning times of Mens 1500 m' ) + theme_bw()
```



If we are interested in predicting the results of the 2008 and 2012 Olympic race, what would we predict?

```
predict(model,
        newdata=data.frame(year=c(2008, 2012)),
        interval="prediction")

##        fit      lwr      upr
## 1 208.1293 199.3971 216.8614
## 2 206.8451 198.0450 215.6453
```

We can compare the predicted intervals with the time actually recorded by the winner of the men's 1500m. In Beijing 2008, Rashid Ramzi from Brunei won the event in 212.94 seconds and in London 2012 Taoufik Makhloufi from Algeria won in 214.08 seconds. Both times are within the corresponding prediction intervals, but clearly the linear relationship must eventually change and therefore our regression could not possibly predict the winning time of the 3112 race.

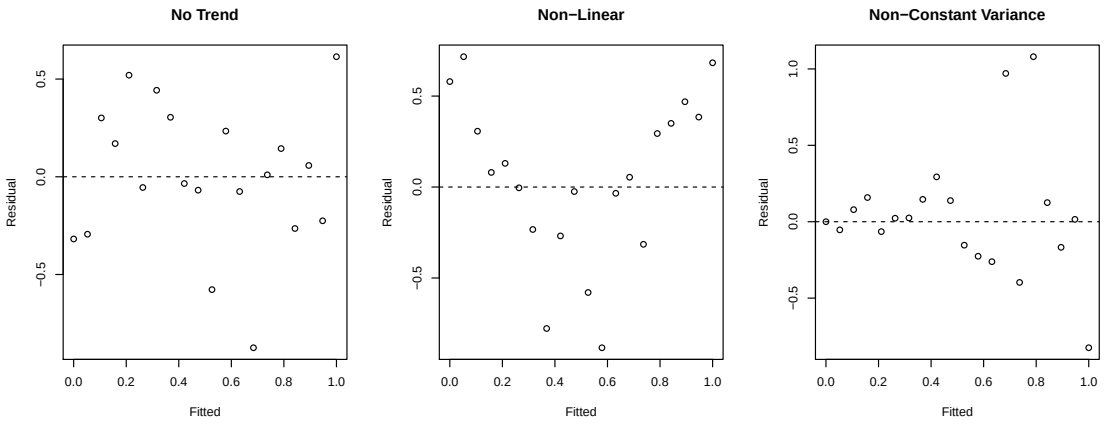
```
predict(model, newdata=data.frame(year=c(3112)), interval="prediction")

##         fit       lwr       upr
## 1 -146.2973 -206.7705 -85.82402
```

1.4 Checking Model Assumptions

As in the anova analysis, we want to be able to check the model assumptions. To do this, we will examine the residuals

$$e_i = y_i - \hat{y}_i$$

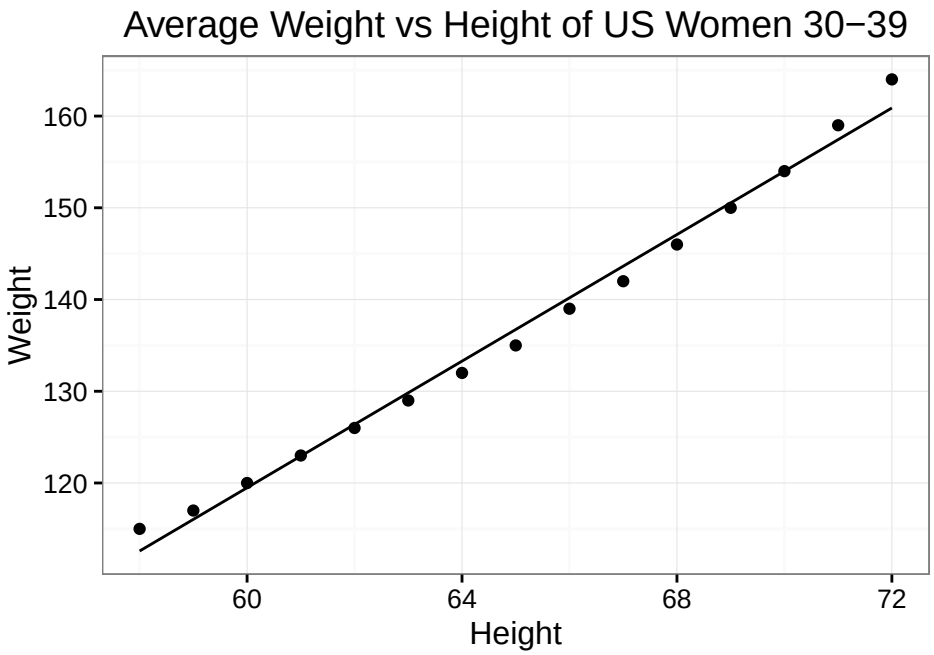
for normality using a QQ-plot as we did in Anova. To address the constant variance and linearity assumptions we will look at scatterplots of the residuals vs the fitted values $\hat{y}_i$. For the regression to be valid, we want the scatterplot to show no discernible trend. There are two patterns that commonly show up that indicate a violation of the regression assumptions.



To illustrate this, we'll look at data about the height and weight values for women between 30 and 39. (The data presented is actually the average weight for women of given heights, but is a useful example).

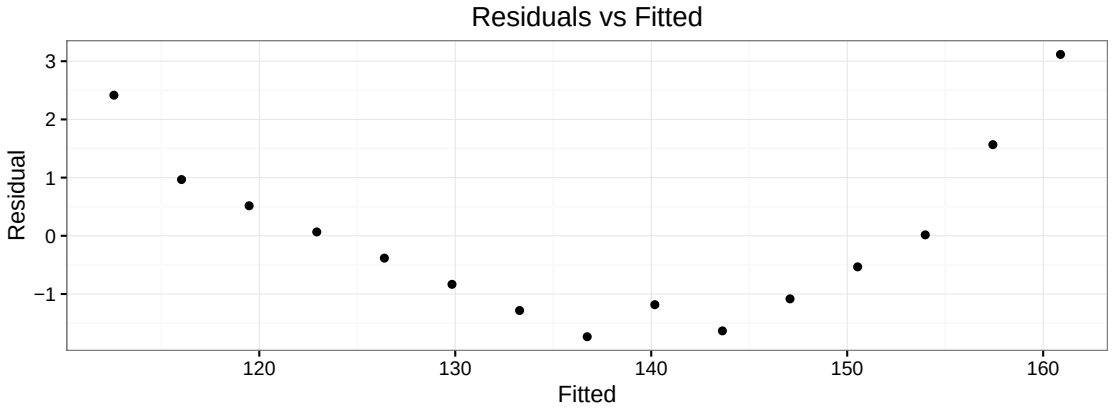
```
data('women')
str(women)

## 'data.frame': 15 obs. of  2 variables:
##  $ height: num  58 59 60 61 62 63 64 65 66 67 ...
##  $ weight: num  115 117 120 123 126 129 132 135 139 142 ...

# fit the regression line and add that info to my data frame
model <- lm(weight ~ height, data=women)
women <- women %>%
  mutate( fit   = fitted(model),
          resid = resid(model)  )
```

```
ggplot(women) +
  geom_point(aes(x=height, y = weight )) +
  geom_line( aes(x=height, y = fit    )) +
  labs( x='Height', y='Weight',
        title='Average Weight vs Height of US Women 30-39' ) +
  theme_bw()
```



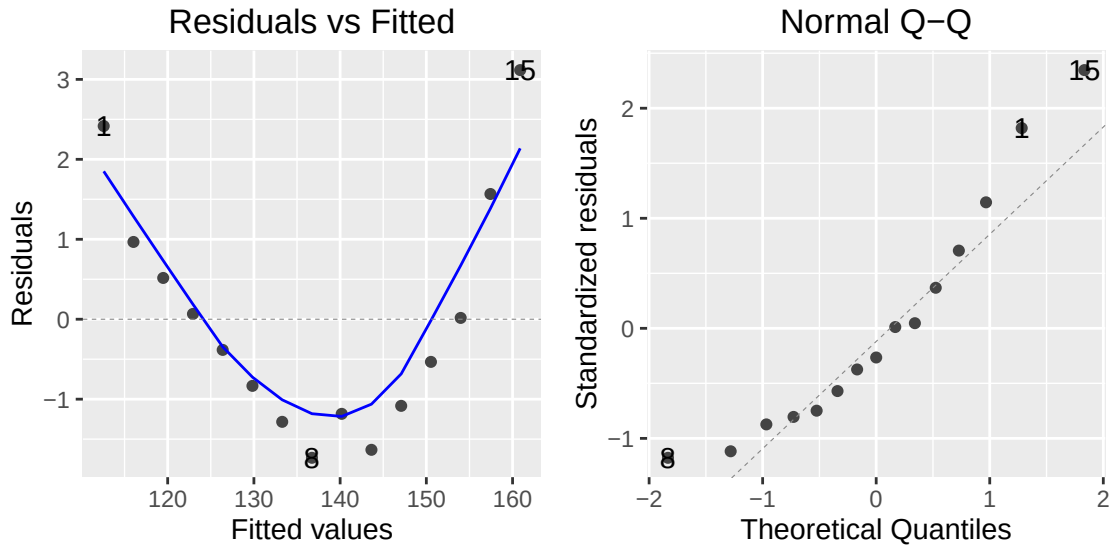
If we squint at this graph, we see that the data are not perfectly linear and there appears to be some curvature. It isn't particularly clear from this graph however. However, when we look at the residuals vs fitted values and immediately conclude that the linearity assumption is violated.

```
ggplot(women, aes( x=fit, y=resid )) +
  geom_point() +
  labs(title='Residuals vs Fitted', x='Fitted', y='Residual') + theme_bw()
```



This sort of graph is useful enough that R provides a quick way to create it. The function `plot( lm.object )` will take the input linear model and produce 6 diagnostic plots. If we've loaded the `ggfortify` package, then the function `autoplot()` will do the same thing.

```
autoplot(model, which=c(1,2))
```

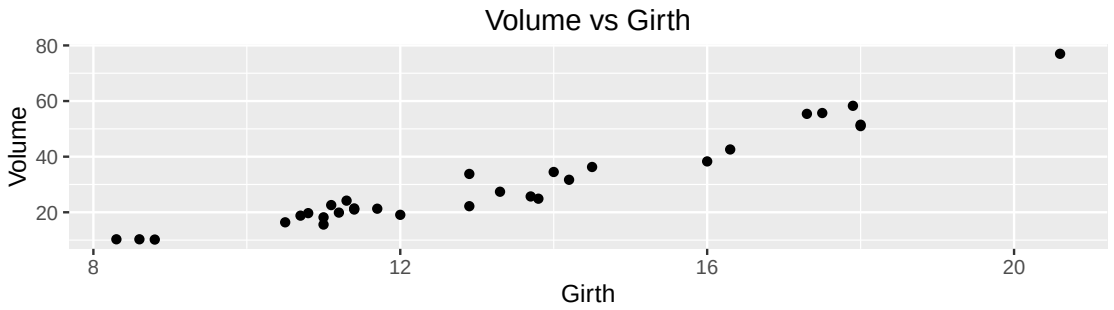


Example: Cherry Trees

To see the entire regression process, we consider the problem of predicting the volume of lumber produced by a cherry tree of a given diameter. The data are given in a dataset pre-loaded in R called `trees`.

Step one: Graph the data. The first step in a regression analysis is to graph the data and think about if a linear relationship makes sense.

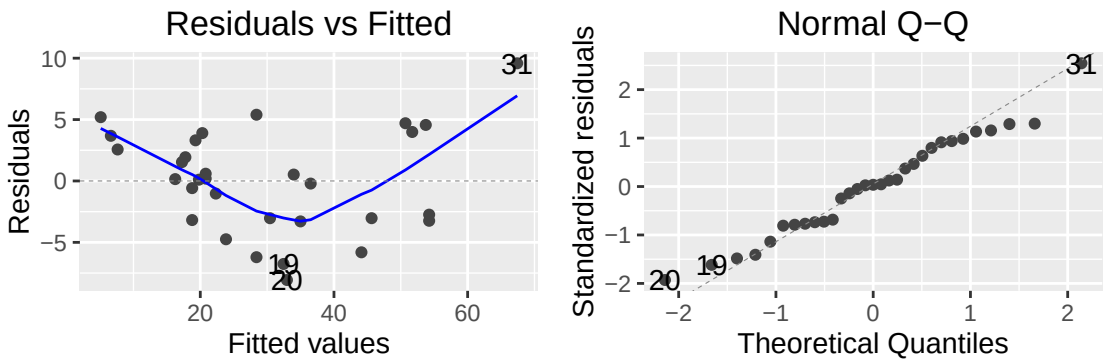
```
head(trees)  # 3 columns Girth, Height, Volume

##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7

ggplot(trees, aes(x=Girth, y=Volume)) +
  geom_point() +
  ggtitle('Volume vs Girth')
```



Initially, it looks like a line is a pretty good description of this relationship.

Step two: Fit a regression and examine the diagnostic plots.

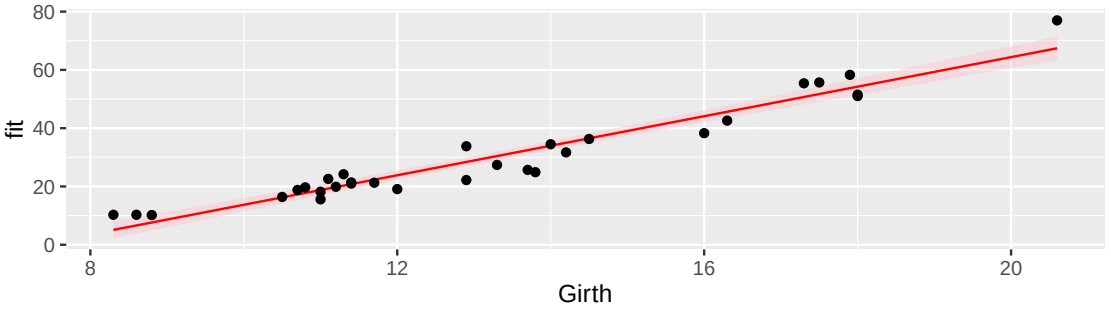
```
model <- lm( Volume ~ Girth, data=trees )
autoplot(model, which=c(1,2))
```



The normality assumption isn't too bad, but there is a trend in the residual plot. At this point I would think about a slightly more complicated model, e.g. should we include height in the model or perhaps `Girth^2`? The implications of both of these possibilities will be explored in STA 571 but for now we'll just continue using the model we have.

Step three: Plot the data and the regression model.

```
trees <- cbind( trees, predict(model, interval='confidence') )
head(trees)  # now we have the fit, lwr, upr columns

##   Girth Height Volume       fit       lwr       upr
## 1   8.3     70   10.3  5.103149  2.152294  8.054004
## 2   8.6     65   10.3  6.622906  3.799685  9.446127
## 3   8.8     63   10.2  7.636077  4.896577 10.375578
## 4  10.5     72   16.4 16.248033 14.156839 18.339228
## 5  10.7     81   18.8 17.261205 15.235884 19.286525
## 6  10.8     83   19.7 17.767790 15.774297 19.761284

ggplot(trees, aes(x=Girth)) +
  geom_ribbon( aes( ymin=lwr, ymax=upr), alpha=.4, fill='pink' ) +
  geom_line( aes(y=fit), color='red') +
  geom_point(aes(y=Volume))
```



While the model we've selected isn't as good as it could be (we are missing the curvature), this isn't horribly bad and might suffice for a first pass[3].

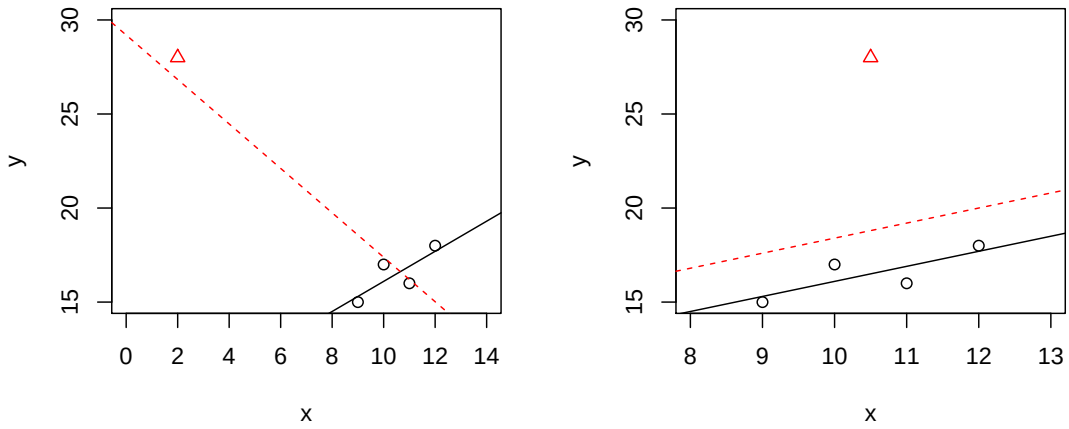
[3] "All models are wrong, but some are useful." George Box

Step four: Evaluate the model coefficients.

```
summary(model)

##
## Call:
## lm(formula = Volume ~ Girth, data = trees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.065 -3.107  0.152  3.495  9.587
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -36.9435     3.3651  -10.98 7.62e-12 ***
## Girth         5.0659     0.2474   20.48  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.252 on 29 degrees of freedom
## Multiple R-squared:  0.9353,Adjusted R-squared:  0.9331
## F-statistic: 419.4 on 1 and 29 DF,  p-value: < 2.2e-16

confint(model)

##                   2.5 %     97.5 %
## (Intercept) -43.825953 -30.060965
## Girth         4.559914   5.571799
```

1.5 Influential Points

Sometimes a dataset will contain one observation that has a large effect on the outcome of the model. Consider the following datasets where the red denotes a highly influential point and the red line is the regression line including the point.

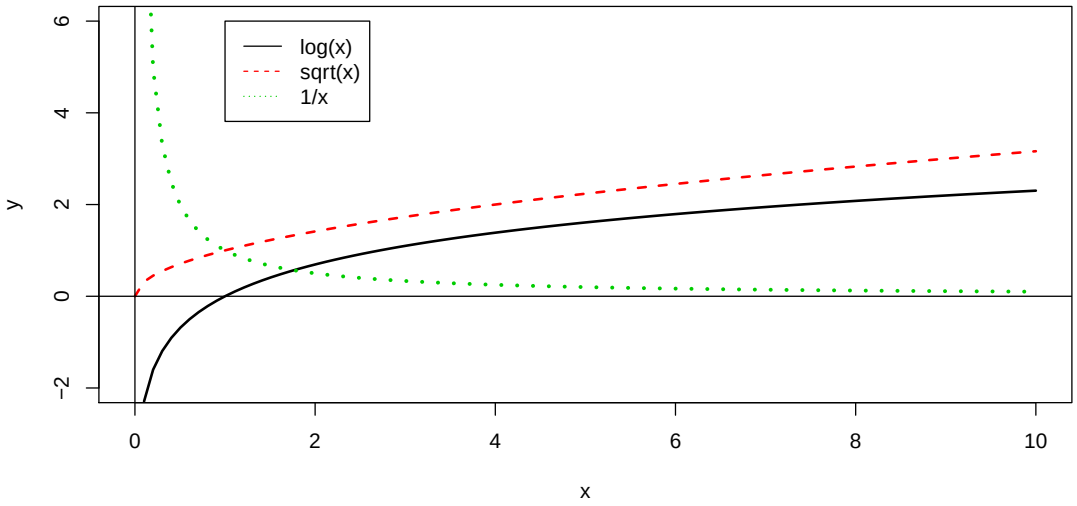


The question of what to do with influential points is not easy to answer. Sometimes these are data points that are a result of lab technician error and should be removed. Sometimes they are the result of an important process that is not well understood by the researcher. It is up to the scientist to figure out which is the case and take appropriate action.

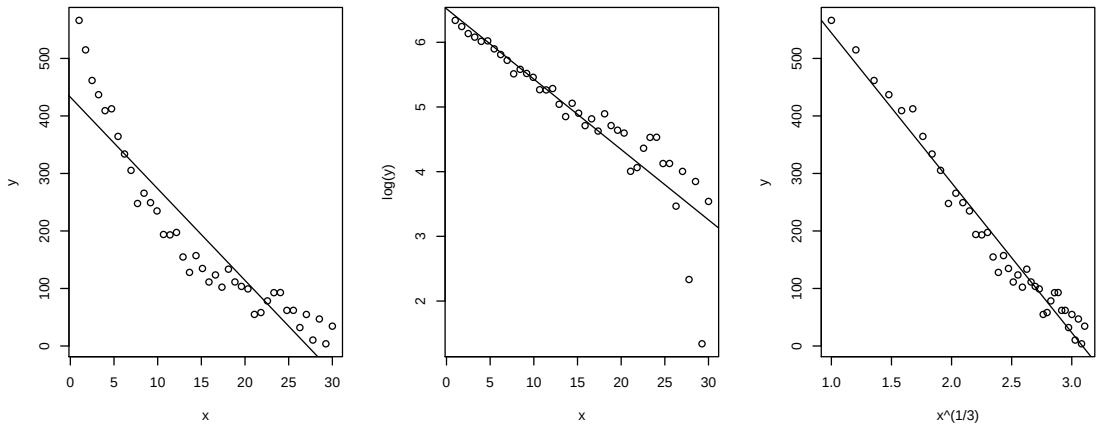
One solution is to run the analysis both with and without the influential point and see how much it affects your inferences.

1.6 Transformations

When the normality or constant variance assumption is violated, sometimes it is possible to *transform* the data to make it satisfy the assumption. Often times count data is analyzed as log(count) and weights are analyzed after taking a square root or cube root transform.



We have the option of either transforming the x-variable or transforming the y-variable or possibly both. One thing to keep in mind, however, is that transforming the x-variable only effects the linearity of the relationship. Transforming the y-variable effects both the linearity and the variance.

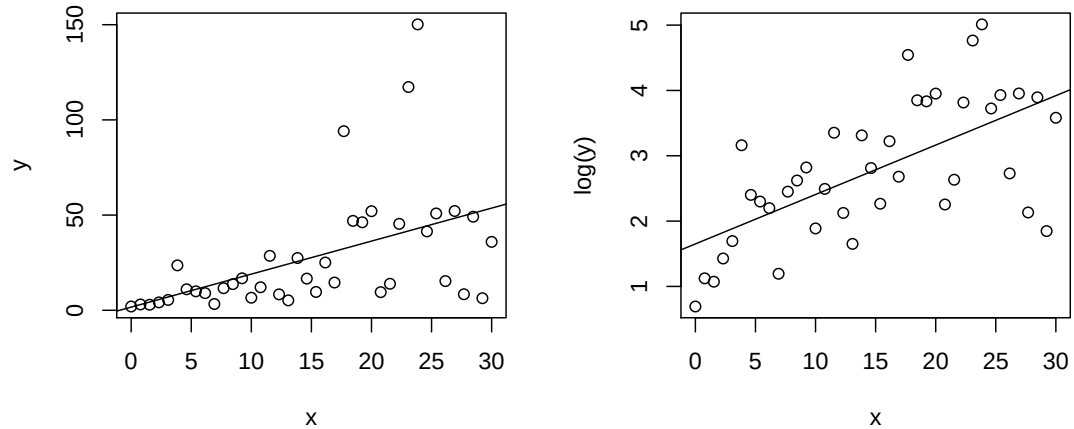


Unfortunately it is not always obvious what transformation is most appropriate. The Box-Cox family of transformations for the y-variable is

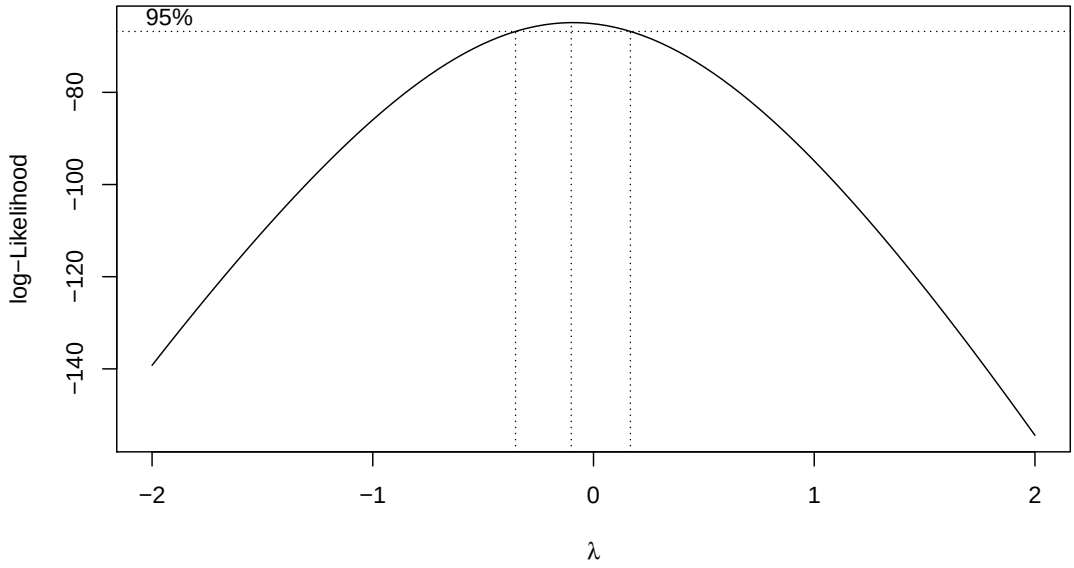
$$f(y \mid \lambda) \quad = \quad \begin{cases} y^{\lambda} & \text{if } \lambda \neq 0 \\ \log y & \text{if } \lambda = 0 \end{cases}$$

which includes squaring ($\lambda = 2$), square root ($\lambda = 1/2$) and as $\lambda \to 0$ the transformation converges to $\log y$. (To do this correctly we should define the transformation in a more complicated fashion, but that level of detail is unnecessary here.) The transformation is selected by looking at the profile log-likelihood value of different values of λ and we want to use the λ that maximizes the log-likelihood.

Of course, we also want to use a transformation that isn't completely obscure and is commonly used in the scientific field, so square roots, reciprocals, and logs are preferred.



```
library(MASS)
str(mydata)

## 'data.frame': 40 obs. of  2 variables:
##  $ x: num  0 0.769 1.538 2.308 3.077 ...
##  $ y: num  2 3.08 2.92 4.17 5.44 ...

boxcox(y~x, data=mydata, plotit=TRUE)
```



Here we see the resulting confidence interval for λ contains 0, so a log transformation would be most appropriate.

In general, deciding on a transformation to use is often a trade-off between statistical pragmatism and interpretability. In cases that a transformation is not possible, or the interpretation is difficult, it is necessary to build more complicated models that are interpretable.