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.

```r
anova(lm.6)
```

Analysis of Variance Table

<table>
<thead>
<tr>
<th>Response</th>
<th>Df</th>
<th>Sum Sq</th>
<th>Mean Sq</th>
<th>F value</th>
<th>Pr(&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>1</td>
<td>603829</td>
<td>603829</td>
<td>121.05</td>
<td>&lt; 2e-16</td>
</tr>
<tr>
<td>x_2</td>
<td>1</td>
<td>202064</td>
<td>202064</td>
<td>40.51</td>
<td>0.0000000000081</td>
</tr>
<tr>
<td>I(x_1^2)</td>
<td>1</td>
<td>13086</td>
<td>13086</td>
<td>2.62</td>
<td>0.106</td>
</tr>
<tr>
<td>I(x_2^2)</td>
<td>1</td>
<td>24838</td>
<td>24838</td>
<td>4.98</td>
<td>0.026</td>
</tr>
<tr>
<td>x_1:x_2</td>
<td>1</td>
<td>131</td>
<td>131</td>
<td>0.03</td>
<td>0.872</td>
</tr>
<tr>
<td>Residuals</td>
<td>276</td>
<td>1376791</td>
<td>4988</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

In the `anova()` output, each term in the regression equation is accompanied by an SS term. They are obtained from a sequential analysis of variance. 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, $SS_{T}$ 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:

```r
y_hat <- predict(lm.6)  # From 9.3.
n <- length(y)
(n - 1) * var(predict(lm.6))
```

[1] 297334

**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:

- $x_1 = 33, x_2 = 9$
- $x_1 = 36, x_2 = 14$

Then we can do the following:

```r
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:
\texttt{lm(formula = y \sim x\_1 + x\_2 + x\_1:x\_2, data = new.dat[1:n, ])}

Residuals:
\begin{tabular}{rrrr}
Min & 1Q & Median & 3Q & Max \\
-157.1 & -50.8 & -14.2 & 36.6 & 264.0 \\
\end{tabular}

Coefficients:
\begin{itemize}
\item \texttt{(Intercept)} & 71.6091 & 34.7693 & 2.06 & 0.040 & * \\
\item \texttt{x\_1} & 0.1989 & 0.9217 & 0.22 & 0.829 \\
\item \texttt{x\_2} & 1.0612 & 2.0269 & 0.52 & 0.601 \\
\item \texttt{x\_1:x\_2} & 0.1030 & 0.0453 & 2.27 & 0.024 & * \\
\end{itemize}

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

\texttt{predict(lm.7, newdata <- new.dat[(n+1):(n+2), ])} # Predict the last 2 cases.

\begin{itemize}
\item 283 \\
\item 284 \\
\item 113.8 \\
\item 143.9 \\
\end{itemize}

### 10.3 Collinearity

Another distressing issue that arises in \textbf{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.

\begin{verbatim}
# 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))
\end{verbatim}
\[ x_1 \leftarrow \text{dat}[1, 1] \]
\[ x_2 \leftarrow \text{dat}[1, 2] \]
\[ y \leftarrow 1 + 2 \times x_1 + 3 \times x_2 + \text{rnorm}(n, 0, 2) \] # Generate y, and add noise.
\[ \text{dat} \leftarrow \text{data.frame}(x_1, x_2, y) \] # Here is the whole data.
\[ \text{plot(dat)} \]
\[ \text{lm.1} \leftarrow \text{lm}(y \sim x_1 + x_2) \] # Fit a plane through the data.
\[ \text{return(lm.1)} \] returns the whole R object lm.1.
\[ \text{return(summary(lm.1))} \] returns only the summary results.
\[ \text{return(summary(lm.1)$coeff)} \] # Returns only the regression coefficients.
\[
\#
\text{Examining data and the regression coefficients for different amounts of}
\text{collinearity.}
\text{make.fit(0)} \] # 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 |

\[ \text{make.fit(0.7)} \] # Some collinearity.

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

\[ \text{make.fit(0.9)} \] # Extreme collinearity.

| Estimate  | Std. Error | t value | Pr(>|t|) |
|-----------|------------|---------|----------|
| (Intercept) | 1.051 | 0.2104 | 4.994 | 0.00002609 |
| $x_1$      | 2.261 | 0.5039 | 4.486 | 0.000199246 |
| $x_2$      | 2.783 | 0.5042 | 5.519 | 0.000002837 |

\[ \text{make.fit(0.999)} \]

| Estimate  | Std. Error | t value | Pr(>|t|) |
|-----------|------------|---------|----------|
| (Intercept) | 1.051 | 0.2104 | 4.9941 | 0.00002609 |
| $x_1$      | 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 ($\beta$) 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)
```
10.4 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.

```r
# Suppose we pick a relatively simple quadratic model for the hail data:
dat <- read.table("http://www.stat.washington.edu/marzban/390/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$coef[1] + lm.g$coef[2] * x + lm.g$coef[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)
4 Sampling Distributions of the Sample Mean and Median

4.1 Sampling Distribution of the Mean

4.1.1 Normal Population

Instead of taking samples from a normal population, using `rnorm()`, we are going to take ONE huge sample from a normal population, using `rnorm()`, and then just treat it as our population. The main reason for this is mostly to set the stage for something called “bootstrapping,” which we will study later.

```r
N <- 100000  # Let N be the population size.
pop <- rnorm(N, 1, 2)  # Take a random sample and treat it as pop.
pop.mean <- mean(pop)  # This is mu, the population mean.
pop.sd <- sd(pop)  # This is sigma, the pop standard deviation.
pop.median <- median(pop)  # Get the population median, for later.
c(p.pop.mean, pop.sd, pop.median)  # Print them for comparison, below.

[1] 1.002  2.000  1.002
```

```r
hist(pop, breaks = 400)  # This shows that the population is pretty normal.
```

# Experiment underlying the sampling distribution.
n.trial <- 10000  # Take 10000 samples of
sample.size <- 10  # size 10 (i.e., small) from the population.
sample.stat <- numeric(n.trial)  # Create space to store the 10000 sample means.

```r
for (i in 1:n.trial) {
  samp <- sample(pop, sample.size, replace = T)  # Take a sample (with replacement).
  sample.stat[i] <- mean(samp)  # Compute each sample’s mean.
}
```

```r
mean(sample.stat)  # Compare mean of sample means

[1] 1.003
```

```r
pop.mean  # with the population mean.

[1] 1.002
```

```r
sd(sample.stat)  # Compare the standard deviation of sample means

[1] 0.6285
```

```r
pop.sd  # with the pop standard deviation.

[1] 2
```

```r
pop.sd / sqrt(sample.size)  # But compare with (pop std dev)/sqrt(n)

[1] 0.6324
```
According to the Central Limit Theorem (CLT), the sampling distribution of the sample mean should be normal. To confirm:

```r
hist(sample.stat, breaks = 40)
qqnorm(sample.stat)  # Pretty normal.
```

As the sample size increases, the mean of the sample means gets pretty close to the population mean, and the standard deviation of the sample means gets pretty close to the \( \frac{\sigma_{\text{pop}}}{\sqrt{n}} \). So, the CLT is confirmed.

4.1.2 Non-normal Population

```r
N <- 100000
pop <- rgamma(N, 1, 1)
```
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)

[1] 1.0036 1.0005 0.6957

hist(pop, breaks = 400) # The distribution of sample means looks non-normal.

n.trial <- 10000 # Take 10000 samples of
sample.size <- 10 # size 10 (i.e., small) from the population.
sample.stat <- numeric(n.trial) # Space for storing the 10000 sample means.

for (i in 1:n.trial) {
    samp <- sample(pop, sample.size, replace=T) # Take a sample (with replacement).
    sample.stat[i] <- mean(samp) # and compute each sample's mean.
}

mean(sample.stat) # Compare mean of sample means with population mean.

[1] 1.003

pop.mean

[1] 1.004

sd(sample.stat) # Compare the sd of sample means with population sd.

[1] 0.3129

pop.sd

[1] 1

pop.sd / sqrt(sample.size) # Compare with (pop sd)/root(n).

[1] 0.3164

hist(sample.stat, breaks = 40)
qqnorm(sample.stat, cex = 0.5)
When the population is NOT normal, for small samples (10) the sampling distribution of the sample mean resists looking normal; but with larger samples (100), it is normal even though the population is not normal.

### 4.2 Sampling Distribution of Median

#### 4.2.1 Non-normal Population

```r
N <- 100000
pop <- rgamma(N, 1, 1)
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)
```
hist(pop, breaks = 400)

n.trial <- 10000  # Take 10000 samples of
sample.size <- 10  # size 10 (i.e., small) from the population.
sample.stat <- numeric(n.trial)  # Space for storing the 10000 sample medians.

for (i in 1:n.trial) {
  samp <- sample(pop, sample.size, replace = T)  # Take a sample (with replacement).
  sample.stat[i] <- median(samp)  # Compute each sample's MEDIAN.
}

mean(sample.stat)  # Compare the MEAN of sample MEDIANS with pop MEDIAN.

[1] 0.7512

pop.median

[1] 0.6973

sd(sample.stat)  # Compare the sd of sample MEDIANS with population sd.

[1] 0.3105

pop.sd

[1] 1.008

# Note that the formula sigma/root(n) applies only to the sample MEAN.

hist(sample.stat, breaks = 40)
qqnorm(sample.stat, cex = 0.5)
The sampling distribution doesn’t look too normal. But if the sample size is relatively large, the distribution of a bunch of sample medians, taken from even a non-normal population, is still normal. Most statistics (e.g., sample mean, sample median, sample standard deviation, ...) ultimately end up having a normal distribution, but some require a larger sample size.