R Users Guide to Stat 201: Chapter 10

Michael Shyne, 2017

Chapter 10: Correlation and Regression

Chapter 10 covers the relationship between two paired numeric variables. Correlation is a measure of the strength of the linear association between the variables. Regression gives us a line of "best fit", the line which best describes the relationship between the variables.

Correlation

In R, the function to calculate the correlation coefficient r is `cor()`. Simply give the function two numeric vectors of equal length. We will use the `faithful` data set. It contains a sample of eruptions of the Old Faithful geyser in Yellowstone National Park. The length of the eruption is in the variable `eruptions` and `waiting` gives the time until the next eruption.

```
# Correlation coefficient of eruption and waiting times
cor(faithful$eruptions, faithful$waiting)
```

```
## [1] 0.9008112
```

We can also just pass the whole data frame into `cor()`. The output will be a correlation matrix, a table of pairwise correlations between every numeric variable in the data frame.

```
# Correlation matrix
cor(faithful)
```

```
##           eruptions   waiting
## eruptions 1.0000000 0.9008112
## waiting   0.9008112 1.0000000
```

You notice some features of this table. The diagonal will always and only contain 1's, since a variable is perfectly correlated with itself. Also, because correlation is a symmetric relation, the matrix is symmetric around the diagonal. While this example is not very interesting, a correlation matrix can be an effective tool in quickly finding correlated variables in larger data sets.

Remember, the vectors passed to the `cor()` function don't have to be in the same data set.

```
# Generate random value vectors
set.seed(23)
x <- runif(20, 1, 10)
y <- runif(20, 3, 8)

cor(x,y)
```

```
## [1] 0.2706635
```

Correlation test

An hypothesis test of the population correlation coefficient ρ is conducted using the `cor.test()` function. This function works similarly to the other hypothesis test functions we saw in chapters 8 and 9. However, note that `cor.test()` takes two vectors as parameters. We can not simply supply a data frame like we did with `cor()`.

```r
# Is populaiton correlation zero?
cor.test(faithful$eruptions, faithful$waiting)
```

```
##
##  Pearson's product-moment correlation
##
## data:  faithful$eruptions and faithful$waiting
## t = 34.089, df = 270, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8756964 0.9210652
## sample estimates:
##       cor
## 0.9008112
```

Regression

To calculate a regression line in R we need to define a linear model using R's formula notation. Recall, regression is not symmetric like correlation. We need to designate a response variable and a predictor variable. Suppose, as is traditional, y is our response variable and x is our predictor. Then, we can define a formula representing this relationship by

$$y \sim x.$$

The separator is a tilde (~). It is located on the upper left of your keyboard, next to the "1".

Now, we can create a linear model by passing the formula to the `lm()` function.

```r
# Linear model with x and y (previously defined)
xy.lm <- lm(y ~ x)
```

Of course, we can define formulas and linear models using data frame columns as well. If we include the `data=` parameter, the `lm()` function will assume the variables in the formula are columns of the data frame designated.

```r
# "Predict" waiting times by eruption length
faith.lm <- lm(faithful$waiting ~ faithful$eruptions)

# This is functionally identical to above, but is aesthetically cleaner
faith.lm2 <- lm(waiting ~ eruptions, data=faithful)
```

In defining the linear model, we have performed our regression. We can see the results by looking at the contents of the linear model variable.

```r
faith.lm
```

```
##
## Call:
## lm(formula = faithful$waiting ~ faithful$eruptions)
##
## Coefficients:
##        (Intercept)  faithful$eruptions
##              33.47               10.73
```

We have the coefficients (intercept and slope) of the regression line. However, there is much more to learn about a regression. The `summary()` function will give us a much more useful output.

```
summary(faith.lm)
```

```
##
## Call:
## lm(formula = faithful$waiting ~ faithful$eruptions)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.0796  -4.4831   0.2122   3.9246  15.9719
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         33.4744     1.1549   28.98   <2e-16 ***
## faithful$eruptions  10.7296     0.3148   34.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.914 on 270 degrees of freedom
## Multiple R-squared:  0.8115, Adjusted R-squared:  0.8108
## F-statistic:  1162 on 1 and 270 DF,  p-value: < 2.2e-16
```

We now have perhaps an excess of information. Much of what is displayed in the summary output is beyond the scope of this course. In is helpful to notice the expanded information on the coefficients. We are given the t statistic and p-values for hypothesis tests of the alternative hypotheses that the coefficients are not equal to zero. The p-value for the slope is the same p-value from the correlation test between the variables. We are also given the coefficient of determination R^2 (`Multiple R-squared`). Ignore the adjusted R^2. It is more relevant to regressions with multiple predictors.

Like with the hypothesis test functions, we can store the results of the `summary()` function to a variable. I encourage you to look at the structure (`str()`) of such a variable. I won't do it here for space reasons, but the summary object contains a wealth of information that will be increasingly useful as you progress in your statistics education.

```
# Store summary results in variable.
xy.sum <- summary(xy.lm)
xy.sum                      # Notice that slope is not sig. different from zero.
```

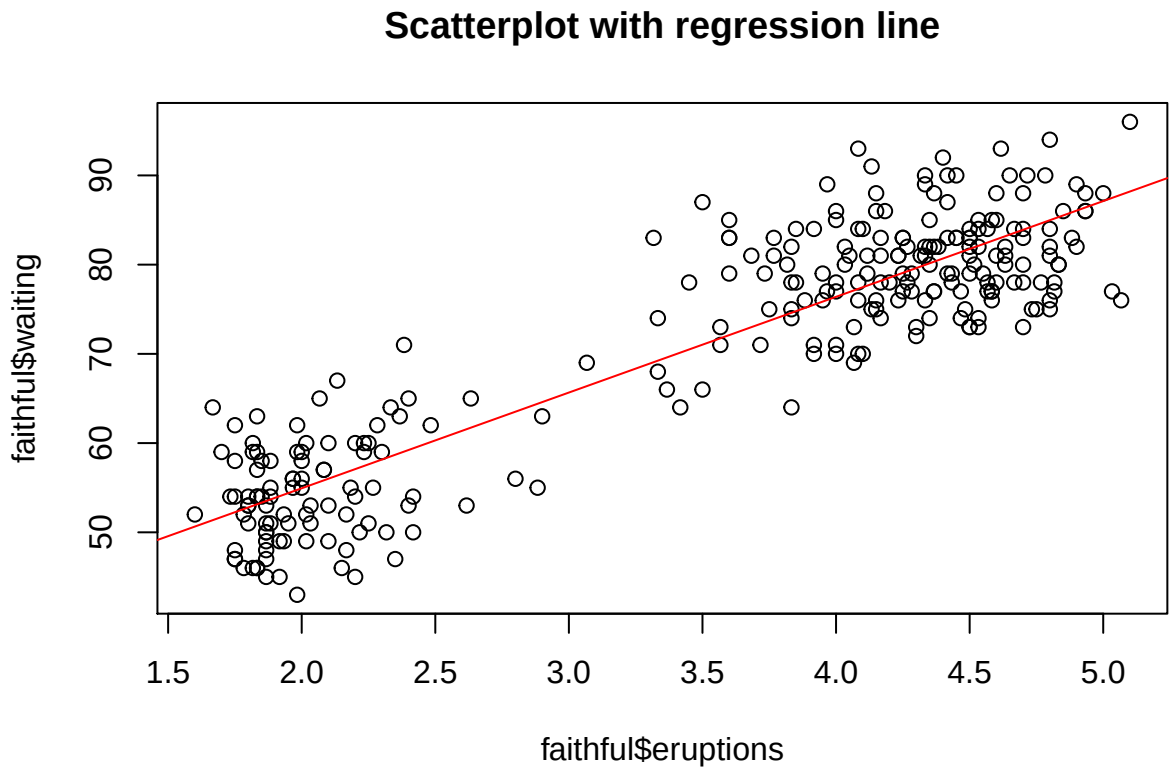
```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.3424 -1.1148 -0.1164  1.2982  2.1560
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.5307     0.9610   4.715 0.000173 ***
## x             0.1601     0.1342   1.193 0.248414
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.413 on 18 degrees of freedom
## Multiple R-squared:  0.07326,    Adjusted R-squared:  0.02177
## F-statistic: 1.423 on 1 and 18 DF,  p-value: 0.2484
```

Plots

It is, of course, often useful to examine graphs of data and regressions. Recall, to produce a scatterplot of two numeric variables, simply pass the to the `plot()` function. Remember that the `x` variable, the predictor in a regression, comes first in the plot command. This is the opposite of the regression formula. To add a regression line, call `abline()` with the linear model object. Note: `abline()` must follow a `plot()` command. It can not be called alone.

```
# Draw scatterplot
plot(faithful$eruptions, faithful$waiting,
     main='Scatterplot with regression line')

# Add regression line in red
abline(faith.lm, col="red")
```



Prediction

One reason to do regression is to be able to predict values of the response variable for given values of the predictor. While this can be done by hand by plugging value into the regression line equation, R provides the handy, if a little complicated to use, function. The `predict()` function takes a linear model object and a data frame with a column that has exactly the same name as the linear model's predictor variable. The data frame should contain the value or values to use for the prediction.

```
# Predict y for x=6
predict(xy.lm, data.frame(x=c(6)))
```

```
##       1
## 5.49135
```

When working with linear models defined by columns from data frames, the requirement that the data frame sent to the `predict()` function presents an additional consideration. When formulas containing vectors in the `data frame$column` format, it is impossible to create a column that has that name (because the `$` has a special meaning, it can't be used in variable names). Thus, linear models defined in that way, like `faith.lm` was defined above, can not be used for predictions. The alternative form with the `data=` parameter, like `faith.lm2` must be used instead.

```
# We can make multiple predictions at once by providing multiple predictor values
faith.pred <- data.frame(eruptions=c(2, 3, 4))

predict(faith.lm2, faith.pred)
```

```
##        1        2        3
## 54.93368 65.66332 76.39296
```

Though it is beyond the scope of the class, notice that we can get prediction or confidence intervals by including the optional `interval=` parameter to the function call.

```
# Predict with 99% prediction interval
predict(faith.lm2, faith.pred, interval="predict", level=.99)
```

```
##        fit      lwr      upr
## 1 54.93368 39.51568 70.35168
## 2 65.66332 50.28810 81.03855
## 3 76.39296 61.01721 91.76872
```

License

This document is distributed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.