Week 10 R tutorial

Aaron Shaw

November 10, 2020

Contents

This week's R tutorial focuses on the basics of correlations and linear regression. I'll work with the `mtcars` dataset that comes built-in with R.

```
data(mtcars)
```

Correlations and covariance

Calculating correlation coefficients is straightforward: use the `cor()` function:

```
with(mtcars, cor(mpg, hp))
```

```
## [1] -0.7761684
```

All you prius drivers out there will be shocked to learn that miles-per-gallon is negatively correlated with horsepower.

The `cor()` function works with two variables or with more—the following generates a correlation matrix for the whole dataset!

```
cor(mtcars)
```

```
##             mpg        cyl       disp         hp        drat         wt
## mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684  0.68117191 -0.8676594
## cyl  -0.8521620  1.0000000  0.9020329  0.8324475 -0.69993811  0.7824958
## disp -0.8475514  0.9020329  1.0000000  0.7909486 -0.71021393  0.8879799
## hp   -0.7761684  0.8324475  0.7909486  1.0000000 -0.44875912  0.6587479
## drat  0.6811719 -0.6999381 -0.7102139 -0.4487591  1.00000000 -0.7124406
## wt   -0.8676594  0.7824958  0.8879799  0.6587479 -0.71244065  1.0000000
## qsec  0.4186840 -0.5912421 -0.4336979 -0.7082234  0.09120476 -0.1747159
## vs    0.6640389 -0.8108118 -0.7104159 -0.7230967  0.44027846 -0.5549157
## am    0.5998324 -0.5226070 -0.5912270 -0.2432043  0.71271113 -0.6924953
## gear  0.4802848 -0.4926866 -0.5555692 -0.1257043  0.69961013 -0.5832870
## carb -0.5509251  0.5269883  0.3949769  0.7498125 -0.09078980  0.4276059
##             qsec         vs          am       gear        carb
## mpg   0.41868403  0.6640389  0.59983243  0.4802848 -0.55092507
## cyl  -0.59124207 -0.8108118 -0.52260705 -0.4926866  0.52698829
```

```
## disp -0.43369788 -0.7104159 -0.59122704 -0.5555692  0.39497686
## hp   -0.70822339 -0.7230967 -0.24320426 -0.1257043  0.74981247
## drat  0.09120476  0.4402785  0.71271113  0.6996101 -0.09078980
## wt   -0.17471588 -0.5549157 -0.69249526 -0.5832870  0.42760594
## qsec  1.00000000  0.7445354 -0.22986086 -0.2126822 -0.65624923
## vs    0.74453544  1.0000000  0.16834512  0.2060233 -0.56960714
## am   -0.22986086  0.1683451  1.00000000  0.7940588  0.05753435
## gear -0.21268223  0.2060233  0.79405876  1.0000000  0.27407284
## carb -0.65624923 -0.5696071  0.05753435  0.2740728  1.00000000
```

Note that if you are calculating correlations with variables that are not distributed normally you should use `cor(method="spearman")` because it calculates rank-based correlations (look it up online for more details).

To calculate covariance, you use the `cov()` function:

```
with(mtcars, cov(mpg, hp))
```

```
## [1] -320.7321
```

While *OpenIntro* does not spend much time on either covariance or correlation, I highly recommend taking the time to review at least the corresponding Wikipedia articles and ideally another statistics textbook) to understand what goes into each one. [1] A key point to notice/understand is that covariance is calculated in such a way that the magnitude may vary depending on the scale of the variables involved. Correlation is not (every correlation coefficient falls between -1 and 1).

Fitting a linear model

Linear models are fit using the `lm()` command. As with `aov()`, the `lm()` function requires a formula as an input and is usually presented with a call to `summary()`. You can enter the formula directly in the call to `lm()` or define it separately. For this example, I'll regress `mpg` on a single predictor, `hp`:

```
model1 <- lm(mpg ~ hp, data=mtcars)

summary(model1)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
## hp          -0.06823    0.01012  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

Notice how much information the output of `summary()` gives you for a linear model! You have details about the residuals, the usual information about the coefficients, standard errors, t-values, etc., little stars

[1] Here are those corresponding Wikipedia articles: correlation and covariance.

corresponding to conventional significance levels, R^2 values, degrees of freedom, F-statistics (remember those?) and p-values for the overall model fit.

There's even more under the hood. Try looking at all the different things in the model object R has created:

```
names(model1)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

You can directly inspect the residuals using `model1$residuals`. This makes plotting and other diagnostic activities pretty straightforward:

```
summary(model1$residuals)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.7121 -2.1122 -0.8854  0.0000  1.5819  8.2360
```

More on that in a moment. In the meantime, you can also use the items generated by the call to `summary()` as well:

```
names(summary(model1))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
summary(model1)$coefficients
```

```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 30.09886054  1.6339210 18.421246 6.642736e-18
## hp          -0.06822828  0.0101193 -6.742389 1.787835e-07
```

CIs around coefficients and predicted values

There are also functions to help you do things with the model such as predict the fitted values for new (unobserved) data. For example, if I found some new cars with horsepowers ranging from 90-125, what would this model predict for the corresponding mpg for each car?

```
new.data <- data.frame(hp=seq(90,125,5))
predict(model1, new.data, type="response")
```

```
##        1        2        3        4        5        6        7        8
## 23.95832 23.61717 23.27603 22.93489 22.59375 22.25261 21.91147 21.57033
```

A call to predict can also provide standard errors around these predictions (which you could use, for example, to construct a 95% confidence interval around the model-predicted values):

```
predict(model1, new.data, type="response", se.fit = TRUE)
```

```
## $fit
##        1        2        3        4        5        6        7        8
## 23.95832 23.61717 23.27603 22.93489 22.59375 22.25261 21.91147 21.57033
##
## $se.fit
##         1         2         3         4         5         6         7         8
## 0.8918453 0.8601743 0.8303804 0.8026728 0.7772744 0.7544185 0.7343427 0.7172804
##
## $df
## [1] 30
```

```
##
## $residual.scale
## [1] 3.862962
```

Linear model objects also have a built-in method for generating confidence intervals around the values of β:

```
confint(model1, "hp", level=0.95) # Note that I provide the variable name in quotes
```

```
##          2.5 %     97.5 %
## hp -0.08889465 -0.0475619
```

Feeling old-fashioned? You can always calculate residuals or confidence intervals (or anything else) "by hand":

```
# Residuals
mtcars$mpg - model1$fitted.values
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive
##         -1.59374995         -1.59374995         -0.95363068         -1.19374995
##   Hornet Sportabout             Valiant          Duster 360           Merc 240D
##          0.54108812         -4.83489134          0.91706759         -1.46870730
##            Merc 230            Merc 280           Merc 280C          Merc 450SE
##         -0.81717412         -2.50678234         -3.90678234         -1.41777049
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental
##         -0.51777049         -2.61777049         -5.71206353         -5.02978075
##   Chrysler Imperial            Fiat 128         Honda Civic      Toyota Corolla
##          0.29364342          6.80420581          3.84900992          8.23597754
##       Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28
##         -1.98071757         -4.36461883         -4.66461883         -0.08293241
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus Europa
##          1.04108812          1.70420581          2.10991276          8.01093488
##      Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
##          3.71340487          1.54108812          7.75761261         -1.26197823
```

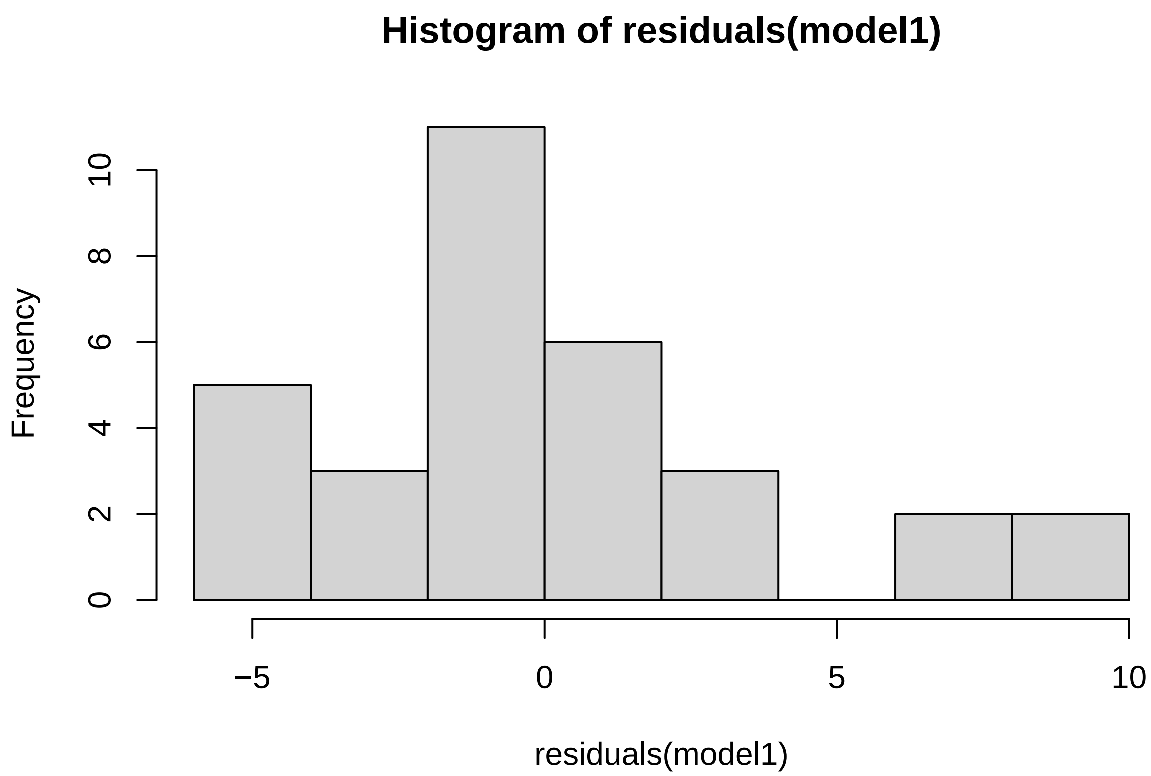
```
# 95% CI for the coefficient on horsepower
est <- model1$coefficients["hp"]
se <- summary(model1)$coefficients[2,2]

est + 1.96 * c(-1,1) * se
```

```
## [1] -0.08806211 -0.04839444
```

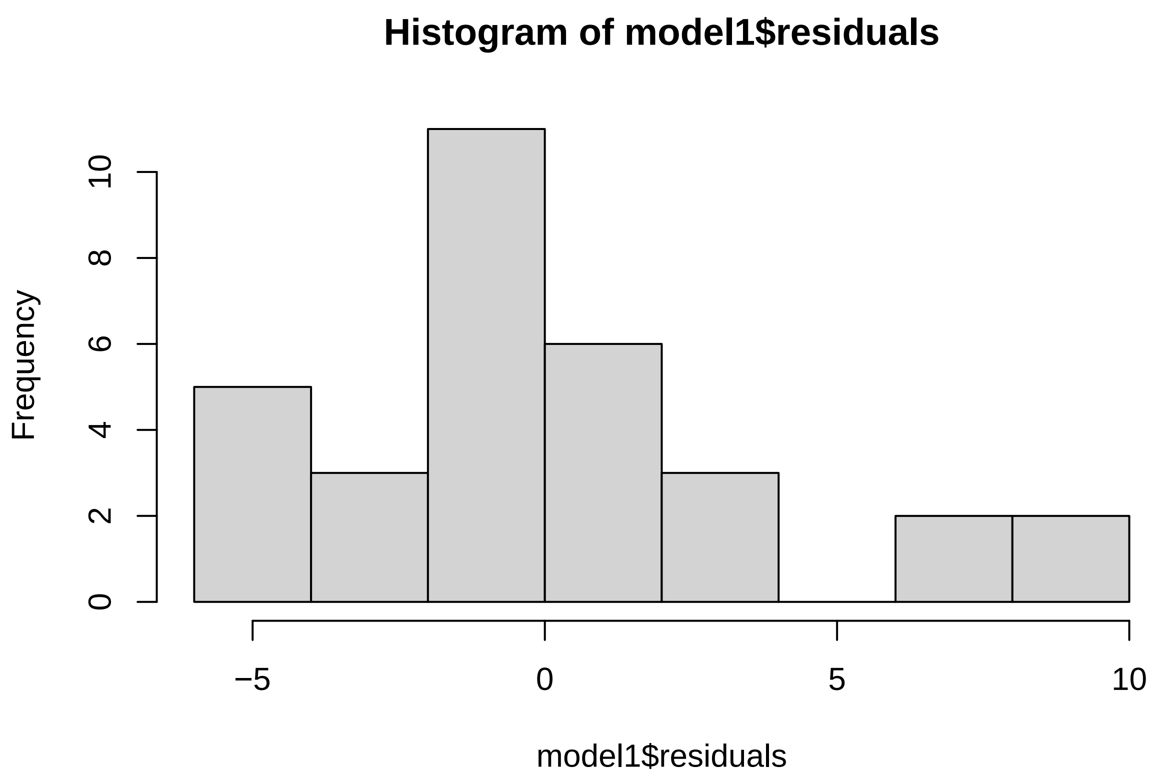
Plotting residuals

You can generate diagnostic plots of residuals in various ways:

```
hist(residuals(model1))
```



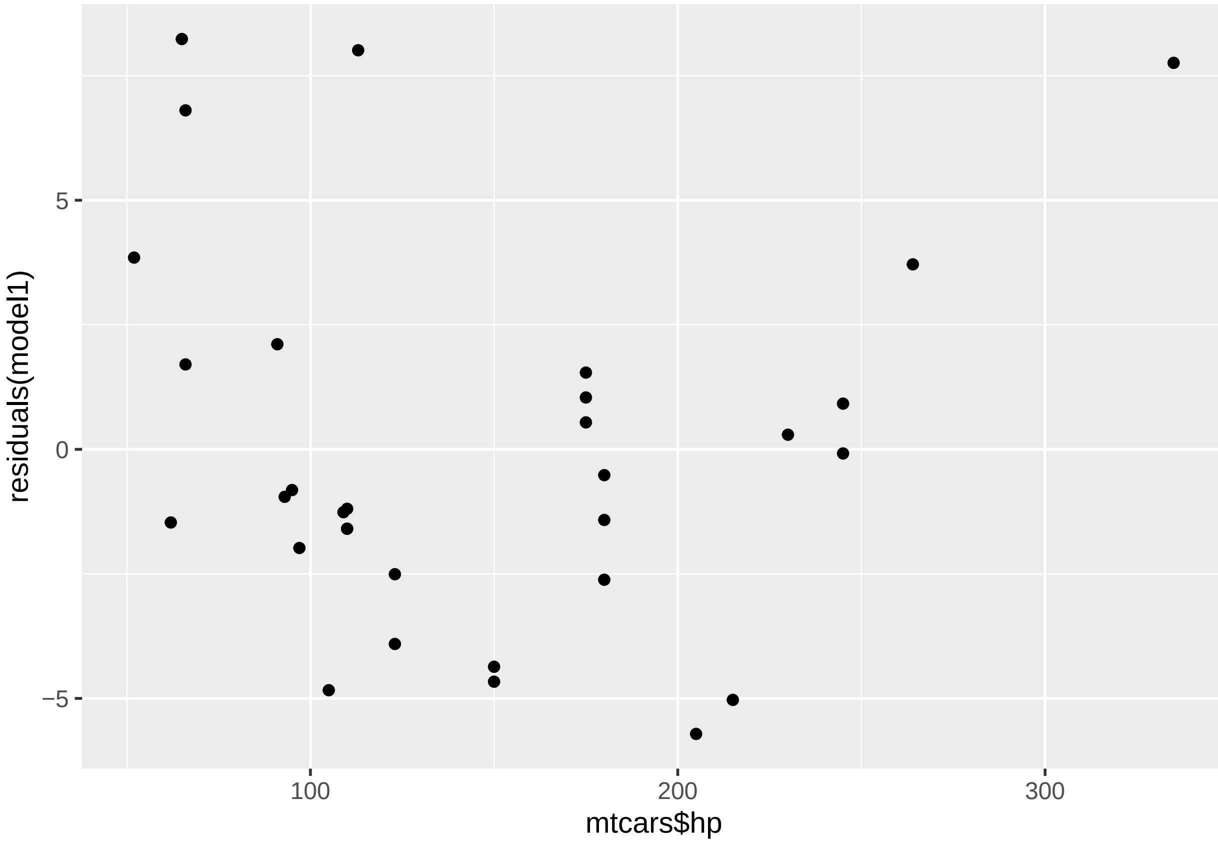

```r
hist(model1$residuals)
```



Plot the residuals against the original predictor variable:

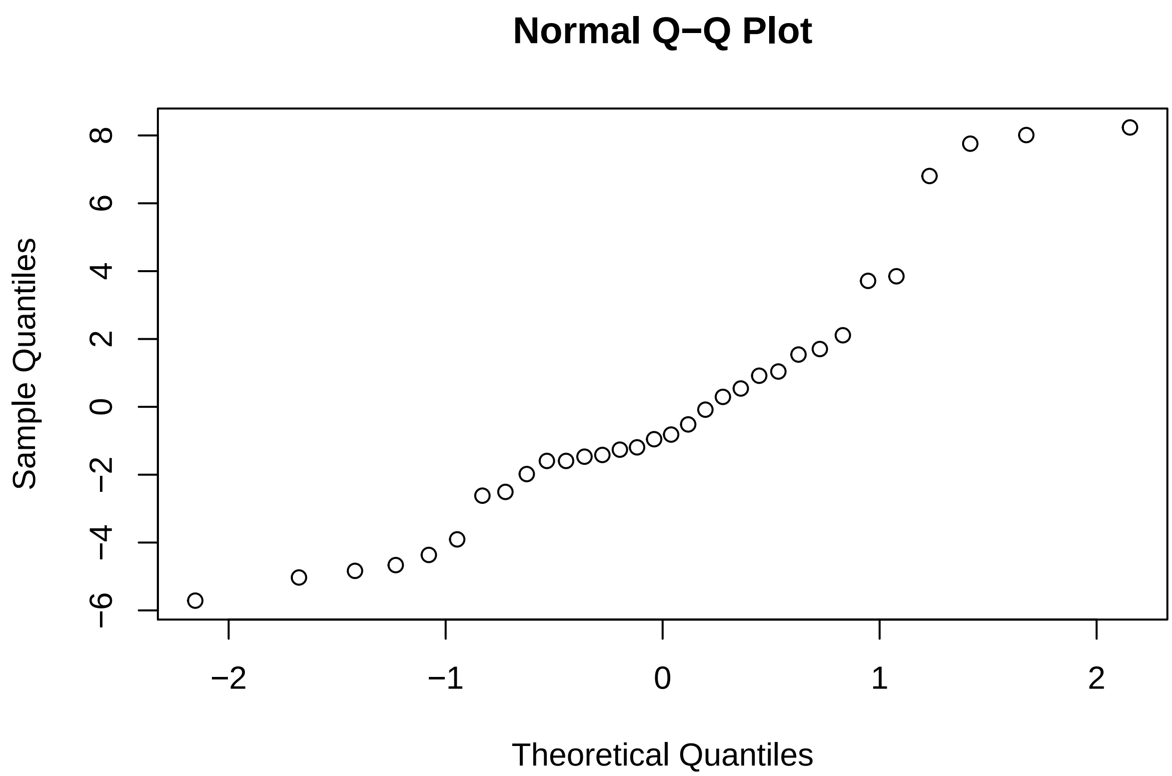
```r
library(ggplot2)
```

```r
qplot(x=mtcars$hp, y=residuals(model1), geom="point")
```



Quantile-quantile plots can be done using `qqnorm()` on the residuals:

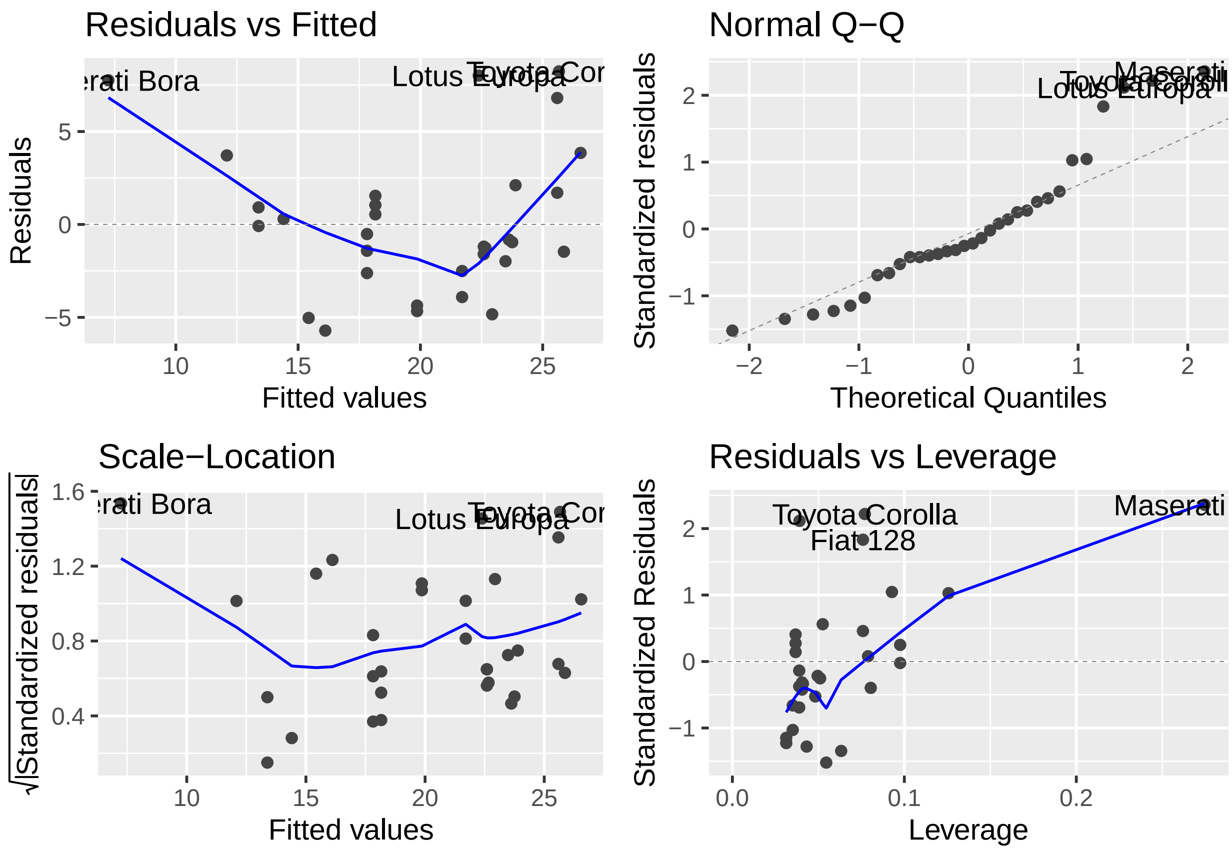
```r
qqnorm(residuals(model1))
```



The easiest way to generate a few generic diagnostic plots in ggplot is documented pretty well on StackExchange and elsewhere:

```
library(ggfortify)

autoplot(model1)
```

```
## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



Adding additional variables (multiple regression—really useful next week)

You can, of course, have models with many variables. This might happen by creating a brand new formula or using a command `update.formula()` to. . . well, you probably guessed it:

```
f1 <- formula(mpg ~ hp)

f2 <- formula(mpg ~ hp + disp + cyl + vs)

f2a <- update.formula(f1, . ~ . + disp + cyl + vs) ## Same as f2 above

model2 <- lm(f2, data=mtcars)

summary(model2)
```

```
##
## Call:
## lm(formula = f2, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0190 -2.1712 -0.7994  1.6104  6.9770
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.00980    4.46560   8.064 1.15e-08 ***
## hp          -0.01594    0.01506  -1.058   0.2992
```

```
## disp        -0.01825    0.01061  -1.720   0.0969 .
## cyl         -1.44613    0.91674  -1.577   0.1263
## vs          -0.96916    1.91824  -0.505   0.6175
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.097 on 27 degrees of freedom
## Multiple R-squared:  0.7701, Adjusted R-squared:  0.736
## F-statistic: 22.61 on 4 and 27 DF,  p-value: 2.745e-08
```

Estimating linear models with predictor variables that are not continuous (numeric or integers) is no problem. Just go for it:

```
mtcars$cyl <- factor(mtcars$cyl)
mtcars$vs <- as.logical(mtcars$vs)

## Refit the same model:
model2 <- lm(f2, data=mtcars)
summary(model2)
```

```
##
## Call:
## lm(formula = f2, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.4430 -1.7856 -0.3927  1.9359  6.0095
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 31.41674    2.43984  12.877 8.65e-13 ***
## hp          -0.02143    0.01457  -1.472   0.1532
## disp        -0.02575    0.01076  -2.392   0.0243 *
## cyl6        -4.16031    1.85492  -2.243   0.0337 *
## cyl8        -2.74149    3.80548  -0.720   0.4777
## vsTRUE      -0.30254    1.85252  -0.163   0.8715
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.941 on 26 degrees of freedom
## Multiple R-squared:  0.8003, Adjusted R-squared:  0.7619
## F-statistic: 20.84 on 5 and 26 DF,  p-value: 2.391e-08
```

We'll talk more about how to interpret these results with categorical predictors next week, but for now you can see that R has no trouble handling multiple types or classes of variables in a regression model.

Producing nice regression tables

Generating regression tables directly from your statistical software is very important for preventing mistakes and typos. There are many ways to do this and a variety of packages that may be helpful (LaTex users: see this StackExchange post for a big list).

One especially easy-to-use package that can output text and html (both eminently paste-able into a variety of typesetting/word-processing systems) is called `stargazer`. Here I use it to generate an ASCII table summarizing the two models we've fit in this tutorial.

```r
library(stargazer)
```

```
##
## Please cite as:
##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
##  R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
```

```r
stargazer(model1, model2, type="text")
```

```
##
## ==================================================================
##                                   Dependent variable:
##                      ---------------------------------------------
##                                           mpg
##                               (1)                     (2)
## ------------------------------------------------------------------
## hp                         -0.068***                -0.021
##                             (0.010)                 (0.015)
##
## disp                                               -0.026**
##                                                     (0.011)
##
## cyl6                                               -4.160**
##                                                     (1.855)
##
## cyl8                                                -2.741
##                                                     (3.805)
##
## vs                                                  -0.303
##                                                     (1.853)
##
## Constant                   30.099***               31.417***
##                             (1.634)                 (2.440)
##
## ------------------------------------------------------------------
## Observations                  32                      32
## R2                           0.602                   0.800
## Adjusted R2                  0.589                   0.762
## Residual Std. Error     3.863 (df = 30)         2.941 (df = 26)
## F Statistic          45.460*** (df = 1; 30) 20.837*** (df = 5; 26)
## ==================================================================
## Note:                                  *p<0.1; **p<0.05; ***p<0.01
```

Back to ANOVAs for a moment

You may recall that I mentioned that R actually calls `lm()` when it estimates an ANOVA. As I said before, I'm not going to walk through the details, but an important thing to note is that the F-statistics and the p-values for those F-statistics are identical when you use `aov()` and when you use `lm()`. That means that you already know what hypothesis is being tested there and how to interpret that part of the regression model output.

```r
summary(aov(data=mtcars, mpg ~ factor(cyl)))
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
```

```
## factor(cyl)  2  824.8   412.4    39.7 4.98e-09 ***
## Residuals   29  301.3    10.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(lm(data=mtcars, mpg ~ factor(cyl)))
```

```
##
## Call:
## lm(formula = mpg ~ factor(cyl), data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.2636 -1.8357  0.0286  1.3893  7.2364
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   26.6636     0.9718  27.437  < 2e-16 ***
## factor(cyl)6  -6.9208     1.5583  -4.441 0.000119 ***
## factor(cyl)8 -11.5636     1.2986  -8.905 8.57e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.223 on 29 degrees of freedom
## Multiple R-squared:  0.7325, Adjusted R-squared:  0.714
## F-statistic:  39.7 on 2 and 29 DF,  p-value: 4.979e-09
```