Tutorial 10: Regression Diagnostics II

Johannes Karreth

RPOS 517, Day 10

This tutorial shows you:

- how to diagnose non-normally distributed errors
- how to diagnose heteroskedasticity
- how to diagnose collinearity
- how to transform variables
- how to obtain robust standard errors

Note on copying & pasting code from the PDF version of this tutorial: Please note that you may run into trouble if you copy & paste code from the PDF version of this tutorial into your R script. When the PDF is created, some characters (for instance, quotation marks or indentations) are converted into non-text characters that R won't recognize. To use code from this tutorial, please type it yourself into your R script or you may copy & paste code from the *source file* for this tutorial which is posted on my website.

Note on R functions discussed in this tutorial: I don't discuss many functions in detail here and therefore I encourage you to look up the help files for these functions or search the web for them before you use them. This will help you understand the functions better. Each of these functions is well-documented either in its help file (which you can access in R by typing `?ifelse`, for instance) or on the web. The *Companion to Applied Regression* (see our syllabus) also provides many detailed explanations.

As always, please note that this tutorial only accompanies the other materials for Day 10 and that you are expected to have worked through the reading for that day before tackling this tutorial.

This week's tutorial focuses on violations of the OLS assumptions. As a reminder, the assumptions you've encountered are:

- [A1] Linear relationships between each x and y
- [A2] Constant error variance
- [A3] Normality of the errors: $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- [A4] Independence of observations
- [A5] No correlation b/w x and ε: $\text{cor}(x, \varepsilon) = 0$

For violations of most of these assumptions, investigating the residuals of a regression model is a good start. Hence this tutorial will focus on residuals quite a bit. Residuals (or errors) are the differences between the observed values of y and the predicted, or fitted, values for y, also known as $\hat{y}$:

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

You recall that we assume these residuals to be distributed normally with a mean of 0 and a *constant* variance. That is, the spread of the residuals around the predicted values is the same across the range of predicted values.

Non-normally distributed errors

You read in section 12.1 of *AR* that non-normally distributed errors do not compromise the validity of OLS estimates in large samples, but they may give you a reason to improve your model. There are two simple graphical ways to diagnose whether the residuals are not normally distributed, and you're already familiar with these methods from the tutorial for Day 3.

For an example, we'll use data from the Canadian Survey of Labour and Income Dynamics, for the province of Ontario. Observations are individuals from Ontario who responded to the 1994 wave of the survey. These data are provided as part of the "car" package, but I uploaded them to my website as well. The following variables are included:

```
slid.dat <- read.csv("http://www.jkarreth.net/files/car_SLID.csv")
summary(slid.dat)
```

```
##      wages          education          age              sex      
##  Min.   : 2.300   Min.   : 0.00   Min.   :16.00   Female:3880  
##  1st Qu.: 9.235   1st Qu.:10.30   1st Qu.:30.00   Male  :3545  
##  Median :14.090   Median :12.10   Median :41.00                
##  Mean   :15.553   Mean   :12.50   Mean   :43.98                
##  3rd Qu.:19.800   3rd Qu.:14.53   3rd Qu.:57.00                
##  Max.   :49.920   Max.   :20.00   Max.   :95.00                
##  NA's   :3278     NA's   :249                                  
##    language   
##  English:5716  
##  French : 497  
##  Other  :1091  
##  NA's   : 121  
##                
##                
##                
```

This dataset contains 5 variables:

Variable	Description
`wages`	Composite hourly wage rate from all jobs in Canadian dollars.
`education`	Number of years of schooling.
`age`	Age in years
`sex`	A factor with levels: Female, Male.
`language`	A factor with levels: English, French, Other.

As often before, we'd like to predict hourly wages (`wages`) with education and, in this case, other variables. To do so, we can fit an OLS regression model as usual:

```
mod <- lm(wages ~ education + age + sex, data = slid.dat)
summary(mod)
```

```
## 
## Call:
## lm(formula = wages ~ education + age + sex, data = slid.dat)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -26.111  -4.328  -0.792   3.243  35.892 
## 
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.905243   0.607771  -13.01   <2e-16 ***
## education    0.918735   0.034514   26.62   <2e-16 ***
## age          0.255101   0.008634   29.55   <2e-16 ***
## sexMale      3.465251   0.208494   16.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.602 on 4010 degrees of freedom
##   (3411 observations deleted due to missingness)
## Multiple R-squared:  0.2972, Adjusted R-squared:  0.2967
## F-statistic: 565.3 on 3 and 4010 DF,  p-value: < 2.2e-16
```

To investigate the residuals of this regression, we can use the `resid()` or (equivalently) `residuals()` functions.

```
summary(resid(mod))
```

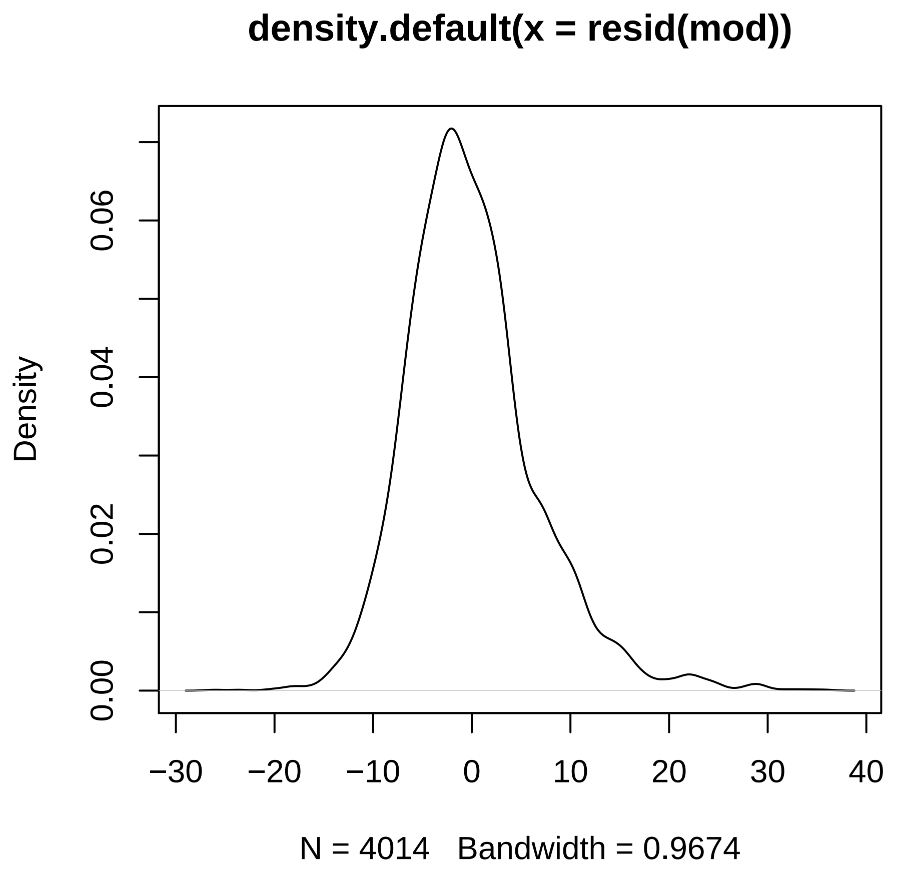
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -26.1100  -4.3280  -0.7925   0.0000   3.2430  35.8900
```

But these quantities don't tell us much about the *distribution* of the residuals.

Density plot

Hence, a density plot is often useful. You're already familiar with density plots (and histograms) from Day 3.

```
plot(density(resid(mod)))
```

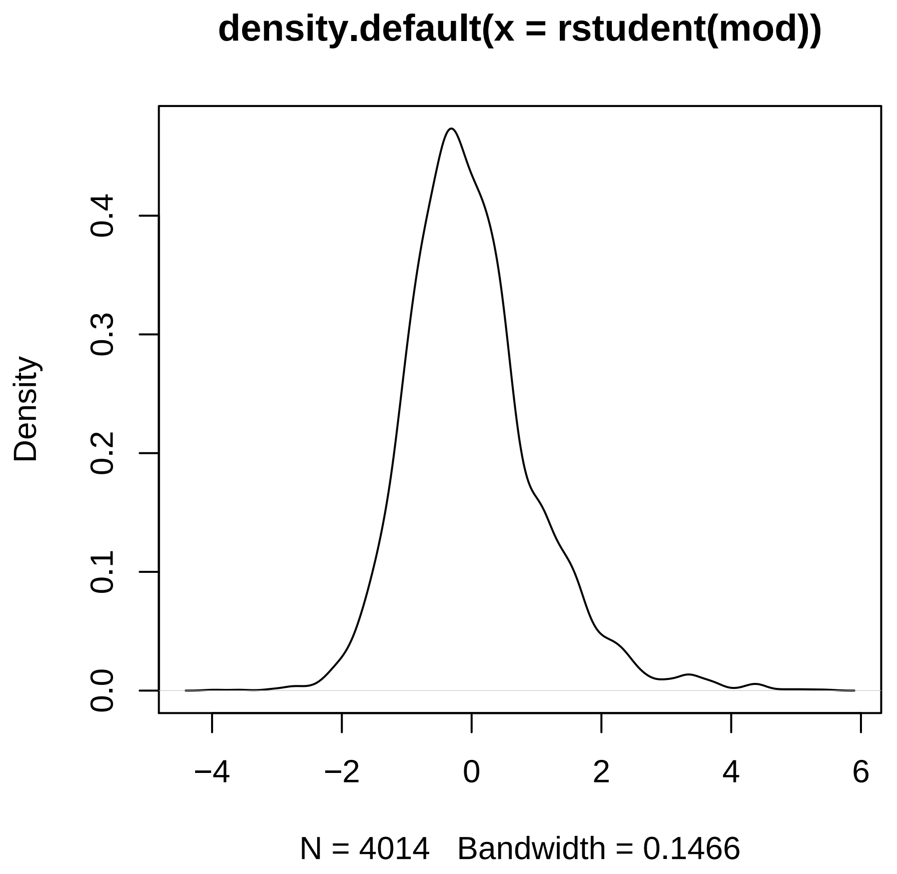


Last week (Day 9), you learned that normalizing/standardizing residuals often is useful because studentized residuals because they follow a t-distribution. You can obtain the studentized residuals using the `rstudent()` function.

```
summary(rstudent(mod))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -3.968000 -0.656000 -0.120100  0.000099  0.491500  5.460000
```

```
plot(density(rstudent(mod)))
```

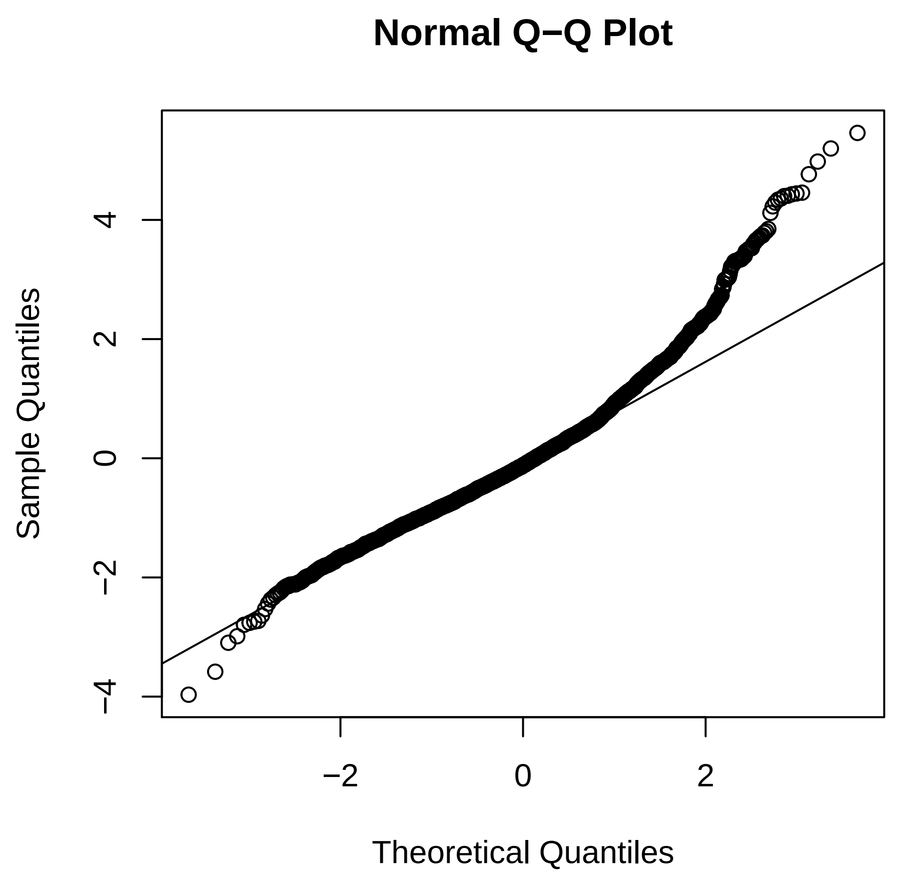


With your knowledge of the normal (and t-) distributions, you can easily tell when the (studentized) residuals are not normally distributed. In this case, you'll note a slight "dent" in the right half of this density plot.

Quantile-comparison plots for residuals

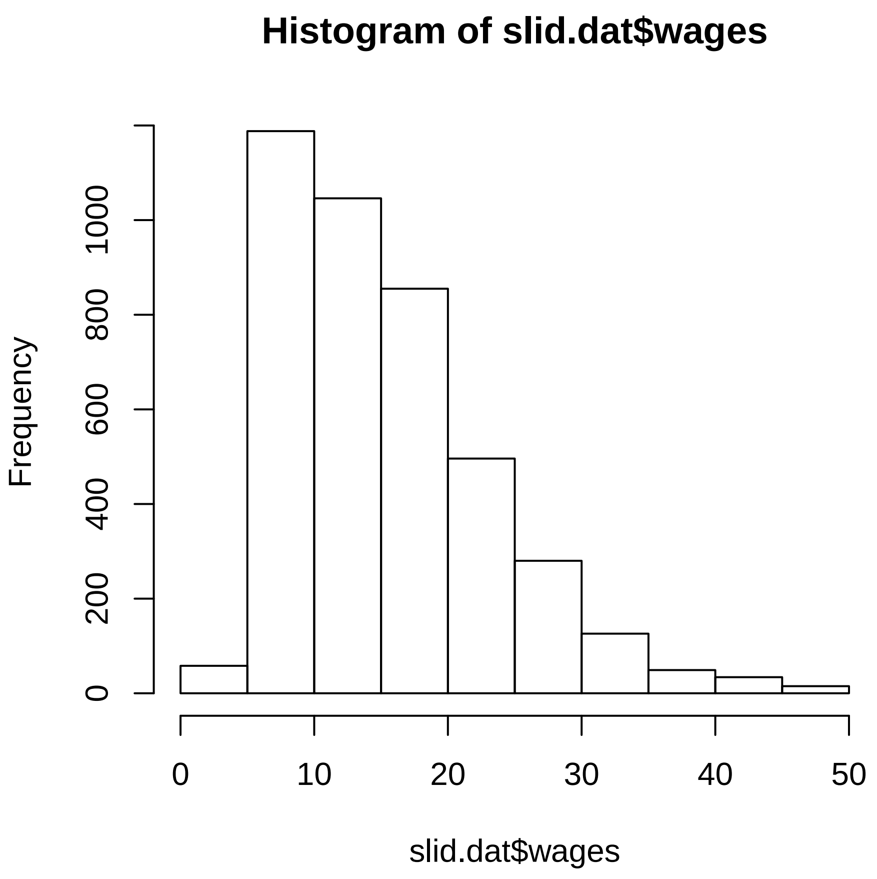
A second graphical test for normality you're already familiar with plots the actual percentiles of the residuals against theoretical percentiles of a normal distribution. The familiar `qqnorm()` function does this for you, in combination with the `qqline()` function.

```
qqnorm(rstudent(mod))
qqline(rstudent(mod))
```



See section 12.1 in *AR* and the tutorial from Day 3 for more detailed guidance on how to interpret these figures. In the quantile-comparison plot above, you probably notice that the tails of the distribution of the residuals deviates from the normal line, especially on the upper end. One reason for this is that our model is not predicting observations on the upper end of y (wages) well. As you'll recall, wages or incomes are often heavily skewed. Let's verify this:

```r
hist(slid.dat$wages)
```



Transforming variables

In this case, it would be useful to create a transformed version of the outcome variable, `wages`, using the natural log. Recall that the log of 0 is undefined, though, so you must check if `wages` contains any values of 0 first. If it does, you can add a very small positive value (depending on the scale of the variable).[1] In this case, `wages` does not contain 0s, so taking the log without any changes is not problematic.

```
summary(slid.dat$wages)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##   2.300   9.235  14.090  15.550  19.800  49.920    3278
```

```
slid.dat$wages.ln <- log(slid.dat$wages)
```

Fitting the model with the logged outcome variable instead also returns a less "concerning" quantile-comparison plot, at least at the upper tail:

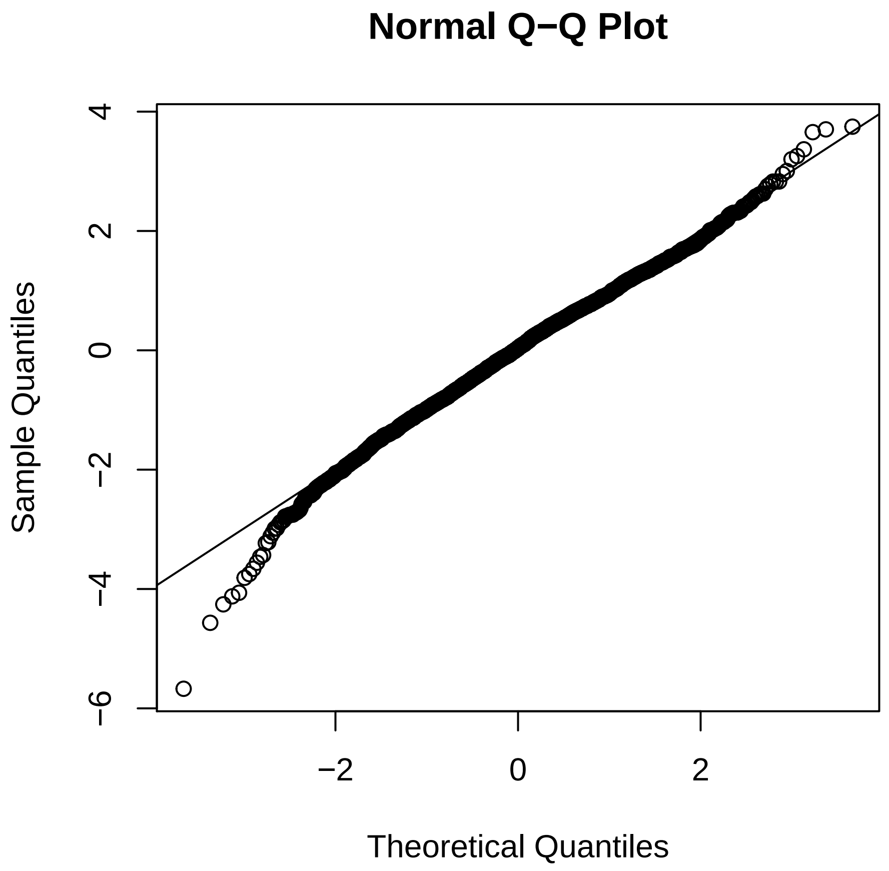
```
mod.ln <- lm(wages.ln ~ education + age + sex, data = slid.dat)
summary(mod.ln)
```

```
##
```

[1] A good discussion of this practice can be found at http://stats.stackexchange.com/questions/1444/how-should-i-transform-non-negative-data-including-zeros.

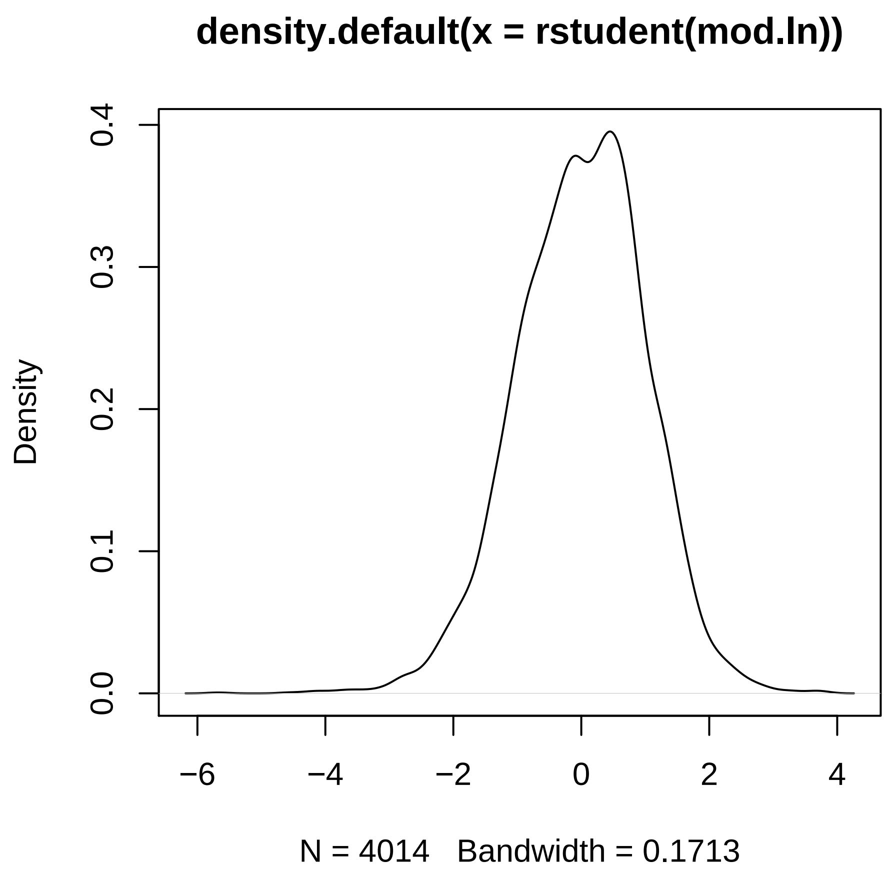
```
## Call:
## lm(formula = wages.ln ~ education + age + sex, data = slid.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.36252 -0.27716  0.01428  0.28625  1.56588
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.1168632  0.0385480   28.97   <2e-16 ***
## education   0.0552139  0.0021891   25.22   <2e-16 ***
## age         0.0176334  0.0005476   32.20   <2e-16 ***
## sexMale     0.2244032  0.0132238   16.97   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4187 on 4010 degrees of freedom
##   (3411 observations deleted due to missingness)
## Multiple R-squared:  0.3094, Adjusted R-squared:  0.3089
## F-statistic: 598.9 on 3 and 4010 DF,  p-value: < 2.2e-16
```

```
qqnorm(rstudent(mod.ln))
qqline(rstudent(mod.ln))
```



A density plot of the studentized residuals reveals the same, as the “dent” in the right has now disappeared:

```r
plot(density(rstudent(mod.ln)))
```



However, note that your interpretation of these estimates has changed with the transformation of the outcome variable. Each β now expresses the "effect" of a one-unit increase in x on the log transformation of y. This requires some extra work and care on interpreting and presenting your results.

Nonconstant error variance (heteroskedasticity)

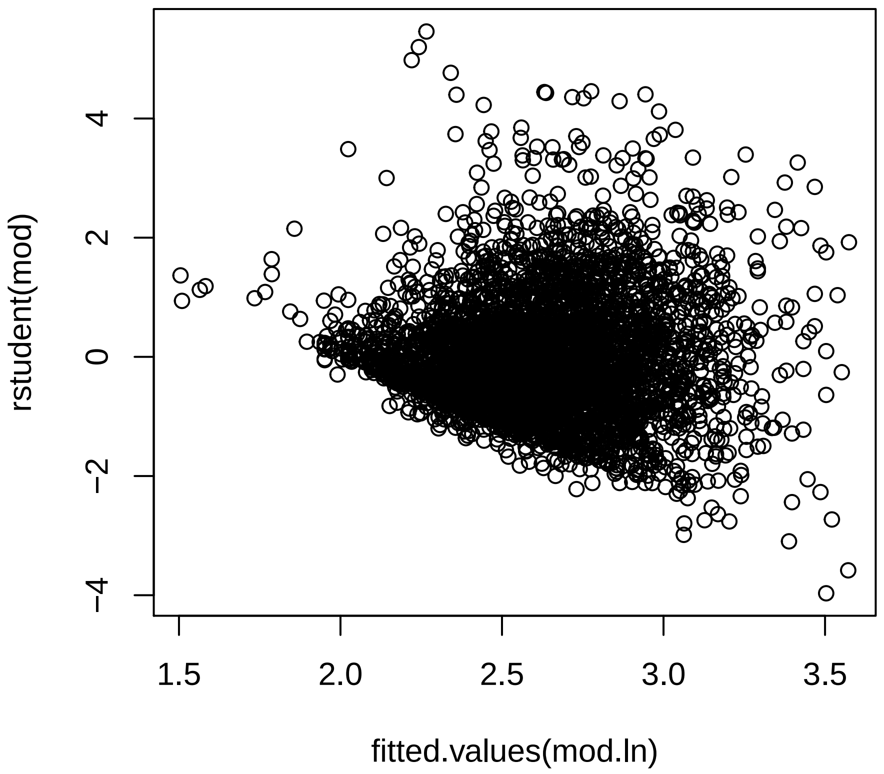
A far bigger concern about inferences from OLS models comes from violations of the assumption of constant error variance. Under heteroskedasticity, the standard errors from OLS are inaccurate. You'll find more details about this problem on p. 272 and pp. 276-7 in *AR*. Broadly speaking, you should consider heteroskedasticity as an indicator for a mis-specified model: your model may be missing an important part of the explanation for variation in your outcome variable y.

Residual plots

You're already familiar with residual plots from bivariate regression: you plotted residuals (on the y-axis) against the predictor (on the x-axis). With multiple regression, you can do the same for each predictor, and you can also plot the residuals against the *fitted values* of y. My writing here supplements what you read in section 12.2.1 in *AR*, using similar data and the same empirical model.

The latter is easily accomplished using the `resid()` (or `rstudent()`) and `fitted.values()` functions. I'm returning to our model with the log-transformed version of the `wages`.

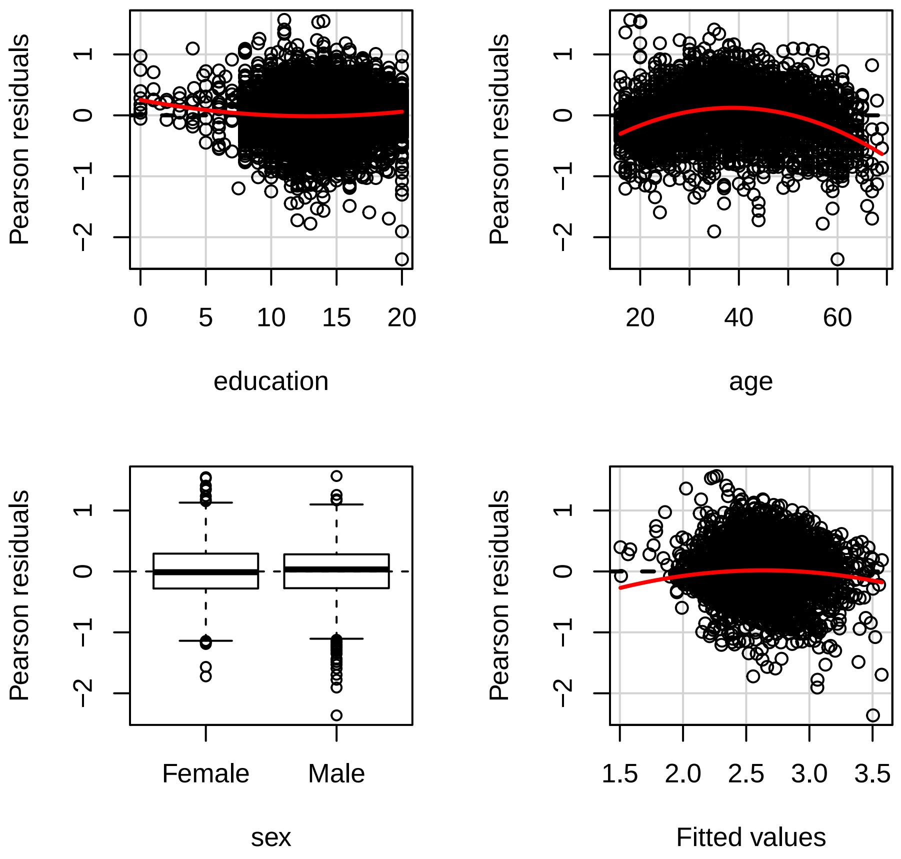
```r
plot(x = fitted.values(mod.ln), y = rstudent(mod))
```



We see here that the variance of the residuals is smaller at lower fitted values and then increases somewhat. This may indicate that the residuals are themselves a function of the fitted values. You should use this as a prompt to investigate this model further.

In addition to plotting residuals against fitted values, you can use the convenient `residualPlots()` function from the "car" package to plot residuals against each individual predictor. The default function usually gives you a useful diagnostic plot, but see the help file at `?residualPlots` for customization options. This function also returns test statistics for the curvatore of the fitted line in each residual plots involving continuous predictors. Notice that the plot on the bottom right reproduces the plot of residuals vs. fitted values that you just made by hand.

```r
library(car)
residualPlots(mod.ln)
```



```
##              Test stat Pr(>|t|)
## education        3.751        0
## age            -21.621        0
## sex                 NA       NA
## Tukey test      -3.931        0
```

Nonlinearity

A very fundamental assumption of linear regression (our assumption "A1") is that the data-generating process is linear: each predictor x is linearly related to y. This means that a one-unit increase in x is associated with *the same* increase in y regardless of the value of x. Revisiting our original model:

```
summary(mod)
```

```
##
## Call:
## lm(formula = wages ~ education + age + sex, data = slid.dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -26.111  -4.328  -0.792   3.243  35.892
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.905243   0.607771  -13.01   <2e-16 ***
## education    0.918735   0.034514   26.62   <2e-16 ***
## age          0.255101   0.008634   29.55   <2e-16 ***
## sexMale      3.465251   0.208494   16.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.602 on 4010 degrees of freedom
##   (3411 observations deleted due to missingness)
## Multiple R-squared:  0.2972, Adjusted R-squared:  0.2967
## F-statistic: 565.3 on 3 and 4010 DF,  p-value: < 2.2e-16
```

We would conclude that one additional year of education is associated with about 1 Canadian dollar (0.91) additional hourly wage. Moving from 5 years of education to 6 years yields the same increase in wage as moving from 19 to 20 years of education.

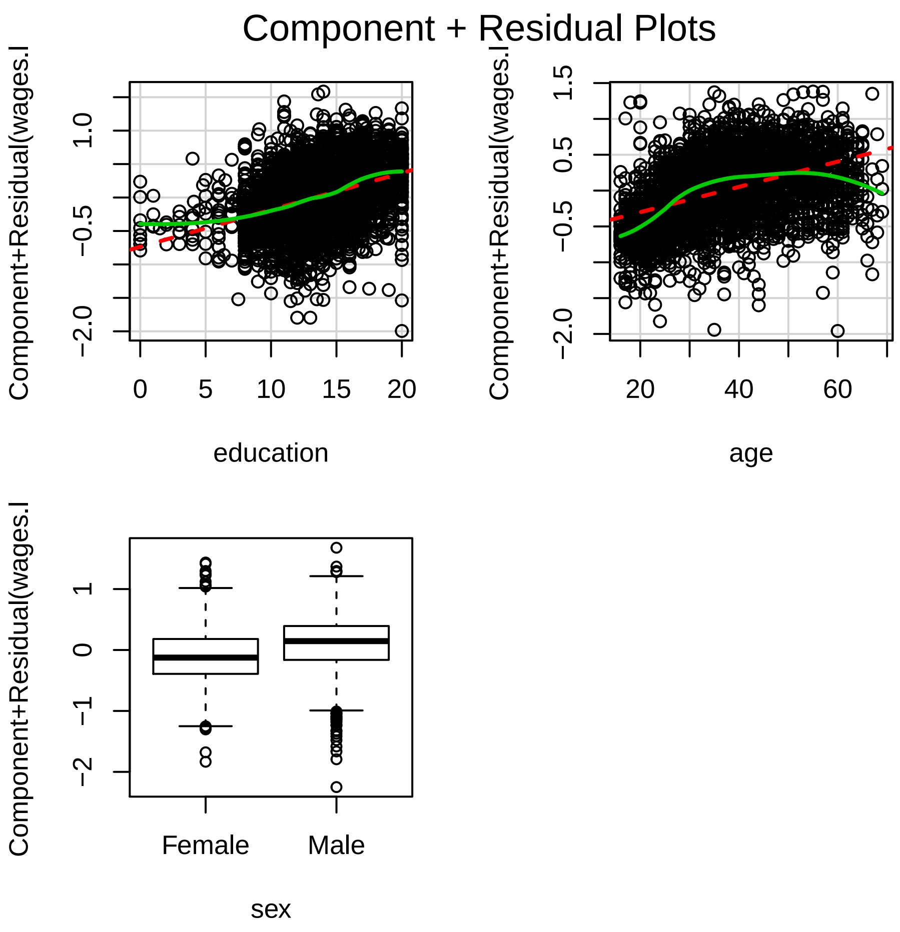
It should be obvious that this assumption is often not realistic. For instance, you could think of the idea of diminishing returns.[2]

Component-plus-residual plots

To check this assumption, you can create component-plus-residual plots as described in section 12.3.1 of *AR*. The "car" package offers the `crPlots()` function for this purpose. Let's return to our model with the log-transformed `wages` outcome variable and create component-plus residual plots for the continuous predictors.

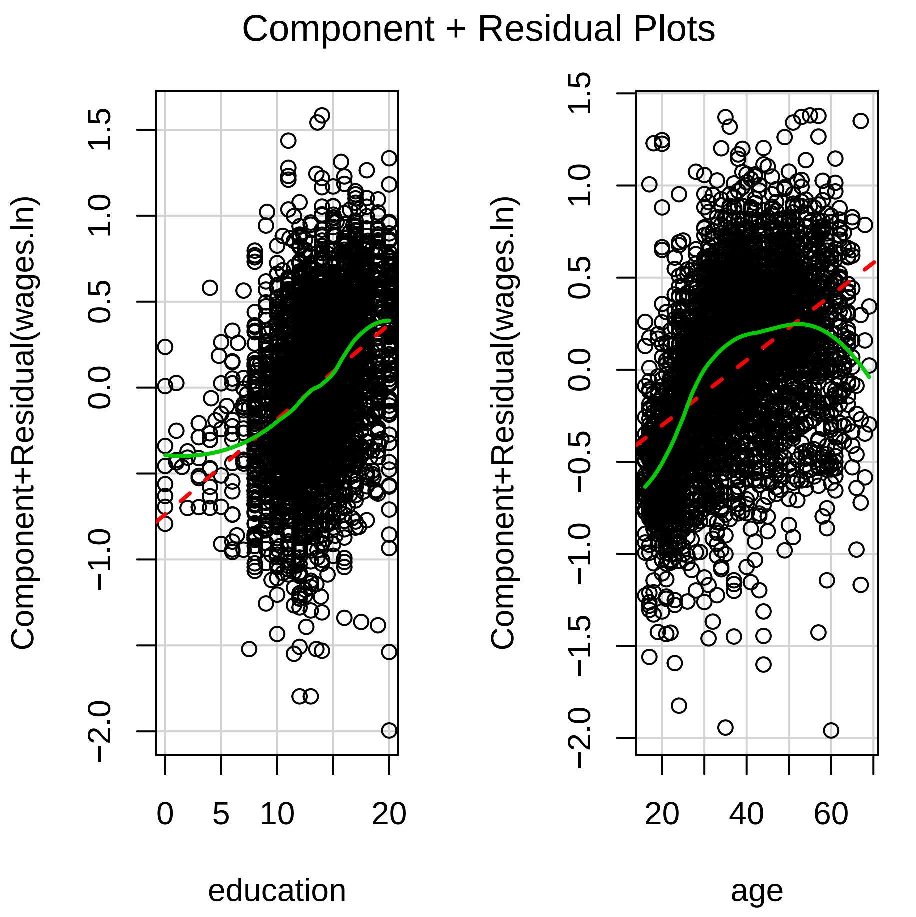
```r
crPlots(mod.ln)
```

[2]See, for instance, http://www.investopedia.com/terms/l/lawofdiminishingmarginalreturn.asp.



I can also use the `terms =` argument to avoid a component-residual plot for the `sex` dummy variable.

```r
crPlots(model = mod.ln, terms = ~ . - sex)
```



Please refer to section 12.3.1 of *AR* for a thorough discussion of this example.

Collinearity

We've already encountered the concept of collinearity (AKA multicollinearity). You recall from the formula for the standard error of $\hat{\beta}$ in multiple regression that strong associations between predictors will increase standard errors, and therefore increase the probability of a type-II error (failing to reject a false null hypothesis).

Variance-inflation factors

You could of course simply examine the correlation between predictors, but that ignores the fact that **multi**collinearity depends upon the association between possibly more than two predictors, as you saw in our calculation of R_j^2, the R^2-value from the regression of one x_j on all other predictors. Therefore, a common diagnostic is the *variance-inflation factor* (VIF). The VIF for predictor x_j is a transformation of R_j^2:

$$\text{VIF}_j = \frac{1}{1 - R_j^2}$$

In R, the "car" package offers the function `vif()` to calculate the VIF for each predictor. Returning to our example:

```
vif(mod.ln)
```

```
## education       age       sex
##  1.012179  1.011613  1.000834
```

AR wisely does not provide "critical values" for VIFs, and you should read *AR*'s discussion of multicollinearity carefully. Multicollinearity is often less of a problem than the suggested "fixes". Let's briefly investigate how the VIFs look for a model with data that we knowingly created using highly correlated predictors. To this end, we make $x2$ a function of $x1$ and some random noise:

```
set.seed(123)
n.sample <- 200
x1 <- rnorm(n.sample, 2, 5)
x2 <- x1 + rnorm(n.sample, 0, 1)
x3 <- runif(n.sample, min = -10, max = 10)
a <- 0.2
b1 <- 1.1
b2 <- 0.2
b3 <- -0.7
e <- rnorm(n.sample, 0, 10)
y <- a + b1 * x1 + b2 * x2 + b3 * x3 + e
mod.sim <- lm(y ~ x1 + x2 + x3)
summary(mod.sim)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.0679  -6.7968  -0.0854   6.8281  25.6651
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.836708   0.759384  -1.102   0.2719
## x1           1.343254   0.717766   1.871   0.0628 .
## x2          -0.001452   0.705987  -0.002   0.9984
## x3          -0.737298   0.121728  -6.057 6.96e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.903 on 196 degrees of freedom
## Multiple R-squared:  0.3842, Adjusted R-squared:  0.3748
## F-statistic: 40.76 on 3 and 196 DF,  p-value: < 2.2e-16
```

```
vif(mod.sim)
```

```
##        x1        x2        x3
## 23.248488 23.231691  1.003858
```

```r
sqrt(vif(mod.sim))
```

```
##       x1       x2       x3 
## 4.821669 4.819926 1.001927
```

```r
mod.sim2 <- lm(y ~ x1 + x3)
summary(mod.sim2)
```

```
## 
## Call:
## lm(formula = y ~ x1 + x3)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -27.0696  -6.7984  -0.0867   6.8293  25.6630 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  -0.8368     0.7565  -1.106     0.27    
## x1            1.3418     0.1486   9.030  < 2e-16 ***
## x3           -0.7373     0.1213  -6.080 6.14e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 9.878 on 197 degrees of freedom
## Multiple R-squared:  0.3842, Adjusted R-squared:  0.378 
## F-statistic: 61.46 on 2 and 197 DF,  p-value: < 2.2e-16
```

```r
vif(mod.sim2)
```

```
##       x1       x3 
## 1.001483 1.001483
```

Robust standard errors

You're learning today (again) that non-constant error variance affects foremost the standard errors of a regression model. This would suggest that if standard errors can be corrected, your OLS estimates are still useful (since they are unbiased anyway, and with corrected standard errors, your inferences and significance tests of $\hat{\beta}$ are also "corrected".) Therefore it might not come as a surprise that econometricians have suggested corrections for the standard error in the presence of heteroskedasticity. One of these corrections goes by the name of the "heteroskedasticity-consistent standard error" or "Huber-White standard error" or "robust standard error." Sometimes different authors refer to different implementations of this idea, but the logic is usually the same. It is explained in section 12.2.3 of *AR* (which was not required for today).

Before we advance, I'd like to emphasize that robust standard errors are often not a good "fix." Heteroskedasticity often suggests deeper problems with your model; perhaps the model is missing an important predictor, or the outcome or one of the predictor variables should be transformed. Today's applied reading, King and Roberts (2015) "How Robust Standard Errors Expose Methodological Problems They Do Not Fix, and What to Do About it" explains this in great detail.

The idea behind robust standard errors is to add to the sampling variance of the coefficient, $V(\hat{\beta})$ a correction that takes into account the non-constant variance of the residuals ε_i. In section 12.2.3 of *AR* you see the equation behind this. In the calculation of $V(\hat{\beta})$, robust standard errors use diag$\{E_1^2/(1-h_1)^2, \ldots E_n^2/(1-h_n)^2\}$ instead of diag$\{E_1^2, \ldots E_n^2\}$. In other words, the standard errors of the coefficients are adjusted for the hat-values (recall hat-valeus from Day 8).

I'm providing you with a small R function, `robSE()`, to calculate robust standard errors for OLS estimates. I'm providing this function on GitHub. You can either copy and paste it into R from there, and then use it with an `lm` object. You have to paste the function once in your R session before you use it. Alternatively, you can read it in using the `source_url()` function from the "devtools" package. I show in this demo https://github.com/jkarreth/JKmisc/blob/master/robclSE.demo.R how that's done.

For this tutorial, I paste in the `robSE()` function after copying it from my GitHub page.

```r
robSE <- function(mod){

  require(lmtest, quiet = TRUE)

    X.mat <- model.matrix(mod) # note that X includes 1 for the intercept
    y <- mod$mod[1] # Response variable
    e <- resid(mod) # Vector of residuals
    n <- dim(X.mat)[1] # Number of observations
    k <- dim(X.mat)[2] # Number of predictors (including intercept)

    vcov.mat <- vcov(mod)   # Variance-covariance matrix
    SE <- sqrt(diag(vcov.mat)) # Standard errors
    E.mat <- matrix(0, ncol = n, nrow = n)  # n-by-n Matrix of residuals
    diag(E.mat) <- e^2  # Squared residuals in the diagonal

    sum.mat <- t(X.mat) %*% E.mat %*% X.mat

    vcov.mat.rob <- abs(solve(t(X.mat) %*% X.mat) %*% sum.mat %*% solve(t(X.mat) %*% X.mat))

    rob.SE <- sqrt(diag(vcov.mat.rob))
    ## Add the degrees-of-freedom correction
    dfc <- sqrt(nrow(X.mat)) / sqrt(nrow(X.mat) - ncol(X.mat))
    rob.SE.dfc <- dfc * sqrt(diag(vcov.mat.rob))
    return(rob.SE.dfc)

    }
```

A couple of notes:

- The first line of the function indicates that this function takes only one argument, "mod." Here, this stands for an `lm()` model object.
- The third line indicates that this function requires the "lmtest" package. Please install this package once before using the `robSE()` function.
- The function then follows the discussion on p. 275 of *AR*. First, it extracts some elements from the model object
  - The matrix of predictors, `X.mat`
  - The outcome variable, `y`
  - The individual residuals or errors, `e`
  - The number of observations, `n`
  - The number of predictors, `k`
  - The variance-covariance matrix of the model, `vcov.mat`

    - The standard errors of the coefficients, expressed as the diagonal of the variance-covariance matrix, `SE`

- The element `vcov.mat.rob` mirrors exactly equation 12.6 on p. 275 of *AR*.
- `rob.SE` is the diagonal of the corrected variance-covariance matrix
- `rob.SE.dfc.` adds the degrees-of-freedom correction (skipped in *AR*)
- Lastly, `return(rob.SE.dfc)` simply prints the "robust" standard errors as the output of the function.

Try it ourself with our model of log-transformed wages:

```
summary(mod.ln)
```

```
##
## Call:
## lm(formula = wages.ln ~ education + age + sex, data = slid.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.36252 -0.27716  0.01428  0.28625  1.56588
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.1168632  0.0385480   28.97   <2e-16 ***
## education   0.0552139  0.0021891   25.22   <2e-16 ***
## age         0.0176334  0.0005476   32.20   <2e-16 ***
## sexMale     0.2244032  0.0132238   16.97   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4187 on 4010 degrees of freedom
##   (3411 observations deleted due to missingness)
## Multiple R-squared:  0.3094, Adjusted R-squared:  0.3089
## F-statistic: 598.9 on 3 and 4010 DF,  p-value: < 2.2e-16
```

```
robSE(mod.ln)
```

```
##  (Intercept)    education          age      sexMale
## 0.0386703770 0.0022252535 0.0005851386 0.0132366014
```

To put both "original" and "robust" standard errors side-by-side, you can use the `screenreg()` function, as explained in more detail at https://github.com/jkarreth/JKmisc/blob/master/robclSE.demo.R.

```
library(texreg)
screenreg(list(mod.ln, mod.ln),
  override.se = list(NA, robSE(mod.ln)),
  custom.model.names = c("Original SEs", "Robust SEs"),
  digits = 5
  )
```

```
##
## ===========================================
##              Original SEs    Robust SEs
## -------------------------------------------
## (Intercept)     1.11686 ***     1.11686 ***
##                (0.03855)       (0.03867)
## education       0.05521 ***     0.05521 ***
##                (0.00219)       (0.00223)
## age             0.01763 ***     0.01763 ***
##                (0.00055)       (0.00059)
## sexMale         0.22440 ***     0.22440 ***
##                (0.01322)       (0.01324)
## -------------------------------------------
## R^2             0.30942         0.30942
## Adj. R^2        0.30890         0.30890
## Num. obs.    4014            4014
## ===========================================
## *** p < 0.001, ** p < 0.01, * p < 0.05
```

You can see that the robust standard errors are pretty close to the original standard errors in this case.

Note: I'm introducing robust standard errors here without endorsement. But, depending on your subfield, you will encounter them in published work, and I want to make sure you see the mechanics of them and how to obtain them in R.