Linear regression in R

Erin Shellman

April 13 & 20, 2015

Contents

Linear regression

In this tutorial we'll learn:

- how to `merge` datasets
- how to fit linear regression models
- how to split data into test and train sets
- how to tune our models and select features

Data preparation

We're working with the Capital Bikeshare again this week, so start by reading in *usage*, *weather*, *stations*.

```
library(dplyr)
library(ggplot2)
library(lubridate)

usage = read.delim('usage_2012.tsv',
                   sep = '\t',
                   header = TRUE)

weather = read.delim('daily_weather.tsv',
                   sep = '\t',
                   header = TRUE)

stations = read.delim('stations.tsv',
                   sep = '\t',
                   header = TRUE)
```

Merging data

We have three related datasets to work with, but we can't really get started until we figure out how to combine them. Let's start with *usage* and *weather*. The *usage* dataframe is at the resolution of the hour, while the *weather* data are at the resolution of a day, so we know we're going to have to either duplicate or compress data to merge. I vote compress, let's summarize!

```r
head(usage)
```

```
##   bike_id          time_start            time_end duration_mins
## 1  W01412 2012-01-01 00:04:00 2012-01-01 00:11:00             7
## 2  W00524 2012-01-01 00:10:00 2012-01-01 00:29:00            19
## 3  W00235 2012-01-01 00:10:00 2012-01-01 00:29:00            19
## 4  W00864 2012-01-01 00:15:00 2012-01-01 00:23:00             8
## 5  W00995 2012-01-01 00:15:00 2012-01-01 00:23:00             8
## 6  W00466 2012-01-01 00:17:00 2012-01-01 00:23:00             6
##                         station_start             station_end  cust_type
## 1         7th & R St NW / Shaw Library           7th & T St NW Registered
## 2       Georgia & New Hampshire Ave NW    16th & Harvard St NW     Casual
## 3       Georgia & New Hampshire Ave NW    16th & Harvard St NW Registered
## 4                       14th & V St NW Park Rd & Holmead Pl NW Registered
## 5                  11th & Kenyon St NW           7th & T St NW Registered
## 6 Court House Metro / 15th & N Uhle St    Lynn & 19th St North Registered
```

```r
custs_per_day = usage %>%
  group_by(time_start = as.Date(time_start), station_start, cust_type) %>%
  summarize(no_rentals = n(),
            duration_mins = mean(duration_mins, na.rm = TRUE))

head(custs_per_day)
```

```
## Source: local data frame [6 x 5]
## Groups: time_start, station_start
##
##   time_start                 station_start  cust_type no_rentals
## 1 2012-01-01           10th & Monroe St NE Registered         10
## 2 2012-01-01                10th & U St NW     Casual          8
## 3 2012-01-01                10th & U St NW Registered         50
## 4 2012-01-01 10th St & Constitution Ave NW     Casual         34
## 5 2012-01-01 10th St & Constitution Ave NW Registered         20
## 6 2012-01-01                11th & H St NE     Casual          4
## Variables not shown: duration_mins (dbl)
```

Perfection, now we can merge! What's the key?

```r
# make sure we have consistent date formats
custs_per_day$time_start = ymd(custs_per_day$time_start)
weather$date = ymd(weather$date)

# then merge. see ?merge for more details about the function
weather_rentals = merge(custs_per_day, weather,
                        by.x = 'time_start', by.y = 'date')

# check dimensions after to make sure they are what you expect
dim(custs_per_day)
```

```
## [1] 99356     5
```

```r
dim(weather)
```

```
## [1] 366  15
```

```r
dim(weather_rentals)
```

```
## [1] 99356    19
```

```r
head(weather_rentals)
```

```
##   time_start                 station_start  cust_type no_rentals
## 1 2012-01-01           10th & Monroe St NE Registered         10
## 2 2012-01-01                10th & U St NW     Casual          8
## 3 2012-01-01                10th & U St NW Registered         50
## 4 2012-01-01 10th St & Constitution Ave NW     Casual         34
## 5 2012-01-01 10th St & Constitution Ave NW Registered         20
## 6 2012-01-01                11th & H St NE     Casual          4
##   duration_mins weekday season_code season_desc is_holiday is_work_day
## 1      16.40000       0           1      Spring          0           0
## 2      16.25000       0           1      Spring          0           0
## 3      10.00000       0           1      Spring          0           0
## 4      20.29412       0           1      Spring          0           0
## 5      14.20000       0           1      Spring          0           0
## 6      10.00000       0           1      Spring          0           0
##   weather_code                                 weather_desc temp
## 1            1 Clear, Few clouds, Partly cloudy, Partly cloudy 0.37
## 2            1 Clear, Few clouds, Partly cloudy, Partly cloudy 0.37
## 3            1 Clear, Few clouds, Partly cloudy, Partly cloudy 0.37
## 4            1 Clear, Few clouds, Partly cloudy, Partly cloudy 0.37
## 5            1 Clear, Few clouds, Partly cloudy, Partly cloudy 0.37
## 6            1 Clear, Few clouds, Partly cloudy, Partly cloudy 0.37
##   subjective_temp humidity windspeed no_casual_riders no_reg_riders
## 1        0.375621   0.6925  0.192167              686          1608
## 2        0.375621   0.6925  0.192167              686          1608
## 3        0.375621   0.6925  0.192167              686          1608
## 4        0.375621   0.6925  0.192167              686          1608
## 5        0.375621   0.6925  0.192167              686          1608
## 6        0.375621   0.6925  0.192167              686          1608
##   total_riders
## 1         2294
## 2         2294
## 3         2294
## 4         2294
## 5         2294
## 6         2294
```

Great, now we want to merge on the last dataset, *stations*. What is the key to link *weather_rentals* with *stations*?

```r
final_data = merge(weather_rentals, stations,
                   by.x = 'station_start', by.y = 'station')
dim(final_data)
```

```
## [1] 98634   154
```

```r
dim(weather_rentals)
```

```
## [1] 99356    19
```

```r
head(final_data[, 1:30])
```

```
##    station_start time_start  cust_type no_rentals duration_mins weekday
## 1 10th & E St NW 2012-07-25     Casual          8      82.37500       3
## 2 10th & E St NW 2012-07-25 Registered         32      13.28125       3
## 3 10th & E St NW 2012-11-13 Subscriber         19      11.73684       2
## 4 10th & E St NW 2012-09-25 Registered         41      12.29268       2
## 5 10th & E St NW 2012-08-09 Registered         34      13.61765       4
## 6 10th & E St NW 2012-11-22 Subscriber          7      12.14286       4
##   season_code season_desc is_holiday is_work_day weather_code
## 1           3        Fall          0           1            1
## 2           3        Fall          0           1            1
## 3           4      Winter          0           1            2
## 4           4      Winter          0           1            1
## 5           3        Fall          0           1            1
## 6           4      Winter          1           0            1
##                                                     weather_desc     temp
## 1              Clear, Few clouds, Partly cloudy, Partly cloudy 0.724167
## 2              Clear, Few clouds, Partly cloudy, Partly cloudy 0.724167
## 3 Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 0.343333
## 4              Clear, Few clouds, Partly cloudy, Partly cloudy 0.550000
## 5              Clear, Few clouds, Partly cloudy, Partly cloudy 0.755833
## 6              Clear, Few clouds, Partly cloudy, Partly cloudy 0.340000
##   subjective_temp humidity windspeed no_casual_riders no_reg_riders
## 1        0.654054 0.450000 0.1648000             1383          6790
## 2        0.654054 0.450000 0.1648000             1383          6790
## 3        0.323225 0.662917 0.3420460              327          3767
## 4        0.544179 0.570000 0.2363210              845          6693
## 5        0.699508 0.620417 0.1561000             1196          6090
## 6        0.350371 0.580417 0.0528708              955          1470
##   total_riders  id terminal_name      lat      long no_bikes
## 1         8173 199         31256 38.89591 -77.02606        6
## 2         8173 199         31256 38.89591 -77.02606        6
## 3         4094 199         31256 38.89591 -77.02606        6
## 4         7538 199         31256 38.89591 -77.02606        6
## 5         7286 199         31256 38.89591 -77.02606        6
## 6         2425 199         31256 38.89591 -77.02606        6
##   no_empty_docks fast_food parking restaurant convenience post_office
## 1              8         5       2         16           0           1
## 2              8         5       2         16           0           1
## 3              8         5       2         16           0           1
## 4              8         5       2         16           0           1
## 5              8         5       2         16           0           1
## 6              8         5       2         16           0           1
```

```r
# probably want to save this now!
write.table(final_data,
            'bikeshare_modeling_data.tsv',
            row.names = FALSE, sep = '\t')

# rename to something more convenient and remove from memory
data = final_data
rm(final_data)
```

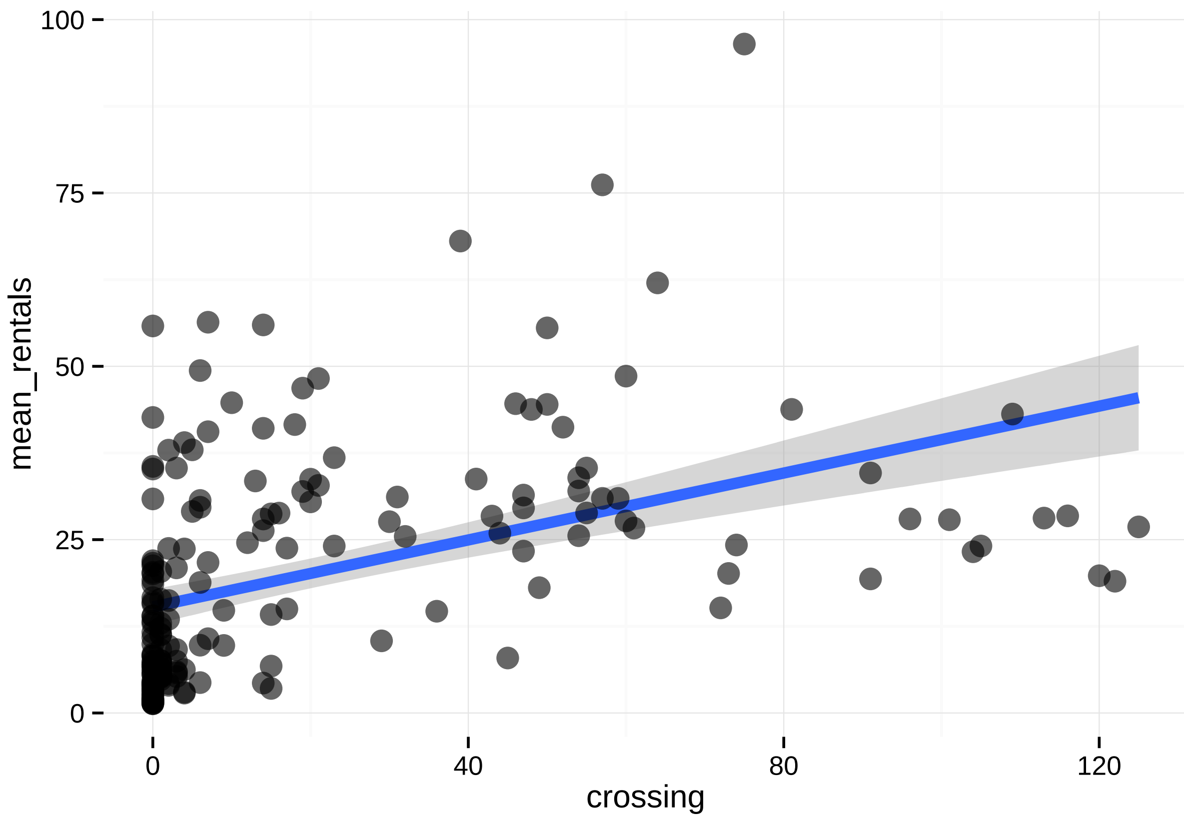
The `lm()` function

The function for creating a linear model in R is `lm()` and the primary arguments are *formula* and *data*. Formulas in R are a little funny, instead of an = sign, they are expressed with a ~. Let's fit the model we saw in the lecture notes: $rentals = \beta_0 + \beta_1 * crossing$. There's a little snag we have to take care of first. Right now we've got repeated measures *i.e.* one measurement per day, so we need to aggregate again. How do we aggregate over date, but still maintain relevant seasonal data?

```r
rentals_crossing = data %>%
  group_by(station_start) %>%
  summarize(mean_rentals = mean(no_rentals),
            crossing = mean(crossing))

head(rentals_crossing)
```

```
## Source: local data frame [6 x 3]
##
##                    station_start mean_rentals crossing
## 1                 10th & E St NW    19.003003      122
## 2            10th & Monroe St NE     7.580517        1
## 3                 10th & U St NW    37.954876        5
## 4 10th St & Constitution Ave NW    28.430362      116
## 5                 11th & H St NE    20.121875       73
## 6            11th & Kenyon St NW    33.718331       20
```

```r
# plot it
ggplot(rentals_crossing, aes(x = crossing, y = mean_rentals)) +
  geom_smooth(method = 'lm', size = 2) +
  geom_point(size = 4, alpha = 0.60) +
  theme_minimal()
```




```r
model = lm(mean_rentals ~ crossing, data = rentals_crossing)

# view what is returned in the lm object
attributes(model)
```

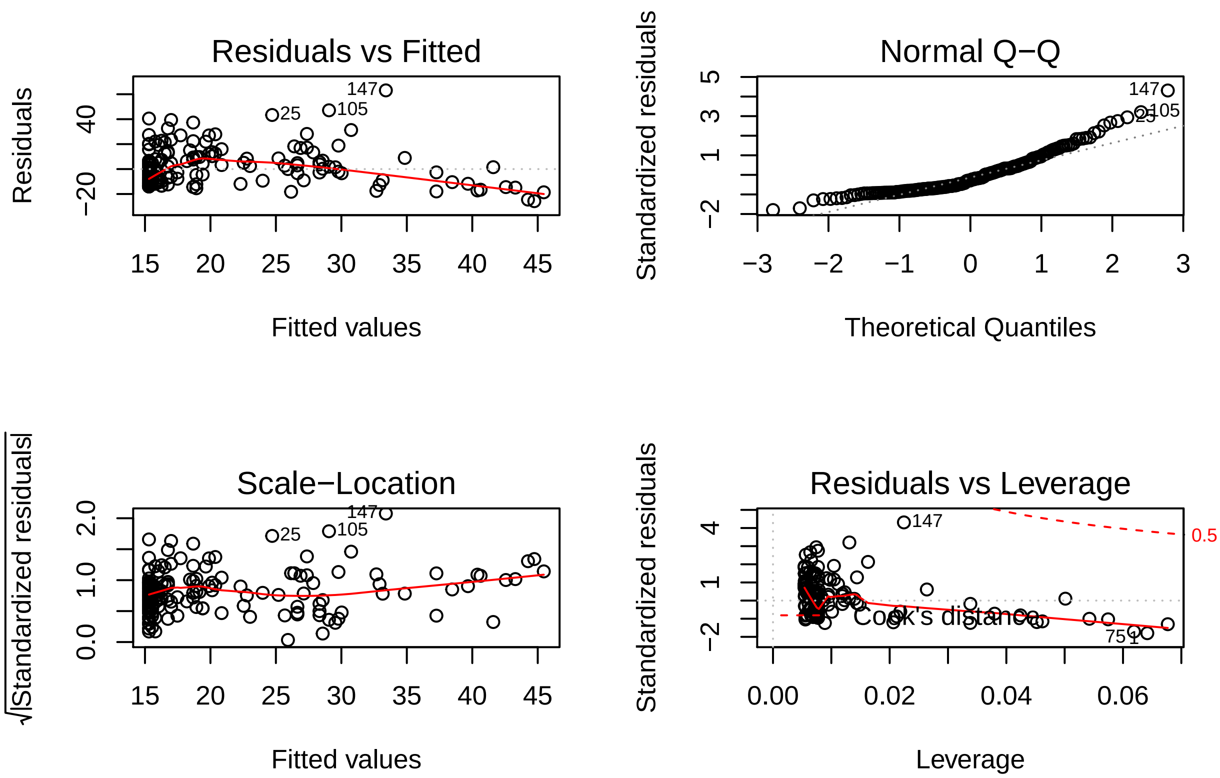
```
## $names
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
##
## $class
## [1] "lm"
```

```r
# get model output
summary(model)
```

```
##
## Call:
## lm(formula = mean_rentals ~ crossing, data = rentals_crossing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.735 -10.767  -4.190   6.755  63.079
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 15.30402    1.29989  11.773  < 2e-16 ***
```

```
## crossing     0.24127    0.03524   6.846 1.11e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.8 on 183 degrees of freedom
## Multiple R-squared:  0.2039, Adjusted R-squared:  0.1996
## F-statistic: 46.87 on 1 and 183 DF,  p-value: 1.109e-10
```

```r
# print model diagnostics
par(mfrow = c(2, 2))
plot(model)
```



The `attributes()` function can be called on just about any object in R and it returns a list of all the things inside. It's a great way to explore objects and see what values are contained inside that could be used in other analysis. For example, extracting the residuals via `model$residuals` is useful if we want to print diagnostic plots like those above.

When we run `summary()` on the `lm` object, we see the results. The *Call* section just prints back the model specification, and the *Residuals* section contains a summary of the distribution of the errors. The fun stuff is in the *Coefficients* section. In the first row contains the covariate names followed by their estimates, standard errors, t- and p-values. Our model ends up being `rentals = 28 + 0.50(crosswalks)` which means that the average number of rentals when there are no crosswalks is 28, and the average increases by 1 rental for every two additional crosswalks.

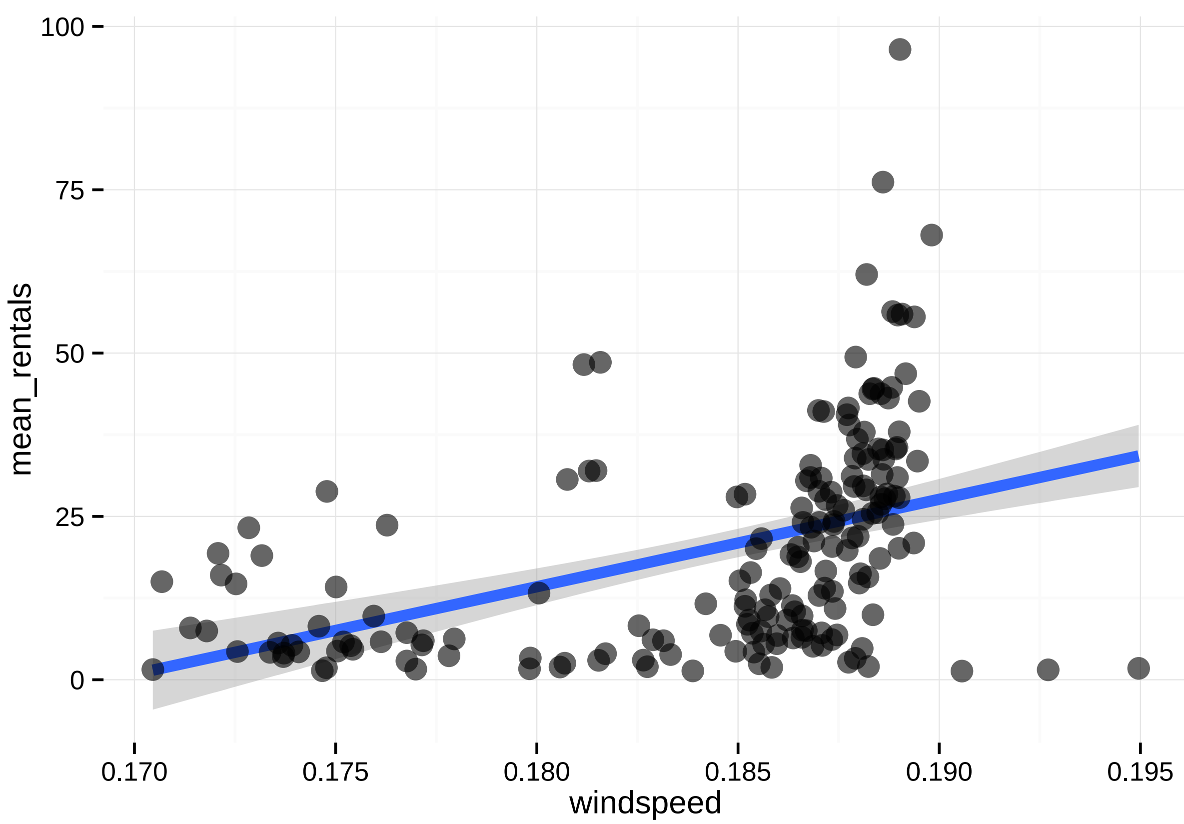
We can fit regressions with multiple covariates the same way:

```r
# lets include windspeed this time
rentals_multi = data %>%
  group_by(station_start) %>%
  summarize(mean_rentals = mean(no_rentals),
```

```r
            crossing = mean(crossing),
            windspeed = mean(windspeed))

head(rentals_multi)
```

```
## Source: local data frame [6 x 4]
##
##                      station_start mean_rentals crossing windspeed
## 1                 10th & E St NW    19.003003      122 0.1731664
## 2            10th & Monroe St NE     7.580517        1 0.1866016
## 3                 10th & U St NW    37.954876        5 0.1890061
## 4 10th St & Constitution Ave NW    28.430362      116 0.1886993
## 5                 11th & H St NE    20.121875       73 0.1889982
## 6            11th & Kenyon St NW    33.718331       20 0.1882405
```

```r
ggplot(rentals_multi, aes(x = windspeed, y = mean_rentals)) +
  geom_smooth(method = 'lm', size = 2) +
  geom_point(size = 4, alpha = 0.60) +
  theme_minimal()
```




```r
model = lm(mean_rentals ~ crossing + windspeed, data = rentals_multi)
summary(model)
```

```
##
## Call:
## lm(formula = mean_rentals ~ crossing + windspeed, data = rentals_multi)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -26.454  -9.202  -1.752   5.080  59.203
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -200.35799   34.20198  -5.858 2.15e-08 ***
## crossing       0.21373    0.03231   6.616 3.99e-10 ***
## windspeed   1172.33663  185.81081   6.309 2.07e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.45 on 182 degrees of freedom
## Multiple R-squared:  0.3468, Adjusted R-squared:  0.3396
## F-statistic: 48.31 on 2 and 182 DF,  p-value: < 2.2e-16
```

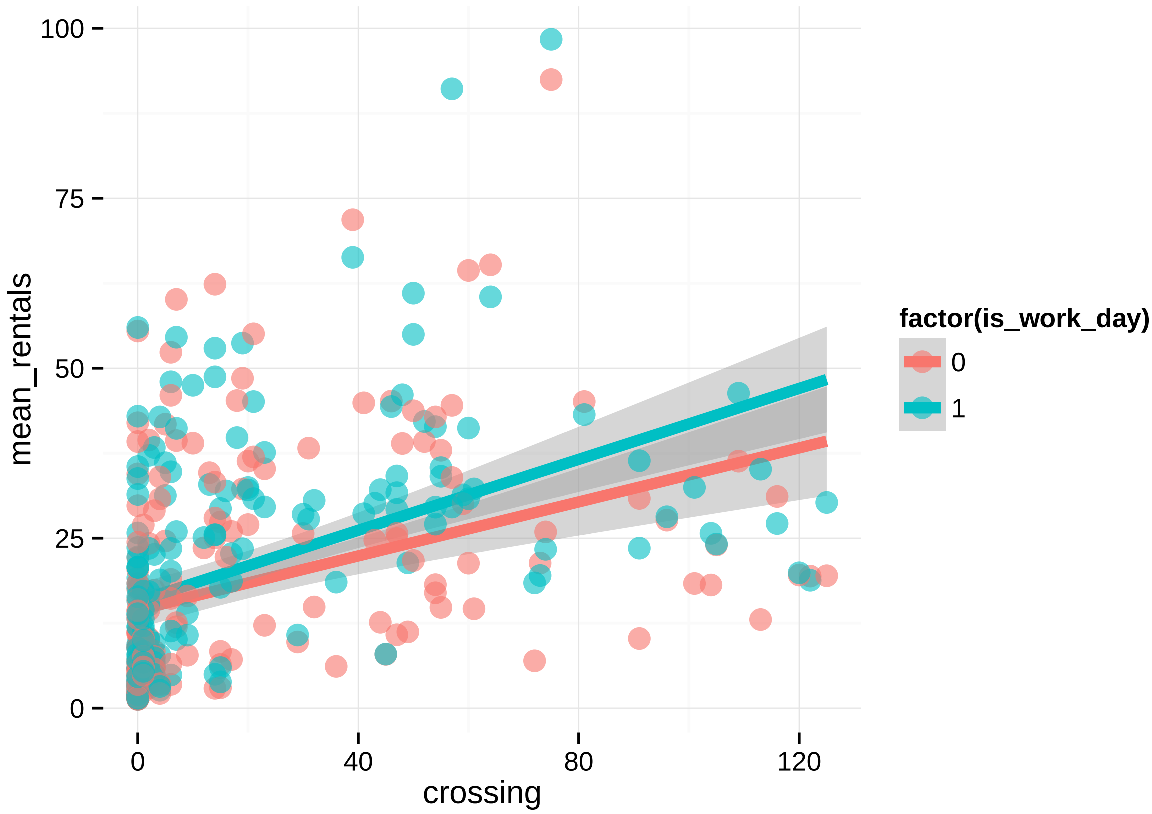
The model coefficients changed quite a lot when we added in windspeed. The intercept is now negative, and the windspeed coefficient is huge! When interpretting coefficients, it's important to keep the scale in mind. Windspeed ranges from 0.05 to 0.44 so when you multiply 2036 by 0.05 for example, you end up with about 102, which is within the range we'd expect.

Let's try one more, this time we'll include a factor variable.

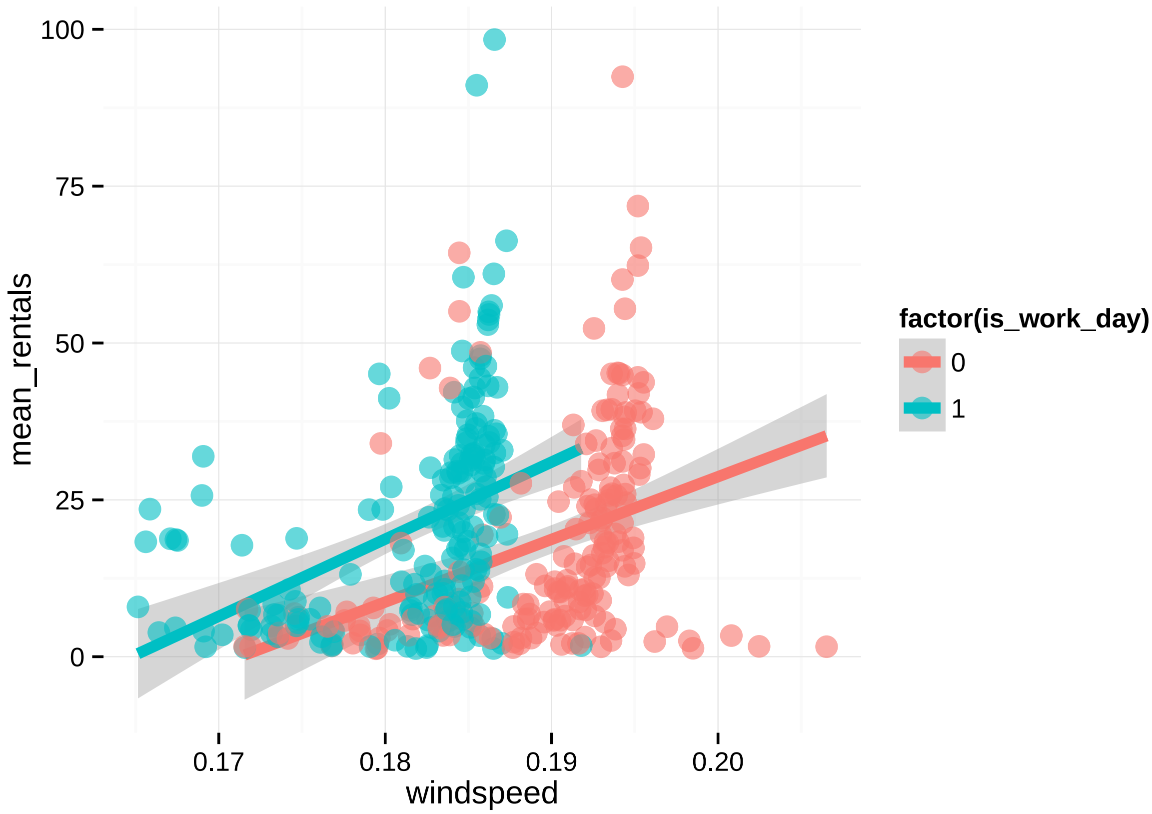
```r
rentals_multi = data %>%
  group_by(station_start, is_work_day) %>%
  summarize(mean_rentals = mean(no_rentals),
            crossing = mean(crossing),
            windspeed = mean(windspeed))

head(rentals_multi)
```

```
## Source: local data frame [6 x 5]
## Groups: station_start
##
##         station_start is_work_day mean_rentals crossing windspeed
## 1      10th & E St NW           0    19.416667      122 0.1858375
## 2      10th & E St NW           1    18.804444      122 0.1670843
## 3 10th & Monroe St NE           0     5.854054        1 0.1912622
## 4 10th & Monroe St NE           1     8.584906        1 0.1838902
## 5      10th & U St NW           0    41.761062        5 0.1939839
## 6      10th & U St NW           1    36.088937        5 0.1865657
```

```r
# plot crossings, colored by is_work_day
ggplot(rentals_multi,
       aes(x = crossing, y = mean_rentals, color = factor(is_work_day))) +
  geom_smooth(method = 'lm', size = 2) +
  geom_point(size = 4, alpha = 0.60) +
  theme_minimal()
```




```r
# plot windspeed, colored by is_work_day
ggplot(rentals_multi,
       aes(x = windspeed, y = mean_rentals, color = factor(is_work_day))) +
  geom_smooth(method = 'lm', size = 2) +
  geom_point(size = 4, alpha = 0.60) +
  theme_minimal()
```




```r
model = lm(mean_rentals ~ crossing + windspeed + factor(is_work_day),
           data = rentals_multi)
summary(model)
```

```
##
## Call:
## lm(formula = mean_rentals ~ crossing + windspeed + factor(is_work_day),
##     data = rentals_multi)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.943  -9.728  -2.500   5.734  61.718
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -165.77396   24.38634  -6.798 4.33e-11 ***
## crossing                0.20358    0.02448   8.316 1.81e-15 ***
## windspeed             949.26045  128.75542   7.373 1.13e-12 ***
## factor(is_work_day)1   10.05016    1.81045   5.551 5.46e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.39 on 366 degrees of freedom
## Multiple R-squared:  0.2868, Adjusted R-squared:  0.281
## F-statistic: 49.06 on 3 and 366 DF,  p-value: < 2.2e-16
```

The output looks a little funny now. There's a term called `factor(is_work_day)1`, what does that mean? Factors are category variables and their interpretation is relative to a baseline. Our factor `is_work_day` only has two levels, 0 and 1, and R sets 0 to the baseline by default. So the interpretation of that term is that we can expect about 17 additional rentals when it is a work day (*i.e.* `is_work_day == 0`) and the other variables are fixed.

Train and test data

For all analyses in this class we'll need to divide our data into train and test sets. We'll do this using a package called *caret*. Check out this nice overview for more details.

The *caret* package

The *caret* package in R contains helper functions that provide a unified framework for data cleaning/splitting, model training, and comparison. I highly recommend the optional reading this week which provides a great overview of the *caret* package.

```
install.packages('caret', dependencies = TRUE)
library(caret)

set.seed(1234) # set a seed
```

Setting a seed in R insures that you get identical results each time you run your code. Since resampling methods are inherently probabilistic, every time we rerun them we'll get slightly different answers. Setting the seed to the same number insures that we get identical randomness each time the code is run, and that's helpful for debugging.

Splitting data into test and train sets

In data mining we're interested in creating models for prediction, and we'll assess the quality of our models by quantifying their prediction accuracy. To measure prediction quality, we hold out a portion of our data called the *test* set. The *training* data is used to build the model.

```
# select the training observations
in_train = createDataPartition(y = data$no_rentals,
                               p = 0.75, # 75% in train, 25% in test.
                               list = FALSE)
head(in_train) # row indices of observations in the training set
```

```
##      Resample1
## [1,]        14
## [2,]        25
## [3,]        27
## [4,]        61
## [5,]        71
## [6,]        83
```

```
training_set = data[in_train, ]
testing_set = data[-in_train, ]

dim(training_set)
```

```
## [1] 73977   154
```

```r
dim(testing_set)
```

```
## [1] 24657   154
```

Note: I recommend doing all data processing and aggregation steps *before* splitting out your train/test sets.

Fitting / Training

A workhorse function in the *caret* package in the `train` function. This function can be used to evaluate performance parameters, choose optimal models based on the values of those parameters, and estimate model performance. For regression we can use it in place of the `lm()` function. Here's our last regression model using the train function.

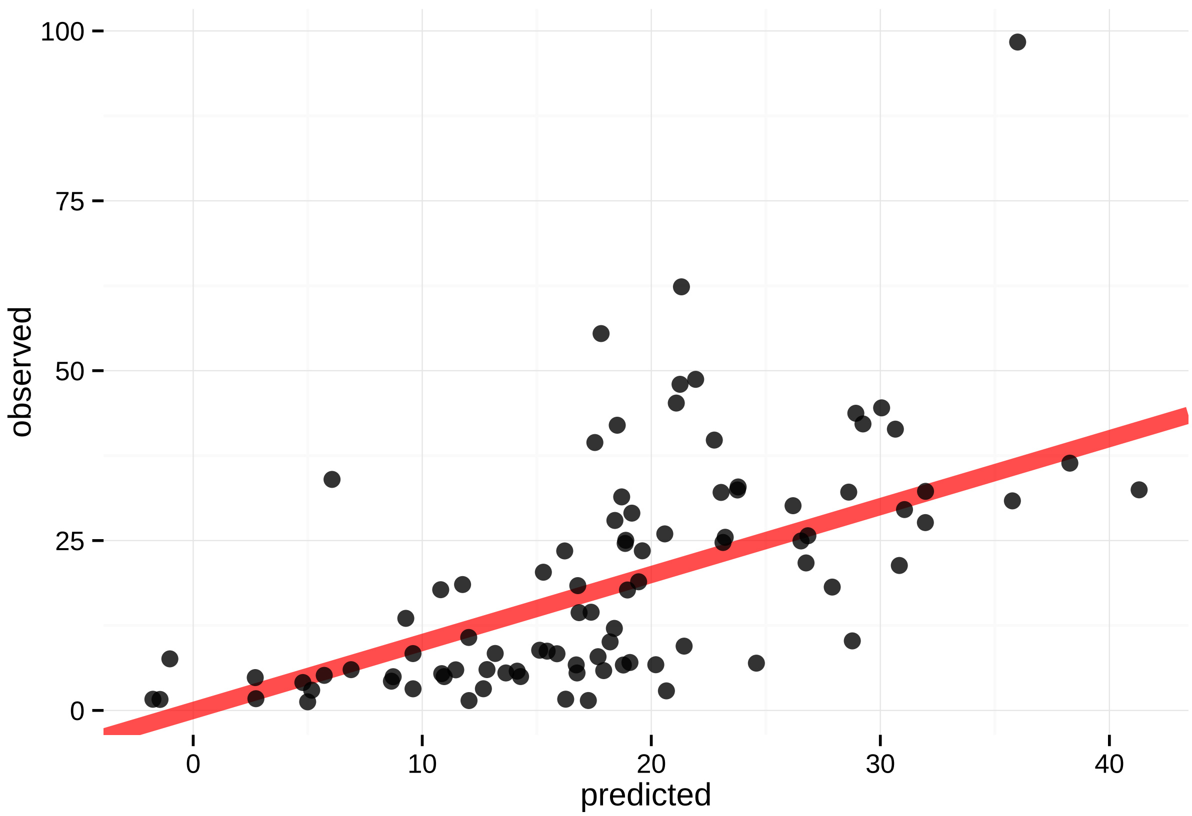
```r
# select the training observations
in_train = createDataPartition(y = rentals_multi$mean_rentals,
                                  p = 0.75,
                                  list = FALSE)
head(in_train)
```

```
##      Resample1
## [1,]         3
## [2,]        17
## [3,]        18
## [4,]        41
## [5,]        44
## [6,]        47
```

```r
training_set = rentals_multi[in_train, ]
testing_set = rentals_multi[-in_train, ]

model_fit = train(mean_rentals ~ crossing + windspeed + factor(is_work_day),
                  data = training_set,
                  method = 'lm',
                  metric = 'RMSE')

print(model_fit)
```

```
## Linear Regression
##
## 278 samples
##   4 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 278, 278, 278, 278, 278, 278, ...
##
## Resampling results
##
##   RMSE     Rsquared   RMSE SD   Rsquared SD
##   14.8344  0.2514065  1.143292  0.05124707
##
##
```

```r
# get predictions
predicted_values = predict(model_fit, newdata = testing_set)

# compare predictions against the observed values
errors = data.frame(predicted = predicted_values,
                    observed = testing_set$mean_rentals)
prediction_error = testing_set$mean_rentals - predicted_values

# eh, not so good
ggplot(data = errors, aes(x = predicted, y = observed)) +
  geom_abline(aes(intercept = 0, slope = 1),
              size = 3, alpha = 0.70, color = 'red') +
  geom_point(size = 3, alpha = 0.80) +
  theme_minimal()
```



Our prediction accuracy is not so great for this model. The RMSE is about 31 which means that on average the predictions are off by about 31 rentals.

Feature Selection

Next time!

Project tips

We saw this issue before when we constructed our SLR.

model_data = data %>% group_by(station_start, weekday, season_code, is_holiday, is_work_day, weather_code) %>% summarize(no_rentals = mean(no_rentals))

head(model_data)