Appendix R4: Data Pre-Processing

Basic Data Transformations: WLS, GLM, Logistic and Trees

J. Alberto Espinosa

8/1/2021

Table of Contents

Overview

Experts state that about 80% of an analytics project goes into data work that needs to be done before analysis can be done with that data. This is generally referred to as feature engineering or data pre-processing. There are many reasons to do pre-processing and various types of pre-processing that can be done. But these can be generally classified into two broad categories: (1) data quality; and (2) data transformactions.

Data quality pre-processing involves things like dealing with things like: missing data; incorrect or inconsistent data; inappropriate data formats or spelling/punctuation issues, etc. A lot of the data quality work in data preprocessing is often done with the analytical tools, as they all have data evaluations and manipulation tools. But the most effective ways of working with data is the realm of database and big data fields, and it is a key skill in the data science profession.

I will cover **data transformations** in this section. These have to do with data manipulations that are done to:

- Comply with a particular model assumption (e.g., lagged models in forecasting; weighted regression models to correct for heteroscedasticity, etc.);
- Improve the predictive accuracy of a model (e.g., logs, Box-Cox, etc.); and
- Convert categorical and other non-quantitative data into quantitative data that can be used for statistical analysis (e.g., categorical to dummy variables, word counts, etc.)

While it is often necessary or desirable to transform predictor variables, many regression models are robust when it comes to predictor variables (e.g., predictor variables often don't need to be normally distributed). In contrast, dependent variables must be transformed to the correct types of values for the particular model. For, example, if the dependent variable is binary or categorical, you must transform the outcome variable and use a logistic, discriminant or other classification models. Good news: R makes this very easy.

1. Transformation: Categorical To Dummy Variable Predictors

Categorical variables (e.g., states, categories, etc.) are not quantitative and they cannot be used as is for statistical analysis. Categorical data has to be converted into some form of quantitative data. If the dependent variable is qualitative then you need to employ classification methods like logistic regression. When the independent variables are qualitative you can use the data with some conversions. R makes this very easy.

The most common way to deal with categorical variables is to create dummy variables. For example, if you have 50 states and you want to analyze whether the state where a house is located has an effect on the house value, you can create 50 dummy variables, one for each state. For example, the variable MD will take a value of 1 if the state is Maryland and 0 otherwise; VA will take a value of 1 if the state is Virginia and 0 otherwise; and so on.

Each state dummy variable would be constructed in a similar manner. You would then leave one of these variables out of the model which will be the "baseline" or "reference" variable (e.g., MD). You would then include in the model the remaining 49 state dummy variables. The intercept coefficient of this model gives you the effect of the baseline or reference variable (e.g., the median housing value prediction for the state of MD). Each of the other 49 state coefficients will give you the price difference for that state, relative to the reference variable (i.e., MD).

There are two ways to do this: (the hard way) you create the dummy variables yourself or (the easy way) you let R do it for you. Let's use the **Carseats** data set as an example:

```r
library(ISLR) # Contains the "Carseats" data set
```

In the model below, categorical variables will be converted into multiple dummy variables. The factor or categorical variable **ShelveLoc** has 3 values: **Bad**, **Medium** and **Good**. These are the shelving categories in various simulated stores and we want to understand how the shelf location affects carseat unit sales.

If we include **ShelveLoc** in the model R will convert this into only 3 binary variables: **ShelveBad** with a value of 1 if the shelf location is bad and 0 if not; **ShelveLocGood** with a value of 1 if the shelf location is medium and 0 otherwise; and so on. and ShelveLocMedium. Note that R will automatically name these variables with the original variable name **ShelveLoc** and attach a suffix equal to the categorical value for that shelf location. When the model is estimated, R will drop one of the three dummy variables, automatically, which is the first one alphabetically, or ShelveLocBad in this case. In a few moments I will explain how to change the variable dropped from the model. But, **why drop a variable?**

The Dummy Variable Trap

If you have a group of binary variables, which are mutually exclusive, then one of these variables is fully dependent on the others. For example if all the 49 state dummy variables have a value of 1, then the MD variable must be 1 (i.e., if it is not one of the 49 included states, it must be the excluded states). If you were to include all 50 variables, any one of these variables will be fully dependent on the other 49. This totally violates the OLS assumption of variable independence and the OLS is literally unsolvable because you would have perfect multicollinearity. This problem is commonly referred to as the "Dummy Variable Trap". Older statistical software would simply hang or give you an error (the standard errors are infinite). But modern statistical software like R will detect this linear dependency and automatically exclude one of the variables to avoid the dummy variable trap. Take a look:

```
class(Carseats$ShelveLoc)

## [1] "factor"

levels(Carseats$ShelveLoc) # Check the unique levels of ShelveLoc

## [1] "Bad"    "Good"   "Medium"

lm.fit <- lm(Sales~.,data=Carseats)
summary(lm.fit) # ShelveLocBad is the default reference category

##
## Call:
## lm(formula = Sales ~ ., data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8692 -0.6908  0.0211  0.6636  3.4115
##
## Coefficients:
```

```
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.6606231  0.6034487   9.380  < 2e-16 ***
## CompPrice        0.0928153  0.0041477  22.378  < 2e-16 ***
## Income           0.0158028  0.0018451   8.565 2.58e-16 ***
## Advertising      0.1230951  0.0111237  11.066  < 2e-16 ***
## Population       0.0002079  0.0003705   0.561    0.575
## Price           -0.0953579  0.0026711 -35.700  < 2e-16 ***
## ShelveLocGood    4.8501827  0.1531100  31.678  < 2e-16 ***
## ShelveLocMedium  1.9567148  0.1261056  15.516  < 2e-16 ***
## Age             -0.0460452  0.0031817 -14.472  < 2e-16 ***
## Education       -0.0211018  0.0197205  -1.070    0.285
## UrbanYes         0.1228864  0.1129761   1.088    0.277
## USYes           -0.1840928  0.1498423  -1.229    0.220
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.019 on 388 degrees of freedom
## Multiple R-squared:  0.8734, Adjusted R-squared:  0.8698
## F-statistic: 243.4 on 11 and 388 DF,  p-value: < 2.2e-16
```

Notice that the categorical variable ShelveLoc was automatically converted into 3 dummy variables. Also notice that ShelveLocBad (the first one alphabetically) was excluded from the model and only ShelveLocMedium and ShelveLocGood were included. The excluded variable is perhaps the most important variable because it will serve as the reference level, against which all other effects are compared.

Let's parse this. The intercept represents the average sales in thousands of units for ShalveLocBad (the excluded variable), when all other variables are 0. If all dummy variables are 0, then it must be the bad location because ShelveLocBad cannot be 0 when the other dummies are 0. As with other binary variable models. All other ShelveLoc dummy variables will measure the effect, but relative to the intercept, which is ShelveLocBad (the reference level). That is, ShelveLocMedium shows, on average and holding everything else constant, how many more carseat thousand units will sell in a medum shelf location, compared to a bad location.

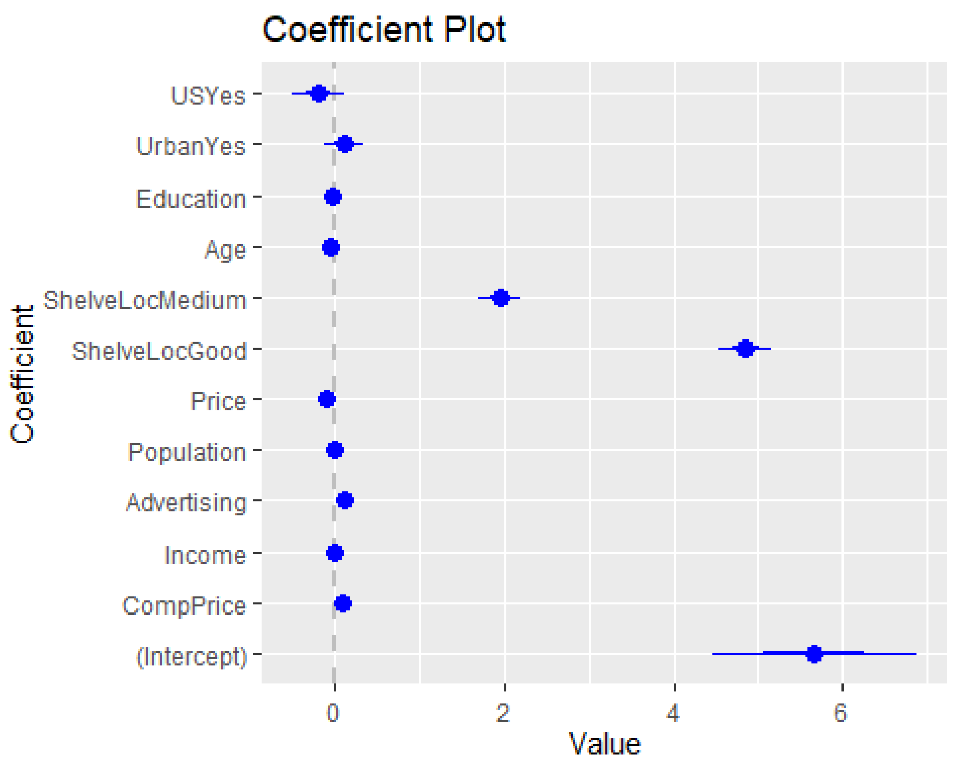
To see the categorical to dummy variable conversion, clearly showing that **Bad** is the reference level enter:

```
contrasts(Carseats$ShelveLoc) # Shows how the dummy variables were coded
```

```
##        Good Medium
## Bad       0      0
## Good      1      0
## Medium    0      1
```

You can inspect the coefficients and price differences across shelving locations visually too:

```
require(coefplot)
coefplot(lm.fit)
```



Re-Leveling

If you want to change the baseline or reference dummy variable you can use the `relevel()` function to select the reference level variable to exclude. The value you enter in `ref=` parameter has to match an existing level in the corresponding categorical variable. data (case sensitive). For example, if you want to use “Good” as the reference level, do the following:

```
Carseats$ShelveLoc <- relevel(Carseats$ShelveLoc, ref="Good")
summary(lm(Sales~.,data=Carseats)) # Check it out

##
## Call:
## lm(formula = Sales ~ ., data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8692 -0.6908  0.0211  0.6636  3.4115
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     10.5108058  0.6039582  17.403  < 2e-16 ***
## CompPrice        0.0928153  0.0041477  22.378  < 2e-16 ***
## Income           0.0158028  0.0018451   8.565 2.58e-16 ***
## Advertising      0.1230951  0.0111237  11.066  < 2e-16 ***
## Population       0.0002079  0.0003705   0.561    0.575
## Price           -0.0953579  0.0026711 -35.700  < 2e-16 ***
## ShelveLocBad    -4.8501827  0.1531100 -31.678  < 2e-16 ***
## ShelveLocMedium -2.8934679  0.1308928 -22.106  < 2e-16 ***
## Age             -0.0460452  0.0031817 -14.472  < 2e-16 ***
## Education       -0.0211018  0.0197205  -1.070    0.285
## UrbanYes         0.1228864  0.1129761   1.088    0.277
## USYes           -0.1840928  0.1498423  -1.229    0.220
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.019 on 388 degrees of freedom
## Multiple R-squared:  0.8734, Adjusted R-squared:  0.8698
## F-statistic: 243.4 on 11 and 388 DF,  p-value: < 2.2e-16
```

Notice that the coefficients ShelveLocBad and ShelveLocMedium now measure the difference in sales, compared to ShelfLocGood. Which reference level to select? If there is a category of interest, select it as the reference level because all other dummy variable effects will be relative to it. If you don't have a category of interest, pick a reference level with the least effect on the response variable, because you will get more noticeable contrasts and stronger effects for the included variables.

2. Transformation: Polynomials

I cover polynomial models in much more depth in Chapter 8, but I present some basic polynomial variable transformations. Let's start by predicting median value of homes in Boston neighborhoods **medv**, using linear, quadratic and cubic polynomials of **lstat** (percentage of low income population). Notice that we use the identity function `I()` for the square and cubic transformations. This is required by R. Without the `I()`, R will not interpret the power function properly.

```
library(MASS)
lm.fit1 <- lm(medv~lstat, data=Boston)
summary(lm.fit1)

##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

```
lm.fit2 <- lm(medv~lstat+I(lstat^2), data=Boston)
summary(lm.fit2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2834  -3.8313  -0.5295   2.3095  25.4148
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.862007   0.872084   49.15   <2e-16 ***
## lstat       -2.332821   0.123803  -18.84   <2e-16 ***
## I(lstat^2)   0.043547   0.003745   11.63   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 2.2e-16
```

```
lm.fit3 <- lm(medv~lstat+I(lstat^2)+I(lstat^3), data=Boston)
summary(lm.fit3)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2) + I(lstat^3), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5441  -3.7122  -0.5145   2.4846  26.4153
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 48.6496253  1.4347240  33.909  < 2e-16 ***
## lstat       -3.8655928  0.3287861 -11.757  < 2e-16 ***
## I(lstat^2)   0.1487385  0.0212987   6.983 9.18e-12 ***
## I(lstat^3)  -0.0020039  0.0003997  -5.013 7.43e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.396 on 502 degrees of freedom
## Multiple R-squared:  0.6578, Adjusted R-squared:  0.6558
## F-statistic: 321.7 on 3 and 502 DF,  p-value: < 2.2e-16
```

Note that the R-squared for the quadratic model is larger than the one for the linear model, and the R-squared for the cubic polynomial is larger than the quadratic. However, to evaluate if the difference is significant we need an ANOVA test (or F-test, which is similar)

```
anova(lm.fit1,lm.fit2, lm.fit3)

## Analysis of Variance Table
##
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + I(lstat^2)
## Model 3: medv ~ lstat + I(lstat^2) + I(lstat^3)
##   Res.Df   RSS Df Sum of Sq       F    Pr(>F)
## 1    504 19472
## 2    503 15347  1    4125.1 141.687 < 2.2e-16 ***
## 3    502 14616  1     731.8  25.134 7.428e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F-statistic is quite high and the p-value is <0.001 for both comparisons, so yes, the quadratic model is statistically superior to the linear model and the cubic model is superior to the quadratic model.

Interpretation: Interpreting a linear model is easy. Interpreting a quadratic model is not so bad. Interpreting a cubic model is trick but doable. Beyond cubic, it becomes almost impossible to interpret polynomial models and should be used primarily when the predictive modeling goal is accuracy. In the cubic model above the effect of **lstat** is negative, so for lower values of lstat, the regression curve goes downwards. But the quadratic effect is positive, so as lstat starts getting larger, it's square becomes really large and begins to pull the curve upwards and it may bottom out at some point. But then the cubic term is negative, so when lstat grows very large, the cubic effect will pull the curve downwards again and the curve may peak at some point. In summary, a polynomial of power n could potentially have n-1 peaks and valleys. The higher the polynomial, the curvier the regression model.

If you want to do high power polynomials, it is easier to use the **poly()** function which does the same as writing out all the power conversions. However, the `poly()` function does **orthogonal polynomials**, which is necessary when the various polynomial terms cause severe multicollinearity. I will discuss orthoghonal polynomials in more detail in Chapter 8, but for now let's just do standard or **raw** polynomials, just like we did with the `I()` function. All we need to do is to include the parameter `raw()=T`. Take for example a model with a polynomial of the 5th. power:

```
lm.fit5=lm(medv~poly(lstat,5, raw=T), data=Boston)
summary(lm.fit5)

##
## Call:
## lm(formula = medv ~ poly(lstat, 5, raw = T), data = Boston)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -13.5433  -3.1039  -0.7052   2.0844  27.1153 
## 
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)                  6.770e+01  3.604e+00  18.783  < 2e-16 ***
## poly(lstat, 5, raw = T)1 -1.199e+01  1.526e+00  -7.859 2.39e-14 ***
## poly(lstat, 5, raw = T)2  1.273e+00  2.232e-01   5.703 2.01e-08 ***
## poly(lstat, 5, raw = T)3 -6.827e-02  1.438e-02  -4.747 2.70e-06 ***
## poly(lstat, 5, raw = T)4  1.726e-03  4.167e-04   4.143 4.03e-05 ***
## poly(lstat, 5, raw = T)5 -1.632e-05  4.420e-06  -3.692 0.000247 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 5.215 on 500 degrees of freedom
## Multiple R-squared:  0.6817, Adjusted R-squared:  0.6785 
## F-statistic: 214.2 on 5 and 500 DF,  p-value: < 2.2e-16
```

```
anova(lm.fit3,lm.fit5)
```

```
## Analysis of Variance Table
## 
## Model 1: medv ~ lstat + I(lstat^2) + I(lstat^3)
## Model 2: medv ~ poly(lstat, 5, raw = T)
##   Res.Df   RSS Df Sum of Sq      F    Pr(>F)    
## 1    502 14616                                  
## 2    500 13597  2    1018.5 18.726 1.438e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Clearly, lm.fit5 is superior to lm.fit3, but good luck interpreting it.

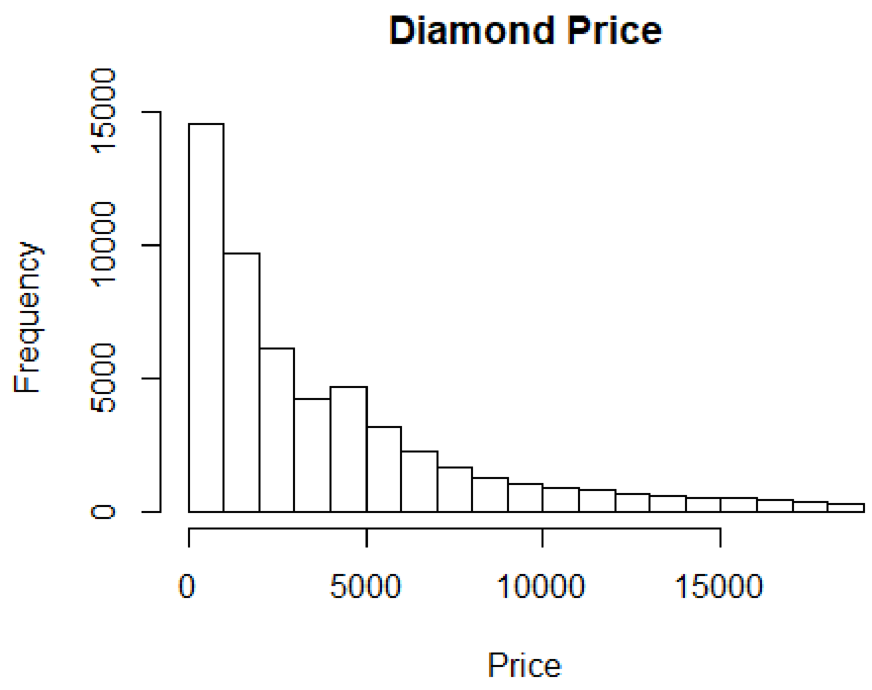
3. Transformation: Log Models

Dependent variables are often logged if they have skewed distributions. A log transformation is also necessary when the outcome variable contains counts (e.g., number of applicants, number of store visitors, etc.) – see Transformation 5 below on count data. Sometimes, we log variables to change the interprtation from unit effects to percentage effects or to improve the statistical fit of the model. Finally, when residuals are not normally distributed, sometimes logging the outcome variable resolves this issue.

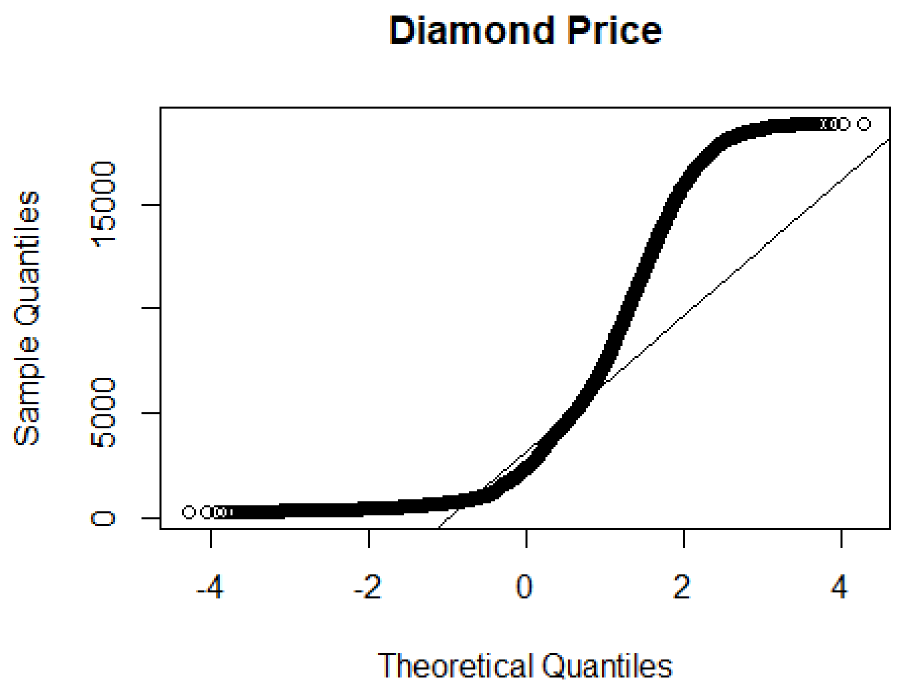
Inspecting for Normality

Predictors with a skewed distributio don't need to be log-transformed for a linear regression if the sample size is large (degrees of freedom > 50). But with smaller samples the sample means can no longer assumed to be normal, so log transformation may be necessary. Let's explore the distribution of diamond prices. We will do this visually with a histogram and a QQ Plot and quantitatively with a Shapiro-Wilks test (there are many tests of normality). Technical note: the Shapior-Wilks test does not work with samples over 5000, so we do the test on a random sample of 5000 observations.

```r
library(ggplot2) # Contains the "diamonds" data set
hist(diamonds$price, main="Diamond Price", xlab="Price")
```



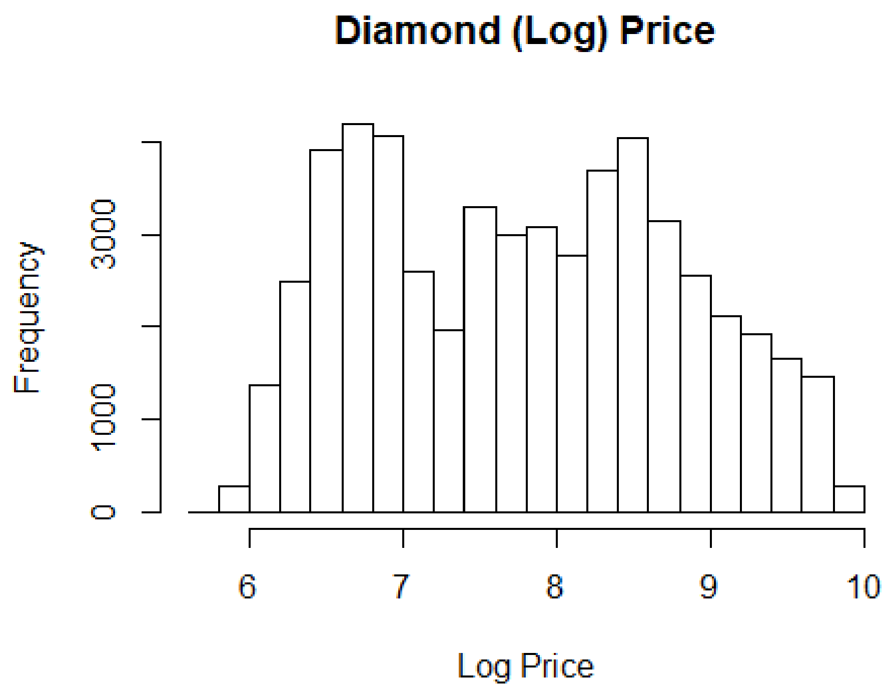
```r
qqnorm(diamonds$price, main="Diamond Price")
qqline(diamonds$price)
```



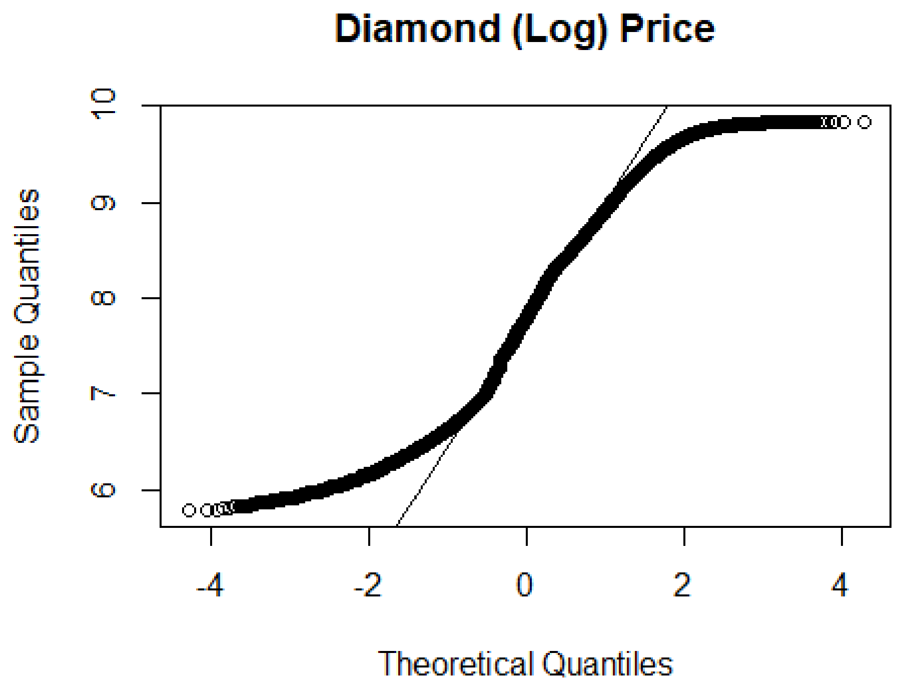
```
shapiro.test(diamonds[sample(nrow(diamonds), 5000),]$price)

##
##  Shapiro-Wilk normality test
##
## data:  diamonds[sample(nrow(diamonds), 5000), ]$price
## W = 0.793, p-value < 2.2e-16
```

It should be pretty obvious that diamond prices don't follow a normal distribution. The histogram is skewed to the right, with very few really expensive diamonds and lots of cheaper ones. Also the QQ Plot shows a sharp departure of the actual data points from the theoretical straight QQ Line. The more the data departs from the line the less normal it is because the straight line has values that match the Gaussian curve formula. Finally, the Shapiro-Wilks test is significant, so we reject the null hypothesis of normality and conclude that the data is not normally distributed. So, we log the price and repeat the process:

```
hist(log(diamonds$price), main="Diamond (Log) Price", xlab="Log Price")
```



```r
qqnorm(log(diamonds$price), main="Diamond (Log) Price")
qqline(log(diamonds$price))
```



```
shapiro.test(log(diamonds[sample(nrow(diamonds), 5000),]$price))

##
##  Shapiro-Wilk normality test
##
## data:  log(diamonds[sample(nrow(diamonds), 5000), ]$price)
## W = 0.96248, p-value < 2.2e-16
```

Interestingly, the histogram and QQ Plots show that the log transformation has made the diamond prices more normally distributed, to some extent. The histogram is more symmetric, but not exactly normally distributed. The QQ Plot shows that the center of the data aligns with the QQ Line but the tail ends depart from normality. These two should be acceptable criteria for a business model. But the Shapiro-Wilks test is still significant, so if we want to be rigorous about it, we would need to reject the null hypothesis of normality. But the Shapiro-Wilks test is notorious for rejecting normality with minor deviations of normality. Again, the model will be approximately correct, but if you need to be rigorous about this, you would need to try a Box-Cox transformation, but the tradeoff will be some loss of interpretability.

Let's look at the various log models and interpret them:

```
summary(lm(price~carat, data=diamonds))

##
## Call:
## lm(formula = price ~ carat, data = diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18585.3   -804.8    -18.9    537.4  12731.7
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2256.36      13.06  -172.8   <2e-16 ***
## carat        7756.43      14.07   551.4   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1549 on 53938 degrees of freedom
## Multiple R-squared:  0.8493, Adjusted R-squared:  0.8493
## F-statistic: 3.041e+05 on 1 and 53938 DF,  p-value: < 2.2e-16

summary(lm(price~log(carat), data=diamonds))

##
## Call:
## lm(formula = price ~ log(carat), data = diamonds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6137.4 -1495.1  -328.3  1141.9 12075.3
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  6237.84      10.73   581.2   <2e-16 ***
## log(carat)   5836.02      15.21   383.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2066 on 53938 degrees of freedom
## Multiple R-squared:  0.7319, Adjusted R-squared:  0.7319 
## F-statistic: 1.473e+05 on 1 and 53938 DF,  p-value: < 2.2e-16
```

```
summary(lm(log(price)~carat, data=diamonds))
```

```
##
## Call:
## lm(formula = log(price) ~ carat, data = diamonds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -6.2844 -0.2449  0.0335  0.2578  1.5642 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 6.215021   0.003348    1856   <2e-16 ***
## carat       1.969757   0.003608     546   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3972 on 53938 degrees of freedom
## Multiple R-squared:  0.8468, Adjusted R-squared:  0.8468 
## F-statistic: 2.981e+05 on 1 and 53938 DF,  p-value: < 2.2e-16
```

```
summary(lm(log(price)~log(carat), data=diamonds))
```

```
##
## Call:
## lm(formula = log(price) ~ log(carat), data = diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1.50833 -0.16951 -0.00591  0.16637  1.33793 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 8.448661   0.001365  6190.9   <2e-16 ***
## log(carat)  1.675817   0.001934   866.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2627 on 53938 degrees of freedom
## Multiple R-squared:  0.933,  Adjusted R-squared:  0.933
## F-statistic: 7.51e+05 on 1 and 53938 DF,  p-value: < 2.2e-16
```

Interpretation

Linear Model: On average, the effect of an additional carat in a diamond is an increase in price of $7,756.

Linear-Log Model: On average, the effect of 1% increase in carats in a diamond is an increase in price of `$5836/100` or `$58.36`. We divide by 100 because a 1% increase is a 0.01 or 1/100 proportion increase, so we need to do the same division on the price.

Log-Linear Model: On average, the effect of 1 additional carat in a diamond is a `1.96.97*100` or `196.97%` increase in price. We multiply the effect because the price increases by a factor of 1.9697, which in percentage points is 100 times that.

Log-Log Model: On average, the effect oa 1% increase in price is a `1.67%` increase in price. We don't need to multiply or divide by 100 because both sides of the model are in either proportions or percentages.

4. Transformation: Count Data

This is a special case of the **Log-Linear** model. OLS is often used with count data, but this is incorrect because counts are not normally distributed. This is so because counts are: (1) not negatives; (2) truncated at 0; and (3) not continuous (i.e., no decimals). Counts follow a **Poisson** distribution. However, if the average count is large the Poisson curve will be very close to a normal distribution and the truncation will happen very far on the left tail and the discrete values will be very close to each other, almost like a continuous variable. So, it is not a major issue to use OLS with count data when the average count of the response variable is very large. But specifying a count model is relatively easy and you would be doing it right. When the average count is small, the truncation on the left will happen close to the mean, so it would be very incorrect to use OLS.

The correct way to specify the model is to use the **Generalized Linear Model** with the `glm()` function, specifying a **Poisson** distribution and with `link="log"`. This link function tells the GLM to transform the outcome to a log value, essentially making the model a Log-Linear model.

In the example below I use the **Carseats** dataset from the **{ISLR}** library. In the model above we used OLS, but this is technically incorrect. The sales are in thousand units, so they have decimals and would appear to be continuous. But if we multiply Sales by 1000 to get actual units, it is easy to realize that they are actual counts. So, I first create a new variable `SalesUnits = 1000 * Sales` to capture actual units sold, that is, a count. Also, neither Sales nor SalesUnits are ever negative, so again, they are truncated at 0.

So, let's fit a reduced model to predict SalesUnits using just the significant predictors from the OLS model above, but using a count data model, using GLM with a Poisson distribution and a Log tranformation. Also, I relevel the ShelveLoc variable back to `ref="Bad"`.

```r
library(ISLR)
options(scipen=4) # To minimize use of scientific notation
Carseats$ShelveLoc <- relevel(Carseats$ShelveLoc, ref="Bad")
Carseats$SalesUnits <- 1000 * Carseats$Sales
glm.fit.count <- glm(SalesUnits ~
                     CompPrice + Income + Advertising + Price + ShelveLoc,
                     family=poisson(link="log"),
                     data=Carseats)
summary(glm.fit.count)

##
## Call:
## glm(formula = SalesUnits ~ CompPrice + Income + Advertising +
##     Price + ShelveLoc, family = poisson(link = "log"), data = Carseats)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -76.923  -11.186    0.626   10.703   45.006
##
## Coefficients:
##                   Estimate  Std. Error z value Pr(>|z|)
## (Intercept)      8.16534700  0.00526008  1552.3   <2e-16 ***
## CompPrice        0.01250654  0.00004616   270.9   <2e-16 ***
## Income           0.00233015  0.00002109   110.5   <2e-16 ***
## Advertising      0.01506094  0.00008523   176.7   <2e-16 ***
## Price           -0.01223072  0.00002990  -409.1   <2e-16 ***
## ShelveLocGood    0.62651643  0.00175152   357.7   <2e-16 ***
## ShelveLocMedium  0.29545561  0.00158949   185.9   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 454394  on 399  degrees of freedom
## Residual deviance: 119682  on 393  degrees of freedom
## AIC: 123954
##
## Number of Fisher Scoring iterations: 4
```

The `family=poisson(link="log")` attribute tells `glm()` that the response variable follows a Poisson distribution and that the outcome variable is to be log-transformed. Because SalesUnit is logged and the predictors are not, the coefficients (multiplied by 100) represent the percent increase in unit sales when a predictor increases by 1 unit. For example, on average, the effect of $1 increase in price is a `0.0122 * 100` or `1.22%` reduction in unit sales.

5. Transformation: Centering Data

Centering data is a straight forward operation. In the code below I first fit an uncentered OLS model to compare results.

```r
library(MASS) # Contains the Boston housing data set
lm.fit <- lm(medv ~ lstat + age, data=Boston) # Un-transformed model
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.981  -3.978  -1.283   1.968  23.158
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.22276    0.73085  45.458  < 2e-16 ***
## lstat       -1.03207    0.04819 -21.416  < 2e-16 ***
## age          0.03454    0.01223   2.826  0.00491 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16
```

To center the data We use the `scale()` function to center the entire Boston housing dataset, but you can actually center individual variables if needed. I give a new name to the centered dataset to preserve the original one. Please note that I also use the `data.frame()` function because the `scale()` function converts the data into a matrix and some functions like `lm()` and `glm()` only work with data frames. Also, the parameter `center=T` centers the data. I will discuss the `scale=F` in a few moments, which is used to standardize the data.

When fitting centered models, it is better to force the regression line through the origin. Including `-1` in the model formula removes the intercept term, that is, it forces the regression line through the origin. Otherwise the regression wih display an intercept that is very very small, which is mathematically almost 0 due to rounding.

```r
Boston.centered <- data.frame(scale(Boston, center=T, scale=F))
Boston.centered[1:5, 1:5] # Check a few of the centered values
```

```
##        crim         zn     indus        chas         nox
## 1 -3.607204   6.636364 -8.826779 -0.06916996 -0.01669506
## 2 -3.586214 -11.363636 -4.066779 -0.06916996 -0.08569506
## 3 -3.586234 -11.363636 -4.066779 -0.06916996 -0.08569506
## 4 -3.581154 -11.363636 -8.956779 -0.06916996 -0.09669506
## 5 -3.544474 -11.363636 -8.956779 -0.06916996 -0.09669506
```

```
lm.centered <- lm(medv ~ lstat + age -1, data=Boston.centered)
summary(lm.centered)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age - 1, data = Boston.centered)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.981  -3.978  -1.283   1.968  23.158
##
## Coefficients:
##       Estimate Std. Error t value Pr(>|t|)
## lstat -1.03207    0.04814 -21.438  < 2e-16 ***
## age    0.03454    0.01221   2.828  0.00486 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.167 on 504 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic: 309.6 on 2 and 504 DF,  p-value: < 2.2e-16
```

Notice that the uncentered and centered regressions above produce identical results because centering does not change the coefficients, except for the intercept.

So, why bother to center your data, if you are going to get identical results? If all you have is a linear model, there is no need to center the data. However, if you have polynomials or interaction models, certain methods will require that you center the data. For example, centering is necessary when including interaction terms involving 2 continuous variables:

```
summary(lm(medv~lstat*age, data=Boston)) # With uncentered data
```

```
##
## Call:
## lm(formula = medv ~ lstat * age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.806  -4.045  -1.333   2.085  27.552
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.0885359  1.4698355  24.553  < 2e-16 ***
## lstat       -1.3921168  0.1674555  -8.313 8.78e-16 ***
## age         -0.0007209  0.0198792  -0.036   0.9711
## lstat:age    0.0041560  0.0018518   2.244   0.0252 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.149 on 502 degrees of freedom
## Multiple R-squared:  0.5557, Adjusted R-squared:  0.5531
## F-statistic: 209.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

```
summary(lm(medv~lstat*age -1, data=Boston.centered)) # With centered data
```

```
##
## Call:
## lm(formula = medv ~ lstat * age - 1, data = Boston.centered)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.177  -4.296  -1.535   1.751  24.476
##
## Coefficients:
##            Estimate Std. Error t value Pr(>|t|)
## lstat     -1.077006   0.054585 -19.731  < 2e-16 ***
## age        0.044915   0.013576   3.308  0.00101 **
## lstat:age  0.002488   0.001434   1.735  0.08339 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.155 on 503 degrees of freedom
## Multiple R-squared:  0.5539, Adjusted R-squared:  0.5513
## F-statistic: 208.2 on 3 and 503 DF,  p-value: < 2.2e-16
```

Interpreting interaction effect with continuous variables is not easy. Centering variables in interaction terms makes the model correct and its interpretation easier. Interaction between 2 continuous variables means that the effect of one variable depends on the value of the other variable. The main effects in the above model show the effect of that variables, when the other variable is at its mean. I will discuss interaction models in Chapter 8, but for now, notice in the regressions above that once you include continuousxcontinuous interaction terms, the results are no longer the same between the centered and uncentered models. The centered data is the correct way to model continuous interaction terms.

Sometimes you want to center a few variables, in that case, center the specific variables, but better give them a new name to avoid messing up the original dataset.

```
Boston$medv.c = scale(Boston$medv, center=T, scale=F)
Boston$lstat.c = scale(Boston$lstat, center=T, scale=F)
Boston$age.c = scale(Boston$age, center=T, scale=F)
centered.data <- cbind(Boston$medv.c, Boston$lstat.c, Boston$age.c)
colnames(centered.data)=c("medv.c", "lstat.c", "age.c") # Add column labels
head(centered.data) # Check some of the results
```

```
##          medv.c   lstat.c      age.c
## [1,]  1.4671937 -7.673063  -3.374901
## [2,] -0.9328063 -3.513063  10.325099
## [3,] 12.1671937 -8.623063  -7.474901
## [4,] 10.8671937 -9.713063 -22.774901
## [5,] 13.6671937 -7.323063 -14.374901
## [6,]  6.1671937 -7.443063  -9.874901
```

6. Transformation: Standardizing Data

You can standardize data with the same `scale()` funtion but using the parameter `scale=T`. Standardizing automatically centers the data, so you can set the parameter to `center=F` or `center=T`. In the end, all variables have a mean of 0 and a standard deviation of 1.

```
Boston.standardized <- data.frame(scale(Boston, center=T, scale=T))
Boston.standardized[1:5, 1:5] # Check a few of the centered values

##         crim         zn      indus       chas        nox
## 1 -0.4193669  0.2845483 -1.2866362 -0.2723291 -0.1440749
## 2 -0.4169267 -0.4872402 -0.5927944 -0.2723291 -0.7395304
## 3 -0.4169290 -0.4872402 -0.5927944 -0.2723291 -0.7395304
## 4 -0.4163384 -0.4872402 -1.3055857 -0.2723291 -0.8344581
## 5 -0.4120741 -0.4872402 -1.3055857 -0.2723291 -0.8344581
```

Let's compare results (note that I did not remove the intercept, so its value is negligible):

```
summary(lm.fit) # Un-transformed model

## 
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -15.981  -3.978  -1.283   1.968  23.158 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 33.22276    0.73085  45.458  < 2e-16 ***
## lstat       -1.03207    0.04819 -21.416  < 2e-16 ***
## age          0.03454    0.01223   2.826  0.00491 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495 
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16

lm.standardized <- lm(medv ~ lstat + age, data=Boston.standardized)
summary(lm.standardized)

## 
## Call:
## lm(formula = medv ~ lstat + age, data = Boston.standardized)
## 
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -1.7376 -0.4325 -0.1396  0.2140  2.5180 
## 
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -4.615e-16  2.984e-02   0.000  1.00000    
## lstat       -8.013e-01  3.742e-02 -21.416  < 2e-16 ***
## age          1.057e-01  3.742e-02   2.826  0.00491 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.6712 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495 
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16
```

Coefficients indicate how many standard deviations **medv** changes when the predictors increase by 1 standard deviation. For example, when lstat increases by 1 standard deviation, medv decreases by 0.801 standard deviations.

As with centering, these instructions will standardize (center and scale) the indicated predictors and save the results in new variables suffixed .std.

```
Boston$medv.std = scale(Boston$medv, center=T, scale=T)
Boston$lstat.std = scale(Boston$lstat, center=T, scale=T)
Boston$age.std = scale(Boston$age, center=T, scale=T)
std.data <- cbind(Boston$medv.std, Boston$lstat.std, Boston$age.std)
colnames(std.data)=c("medv.std", "lstat.std", "age.std") # Add column labels
head(std.data) # Check some of the results

##        medv.std  lstat.std    age.std
## [1,]  0.1595278 -1.0744990 -0.1198948
## [2,] -0.1014239 -0.4919525  0.3668034
## [3,]  1.3229375 -1.2075324 -0.2655490
## [4,]  1.1815886 -1.3601708 -0.8090878
## [5,]  1.4860323 -1.0254866 -0.5106743
## [6,]  0.6705582 -1.0422909 -0.3508100
```

Standardized regression

Instead of standardizing the entire dataset, you can also run a regular linear model function and then extract standardized coefficients with the **{lm.beta}** package, which will yield identical results than runing a regression with all variables standardized.

```
library(lm.beta)
lm.standardized = lm.beta(lm.fit) # Extract standardized coefficients
summary(lm.standardized) # Check it out

## 
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
```

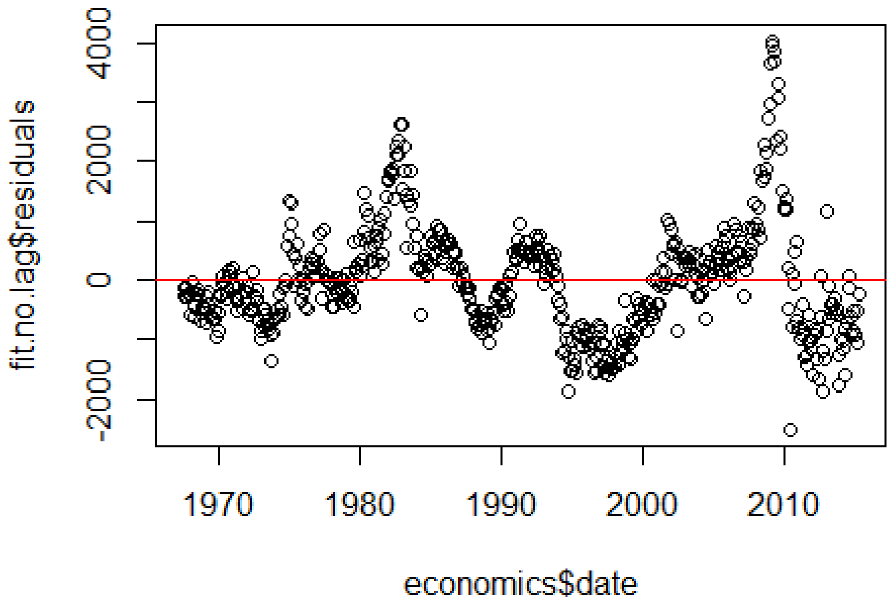
```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.981  -3.978  -1.283   1.968  23.158
##
## Coefficients:
##             Estimate Standardized Std. Error t value Pr(>|t|)
## (Intercept) 33.22276      0.00000    0.73085  45.458  < 2e-16 ***
## lstat       -1.03207     -0.80135    0.04819 -21.416  < 2e-16 ***
## age          0.03454      0.10573    0.01223   2.826  0.00491 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16
```

7. Transformation: Lagging data

Let's work the **economics** dataset in the **{ggplot2}** library. Let's plot a plain OLS **no-lag** model to predict unemployment (**unemploy** - number of unemployed in thousands) using **date** (month of data collection); **pce** (personal consumption expenditures in $ billions); **uempmed** (median duration of unemployment in weeks); **psavert** (personal savings rate); and **pop** (total population in thousands). Let's also plot the residuals against the month.

```
library(ggplot2)
# There is a minor data glitch in economics
economics = as.data.frame(economics) # To fix the glitch
options(scipen=4) # Reduce use of scientific notation
fit.no.lag <- lm(unemploy ~ date + pce + uempmed + psavert + pop, data=econom
ics)
summary(fit.no.lag)

##
## Call:
## lm(formula = unemploy ~ date + pce + uempmed + psavert + pop,
##     data = economics)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2522.2  -586.2   -76.0   439.9  4031.1
##
## Coefficients:
##                Estimate  Std. Error t value  Pr(>|t|)
## (Intercept) 26896.43520  6200.89392   4.338 0.0000171 ***
## date            1.64376     0.16036  10.250   < 2e-16 ***
## pce            -0.89754     0.10103  -8.884   < 2e-16 ***
## uempmed       581.60772    17.46624  33.299   < 2e-16 ***
## psavert       123.78794    32.45718   3.814  0.000152 ***
## pop             -0.13104     0.03004  -4.362 0.0000153 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 926.3 on 568 degrees of freedom
## Multiple R-squared:  0.8782, Adjusted R-squared:  0.8771
## F-statistic: 818.7 on 5 and 568 DF,  p-value: < 2.2e-16
```

```r
plot(economics$date, fit.no.lag$residuals)
abline(0,0, col="red") # Red line with intercept and slope equal to 0
```



The results above clearly shows a cyclical pattern in the residuals suggesting possible positive serial correlation. The plot shows an obvious cyclical pattern. The OLS results look really good, but that is part of the problem with serial correlation in that variables show as significant when sometimes they are not. So, it sounds like we will need to test this model for serial correlation. So, let's test it statistically.

The first thing is to ensure that the data is sorted by the ordinal variable, **date** in this case. That is, if we want to test that the errors are serially correlated across time, the cases must also be orderd by time. The **economics** dataset is already orderd.

Testing for Serial Correlation

We start with a **Durbin-Watson (DW)** test for serial correlation using the `dwtest()` function in the **{lmtest}** library.

```r
library(lmtest)
dwtest(fit.no.lag)

## 
##  Durbin-Watson test
## 
## data:  fit.no.lag
## DW = 0.18485, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0
```

DW = 0.185, which is far to the left of 1.5 and the p-value is also significant. So, we reject the null hypothesis of no serial correlation and conclude that serial correlation is present.

But this DW test only evaluates serial correlation with the data in the prior period. However, we suspect that there may be serial correlation with data in the same month one year (i.e., 12 periods) back. So we need another test.We can use either the Box-Pierce or Ljung-Box test, both of which allow you to specify the number of lagged periods to test. Let's try both tests with `lag=1` and `lag=12`. Notice that, in contrast to the DW test, the two Box tests only required the ordered residual for the test.

```r
Box.test(residuals(fit.no.lag), lag=1) # The default is Box-Pierce

## 
##  Box-Pierce test
## 
## data:  residuals(fit.no.lag)
## X-squared = 472.72, df = 1, p-value < 2.2e-16

Box.test(residuals(fit.no.lag), lag=12)

## 
##  Box-Pierce test
## 
## data:  residuals(fit.no.lag)
## X-squared = 3703.1, df = 12, p-value < 2.2e-16

Box.test(residuals(fit.no.lag), lag=1, type="Ljung-Box")

## 
##  Box-Ljung test
## 
## data:  residuals(fit.no.lag)
## X-squared = 475.2, df = 1, p-value < 2.2e-16

Box.test(residuals(fit.no.lag), lag=12, type="Ljung-Box")

## 
##  Box-Ljung test
## 
## data:  residuals(fit.no.lag)
## X-squared = 3751.8, df = 12, p-value < 2.2e-16
```

Both tests confirm the serial correlation with residuals lagged 1 period, but they also show serial correlation with a lag of 12 periods.

Correcting for Serial Correlation

An AR(1) model is an auto-regressive model with a 1 period lag. An AR(12) model would include 12 lags of 1, 2, etc., to all 12 perdiod lags. We can certainly try this, which would be standard in econometric forecasting models, but since we are building this model using business intuition and would like a good degree of interpretability, we will lag just 1 and 12 periods and observe our results. However, theres is nothing stopping you from trying a full AR(12) model on your own.

Let's lag the unemploy variable 1 and 12 periods and store the lagged results in new variables suffixed **L1** and **L2**. We use the `slide()` function from the from the **{DataCombine}** library. Negative slides lag values and positive slides move values forward. The `var=` parameter shows which variable to lag; the `NewVar=` parameter contains the name of the variable where the lagged values will be stored; and the `slideBy=` parameter indicates how many periods to lag (negative) or advance (positive).

```
library(DataCombine)
economics = slide(economics, Var="unemploy", NewVar="unemploy.L1", slideBy=-1
)
economics = slide(economics, Var="unemploy", NewVar="unemploy.L12", slideBy=-
12)
economics[1:15,c("date", "unemploy", "unemploy.L1", "unemploy.L12")]

##          date unemploy unemploy.L1 unemploy.L12
## 1  1967-07-01     2944          NA           NA
## 2  1967-08-01     2945        2944           NA
## 3  1967-09-01     2958        2945           NA
## 4  1967-10-01     3143        2958           NA
## 5  1967-11-01     3066        3143           NA
## 6  1967-12-01     3018        3066           NA
## 7  1968-01-01     2878        3018           NA
## 8  1968-02-01     3001        2878           NA
## 9  1968-03-01     2877        3001           NA
## 10 1968-04-01     2709        2877           NA
## 11 1968-05-01     2740        2709           NA
## 12 1968-06-01     2938        2740           NA
## 13 1968-07-01     2883        2938         2944
## 14 1968-08-01     2768        2883         2945
## 15 1968-09-01     2686        2768         2958
```

Now let's try the regression model above, but with the 2 lagged variables included.

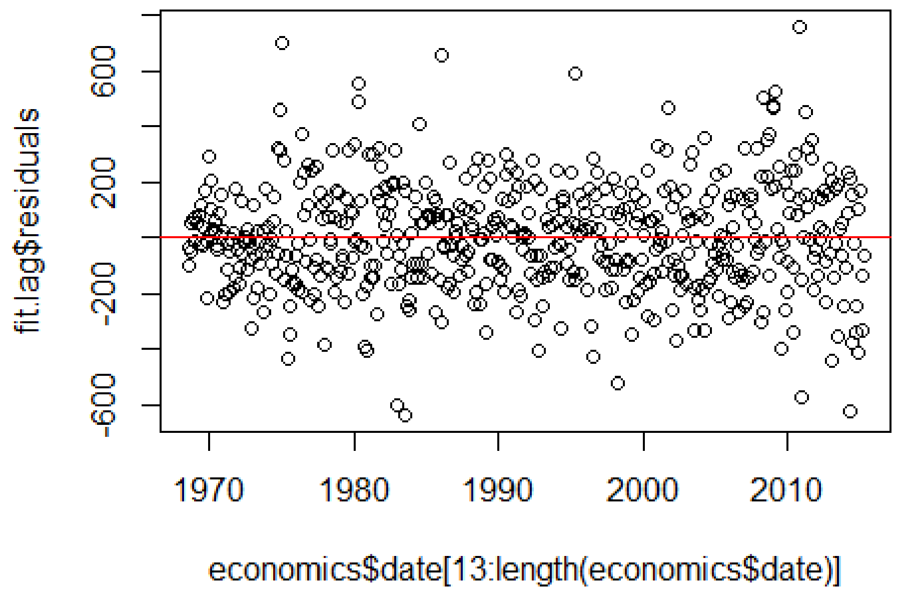
```
fit.lag <- lm(unemploy ~ date + unemploy.L1 + unemploy.L12 +
                         pce + uempmed + psavert + pop,
                         data=economics)
summary(fit.lag)
```

```
##
## Call:
## lm(formula = unemploy ~ date + unemploy.L1 + unemploy.L12 + pce +
##     uempmed + psavert + pop, data = economics)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -636.48 -126.15   -7.57  127.65  757.56
##
## Coefficients:
##                  Estimate   Std. Error t value Pr(>|t|)
## (Intercept)  -2172.974745  1578.301787  -1.377 0.169136
## date            -0.052880     0.047746  -1.108 0.268544
## unemploy.L1      1.065303     0.009687 109.972  < 2e-16 ***
## unemploy.L12    -0.055135     0.009076  -6.075 2.31e-09 ***
## pce              0.016157     0.023327   0.693 0.488837
## uempmed        -31.500533     8.560124  -3.680 0.000256 ***
## psavert         29.676306     6.969609   4.258 2.42e-05 ***
## pop              0.009526     0.007643   1.246 0.213176
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 196 on 554 degrees of freedom
##   (12 observations deleted due to missingness)
## Multiple R-squared:  0.9943, Adjusted R-squared:  0.9942
## F-statistic: 1.373e+04 on 7 and 554 DF,  p-value: < 2.2e-16
```

The change in results should be obvious. data, pce and pop were significant in the no-lag model, but they are not significant now. That is, once we control for the unemployment values 1 and 12 periods back, these variables lose their significance.

Let's see if the serial correlation problem was corrected. I will start with a residual plot against date. Note that I exclude the first 12 date values and start with the 13th value, because there are no residuals for these data points, since there are no lag 12 unemployment values.

```r
plot(economics$date[13:length(economics$date)], fit.lag$residuals)
abline(0,0, col="red") # Red line with intercept and slope equal to 0
```



The residual plot above shows an even cloud of residuals, suggesting that serial correlation was resolved. To test this statistically we conduct a DW test and the two Box tests for lag-1 and lag-12

```
dwtest(fit.lag)

##
##  Durbin-Watson test
##
## data:  fit.lag
## DW = 2.1188, p-value = 0.8705
## alternative hypothesis: true autocorrelation is greater than 0

Box.test(residuals(fit.lag), lag=1)

##
##  Box-Pierce test
##
## data:  residuals(fit.lag)
## X-squared = 1.9927, df = 1, p-value = 0.1581

Box.test(residuals(fit.lag), lag=1, type="Ljung-Box")

##
##  Box-Ljung test
##
## data:  residuals(fit.lag)
## X-squared = 2.0034, df = 1, p-value = 0.157
```

```
Box.test(residuals(fit.lag), lag=12)

##
##  Box-Pierce test
##
## data:  residuals(fit.lag)
## X-squared = 24.036, df = 12, p-value = 0.02011

Box.test(residuals(fit.lag), lag=12, type="Ljung-Box")

##
##  Box-Ljung test
##
## data:  residuals(fit.lag)
## X-squared = 24.384, df = 12, p-value = 0.01803
```

The DW test shows that serial correlation for a 1-period lag was corrected. However, many statiscicians argue that DW is not an appropriate test when a lagged response variable is included as a predictor because the DW will be biased towards a value of 2. From a practical perspective, the DW test is not precise, but sufficient for most business applications, particularly since the residual plot looks OK. However, it does not hurt to try the box tests and they are both non-significant, which provide some reassurance that the serial correlation problem was corrected. However, it appears that the serial correlation with lag-12 was not fully correcte, as both Box tests were significant. Again, this should be OK for most business models, but a more rigorous approach would require to test all 12 lags of the AR(12) model to find which lags are significant and which ones correct their respective serial correlation. I invite the readers to try this on their own.

Interpretation:

If we accept the lagged model above, let's analyze it. The R-squared is quite high at 99.43%. However, this is not entirely unusual with lagged variables, which tend to be very strong predictors. As I discussed above, date, pce and pop are no longer significant. uempmed has a strong negative effect, but in the lagged model the effect flipped to negative. psavert retain its high significance and sign. These results illustrate how a model with serial correlation without correction will yield misleading results and show significant predictors, which are not so in reality, and which may actually have the opposite effect.

The Lag-1 variable had a strong positive effect, suggesting that, holding everything else constant, on average, for each person unemployed in one period, there will be 1.06 persons unemployed in the subsequent period. However, for every person unemployed 12 months ago, there are 0.05 fewer persons unemployed today. These resulsts are illustrative of the business cycle.