Introduction to Linear Regression in R

Miratrix

2019-09-05

This handout shows how to fit simple linear regression models in R. Linear regression is the main way researchers tend to examine the relationships between multiple variables. This document runs through some code without too much discussion, with the assumption that you are already familiar with interpretation of such models.

We begin with an example of doing a simple two-way linear regression on some data we borrow from a textbook's data package:

```
library( Lock5Data )
data( HappyPlanetIndex )
head( HappyPlanetIndex )
```

```
##     Country Region Happiness LifeExpectancy Footprint  HLY  HPI HPIRank
## 1   Albania      7       5.5           76.2       2.2 41.7 47.9      54
## 2   Algeria      3       5.6           71.7       1.7 40.1 51.2      40
## 3    Angola      4       4.3           41.7       0.9 17.8 26.8     130
## 4 Argentina      1       7.1           74.8       2.5 53.4 59.0      15
## 5   Armenia      7       5.0           71.7       1.4 36.1 48.3      48
## 6 Australia      2       7.9           80.9       7.8 63.7 36.6     102
##   GDPperCapita   HDI Population
## 1         5316 0.801       3.15
## 2         7062 0.733      32.85
## 3         2335 0.446      16.10
## 4        14280 0.869      38.75
## 5         4945 0.775       3.02
## 6        31794 0.962      20.40
```

We see some statistics for various countries. Type `?HappyPlanetIndex` to learn more.

To install this package, type `install.packages( "Lock5Data" )` once; once a package is installed on your computer it will stay there until you remove it or upgrade R.

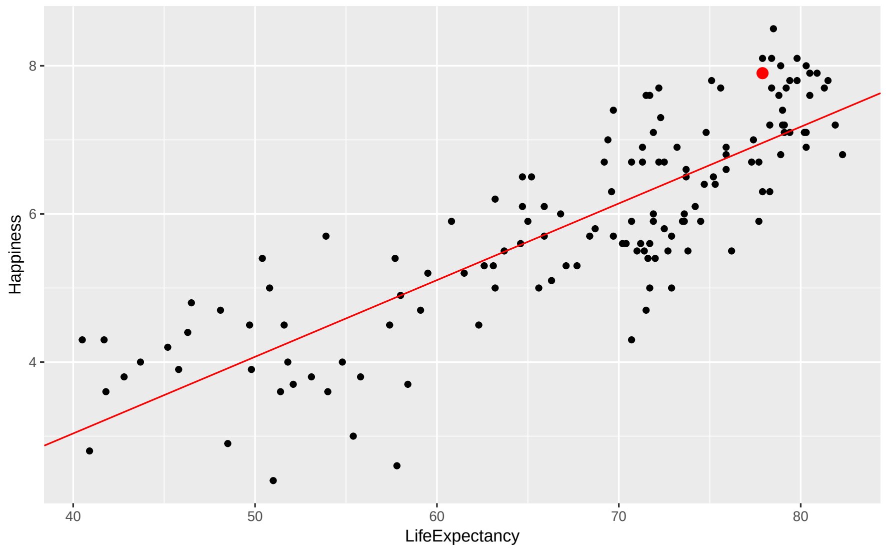
To fit a regression simply type:

```
my.lm = lm( Happiness ~ LifeExpectancy, data=HappyPlanetIndex )
my.lm
```

```
##
## Call:
## lm(formula = Happiness ~ LifeExpectancy, data = HappyPlanetIndex)
##
## Coefficients:
##    (Intercept)  LifeExpectancy
##         -1.104           0.104
```

We can plot it on top of a scatterplot. (For fun, we get fancy and make the US marked in a big red point):

```
usa = filter( HappyPlanetIndex, Country == "United States of America" )
ggplot( HappyPlanetIndex, aes( LifeExpectancy, Happiness ) ) +
  geom_point() +
  geom_point( data=usa, col="red", size=3 ) +
  geom_abline( intercept = -1.1037, slope =  0.1035, col="red" )
```



The US at this time had more happiness than we would have expected given its life expectancy.

You can pull parts of your model out with the `coef()` command:

```
coef(my.lm)
```

```
##    (Intercept) LifeExpectancy
##         -1.104          0.104
```

```
coef(my.lm)[1] # intercept
```

```
## (Intercept)
##        -1.1
```

```
coef(my.lm)[2] # slope
```

```
## LifeExpectancy
##          0.104
```

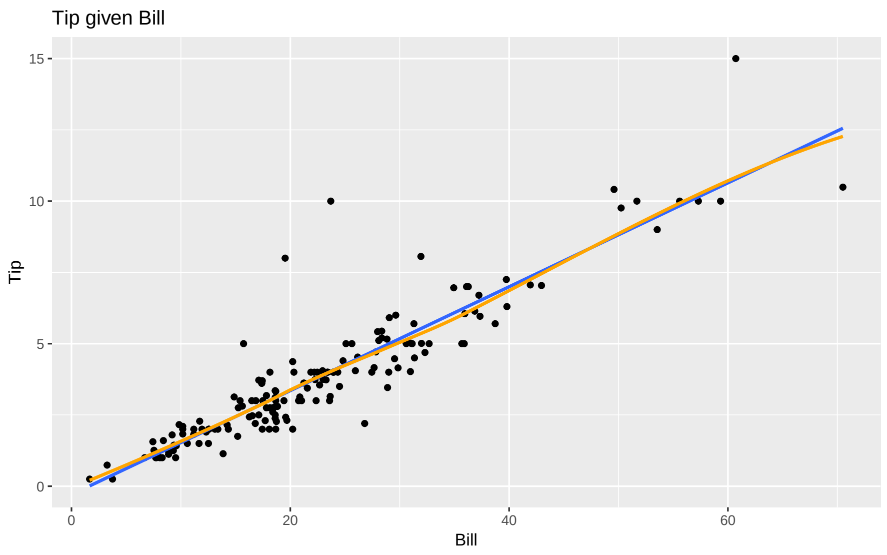
```
coef(my.lm)["LifeExpectancy"] # alternate way.
```

```
## LifeExpectancy
##          0.104
```

Regression with Multiple Variables

We here present an example on the `RestaurantTips` dataset. Let's first regress Tip on Bill. Before doing regression, we should plot the data to make sure using simple linear regression is reasonable. For kicks, we add in the regression line as well, taking advantage of ggplot's `geom_smooth()` method:

```
data(RestaurantTips)
qplot( Bill, Tip, data=RestaurantTips, main="Tip given Bill") +
  geom_smooth( method="lm", se=FALSE ) +
  geom_smooth( method="loess", se=FALSE, col="orange" )
```



```r
mod = lm(Tip ~ Bill, data=RestaurantTips)
summary(mod)
```

```
##
## Call:
## lm(formula = Tip ~ Bill, data = RestaurantTips)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.391 -0.489 -0.111  0.284  5.974
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.29227    0.16616   -1.76    0.081 .
## Bill         0.18221    0.00645   28.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.98 on 155 degrees of freedom
## Multiple R-squared:  0.837,  Adjusted R-squared:  0.836
## F-statistic:  798 on 1 and 155 DF,  p-value: <2e-16
```

Looking at the plot, the trend appears to be approximately linear. There are a few unusually large tips, but no extreme outliers, and variability appears to be constant as Bill increases, so we proceed. The simple linear regression model is now saved under the name mod (short for model - you can call it anything you want). Once we fit the model, we used summary to see the output.

Results relevant to the intercept are in the (Intercept) row and results relevant to the slope are in the Bill (the explanatory variable) row. The estimate column gives the estimated coefficients, the std. error column gives the standard error for these estimates, the t value is simply estimate/SE, and the p-value is the result of a hypothesis test testing whether that coefficient is significantly different from 0.

We also see the standard error of the error as “Residual standard error” and R2 as “Multiple R-squared”. The last line of the regression output gives details relevant to an ANOVA table for testing our model against no model. It has the F-statistic, degrees of freedom, and p-value.

We now include the additional explanatory variables of number in party (Guests) and whether or not they pay with a credit card:

```
tip.mod = lm(Tip ~ Bill + Guests + Credit, data=RestaurantTips )
summary(tip.mod)
```

```
##
## Call:
## lm(formula = Tip ~ Bill + Guests + Credit, data = RestaurantTips)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.384 -0.478 -0.108  0.272  5.984
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.25468    0.20273   -1.26     0.21
## Bill         0.18302    0.00846   21.64   <2e-16 ***
## Guests      -0.03319    0.10282   -0.32     0.75
## Credity      0.04217    0.18282    0.23     0.82
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.985 on 153 degrees of freedom
## Multiple R-squared:  0.838,  Adjusted R-squared:  0.834
## F-statistic:  263 on 3 and 153 DF,  p-value: <2e-16
```

This output should look very similar to the output for one variable, except now there is a row corresponding to each explanatory variable.
Our two-category (y, n) `Credit` variable was automatically converted to a 0-1 dummy variable (with "y" being 1 and "n" our baseline).

Categorical Variables (and Factors)

You can include any explanatory categorical variable in a multiple regression model, and R will automatically create corresponding 0/1 variables. For example, if you were to include gender coded as male/female, R would create a variable GenderMale that is 1 for males and 0 for females.

Numbers Coding Categories.

If you have multiple levels of a category, but your levels are coded with numbers you have to be a bit careful because R can treat this as a quantitative (continuous) variable by mistake in some cases. You will know it did this if you only see the single variable on one line of your output. For categorical variables with k categories, you should see $k-1$ lines.

To make a variable categorical, even if the levels are numbers, convert the variable to a factor with `as.factor()`:

```
data( USStates )
USStates$Region = as.factor( USStates$Region )
```

Setting new baselines.

We can reorder the levels if desired (the first is our baseline).

```
levels( USStates$Region )
```

```
## [1] "MW" "NE" "S"  "W"
```

```
USStates$Region = relevel(USStates$Region, "S" )
levels( USStates$Region )
```

```
## [1] "S"  "MW" "NE" "W"
```

Now any regression will use the south as baseline.

Missing levels in a factor

R often treats categorical variables as factors. This is often useful, but sometimes annoying. A factor has different *levels* which are the different values it can be. For example:

```
data(FishGills3)
levels(FishGills3$Calcium)
```

```
## [1] ""       "High"   "Low"    "Medium"
```

```
table(FishGills3$Calcium)
```

```
##
##          High    Low Medium
##      0     30     30     30
```

Note the weird nameless level; it also has no actual observations in it. Nevertheless, if you make a boxplot, you will get an empty plot in addition to the other three. This error was likely due to some past data entry issue. You can drop the unused level:

```
FishGills3$Calcium = droplevels(FishGills3$Calcium)
```

You can also turn a categorical variable into a numeric one like so:

```
summary( FishGills3$Calcium )
```

```
##   High    Low Medium
##     30     30     30
```

```
asnum = as.numeric( FishGills3$Calcium )
asnum
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
## [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1
## [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Regression on only a categorical variable is fine:

```
mylm = lm( GillRate ~ Calcium, data=FishGills3 )
mylm
```

```
##
## Call:
## lm(formula = GillRate ~ Calcium, data = FishGills3)
##
## Coefficients:
##   (Intercept)     CalciumLow  CalciumMedium
##          58.2           10.3            0.5
```

R has made you a bunch of dummy variables automatically. Here "high" is the baseline, selected automatically. We can also force it so there is no baseline by removing the intercept:

```
mymm = lm( GillRate ~ 0 + Calcium, data=FishGills3 )
mymm
```

```
##
## Call:
## lm(formula = GillRate ~ 0 + Calcium, data = FishGills3)
##
## Coefficients:
##   CalciumHigh     CalciumLow  CalciumMedium
##          58.2           68.5           58.7
```

Some extra stuff (optional)

Removing Outliers

If you can identify which rows the outliers are on, you can do this by hand (say the rows are 5, 10, 12).

```
new.data = old.data[ -c(5,10,12), ]
lm( Y ~ X, data=new.data )
```

Some technical details: The `c(5,10,12)` is a list of 3 numbers. The `c()` is the concatenation function that takes things makes lists out of them. The "-list" notation means give me my old data, but without rows 5, 10, and 12. Note the comma after the list. This is because we identify elements in a dataframe with row, column notation. So `old.data[1,3]` would be row 1, column 3.

If you notice your points all have X bigger than some value, say 20.5, you could use filtering to keep everything less than some value:

```
new.data = filter( old.data, X <= 20.5 )
```

Missing data

If you have missing data, `lm` will automatically drop those cases because it doesn't know what else to do. It will tell you this, however, with the `summary` command.

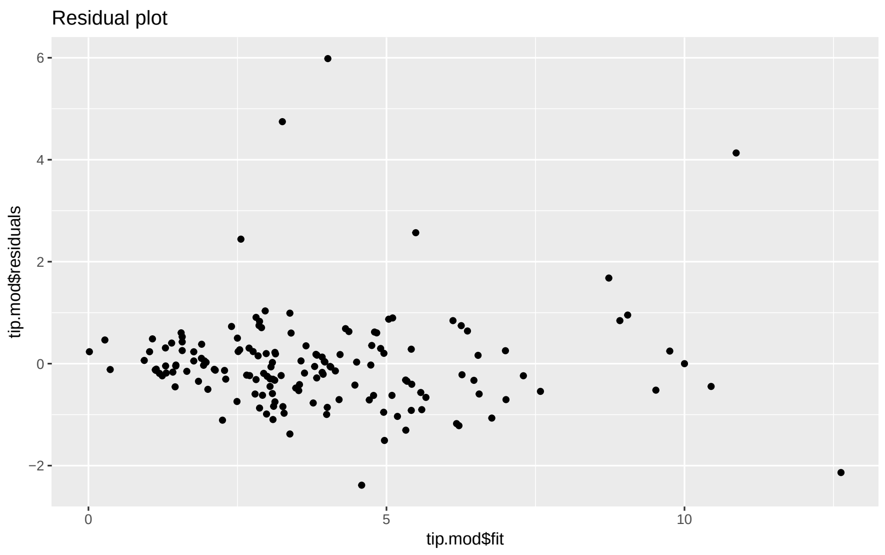
```
data(AllCountries)
dev.lm = lm( BirthRate ~ Rural + Health + ElderlyPop, data=AllCountries )
summary( dev.lm  )
```

```
##
## Call:
## lm(formula = BirthRate ~ Rural + Health + ElderlyPop, data = AllCountries)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -13.549  -4.382  -0.779   4.096  16.999
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  24.6255     1.8937   13.00  < 2e-16 ***
## Rural         0.1165     0.0234    4.98  1.5e-06 ***
## Health        0.1905     0.1033    1.84    0.067 .
## ElderlyPop   -1.2923     0.1026  -12.60  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.26 on 175 degrees of freedom
##   (38 observations deleted due to missingness)
## Multiple R-squared:  0.658,  Adjusted R-squared:  0.652
## F-statistic:  112 on 3 and 175 DF,  p-value: <2e-16
```

Checking conditions

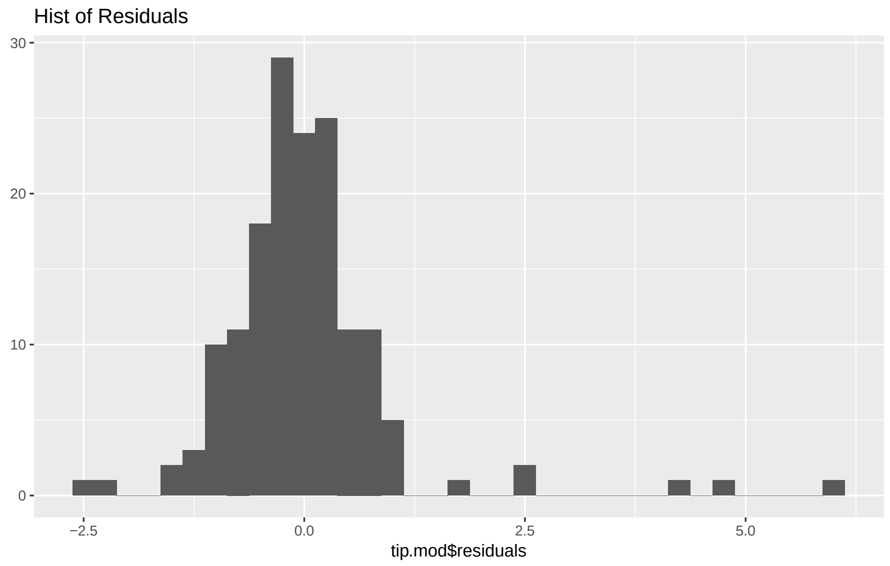
To check the conditions, we need to calculate residuals, make a residual versus fitted values plot, and make a histogram of the residuals:

```
qplot( tip.mod$fit, tip.mod$residuals, main="Residual plot" )
```



We see no real pattern other than some extreme outliers.

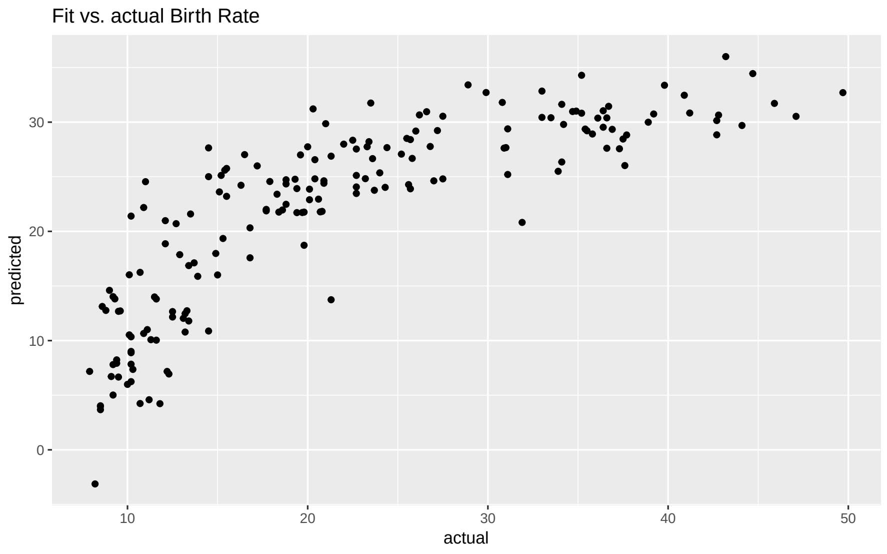
```
qplot( tip.mod$residuals, main="Hist of Residuals", binwidth=0.25)
```



The residual histogram suggests we are not really normally distributed, so we should treat our SEs and p-values with caution. These plots are the canonical "model-checking" plots you might use.

Another is the "fitted outcomes vs. actual outcomes" plot of:

```r
predicted = predict( dev.lm )
actual = dev.lm$model$BirthRate
qplot( actual, predicted, main="Fit vs. actual Birth Rate" )
```



Note the `dev.lm` variable has a `model` variable inside it. This is a data frame of the *used* data for the model (i.e., if cases were dropped due to missingness, they will not be in the model). We then grab the birth rates from this, and make a scatterplot. If we tried to skip this, and use the original data, we would get an error because our original data set has some observations that were dropped.