R computing for Business Data Analytics

Instructor: Dr. Howard Hao-Chun Chuang

Assistant Professor, Department of Management Information Systems

National Chengchi University, Taipei, Taiwan 116

chuang@nccu.edu.tw

Last Revised: October 2014

In lectures 3-4 we have covered basic probability theory and a set of widely-used probability distributions that enable us to model *population* behaviors. In lecture 5 we go through parameter estimation and Monte-Carlo simulation. From now on, we will transit to statistical data analysis, which involves the application of probability distributions to assess *sample* characteristics, to make inferences in population, and most importantly, to obtain **information grounded on data** in order to facilitate our decision-making. My hope is that you will gradually be able to connect the dots as we progress, instead of viewing probability and statistics as isolated parts.

In the coming lectures we will look into several important data analysis techniques. Most of them are regression-based. Regression is a statistical method to assess relationships among variables. The objective is to find a model for predicting the dependent (response) variable given one or multiple independent (predictor) variables. Being proficient in regression modeling will enable you to answer many questions by using data formally and provide a solid foundation for business data analytics.

Given limited time, I will try to minimize mathematical foundation of statistical models in the remaining lectures. Without compromising your learning, I will put some technical details aside and focus on applied statistical modeling in *R*.

6.1 Simple linear regression

A *simple* linear regression model is defined by

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,\ i = 1, 2, \ldots, n$$

where ε_i is a random error/term with mean zero and variance σ^2. The word *simple* means that there is one and only one independent variable in the model.

As we collect a data set of sample size *n*, in most cases we estimate β_0 (intercept) and β_1 (slope) using ordinary least squares (OLS), which basically solves the optimization problem:

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

$$\text{s.t. } \hat{y}_i=\hat{\beta}_0+\hat{\beta}_1 x_i$$

We can re-write the objective function as:

$$Min\ (y_1-\hat{\beta}_0+\hat{\beta}_1x_1)^2+(y_2-\hat{\beta}_0+\hat{\beta}_1x_2)^2+\cdots+(y_n-\hat{\beta}_0+\hat{\beta}_1x_n)^2$$

Using some calculus, we can show that

$$\hat{\beta}_1=\frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sum_{i=1}^{n}(x_i-\overline{x})^2}\ \&\ \hat{\beta}_0=\overline{y}-\hat{\beta}_1\overline{x}$$

In addition to estimated coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$, we are interested in estimated residua.

$$\hat{\varepsilon}_i=y_i-\hat{y}_i$$

Remember in lecture 5 we have learnt that parameter estimation/statistical inference involves *uncertainty*, which is quantified through standard errors associated with $\hat{\beta}_0$ and $\hat{\beta}_1$.

$$\text{SE}(\hat{\beta}_1)^2=\sigma^2\left[\frac{1}{n}+\frac{\overline{x}^2}{\sum_{i=1}^{n}(x_i-\overline{x})^2}\right]\ \&\ \text{SE}(\hat{\beta}_0)^2=\frac{\sigma^2}{\sum_{i=1}^{n}(x_i-\overline{x})^2}\ \ (\sigma^2=\text{Var}(\varepsilon_i))$$

Standard errors have two applications. First, standard errors can be used to construct the *95%* (or ??%) *confidence interval* for β_1: $[\hat{\beta}_1-1.96\cdot\text{SE}(\hat{\beta}_1),\ \hat{\beta}_1+1.96\cdot\text{SE}(\hat{\beta}_1)]$. Second, standard errors allow us to perform *hypothesis testing*

$$H0:\beta_1=0\ \left(\text{There is no relationship between } x \text{ and } y\right)$$
$$Ha:\beta_1\neq 0\ \left(\text{There is some relationship between } x \text{ and } y\right)$$

Essentially, we want to determine whether $\hat{\beta}_1$ is **sufficiently far from zero** so that we can be confident that β_1 is non-zero. In practice, we compute a *t-statistic*

$$t=\frac{\hat{\beta}_1-0}{\text{SE}(\hat{\beta}_1)}$$

which measures the number of standard deviations that $\hat{\beta}_1$ is away from 0. The *p-value* is nothing but the probability of observing any value $\geq|t|$, assuming β_1=0. A small p-value indicates that the observed association between *x* and *y* is *probably not due to chance*. We reject *H*0 and declare a relationship between *x* and *y* to exist, if the p-value is **small enough**.

Let's try to fit this model using a simple dataset in *R*.

```
> data( )
> attach(cars)
> plot(cars)
```

The plot reveals a _________ association between distance and speed of cars. Suppose we want to estimate the following model

$$dist_i = \beta_0 + \beta_1 speed_i + \varepsilon_i,\ i = 1, 2, \ldots, 50$$

Here is how we do it in *R*:

> L1=lm(dist ~ speed)
>
> summary(L1) #What are $\hat{\beta}_0$ and $\hat{\beta}_1$? (coef(L1))

The statistical significance of estimated betas is contingent on estimates of standard errors and hypothesis testing. However, most of the estimates and tests are derived based on *large-sample* properties. For the *cars* data, the sample size is merely 50, a pretty small number of observations from a contemporary standpoint. How can we tackle the issue? Nonparametric *bootstrapping* (i.e., sample with replacement) turns out to be an attractive option.

```
> n=nrow(cars) #the sample size
> S=500 #the number of bootstrapped/simulated sample
> betas( )
> for(s in 1:S){
>       new=sample(1:n, size=n, replace=TRUE)
>       boot.speed=speed[new]
>       boot.dist=dist[new]
>       boot.fit=lm(boot.dist~boot.speed)
>       betas[s]=coef(boot.fit)[2] #retrieve beta1 only
> }
> hist(betas)
> c(mean(betas), sd(betas)) #are the numbers close to coef(L1)?
> c(quantile(betas, 0.025), quantile(betas, 0.975))
```

We are also interested in estimated residuals

```
> res=residuals(L1)
> mean(res) #Check the assumption of mean zero
```

Residuals allow us to detect potential outliers that we may fail to predict well (DON'T throw outliers away if you have a large sample) and help us assess model specification errors.

```
> yhat=fitted(L1)
> plot(yhat, res) #Do you notice outliers?
> plot(L1, which=1) #This generates the same plot with a smoothed line
```

Despite the decent fit of L1, $\hat{\beta}_0$ =-17.579 feet does not make sense in the real world because it suggests that a car moves backward when *x*(speed)=0. Instead, we can fit

$$dist_i = \beta_1 speed_i + \varepsilon_i,\ i = 1, 2, ..., 50$$

```
> L2=lm(dist ~ 0 + speed)
> summary(L2)
```

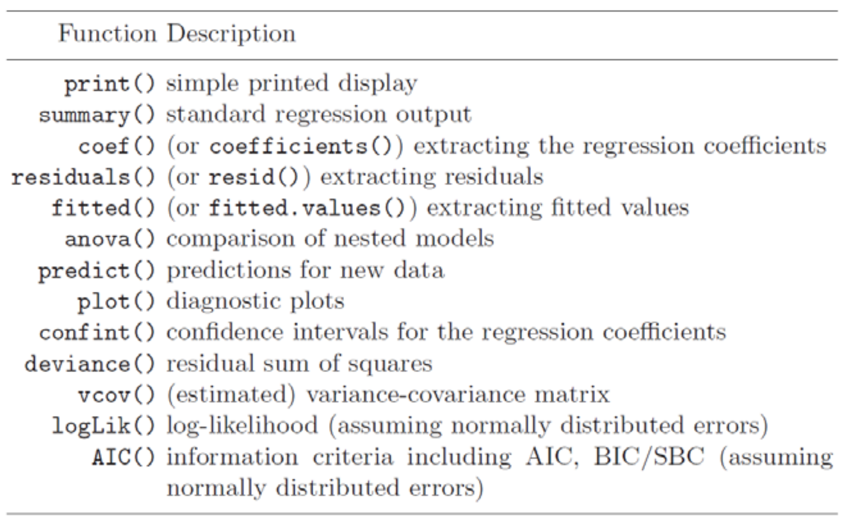
Let's make the model NOT so simple and fit the following

$$dist_i = \beta_1 speed_i + \beta_2 speed_i^2 + \varepsilon_i,\ i = 1, 2, ..., 50$$

```
> L3=lm(dist ~ 0 + speed + I(speed^2))
> plot(cars)
> abline(L1)
> abline(L2, col= 'red', lwd=2, lty=2)
> curve(1.239*x+0.090*x^2 , add=TRUE, col= 'blue', lwd=2, lty=3)
```

I hope the example helps refresh your memory of regression and build more confidence in data analysis. Also, *lm*( ) is a very powerful function and we will see it so many times. Below lists functions you can apply to a fitted object such as L1 (Kleiber & Zeileis, 2008)

Table 3.1. Generic functions for fitted (linear) model objects.



Function	Description
`print()`	simple printed display
`summary()`	standard regression output
`coef()`	(or `coefficients()`) extracting the regression coefficients
`residuals()`	(or `resid()`) extracting residuals
`fitted()`	(or `fitted.values()`) extracting fitted values
`anova()`	comparison of nested models
`predict()`	predictions for new data
`plot()`	diagnostic plots
`confint()`	confidence intervals for the regression coefficients
`deviance()`	residual sum of squares
`vcov()`	(estimated) variance-covariance matrix
`logLik()`	log-likelihood (assuming normally distributed errors)
`AIC()`	information criteria including AIC, BIC/SBC (assuming normally distributed errors)

6.2 Monte-Carlo simulation of simple linear regression

You have learnt how to estimate a simple linear regression model using data with the aid of *lm*( ). Can you say something about how good the estimation approach – OLS – is? One can address the question by designing a Monte-Carlo simulation experiment, assuming the data generating process (DGP) is known.

```
> set.seed(123)
> b0=0.2
> b1=0.5
> n=1000
> x=runif(n, -1, 1)
> S=500
> par.est=matrix(NA, nrow=S, ncol=4)
> for(s in 1:S){
>     y=b0+b1*x+rnorm(n, 0, 1)
>     model=lm(y ~ x)
>     vcv=vcov(model)
>     par.est[s, 1]=model$coef[1]
>     par.est[s, 2]=model$coef[2]
>     par.est[s, 3]=sqrt(diag(vcv)[1])
>     par.est[s, 4]= sqrt(diag(vcv)[2])
> }
```

Assess estimated betas versus true betas

```
> dev.new(width=12, height=5)
> par(mfrow=c(1,2))
> hist(par.est[,1])
> abline(v=b0,col= ‘red’, lwd=2)
> hist(par.est[,2])
> abline(v=b0, col= ‘red’, lwd=2)
```

Ass the standard deviation

```
> sd.beta0=sd(par.est[ , 1])
> mean.se.beta0=mean(par.est[ , 3])
> sd.beta1=sd(par.est[ , 2])
> mean.se.beta1=mean(par.est[ , 4])
```

We can further assess the validity of 95% confidence intervals.

```
> lower.beta0=par.est[ ,1]-qt(0.975, df=n-2)*par.est[,3]
> upper.beta0=par.est[ ,1]+qt(0.975, df=n-2)*par.est[,3]
> lower.beta1=par.est[ ,2]-qt(0.975, df=n-2)*par.est[,4]
> upper.beta1=par.est[ ,2]+qt(0.975, df=n-2)*par.est[,4]
```

Calculate the coverage probability

```
> trueinCI.beta0=ifelse(b0>= lower.beta0 & b0 <=upper.beta0, 1, 0)
> trueinCI.beta1=ifelse(b1>= lower.beta1 & b1 <=upper.beta1, 1, 0)
> cover.beta0=mean(trueinCI.beta0)
> cover.beta1=mean(trueinCI.beta1)
```

What is the overall performance the OLS model ?

6.3 Multiple linear regression

Multiple linear regression refers to a model with more than one independent variables

$$y_i = \beta_0 + \beta_1 x_{1i} + \cdots \beta_k x_{ki} + \varepsilon_i \;\; i = 1, 2, ..., n$$

Again, what OLS does is to solve the optimization problem

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

$$\text{s.t. } \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \cdots + \hat{\beta}_k x_{ki}$$

The objective function in the optimization problem is called *SSE* (sum of squared errors). We can modify it into *MSE* (mean squared errors)

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$E[\text{MSE}] = \sigma^2 + (\text{Model Bias})^2 + \text{Model Variance}$$

The first term σ^2 is called "irreducible noise" and cannot be eliminated by modeling. The second term is the squared bias that reflects how close the functional form of the model can get to the true relationship between the predictors and the outcome. The last term is model variance. Here a famous *variance-bias trade-off* arises. Complex models tend to have high variance and *over-fit*. Simple models, on the other hand, tend to have high bias and *under-fit*. Balancing the trade-off makes regression modeling an art rather than a mechanic.

As mentioned earlier, we impose assumptions on the error term ε_i. Essentially we assume

$$y_i \sim N(\beta_0 + \beta_1 x_{1i} + \cdots \beta_k x_{ki}, \sigma^2),\ i = 1, 2, ..., n$$
$$\text{or } \varepsilon_i \sim N(0, \sigma^2),\ i = 1, 2, ..., n$$

A direct consequence of this assumption is that the response variable *y* must be *continuous* and *iid*. This assumption is so critical but often ignored by many analysts (including a lot of professors…). On some occasions you will have to deal with *non-continuous* and/or *non-iid y* observations, which require a set of different techniques (in lectures 8 & 9). The *normality*, if violated, is NOT as important per se. What else do we assume?

Let's look into a dataset.

```
> install.packages("foreign")
> library(foreign)
> kidiq=read.dta("…./kidiq.dta", convert.underscore=T)
> attach(kidiq)
```

Consider a linear regression model with two predictors:

$$kid.score_i = \beta_0 + \beta_1 mom.hs_i + \beta_2 mom.iq_i + \varepsilon_i\ \ i = 1, 2, ..., n$$

```
> fit.1=lm(kid.score ~ mom.hs + mom.iq)
> coef(fit.1)
```

Write out the fitted model and interpret the coefficients.

```
> summary(fit.1)
```

What is the meaning of R^2 and adjusted R^2? I believe you ALL have seen R^2 in your statistics course. What is the intuition?

R^2 is approximately equal to the *squared correlation between y_i and fitted* $\hat{y}_i$. How can we verify this in *R*?

```
> ?
```

On top of R^2, what is the meaning of the *F*-statistic?

```
> install.packages("AER")
> library(AER)
> linear.hypothesis(fit.1, c("mom.hs=0", "mom.iq=0"))
```

The hypothesis test answers a question: Is at least one of $x_1, x_2, \ldots, x_k$ useful in predicting the response variable *y* (write out the *H*0 in the test above)?

You may have noticed that the variable *mom.hs* is a dummy variable that takes binary values (0/1). Dummy variables are useful for us to model categorical data (e.g., education, residence, ethnic group). The most common way to code dummy variables is to create *j*-1 dummies for *j* categories, in which the excluded category serves as the base level (including *j* variables for *j* categories makes the regression model NON-identifiable).

```
> lm(kid.score ~ mom.work)
> lm(kid.score ~ as.factor(mom.work)) #can you see the difference?
```

mom.work=1: mother did not work in first three years of child's life

mom.work=2: mother worked in second or third years of child's life

mom.work=3: mother worked part-time in first year of child's life

mom.work=4: mother worked full-time in first year of child's life

How many dummy variables are included? Which category is the base level?

Consider a linear regression model with *interactions*:

$$kid.score_i = \beta_0 + \beta_1 mom.hs_i + \beta_2 mom.iq_i + \beta_3 mom.hs_i \times mom.iq_i + \varepsilon_i \quad i = 1, 2, ..., n$$

```
> fit.2=lm(kid.score ~ mom.hs + mom.iq + mom.hs:mom.iq)
> coef(fit.2)
> confint(fit.2)
```

Write out the fitted model and interpret the coefficients as well as confidence intervals!

There are three types of interaction effects:

Synergistic (enhancing) interactions: Both variables affect *y* in the same direction, and together they produce an even stronger effect. The additive effects and the interactive effect are all of the same sign.

Buffering interactions: Two predictors have coefficients of opposite sign. One predictor weakens the effect of the other predictor. The interaction "buffers" the weakening.

Interference (antagonistic) interactions: Both predictors have coefficients of the same sign in additive form, but the interaction has a coefficient of opposite sign.

Which type of interaction does fit.2 reveal?

We can further visualize the fitted model.

```
> plot(mom.iq, kid.score, xlab= "Mother IQ", ylab= "Child test score", pch=20)
> curve(cbind(1, 1, x, 1*x) %*% coef(fit.2), add=T, col= "red")
> curve(cbind(1, 0, x, 0*x) %*% coef(fit.2), add=T, col= "green")
```

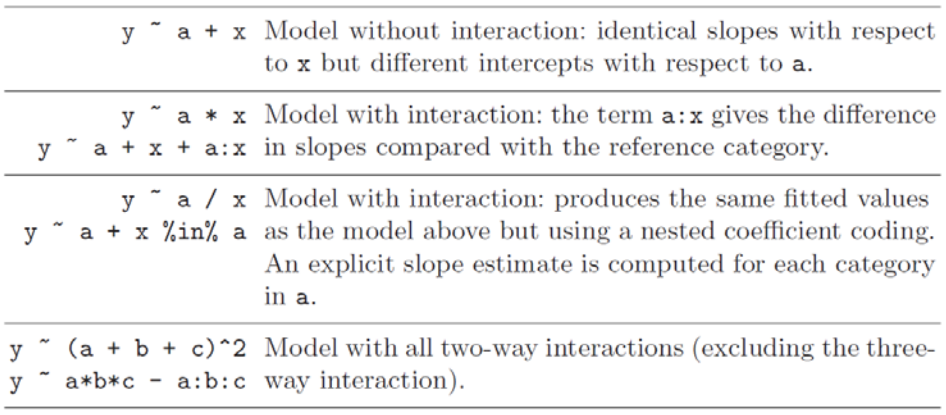
The coefficient on the interaction term represents the difference in the slope for *mom.iq*, comparing kids with mothers who did and did not complete high school. That is, the difference between the slopes of the red and green lines.

Let's put everything into the regression model

```
> fit.all=lm(kid.score ~ mom.hs + mom.iq + mom.hs:mom.iq + mom.age +
as.factor(mom.work))
> summary(fit.all)
```

This is by far the most complicated model we have fitted. Including all variables doesn't help much in this case. Do you notice any changes in the statistical significance of variables?

Below is another useful table for you learn how to manipulate *lm*( ).



`y ~ a + x`	Model without interaction: identical slopes with respect to `x` but different intercepts with respect to `a`.

`y ~ a * x`
`y ~ a + x + a:x`
	Model with interaction: the term `a:x` gives the difference in slopes compared with the reference category.

`y ~ a / x`
`y ~ a + x %in% a`
	Model with interaction: produces the same fitted values as the model above but using a nested coefficient coding. An explicit slope estimate is computed for each category in `a`.

`y ~ (a + b + c)^2`
`y ~ a*b*c - a:b:c`
	Model with all two-way interactions (excluding the three-way interaction).

In addition to explaining the *in-sample variation* of the response variable *y*, regression allows us to generate *out-of-sample prediction* (i.e., extrapolation), given a fitted model.

$$\tilde{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \cdots \hat{\beta}_k x_{ki} \quad i = n+1, n+2, ...N$$

where $\tilde{y}_i$ is the predictive value conditional on new observations of $x_1,...,x_k$.

```
> x.new=data.frame(mom.hs=1, mom.iq=100)
> predict(fit.1, x.new, interval= "prediction", level=0.95)
```

The reason we need an interval for the predictive value is so simple – the future outcome is *uncertain* and all forecasts are likely to be *wrong*. Let's do some simulation.

```
> sigma=sd(residuals(fit.1))
> ytilda.fixed=coef(fit.1)[1]+coef(fit.1)[2]*1+coef(fit.1)[3]*100
> ytilda=ytilda.fixed+ rnorm(5000, 0, sigma)
> c(quantile(ytilda, 0.025), quantile(ytilda, 0.975))
```

Is the simulated interval close to the one generated from *predict*( )?

Below are a few tips for building regression models for prediction (Gelman & Hill, 2009)

1. Include all variables that you expect to be relevant and important.
2. Sometimes you can average or sum several inputs to create an index/score.
3. For inputs with large effects, consider including their interactions well.
4. If a predictor variable is not statistically significant with the expected sign, keep it.
5. If a predictor variable is statistically significant with the expected sign, you NEED it.
6. If a predictor variable is NOT statistically significant with the non-expected sign, drop it.
7. If a predictor variable is statistically significant with the non-expected sign, think hard!
8. DO NOT throw away observations (e.g., outliers) arbitrarily. You are losing information.

If some of you decide to conduct *regression-based prediction* in the term project, you will have to show how good your prediction is. Below are three standard performance metrics (*R^2* is a measure of *correlation*, not *accuracy*):

Mean error: $\text{ME} = \dfrac{1}{N-(n+1)+1}\sum_{i=n+1}^{N}(y_{i-}\tilde{y}_i)$

Root mean square error: $\text{RMSE} = \sqrt{\dfrac{1}{N-(n+1)+1}\sum_{i=n+1}^{N}(y_{i-}\tilde{y}_i)^2}$

Mean absolute percentage error: $\text{MAPE} = \dfrac{100}{N-(n+1)+1}\sum_{i=n+1}^{N}\dfrac{|\,y_{i-}\tilde{y}_i\,|}{y_i}$

ME should be close to zero for un-biased prediction. RMSE expresses the magnitude of the forecast error in the units of the response variable *y*. MAPE expresses the forecast error in percentage terms and MAPE should not be used for a response that is close to zero as the division by a small number becomes unstable (Ledolter, 2013).

6.4 Variable transformation

I have tried really hard to sharpen the focus of this lecture by only showing you the nuts and bolts of linear regression. In the last section of lecture 7 I want to talk about practical issues in transforming variables. Let's revisit the *kidiq* model.

```
> fit.2=lm(kid.score ~ mom.hs + mom.iq + mom.hs:mom.iq)
> summary(fit.2)
```

The coefficient on *mom.hs* is 51.3 – does this mean that kids with mothers who graduated from high school do, on average, 51.3 points better on their tests?

NO! The model includes an interaction term, and 51.3 is the predicted difference for kids that differ in *mom.hs*, among those with *mom.iq*=0. Since *mom.iq* can hardly be close to zero, the comparison at *zero*, and the coefficient 51.3 is meaningless.

We can simplify the interpretation by first subtracting the mean of each input variable:

```
> c.mom.hs=mom.hs-mean(mom.hs)
> c.mom.iq=mom.iq-mean(mom.iq)
> fit.3=lm(kid.score ~ c.mom.hs + c.mom.iq + c.mom.hs:c.mom.iq)
```

The R^2 and the coefficient of the interaction term are invariant with linear transformation of variables. But the main effects are now interpretable since each main effect corresponds to a predictive difference with the other input at its *average*.

An alternative is to leave the binary input (i.e., *mom.hs*) as it is. Then rescale the continuous variable (i.e., mom.iq) using *z*-scores (see lecture 5).

```
> z.mom.iq=(mom.iq-mean(mom.iq))/(sd(mom.iq))
```

As for models with NO interactions, a procedure that is equivalent to centering and rescaling is to leave the independent variables as is, and rescale estimated coefficients by multiplying each $\hat{\beta}$ by two times the standard deviation of the corresponding *x*. This gives a sense of the importance of each variable after controlling for others in the linear model.

The last type of variable transformation I want to mention is logarithmic transformations, For instance, if you believe the relationship among variables is multiplicative (*y*=*abc*) rather than additive (*y*=*a*+*b*+*c*), it makes sense to take natural log such that $\log y = \log a + \log b + \log c$.

Example: Estimating the Cobb-Douglas production function – $Y = \alpha L^{\beta} K^{\gamma}$.

For a linear regression model – $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, there are three types of transformations involving logarithms. The first type is the *linear-log* model

$$y_i = \beta_0 + \beta_1 \log(x_i) + \varepsilon_i$$

where the interpretation of estimated β_1 will be different from the linear case. We know that

$$\log(x) + 1 = \log(x) + \log(e) = \log(x \cdot e)$$

Adding 1 unit to log(*x*) means multiplying *x* itself by $e \approx 2.72$. Hence, the estimated β_1 is the expected change in *y* when *x* is multiplied by *e*. The second type is the *log-linear* model

$$\log(y_i) = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Adding 1 unit to *x* now corresponds to β_1 units increase in log(*y*). That is, a 1-unit increase in *x* makes the value of *y* multiplied e^{β_1}. The last type is the *log-log* model

$$\log(y_i) = \beta_0 + \beta_1 \log(x_i) + \varepsilon_i$$

Where the interpretation of the estimated β_1 is simply a combination of the two cases above. Multiplying *x* by *e* will multiply *y* by e^{β_1}.

In practice, it makes sense to take log of the response variable *y* if all observed outcomes are positive. The log transformation oftentimes will make skewed data look more like normally distributed data. If a variable has a narrow dynamic range (i.e., the ratio between the high and low values is close to 1), log transformations will not make much of a difference. When both independent and dependent variables have a large amount of variation (potentially caused by potential outliers), working with logarithm would be likely to improve model fitness.

6.5 Model diagnostics

We have talked very little about *model diagnostics* due to time constraints. Model diagnostics enable us to detect potential problems that may occur when we fit a linear regression data to a particular data set. The following contains some of the most common problems

1. Non-linearity of the (x_1, x_2,…, x_k) – *y* relationships
2. Correlation of error terms
3. Collinearity
4. Outliers
5. High-leverage points
6. Non-constant variance of error terms
7. Non-normal errors
8. Endogeneity

With the aid of two simulation experiments, below I will illustrate the aftermath of naively using OLS and ignoring 6 and 7.

Experiment 1: *Heteroskedasticity*

```
> set.seed(123)
> b0=0.2
> b1=0.5
> n=1000
> x=runif(n, -1, 1)
> S=1000
> gamma=1.5
> par.est=matrix(NA, nrow=S, ncol=2)
> for(s in 1:S){
>     y1=b0+b1*x+rnorm(n, 0, exp(x*gamma))
>     model1=lm(y1~x)
>     sigma=summary(model1)$sigma
>     y2=b0+b1*x+rnorm(n, 0, sigma)
>     model2=lm(y2~x)
>     par.est[s, 1]=model1$coef[2]
>     par.est[s, 2]=model2$coef[2]
> }
```

Assess OLS estimates under constant and non-constant error variances

```
> dev.new(width=8, height=5)
> plot(density(par.est[,2]), lwd=2, xlab= “”)
> lines(density(par.est[,1]),col= ‘red’, lwd=2, lty=2)
> legend(0.12,3,c("Homoskedastic","Heteroskedastic"),
lty=c(1,2),lwd=c(2,2),bty="n",cex=1.1,col=c('black','red'))
> mtext(expression(paste(hat(beta[1]))),side=1,line=2,cex=1.1)
> abline(v=0.5,col='green')
```

Experiment 2: *Non-normal errors*

```
> set.seed(123)
> b0=0.2
> b1=0.5
> n=1000
> x=runif(n, -1, 1)
> S=1000
> library(VGAM)
> par.est=matrix(NA, nrow=S, ncol=2)
> for(s in 1:S){
>       y1=b0+b1*x+rnorm(n, 0, 1)
>       model1=lm(y1~x)
>       y2=b0+b1*x+rlaplace(n, 0, 1)
>       model2=lm(y2~x)
>       par.est[s, 1]=model1$coef[2]
>       par.est[s, 2]=model1$coef[2]
> }
```

Assess OLS estimates under normal and Laplace errors

```
> dev.new(width=8, height=5)
> plot(density(par.est[, 1]), lwd=2, xlab= “”)
> lines(density(par.est[, 2]),col= ‘red’, lwd=2, lty=2)
> legend(0.3, 0.6,c("Normal Errors", "Laplace Errors"),
lty=c(1,2),lwd=c(2,2),bty="n",cex=1.1,col=c('black', 'red'))
> mtext(expression(paste(hat(beta[1]))),side=1,line=2,cex=1.1)
> abline(v=0.5,col='green')
```

Hopefully the two simulation experiments help you better understand the consequences of directly applying OLS regression to data when modeling assumptions are violated. You will be asked to do similar exercises for other OLS assumptions in HW4.

To sum up, the aforementioned regression modeling techniques introduced in lecture 6 are not exhaustive but provide you with the set of elements for data analytics. The models are appropriate for *cross-sectional data* and *single-equation modeling*. For *panel data* and/or *systems-of-equation modeling*, you will need to apply more sophisticated methods, which are beyond the scope of this course (I will be happy to offer an advanced course if you really enjoy being tortured by me, LOL). Nonetheless, those materials are easy to find and study given you already have a solid grasp of regression through this lecture (do you?).

I truly hope you will have more confidence in performing regression analysis from now on.