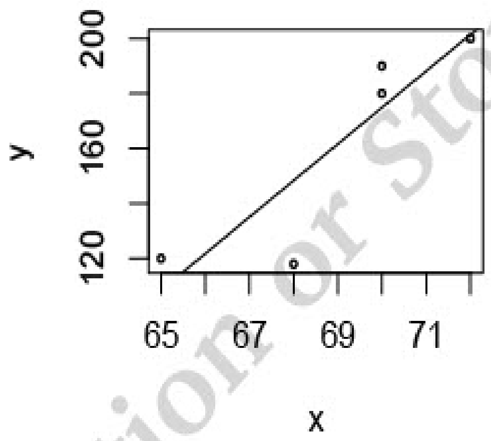
```
Residuals:
     1      2      3      4      5
 -1.46   5.11  11.54 -30.31  15.11

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -755.11     272.53   -2.77    0.070 .
x              13.29       3.95    3.37    0.044 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 20.9 on 3 degrees of freedom
Multiple R-squared:  0.791,Adjusted R-squared:  0.721
F-statistic: 11.3 on 1 and 3 DF,  p-value: 0.0436
```

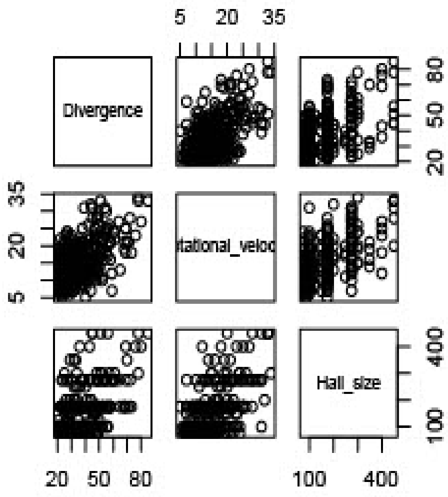


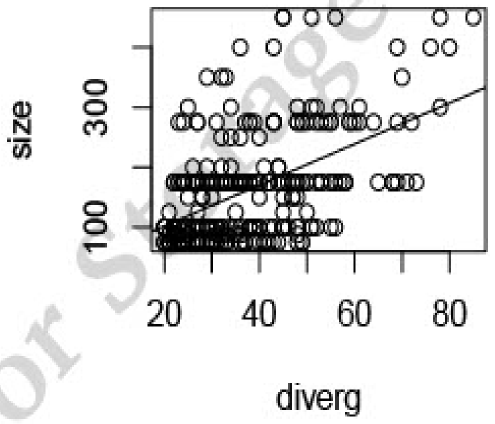
Example: Regression on Hail Data

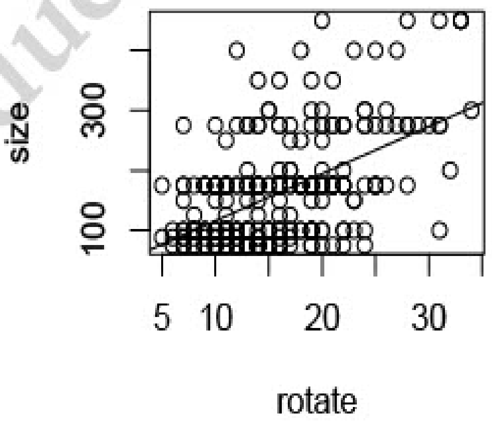
In practice, two quantities called "divergence" and "rotate" are measured by Doppler radar, while hail size is measured directly, i.e., on the ground. But if we can relate hail size to divergence and rotate, then we can predict hail size from Doppler radar. In regression lingo, size is the response (or dependent) variable, and the others are predictors (or independent variables, or covariates).

```
dat <- read.table("hail_dat.txt", header=T)

plot(dat)
cor(dat)  # This shows the correlations between ALL the vars in the hail data

                    Divergence Rotational_velocity Hail_size
Divergence              1.0000              0.5496    0.5214
Rotational_velocity     0.5496              1.0000    0.5386
Hail_size               0.5214              0.5386    1.0000

size <- dat[, 3]  # Name the 3 columns in dat. Size is in 100th-of-an-inch.
```

```r
rotate <- dat[, 2]
diverg <- dat[, 1]

model.1 <- lm(size ~ diverg)  # Regression of size and divergence.
plot(diverg, size)  # Viewing the scatterplot.
abline(model.1)  # Viewing the regression line.

model.2 <- lm(size ~ rotate)  # Regression on size and rotation.
plot(rotate, size)
abline(model.2)
```







Note that it looks like the line is not really going "through" the data; it seems like the line's slope should be larger. The fit is in fact correct. The line that intuitively (or visually) goes "through" the scatterplot is NOT the regression line, but something else called the "sd line."

3.5 Analysis of Variance (ANOVA) in Regression

ANOVA decomposes SST (total sum of squares) into $SS_{explained}$ and $SS_{unexplained}$ (SSE).

$$SS_{explained} = \sum_{i=1}^{n} (\hat{y}_i - \overline{y})^2 \tag{1}$$

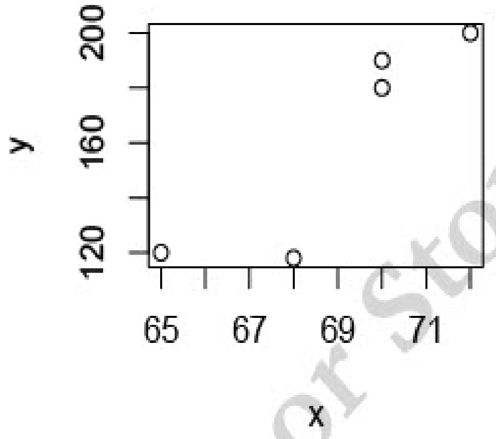
$$SS_{unexplained} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2}$$

$$SST = \sum_{i=1}^{n} (y_i - \overline{y})^2 \tag{3}$$

$SS_{explained}$ is converted to a proportion called R-squared (a.k.a. coefficient of determination). It measures the proportion of the variability in y that is explained by x. It's a measure of goodness-of-fit.

```
x <- c(72, 70, 65, 68, 70) # Enter data into R.
y <- c(200, 180, 120, 118, 190) # See 1.1 for alternative ways to enter data.
plot(x, y)  # Plot the scatterplot.
cor(x, y)  # Correlation between x and y.

[1] 0.8892

model.1 <- lm(y ~ x)  # Fitting the regression.
anova(model.1)  # Note that SS_explained = 4942 and SSE = 1309

Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
x          1   4942    4942    11.3  0.044 *
Residuals  3   1309     436
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

summary(model.1)  # R-squared = 0.7906


Call:
lm(formula = y ~ x)

Residuals:
     1      2      3      4      5
 -1.46   5.11  11.54 -30.31  15.11

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -755.11     272.53   -2.77    0.070 .
x              13.29       3.95    3.37    0.044 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 20.9 on 3 degrees of freedom
Multiple R-squared:  0.791,Adjusted R-squared:  0.721
F-statistic: 11.3 on 1 and 3 DF,  p-value: 0.0436

# The R^2 is reported as "Multiple R-squared".

# Note that the R-squared from summary() agrees with 1-(SSE/SST):
1 - (1308.9 / (4942.3 + 1308.9))

[1] 0.7906
```



$SS_{unexplained}$ (SSE) is converted to a standard deviation (of errors), and denoted as *se*. This standard deviation of errors is also called standard deviation about regression. Either way it is reported as “Residual standard error: 20.9”. Note that it is equal to $\sqrt{\frac{SSE}{n-2}}$:

```
sqrt(1308.9 / (5 - 2))

[1] 20.89
```

In sum, R-squared and *se* together tell you how good the model is. R-squared tells you what percent of the variance in y can be attributed to x, and *se* tells you the typical error, i.e., deviation of data from the line.

To get R^2 (and nothing else), use the following command:

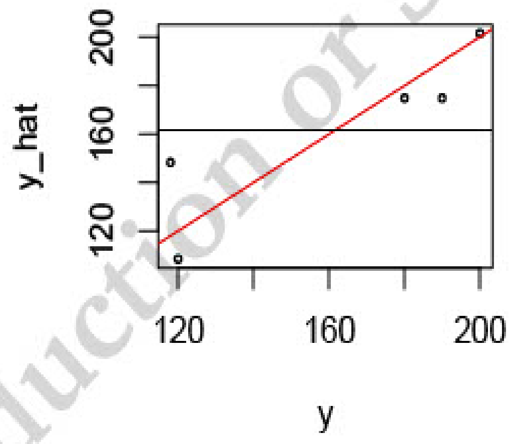
```
summary(model.1)$r.squared

[1] 0.7906

# Do the following to check that R's SSE really is Sum of Squared Errors:
y_hat <- predict(model.1)  # This is a quick way of getting y_hat.
y_hat  # To see the predictions.

    1     2     3     4     5
201.5 174.9 108.5 148.3 174.9
```

```
sum((y - y_hat) ^ 2)  # 1308.914 = SSE above.

[1] 1309
```

3.6 Visual Assessment of Goodness-of-Fit

One way to access the goodness of fit is to examine the scatterplot of predicted y versus actual y.

```
y_hat <- predict(model.1)

# Alternatively, you can use the predictions stored in lm() itself:
y_hat <- model.1$fitted.values

# Here is the scatterplot of predicted size vs. actual size:
plot(y, y_hat, cex = 0.5)

abline(0, 1, col = "red")  # Add a diagonal line.
abline(h = mean(y))  # Add a horizontal line at the mean of y (i.e., size).
```

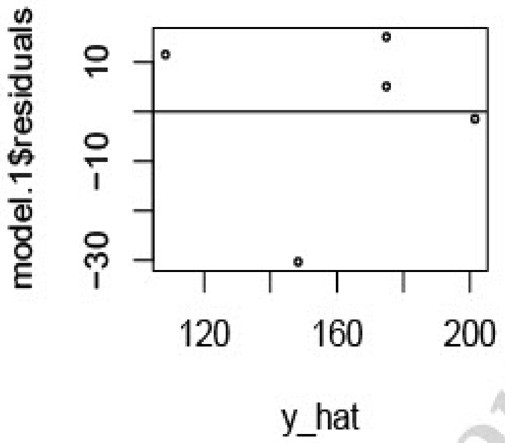


If the model were good, this scatterplot would be symmetrically spread about the red line. But, clearly our model is not good. This scatterplot (of predicted vs. actual) is often a great way of visualizing how well the model is doing. For example, we see that for smaller hail size (i.e., small x value), the predictions of size are all above the diagonal indicating that the model over-predicts the size of small hail. Looking at larger x values, it's clear that the model under-predicts the size of large hail.

If a model is completely useless, then the predictions will be symmetrically spread about the horizontal line at the mean of y. Here, we can see that our model is nearly (but not completely) useless. This kind of model diagnosis can help in coming up with a better model.

Another visual assessment tool is the residual plot. This plot checks different facet of "goodness" (or quality) than the above plot.

```
# The residuals are already contained in what lm() returns.
plot(y_hat, model.1$residuals, cex = 0.5)
abline(h = 0)
```



In a good fit, these residuals (or errors) should NOT display any relationship with the predicted values. One way to confirm that there is no relationship is to compute the correlation:

```
cor(y_hat, model.1$residuals)

[1] -1.237e-16
```

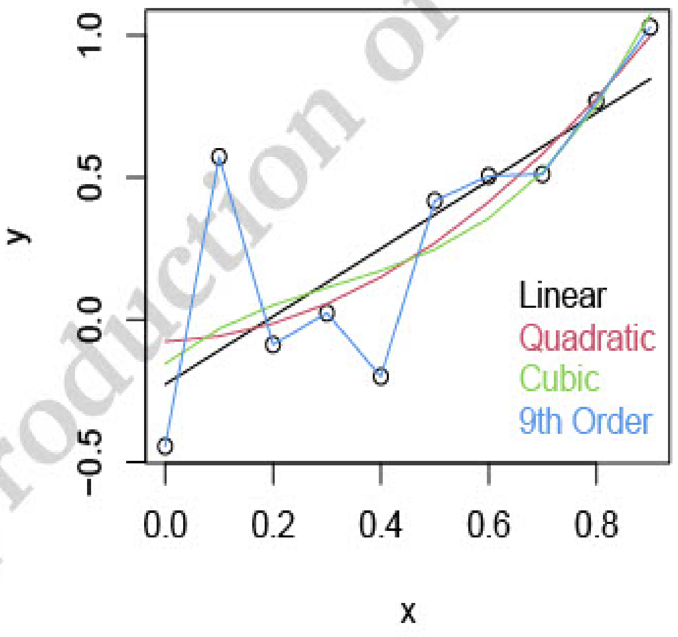
The fact that the correlation is zero is not a direct reflection of the goodness of fit, because that correlation is zero by construction. If it's not zero, one has a bug! In fact, it is the identically-zero nature of this correlation which makes a plot of the residuals vs. $\hat{y}$ a useful plot to examine. The correlation between the residuals and the observed y values is not identically zero; and for that reason the corresponding scatterplot is not readily interpretable.

3.7 Nonlinear Fits (Linear Regression with Higher Order Terms)

Linear regression is actually NOT linear when it comes to allowing nonlinear relationships between x and y. (The term "linear" refers to the parameters of the model, i.e., the regression coefficients.) This is good news, because linear regression can fit any nonlinear data. But it's also bad news, because the ability to fit nonlinear data also allows for overfitting. In developing regression models of data it is important to assure that the model is not overfitting the data, because such a model will have poor *predictive* capability. Toward the end of this book we will see how to assess the predictive capability of a regression model. Here, let's first confirm that linear regression *can* overfit (memorize) data.

```
set.seed(12)  # Set a seed to ensure reproducable results.
x <- seq(0, 0.9, 0.1)  # Pick 10 x's between 0 and 1.
y <- x + rnorm(10, 0, 0.3)  # x and y are "truly" linear, plus error.
plot(x,y)  # Look at the data.

lm.1 <- lm(y ~ x)  # Fit the simplest regression model
```

```
lines(x, lm.1$fitted.values)

lm.2 <- lm(y ~ x + I(x^2))  # Fit a regression model including the quadratic term.
lines(x, lm.2$fitted.values, col = 2) # The I() is necessary. Don't ask why!

lm.3 <- lm(y ~ x + I(x^2) + I(x^3))  # Add a cubic term.
lines(x, lm.3$fitted.values, col = 3)  # Note that the fit is getting more curvy.

lm.4 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8)
           + I(x^9))
# Fit a 9th order polynomial.
lines(x, lm.4$fitted.values, col = 4)
summary(lm.4)$r.squared  # Examine the R-squared.

[1] 1

legend('bottomright', c('Linear', 'Quadratic', 'Cubic', '9th Order'),
       text.col = c(1, 2, 3, 4), bty = 'n')

# Note that the last model will have no predictive power since it overfits the data.
```



3.8 Model Comparison

Example: Hail Data

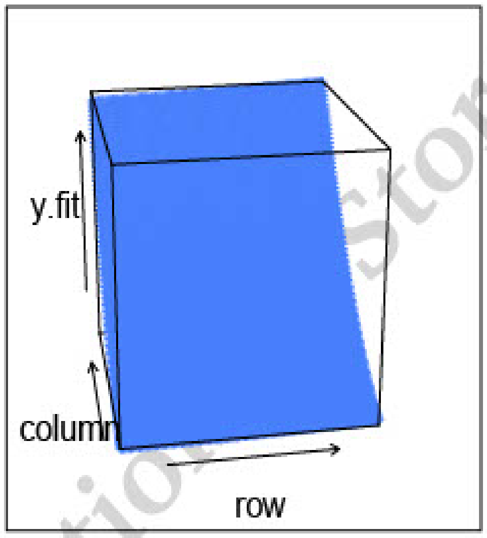
Note that the closer R^2 is to 1, the "better" the fit and the closer it is to 0, the worse. But do recall that because of overfitting concerns, higher R^2 does not necessarily mean better predictions on new/future data.

```
dat <- read.table("hail_dat.txt", header = T)

x_1 <- dat[, 1]  # Divergence.
x_2 <- dat[, 2]  # Rotate.
y <- dat[, 3]  # Hail size. Size is in 100th-of-an-inch.
# Renaming the columns in dat:
colnames(dat) <- c("x_1", "x_2", "y")

lm.1 <- lm(y ~ x_1)  # Predicting size from divergence (simple regression).
summary(lm.1)


Call:
lm(formula = y ~ x_1)

Residuals:
   Min     1Q Median     3Q    Max
-126.1  -50.9  -19.8   44.8  262.6

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   33.673     13.361    2.52    0.012 *
x_1            3.417      0.334   10.23   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 76 on 280 degrees of freedom
Multiple R-squared:  0.272,Adjusted R-squared:  0.269
F-statistic:  105 on 1 and 280 DF,  p-value: <2e-16

lm.2 <- lm(y ~ x_2)  # Predicting size from rotation (simple regression).
summary(lm.2)


Call:
lm(formula = y ~ x_2)

Residuals:
   Min     1Q Median     3Q    Max
-180.9  -55.1  -11.6   36.6  268.4

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   37.268     12.508    2.98   0.0031 **
x_2            7.858      0.735   10.70   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 75 on 280 degrees of freedom
Multiple R-squared:  0.29,Adjusted R-squared:  0.288
F-statistic:  114 on 1 and 280 DF,  p-value: <2e-16
```

```
lm.3 <- lm(y ~ x_1 + x_2) # Predicting size from both (multiple regression).
summary(lm.3)


Call:
lm(formula = y ~ x_1 + x_2)

Residuals:
   Min     1Q Median     3Q    Max 
-157.4  -52.0  -12.2   35.5  261.8 

Coefficients:
            Estimate Std. Error t value     Pr(>|t|)    
(Intercept)   -1.179     13.684   -0.09         0.93    
x_1            2.117      0.375    5.65 0.0000000401 ***
x_2            5.268      0.835    6.31 0.0000000011 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 71.2 on 279 degrees of freedom
Multiple R-squared:  0.363,Adjusted R-squared:  0.358 
F-statistic: 79.5 on 2 and 279 DF,  p-value: <2e-16

lm.4 <- lm(y ~ x_1 + x_2 + x_1:x_2)  # Multiple regression with interaction.
summary(lm.4)


Call:
lm(formula = y ~ x_1 + x_2 + x_1:x_2)

Residuals:
   Min     1Q Median     3Q    Max 
-157.1  -50.8  -14.2   36.6  264.0 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept)  71.6091    34.7693    2.06    0.040 *
x_1           0.1989     0.9217    0.22    0.829  
x_2           1.0612     2.0269    0.52    0.601  
x_1:x_2       0.1030     0.0453    2.27    0.024 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.7 on 278 degrees of freedom
Multiple R-squared:  0.375,Adjusted R-squared:  0.368 
F-statistic: 55.5 on 3 and 278 DF,  p-value: <2e-16

lm.5 <- lm(y ~ x_1 + x_2 + I(x_1 ^ 2) + I(x_2 ^ 2)) # Multiple quadratic regression.
summary(lm.5)


Call:
```

```
lm(formula = y ~ x_1 + x_2 + I(x_1^2) + I(x_2^2))

Residuals:
   Min     1Q Median     3Q    Max 
-180.8  -48.6  -16.1   38.1  266.0 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept)  88.4961    38.1671    2.32    0.021 *
x_1           0.4789     1.6626    0.29    0.774  
x_2          -1.9992     3.3297   -0.60    0.549  
I(x_1^2)      0.0175     0.0186    0.94    0.346  
I(x_2^2)      0.2053     0.0918    2.24    0.026 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.5 on 277 degrees of freedom
Multiple R-squared:  0.38,Adjusted R-squared:  0.371 
F-statistic: 42.4 on 4 and 277 DF,  p-value: <2e-16

lm.6 <- lm(y ~ x_1 + x_2 + I(x_1 ^ 2) + I(x_2 ^ 2) + x_1:x_2)
summary(lm.6)


Call:
lm(formula = y ~ x_1 + x_2 + I(x_1^2) + I(x_2^2) + x_1:x_2)

Residuals:
   Min     1Q Median     3Q    Max 
-179.7  -48.9  -16.1   38.0  265.7 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept)  89.0739    38.4006    2.32    0.021 *
x_1           0.4974     1.6695    0.30    0.766  
x_2          -2.0870     3.3794   -0.62    0.537  
I(x_1^2)      0.0145     0.0264    0.55    0.584  
I(x_2^2)      0.1921     0.1228    1.56    0.119  
x_1:x_2       0.0137     0.0843    0.16    0.872  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.6 on 276 degrees of freedom
Multiple R-squared:  0.38,Adjusted R-squared:  0.369 
F-statistic: 33.8 on 5 and 276 DF,  p-value: <2e-16

# Plotting a surface that goes through the cloud of points in 3d.
# Suppose we decided that the best model is the most complex model, above:
lm.e <- lm(size ~ diverg + rotate + I(diverg^2) + I(rotate^2) + I(diverg * rotate))

x <- seq(min(rotate), max(rotate), length = 100)  # x and y simply define a
y <- seq(min(diverg), max(diverg), length = 100)  # grid in the x-y plane.
```

```
f <- function(x, y) {
  r <- lm.e$coeff[1] + lm.e$coeff[2] * x + lm.e$coeff[3]*y
  + lm.e$coeff[4] * x ^ 2 + lm.e$coeff[5] * y ^ 2 + lm.e$coeff[6] * x * y
}
y.fit <- outer(x, y, f)  # y contains the "height" values of the surface,
                         # over the x-y grid; that's what outer() does. Look it up.

library(lattice)  # Loading the library that contains the function cloud().
# Making a 3d plot of the points of the plane.
cloud(y.fit, type = "p", screen = list(z = 10, x = -70, y = 0))

# Note that in spite of the nonlinearity of the regression function
# itself, i.e. with quadratic and an interaction terms, the surface is
# mostly planar in the range of our data (i.e., x and y).
```



Here is a discussion of all of the above results based on the R^2 values: It seems like

1. Rotate is a better predictor of size than diverge.
2. The two of them together make for an even better model.
3. Quadratic terms for each, make the model even better, but not by much
4. R^2 goes from 0.3629 to 0.3800.
5. An interaction term, without quadratic terms, gives a model that is comparable to what we got from a quadratic model with no interaction.
6. Quadratic and interaction terms, together, "seem" to give the best model.

Note that R^2 increases as the complexity of the model increases (adding more terms). The main question (which can be addressed only qualitatively at this point) is this: is the gain in R^2 big enough to warrant the new term, knowing that a new term can lead to over-fitting. In this example, the gain from $R^2 = 0.3629$ to $R^2 = 0.3800$ is probably NOT worth the risk of overfitting. So, we should keep the simpler model. That's called the principle of "Occam's Razor," which posits that one should go with simpler things.

```
anova(lm.6)

Analysis of Variance Table

Response: y
           Df  Sum Sq Mean Sq F value        Pr(>F)
x_1         1  603829  603829  121.05       < 2e-16 ***
x_2         1  202064  202064   40.51 0.00000000081 ***
I(x_1^2)    1   13086   13086    2.62         0.106
I(x_2^2)    1   24838   24838    4.98         0.026 *
x_1:x_2     1     131     131    0.03         0.872
Residuals 276 1376791    4988
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the anova() output, there is an SS term for **each term** in the regression equation, followed by an SS term for "Residuals." The latter is what we have been calling SSE (or $SS_{unexplained}$); so, where is $SS_{explained}$? It can be obtained by adding the other SS terms. The reason R produces separate terms for each term is because R performs what is called **sequential analysis of variance**, which is a bit different from what we do. In fact, these SS terms will change depending on the order of the terms in the regression equation!

It's important to realize that all of these SS terms are measures of variability. Specifically, SST is the numerator of the sample variance of the y's, $SS_{explained}$ is the numerator of the sample variance of the **predictions**, and $SS_{unexplained}$ (SSE) is converted to variance when it's divided by $n-(k+1)$, where k is the number of parameters in the regression model. You can confirm these:

```
y_hat <- predict(lm.6) # From 9.3.
n <- nrow(dat)
(n - 1) * var(y_hat) # 843948 = 603829 + 202064 + 13086 + 24838 + 131 = SS_explained

[1] 843948
```

3.8.1 Prediction on New Data

The best way to do prediction on new data is to just attach the new data to the bottom of the old data. Suppose the new data consists of the following 2 case:

$$x_1 = 33,\ x_2 = 8$$
$$x_1 = 35,\ x_2 = 14$$

Then we can do the following:

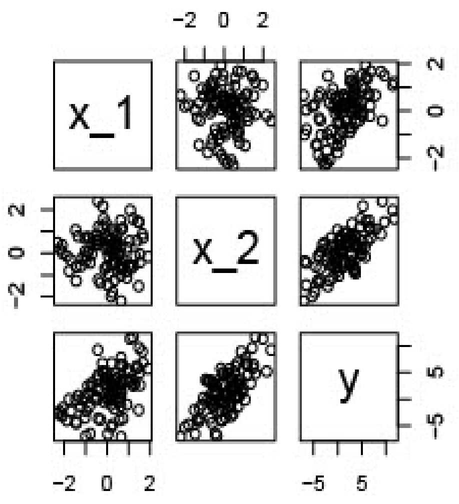
```
n <- nrow(dat)  # number of cases in dat.
new_1 = c(33, 8, NA)  # y = NA because we don't know y for new data.
new_2 = c(35, 14, NA)
new.dat = rbind(dat, new_1, new_2) # Using row-bind to attach new data to old data.

# In the next line, we redevelop lm.4, but on the first n cases:
lm.7 <- lm(y ~ x_1 + x_2 + x_1:x_2, dat = new.dat[1:n, ])  # NOTE: dat=new.dat[1:n,].
summary(lm.7)  # Same as lm.4.
```

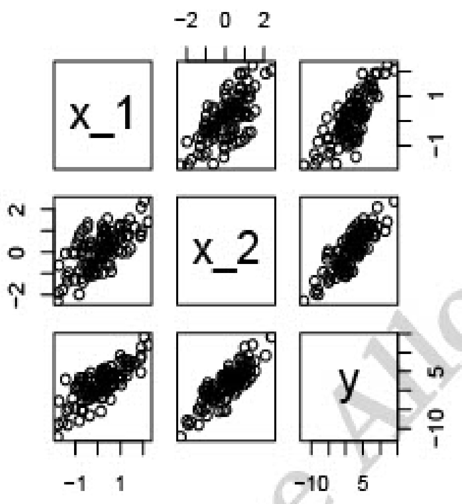
```
Call:
lm(formula = y ~ x_1 + x_2 + x_1:x_2, data = new.dat[1:n, ])

Residuals:
   Min     1Q Median     3Q    Max
-157.1  -50.8  -14.2   36.6  264.0

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  71.6091    34.7693    2.06    0.040 *
x_1           0.1989     0.9217    0.22    0.829
x_2           1.0612     2.0269    0.52    0.601
x_1:x_2       0.1030     0.0453    2.27    0.024 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.7 on 278 degrees of freedom
Multiple R-squared:  0.375,Adjusted R-squared:  0.368
F-statistic: 55.5 on 3 and 278 DF,  p-value: <2e-16

colnames(new.dat) <- c("x_1", "x_2", "x_1:x_2")
predict(lm.7, newdata <- new.dat[(n+1):(n+2), ])  # Predict the last 2 cases.

  283   284
113.8 143.9
```

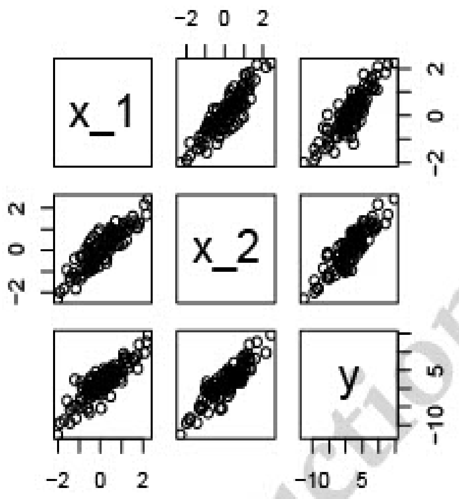
3.9 Collinearity

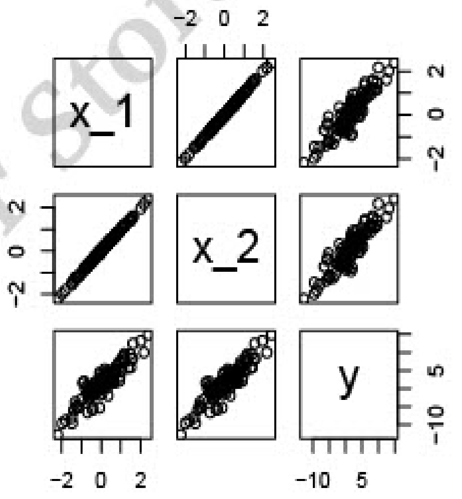
Another distressing issue that arises in **multiple** regression is collinearity, i.e., a linear association between the predictors themselves. One reason collinearity is distressing is that it renders the regression coefficients uninterpretable; i.e., a given beta can no longer be interpreted as the average rate of change of y with respect to a unit change in x with everything else held fixed. Insisting on that kind of interpretation, in the presence of collinearity, can lead to wrong (or even absurd) conclusions. Collinearity also makes the predictions more uncertain, but here we will focus on the effect of collinearity on the regression coefficients.

```
# To that end, we'll write an R function, which is nothing but some lines
# of code intended to be used over and over again.
make.fit <- function(r) {
# The function first makes data on x_1, x_2, and y, with collinearity
# (i.e., correlation between x_1 and x_2) equal to r.
# The input of the function is r (i.e., correlation between x_1 and x_2.
# NOT between y and anything).
# The function then fits that data using y, and returns some stats about
# the estimated regression coefficients.
  library(MASS)  # This library contains mvrnorm(); see below.
  set.seed(1)  # Ensures reproducible outputs.
  n <- 100
  # The R function mvrnorm() below takes a sample from a multivariate normal,
  # which is a higher-dimensional analog of the normal distribution.
  dat <-  mvrnorm(n, rep(0, 2), matrix(c(1, r, r, 1), 2, 2))
  x_1 <- dat[, 1]
  x_2 <- dat[, 2]
  y <- 1 + 2 * x_1 + 3 * x_2 + rnorm(n, 0,2)  # Generate y, and add noise.
  dat <- data.frame(x_1, x_2, y)  # Here is the whole data.
  plot(dat)
  lm.1 <- lm( y ~ x_1 + x_2)  # Fit a plane through the data.
  # return(lm.1) returns the whole R object lm.1.
  # return(summary(lm.1)) returns only the summary results.
  return(summary(lm.1)$coeff)  # Returns only the regression coefficients.
  }

# Examining data and the regression coefficients for different amounts of
# collinearity.
make.fit(0)  # No collinearity.

            Estimate Std. Error t value  Pr(>|t|)
(Intercept)    1.051     0.2104   4.994 2.609e-06
x_1            2.107     0.2190   9.623 8.760e-16
x_2            3.042     0.2335  13.028 4.968e-23

make.fit(0.7)  # Some collinearity.

            Estimate Std. Error t value  Pr(>|t|)
(Intercept)    1.051     0.2104   4.994 2.609e-06
x_1            2.161     0.3096   6.979 3.684e-10
x_2            2.885     0.3099   9.310 4.141e-15

make.fit(0.9)   # Extreme collinearity.

            Estimate Std. Error t value     Pr(>|t|)
(Intercept)    1.051     0.2104   4.994 0.0000026091
x_1            2.261     0.5039   4.486 0.0000199246
x_2            2.783     0.5042   5.519 0.0000002837

make.fit(0.999)

            Estimate Std. Error t value    Pr(>|t|)
(Intercept)    1.051     0.2104  4.9941 0.000002609
x_1            4.412     4.8973  0.9009 0.369846805
x_2            0.630     4.8975  0.1286 0.897910878
```



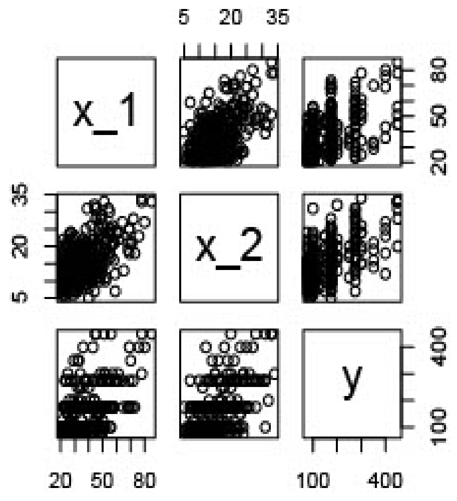






When collinearity is extreme, not only are the standard errors huge, but the estimated regression coefficients themselves (β) are way off. As collinearity increases, the regression coefficients become more uncertain, and so we are unable to interpret them, like we would if there were no collinearity. The regression equation is still OK to use for predictions. But, of course, the predictions will be less certain as well. Note that in practice we don't control/adjust the data or the collinearity; all we see are the scatterplots, and based on the scatterplots between the predictors, we decide how much collinearity there is. For example, for the hail data:

```
plot(dat)
```

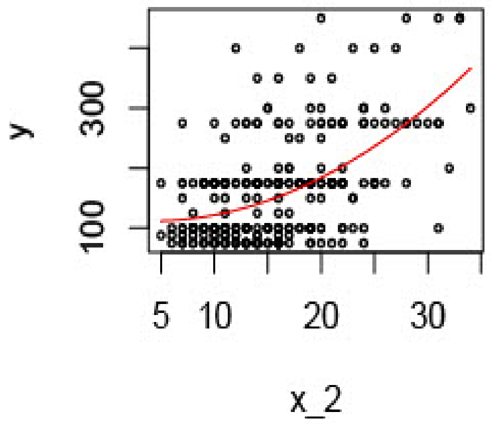


In the scatterplots, the one for x_1 versus x_2 (or vice versa) suggests some collinearity; it is not extreme, but it is present. As such, we have to be cautious in interpreting the regression coefficients. But the regression model is still okay for making predictions (as long as it does not overfit, of course).

3.10 Plotting Curved Fits on a Scatterplot

We plotted polynomial fits, but the "curves" were just the result of connecting points with straight lines, and as a result, the "curves" did not look smooth. Here is a way to get a smoother looking fit on the scatterplot.

```
# Suppose we pick a relatively simple quadratic model for the hail data:
dat <- read.table("hail_dat.txt", header = T)

x_1 <- dat[, 1] # Divergence.
x_2 <- dat[, 2] # Rotate.
y <- dat[, 3] # Hail size. Size is in 100th-of-an-inch.
lm.g <- lm(y ~ x_2 + I(x_2 ^ 2))
lm.g$coef  # Examine the regression coeffs.

(Intercept)         x_2    I(x_2^2)
   116.3519     -2.2682      0.2827

x <- seq(min(x_2), max(x_2), .01)  # Generate a fake x.
y.fit <- lm.g$coeff[1] + lm.g$coeff[2] * x + lm.g$coeff[3] * x^2
plot(x_2, y, cex = 0.5)
points(x , y.fit, col = "red", type = "l")
```



```
# Alternatively, a fancier way is as follows.
x <- matrix(seq(min(x_2), max(x_2), .01), byrow = T)  # Generate a fake x .
colnames(x) <- "x_2"
plot(x_2, y)
lines(x, predict(lm.g, newdata = data.frame(x)), col=2)
```