Linear regression in R

Erin Shellman

April 13 - 27, 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 they're combined. 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)
```

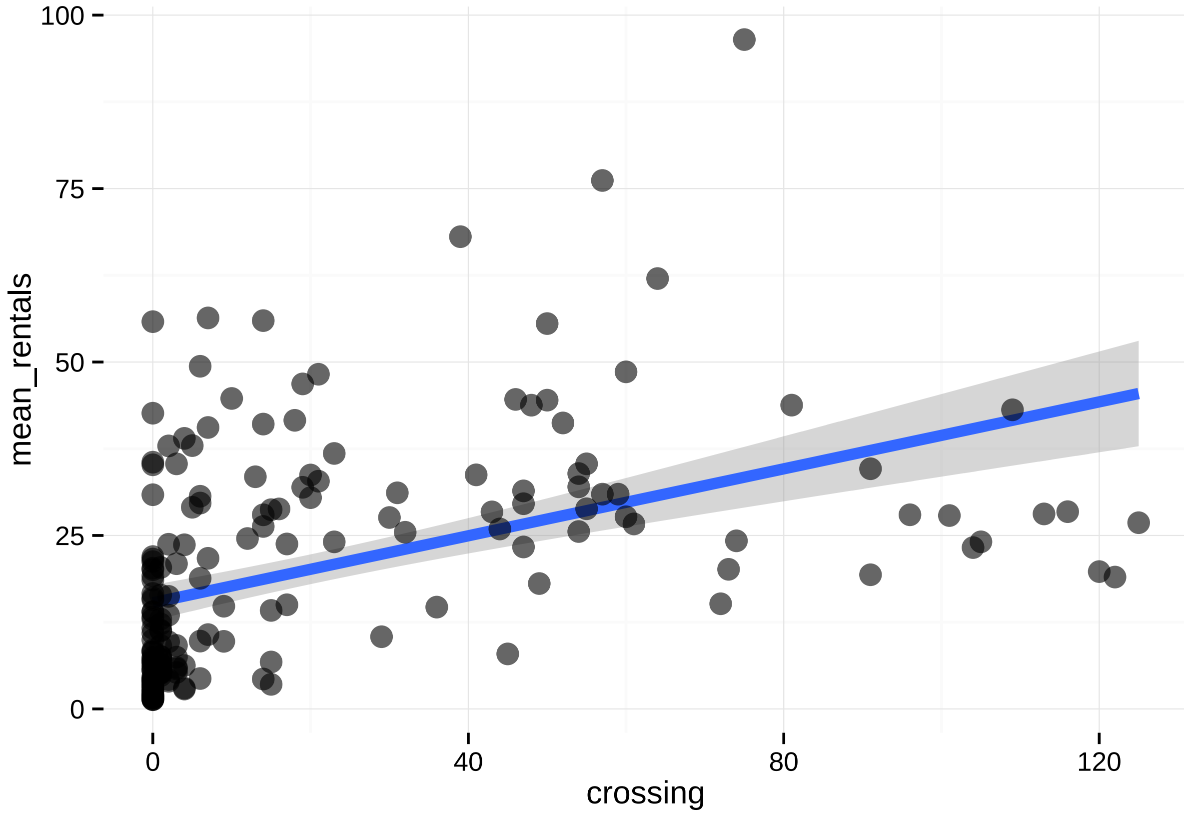
The `lm()` function

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 this time over date.

```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()
```




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

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

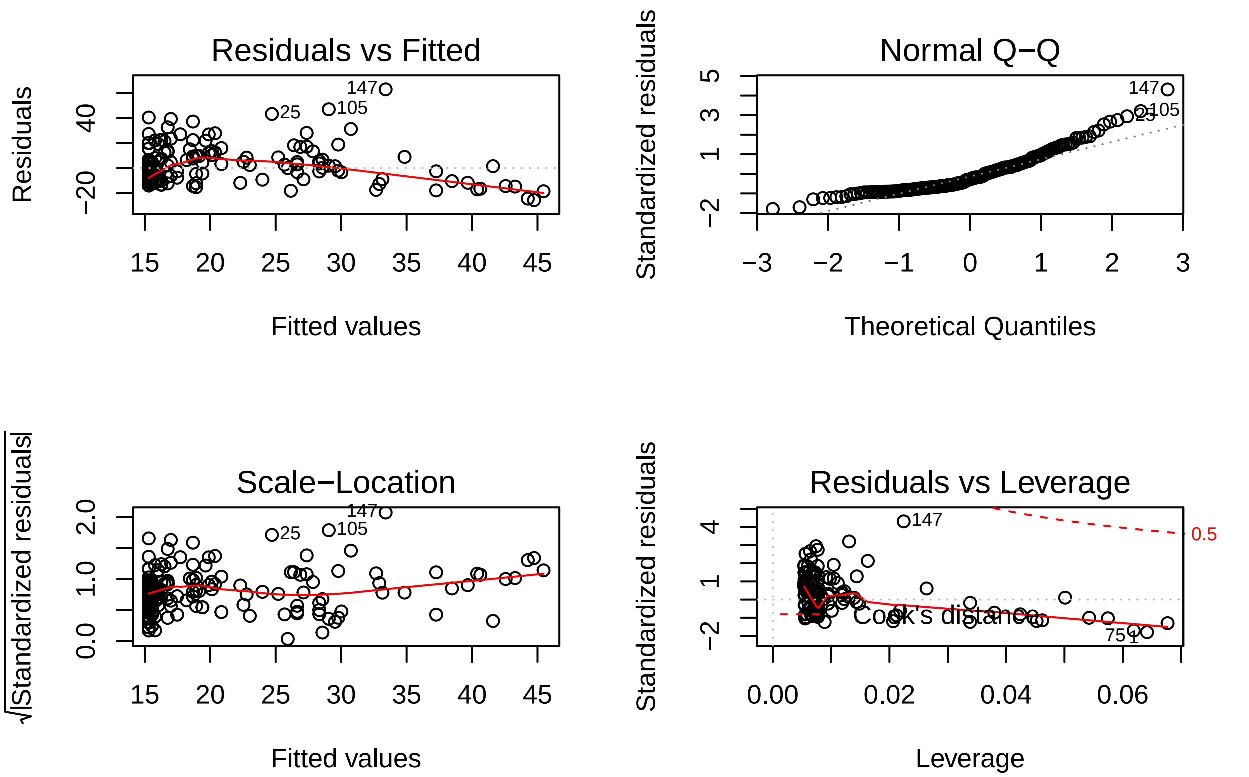
```
## $names
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
##
## $class
## [1] "lm"
```

```
# 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 = 15 + 0.24(crosswalks)` which means that the average number of rentals when there are no crosswalks is 15, and the average increases by 1 rental for every four additional crosswalks.

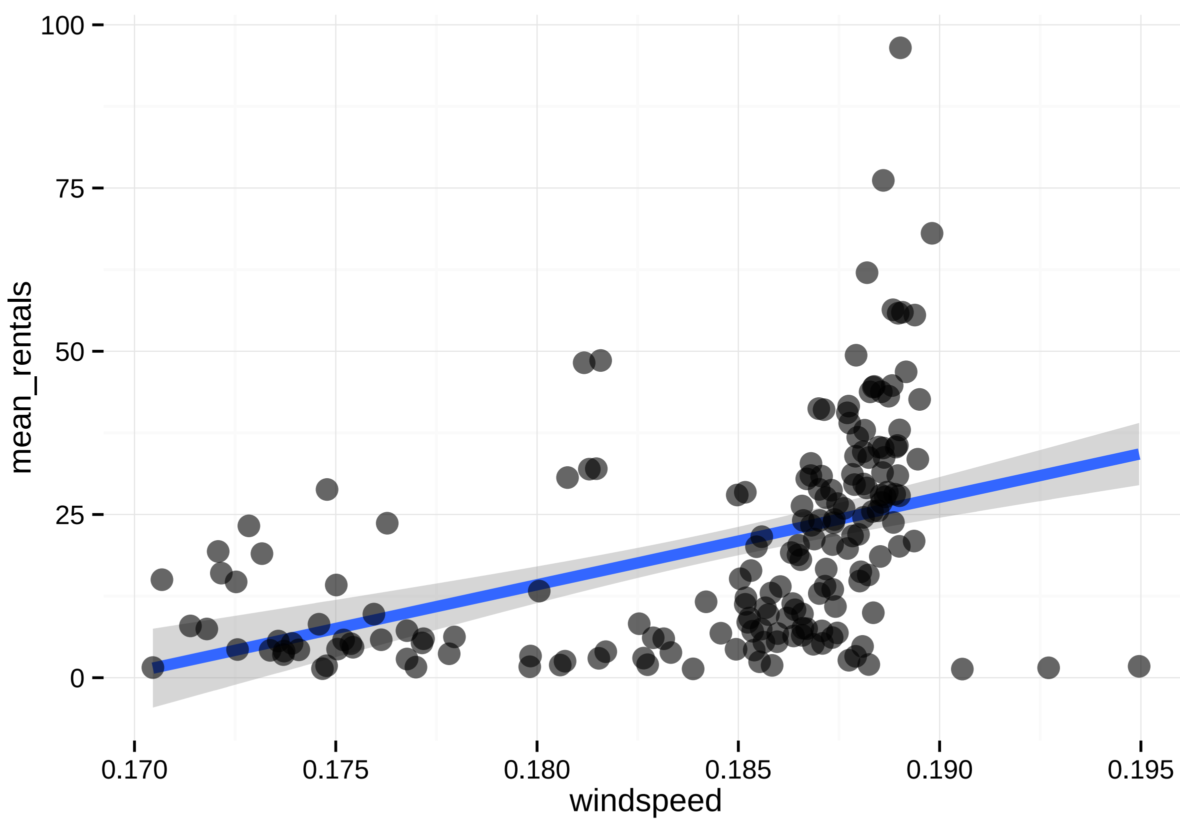
We can fit regressions with multiple covariates the same way:

```r
# lets include windspeed this time
rentals_multi =
  data %>%
    group_by(station_start) %>%
```

```
    summarize(mean_rentals = mean(no_rentals),
              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
```

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




```
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
```

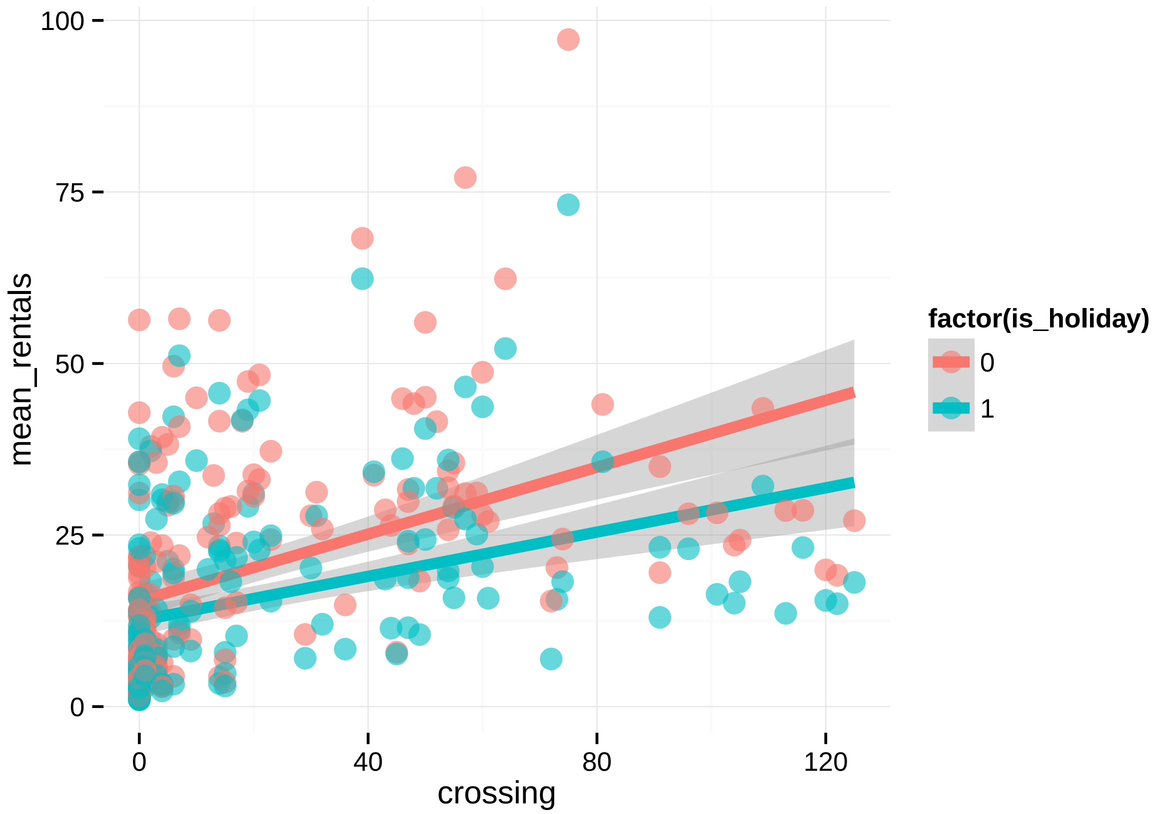
The model coefficients changed quite a lot when we added in wind speed. The intercept is now negative, and the wind speed coefficient is huge! When interpreting coefficients, it's important to keep the scale in mind. Wind speed ranges from 0.05 to 0.44 so when you multiply 1172 by 0.05 for example, you end up with about 60, which is within the range we'd expect.

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

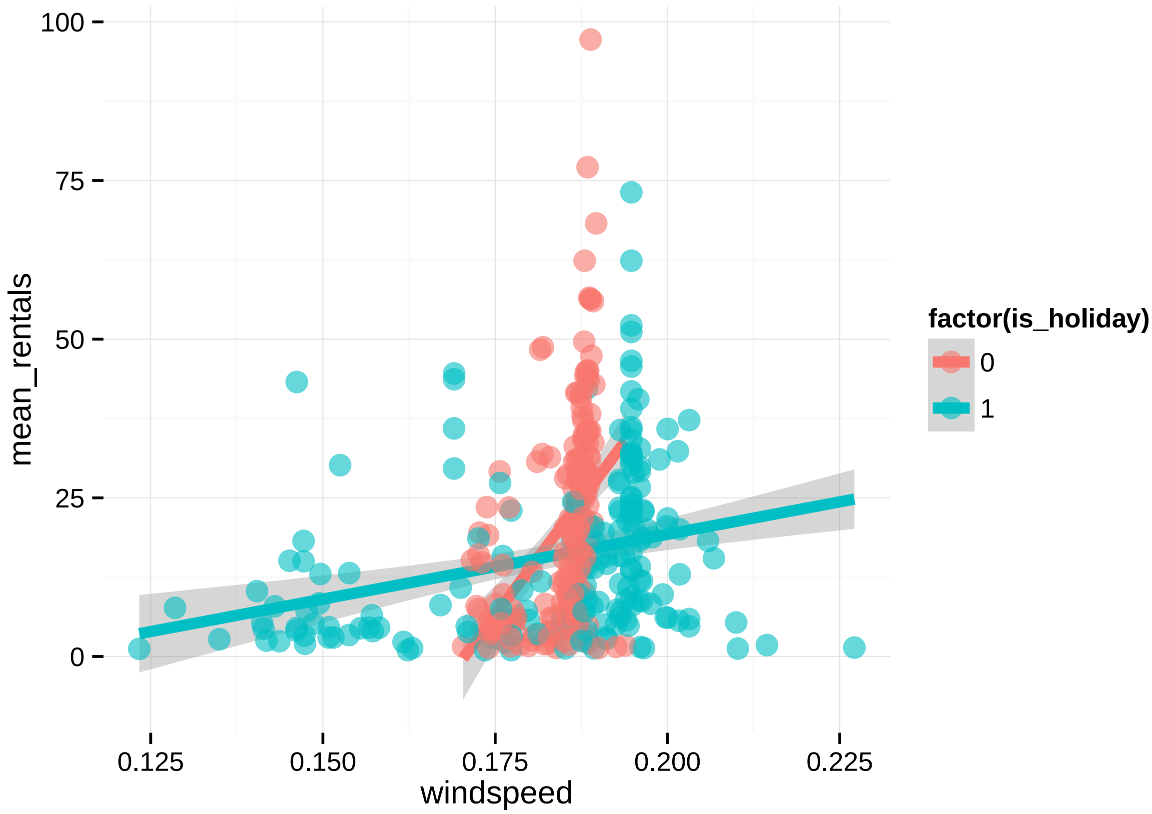
```
rentals_multi =
  data %>%
    group_by(station_start, is_holiday) %>%
    summarize(mean_rentals = mean(no_rentals),
              crossing = mean(crossing),
              windspeed = mean(windspeed))

head(rentals_multi)
```

```
## Source: local data frame [6 x 5]
## Groups: station_start
##
##         station_start is_holiday mean_rentals crossing windspeed
## 1      10th & E St NW          0    19.126935      122 0.1739709
## 2      10th & E St NW          1    15.000000      122 0.1471828
## 3 10th & Monroe St NE          0     7.670782        1 0.1863943
## 4 10th & Monroe St NE          1     5.000000        1 0.1925257
## 5      10th & U St NW          0    38.210210        5 0.1887857
## 6      10th & U St NW          1    29.857143        5 0.1959959
```

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




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




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

```
##
## Call:
## lm(formula = mean_rentals ~ crossing + windspeed + factor(is_holiday),
##     data = rentals_multi)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.688  -9.523  -3.167   5.672  65.164
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -30.32142    9.19757  -3.297  0.00107 **
## crossing              0.19401    0.02249   8.626  < 2e-16 ***
## windspeed           253.19832   49.69175   5.095 5.59e-07 ***
## factor(is_holiday)1  -4.14508    1.38743  -2.988  0.00300 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.33 on 366 degrees of freedom
## Multiple R-squared:  0.244,  Adjusted R-squared:  0.2378
## F-statistic: 39.37 on 3 and 366 DF,  p-value: < 2.2e-16
```

The output looks a little funny now. There's a term called `factor(is_holiday)1`, what does that mean? Factors are category variables and their interpretation is relative to a baseline. Our factor `is_holiday` 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 10 additional rentals when it is a holiday (*i.e.* `is_holiday == 0`) and the other variables are fixed.

The *caret* package

For this section, we'll use the fully cleaned and combined data from the project-1-data-cleanup file, so make sure you've gone through and cleaned your data up like that, or download the clean file from here.

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

We'll be using the *caret* package (short for **c**lassification **a**nd **re**gression **t**raining) for model development because it integrates many modeling packages in R into one unified syntax. That means more reusable code for us! *caret* 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.

Train and test data

Before any analysis in this class we'll need to divide our data into train and test sets. Check out this nice overview for more details. The *training* set is typically about 75% of the data and is used for all the model development. Once we have a model we're satisfied with, we use our *testing* set, the other 25% to generate model predictions. Splitting the data into the two groups, train and test, generates two types of errors, in-sample and out-of-sample errors. *In-sample* errors are the errors derived from same data the model was built with. *Out-of-sample* errors are derived from measuring the error on a fresh data set. We are interested in the out-of-sample error because this quantity represents how'd we'd expect the model to perform in the future on brand new data.

Here's how to split the data with *caret*:

```
# select the training observations
in_train = createDataPartition(y = data$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,]         7
## [2,]        76
## [3,]        90
## [4,]        92
## [5,]       100
## [6,]       103
```

```r
train = data[in_train, ]
test = data[-in_train, ]

dim(train)
```

```
## [1] 17544   119
```

```r
dim(test)
```

```
## [1] 5846  119
```

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

Training

Our 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
model_fit = train(rentals ~ crossing + windspeed + factor(is_holiday),
                  data = train,
                  method = 'lm',
                  metric = 'RMSE')
print(model_fit)
```

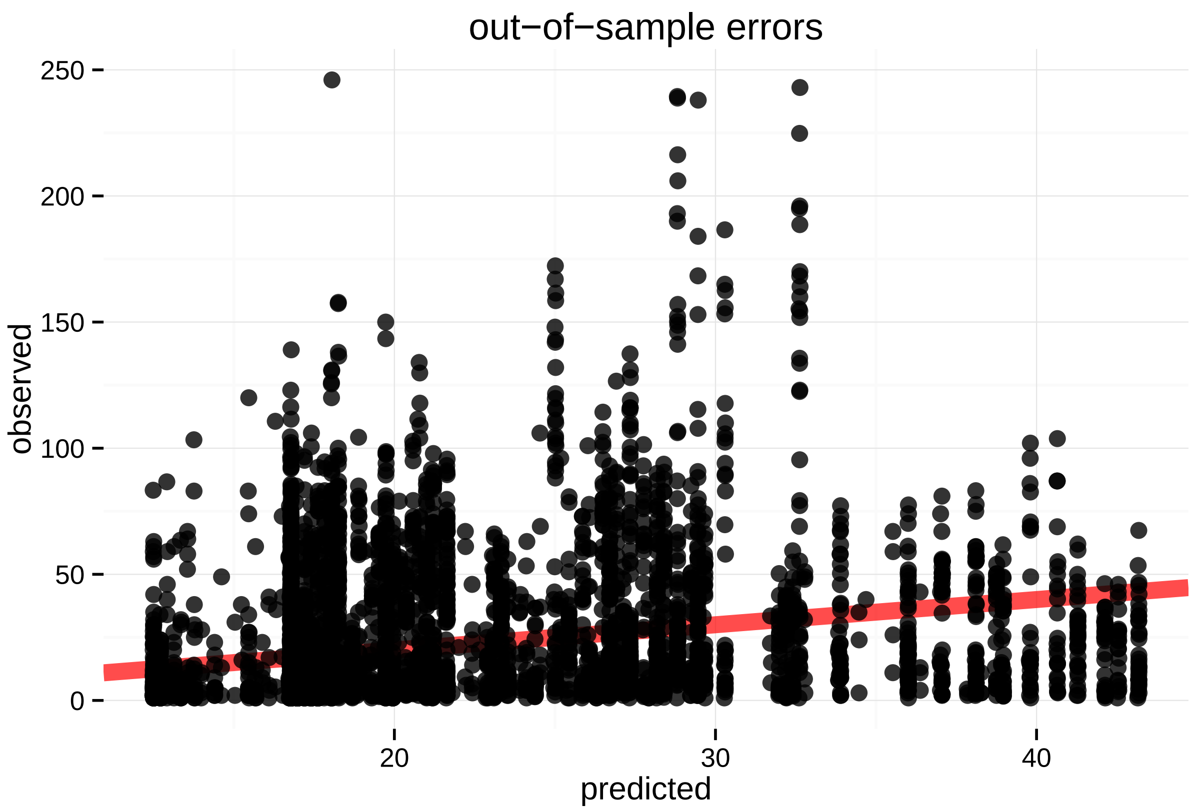
```
## Linear Regression
##
## 17544 samples
##   118 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 17544, 17544, 17544, 17544, 17544, 17544, ...
##
## Resampling results
##
##   RMSE      Rsquared    RMSE SD   Rsquared SD
##   26.55475  0.06312135  0.349779  0.004381095
##
##
```

```r
summary(model_fit)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
```

```
##      Min       1Q  Median       3Q      Max
## -41.532 -14.898 -10.152   6.068 235.384
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)            16.806596   0.722069  23.276  < 2e-16 ***
## crossing                0.211239   0.006345  33.294  < 2e-16 ***
## windspeed              -0.143815   3.614209  -0.040    0.968
## `factor(is_holiday)1` -4.279564   0.672116  -6.367 1.97e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.55 on 17540 degrees of freedom
## Multiple R-squared:  0.06137,    Adjusted R-squared:  0.06121
## F-statistic: 382.3 on 3 and 17540 DF,  p-value: < 2.2e-16
```

```r
# get predictions
out_of_sample_predictions = predict(model_fit, newdata = test)

# compare predictions against the observed values
errors = data.frame(predicted = out_of_sample_predictions,
                    observed = test$rentals,
                    error = out_of_sample_predictions - test$rentals)

# 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) +
  ggtitle('out-of-sample errors') +
  theme_minimal()
```



Our prediction accuracy is not so great for this model. The in-sample RMSE is about 27 which means that on average the predictions are off by about 27 rentals. Let's fit the giant model we made before:

```
full_model = train(rentals ~ .,
                   data = train,
                   method = 'lm')
```

The in-sample RMSE is about 19, so definitely an improvement over the previous model, but this model is really complex and probably not going to be usable by Pronto. How can we reduce the complexity of the model, but maintain reasonable predictive accuracy?

Preprocessing

Shrinkage methods require that the predictors are normalized to be on the same scale. We can accomplish this by centering and scaling the data. You center a variable by subtracting the mean of the variable from from each observation. To scale your observations you then divide the centered observation by the variable standard deviation. Now the variable follows a standard normal distribution with mean = 0 and standard deviation = 1.

The *caret* package has lots of convenient functions for preprocessing data, check 'em out!

Converting factors to dummy variables We run into some trouble if we try to just center and scale the data because its got factor variables and you can't subtract a number from a category. We can use the `model.matrix` function to fix that really quickly.

```
no_factors = as.data.frame(model.matrix(rentals ~ . -1, data = data))
```

```r
# put rentals back on
no_factors$rentals = data$rentals

full_model_scaled = train(rentals ~ .,
                  data = no_factors,
                  method = 'lm',
                  preProcess = c('center', 'scale'))
```

Coefficients estimated with normalized data have a different interpretation than coefficients from un-normalized data. In this case when the data are scaled the intercept has a better interpretation, it's the expected number of rentals when all the predictors are at their average value. So, in this case, when all the predictors are at their average values, we expect about 21 rentals per day. In the previous full-model we had an intercept of about -28, which could be interpreted as the expected number of rentals when all the other predictors have a value of 0. That's pretty unsatisfying for a couple reasons. First, we can't have negative rentals! Second, for a lot of the predictors it doesn't make sense to plug in 0's. What does it mean to have a duration of 0? Or a temp of 0? Centering and scaling fix the non-interpret ability of the previous models.

Since we divide by the standard deviation during scaling, the non-intercept coefficients in the centered and scaled model can be interpreted as the increase in y associated with a 1 standard deviation increase in x.

Model Selection

Variable combination

A simple method to reduce model complexity is to combine some of the variables. For example the dataset contains a variable for *alcohol*, *pub* and *bar*, likewise there's a variable for *food_court*, *restaurant*, *food_cart*, and *fast_food*. Maybe we can retain information and remove some variables.

```r
no_factors$food = no_factors$fast_food + no_factors$restaurant +
  no_factors$food_court + no_factors$bar.restaurant +
  no_factors$cafe + no_factors$food_cart

no_factors$nightlife = no_factors$bar + no_factors$club +
  no_factors$pub + no_factors$nightclub

no_factors$seedy_stuff = no_factors$stripclub + no_factors$strip_club +
  no_factors$alcohol + no_factors$check_cashing + no_factors$motel +
  no_factors$hostel

no_factors$tourism = no_factors$theatre + no_factors$arts_centre +
  no_factors$tourist + no_factors$school..historic. + no_factors$hotel +
  no_factors$gallery + no_factors$artwork + no_factors$sculpture +
  no_factors$museum + no_factors$tour_guide + no_factors$car_rental +
  no_factors$guest_house + no_factors$landmark + no_factors$attraction +
  no_factors$information

dim(no_factors)
```

```
## [1] 23390   133
```

```r
# now remove those variables from the no_factorsset
no_factors =
```

```r
  no_factors %>%
  select(-fast_food, -restaurant, -food_court, -bar.restaurant, -cafe,
         -food_cart, -bar, -club, -pub, -nightclub, -stripclub, -strip_club,
         -alcohol, -check_cashing, -motel, -hostel, -theatre, -arts_centre,
         -tourist, -school..historic., -hotel, -gallery, -artwork, -sculpture,
         -museum, -tour_guide, -car_rental, -guest_house, -landmark,
         -attraction, -information)

# Reduced the dataset by 31 variables!
dim(no_factors)
```

```
## [1] 23390   102
```

Try out your own categories, these are just a few to get you started. We'll learn how to make categories computationally when we cover clustering.

We've change the dataframe, don't forget to redefine the train and test sets!

```r
train = no_factors[in_train, ]
test = no_factors[-in_train, ]

dim(train)
```

```
## [1] 17544   102
```

```r
dim(test)
```

```
## [1] 5846  102
```

```r
# how does our new full-model compare?
full_model = train(rentals ~ .,
                   data = train,
                   method = 'lm')
```

Subset selection

We haven't talked much about computational limitations yet, but it's a good time to start. Selection methods can be *extremely* slow. Why? Because we have $2^p = 2^{117}$ possible variable combinations. I recommend doing some combining before trying these methods. I'll leave the combining up to you, but to make sure these models can run in less than infinite time, I'm going to remove a bunch of predictors so you get the idea.

```r
train =
  train %>%
    select(rentals, cust_typeCasual, cust_typeRegistered, cust_typeSubscriber,
           weekday1, weekday2, weekday3, weekday4, weekday5, weekday6,
           season_code2, season_code3, season_code4, is_holiday, weather_code2,
           weather_code3, humidity, windspeed, temp, duration, food, nightlife,
           seedy_stuff, tourism)

test =
  test %>%
```

```r
    select(rentals, cust_typeCasual, cust_typeRegistered, cust_typeSubscriber,
           weekday1, weekday2, weekday3, weekday4, weekday5, weekday6,
           season_code2, season_code3, season_code4, is_holiday, weather_code2,
           weather_code3, humidity, windspeed, temp, duration, food, nightlife,
           seedy_stuff, tourism)

# forward selection
forward_model = train(rentals ~ .,
                      data = na.omit(train),
                      method = 'leapForward',
                      preProcess = c('center', 'scale'),
                      # try models of size 1 - 23
                      tuneGrid = expand.grid(nvmax = 1:23),
                      trControl = trainControl(method = 'cv', number = 5))
```

```
## Loading required package: leaps

## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```r
# what does this return?
attributes(forward_model)
```

```
## $names
##  [1] "method"       "modelInfo"    "modelType"    "results"
##  [5] "pred"         "bestTune"     "call"         "dots"
##  [9] "metric"       "control"      "finalModel"   "preProcess"
## [13] "trainingData" "resample"     "resampledCM"  "perfNames"
## [17] "maximize"     "yLimits"      "times"        "terms"
## [21] "coefnames"    "xlevels"
##
## $class
## [1] "train"         "train.formula"
```

```r
# what what should the number of variables, k, be?
forward_model$bestTune
```

```
##    nvmax
## 22    22
```

```r
# what metric was used?
forward_model$metric
```

```
## [1] "RMSE"
```

```r
# here's a handful of other useful plots and summaries
print(forward_model)
```

```
## Linear Regression with Forward Selection
##
## 17544 samples
##    23 predictor
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (5 fold)
##
## Summary of sample sizes: 14036, 14034, 14036, 14034, 14036
##
## Resampling results across tuning parameters:
##
##   nvmax  RMSE      Rsquared   RMSE SD    Rsquared SD
##    1     25.12277  0.1592009  0.6991738  0.008738563
##    2     24.70043  0.1871617  0.7078603  0.007491652
##    3     24.66559  0.1894340  0.7300260  0.006809702
##    4     24.66559  0.1894340  0.7300260  0.006809702
##    5     24.62280  0.1922227  0.7459557  0.006081438
##    6     24.46921  0.2023450  0.9060200  0.016729246
##    7     23.92798  0.2371929  0.7135446  0.010194007
##    8     23.91284  0.2381279  0.7027733  0.010627555
##    9     23.89126  0.2395247  0.7100072  0.010066808
##   10     23.89113  0.2395232  0.7091856  0.010121113
##   11     23.65259  0.2538549  0.5074372  0.035077089
##   12     23.21621  0.2811226  0.5943756  0.042464328
##   13     23.19920  0.2821733  0.5813361  0.042546798
##   14     22.99002  0.2955359  0.8050175  0.038555975
##   15     22.77455  0.3091486  0.6720125  0.009238166
##   16     22.77545  0.3090951  0.6716052  0.009215905
##   17     22.76392  0.3098084  0.6541484  0.009762380
##   18     22.73370  0.3116430  0.6668330  0.009664536
##   19     22.73610  0.3114909  0.6650696  0.009688894
##   20     22.73052  0.3118367  0.6728287  0.010085408
##   21     22.71691  0.3126163  0.6743395  0.009919237
##   22     22.68428  0.3146378  0.6645330  0.009855135
##   23     22.68428  0.3146378  0.6645330  0.009855135
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was nvmax = 22.
```

```r
summary(forward_model)
```

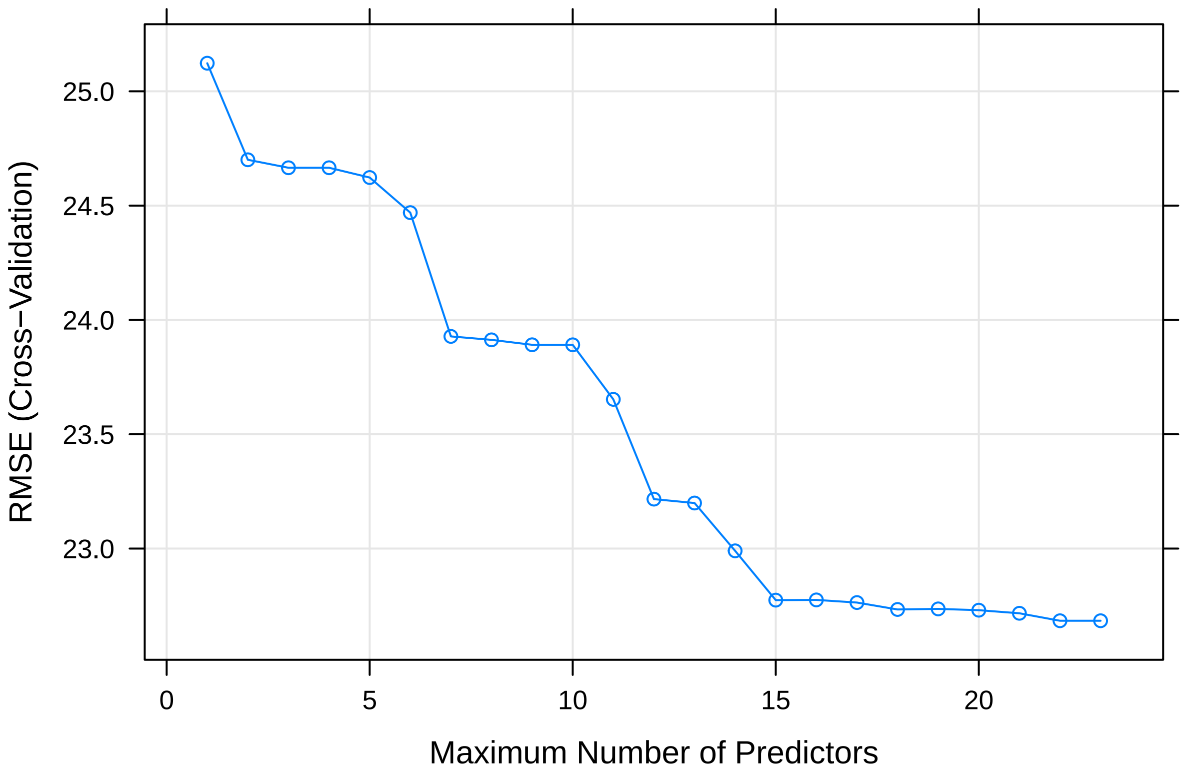
```
## Subset selection object
## 23 Variables  (and intercept)
##                     Forced in Forced out
## cust_typeCasual         FALSE      FALSE
## cust_typeRegistered     FALSE      FALSE
## weekday1                FALSE      FALSE
## weekday2                FALSE      FALSE
```

```
## weekday3               FALSE      FALSE
## weekday4               FALSE      FALSE
## weekday5               FALSE      FALSE
## weekday6               FALSE      FALSE
## season_code2           FALSE      FALSE
## season_code3           FALSE      FALSE
## season_code4           FALSE      FALSE
## is_holiday             FALSE      FALSE
## weather_code2          FALSE      FALSE
## weather_code3          FALSE      FALSE
## humidity               FALSE      FALSE
## windspeed              FALSE      FALSE
## temp                   FALSE      FALSE
## duration               FALSE      FALSE
## food                   FALSE      FALSE
## nightlife              FALSE      FALSE
## seedy_stuff            FALSE      FALSE
## tourism                FALSE      FALSE
## cust_typeSubscriber    FALSE      FALSE
## 1 subsets of each size up to 22
## Selection Algorithm: forward
##            cust_typeCasual cust_typeRegistered cust_typeSubscriber weekday1
## 1  ( 1 )  " "             "*"                 " "                 " "
## 2  ( 1 )  " "             "*"                 " "                 " "
## 3  ( 1 )  " "             "*"                 " "                 " "
## 4  ( 1 )  "*"             "*"                 " "                 " "
## 5  ( 1 )  "*"             "*"                 " "                 " "
## 6  ( 1 )  "*"             "*"                 " "                 " "
## 7  ( 1 )  "*"             "*"                 " "                 " "
## 8  ( 1 )  "*"             "*"                 " "                 " "
## 9  ( 1 )  "*"             "*"                 " "                 " "
## 10  ( 1 ) "*"             "*"                 " "                 " "
## 11  ( 1 ) "*"             "*"                 " "                 " "
## 12  ( 1 ) "*"             "*"                 " "                 " "
## 13  ( 1 ) "*"             "*"                 " "                 " "
## 14  ( 1 ) "*"             "*"                 " "                 " "
## 15  ( 1 ) "*"             "*"                 " "                 " "
## 16  ( 1 ) "*"             "*"                 " "                 "*"
## 17  ( 1 ) "*"             "*"                 " "                 "*"
## 18  ( 1 ) "*"             "*"                 " "                 "*"
## 19  ( 1 ) "*"             "*"                 " "                 "*"
## 20  ( 1 ) "*"             "*"                 " "                 "*"
## 21  ( 1 ) "*"             "*"                 " "                 "*"
## 22  ( 1 ) "*"             "*"                 " "                 "*"
##            weekday2 weekday3 weekday4 weekday5 weekday6 season_code2
## 1  ( 1 )  " "      " "      " "      " "      " "      " "
## 2  ( 1 )  " "      " "      " "      " "      " "      " "
## 3  ( 1 )  " "      " "      " "      " "      " "      " "
## 4  ( 1 )  " "      " "      " "      " "      " "      " "
## 5  ( 1 )  " "      " "      " "      " "      " "      " "
## 6  ( 1 )  " "      " "      " "      " "      " "      " "
## 7  ( 1 )  " "      " "      " "      " "      " "      " "
## 8  ( 1 )  " "      " "      " "      " "      " "      " "
## 9  ( 1 )  " "      " "      " "      " "      " "      " "
```

```
## 10  ( 1 ) " "       " "       " "       " "       " "       " "
## 11  ( 1 ) " "       " "       " "       "*"       " "       " "
## 12  ( 1 ) " "       " "       " "       "*"       " "       " "
## 13  ( 1 ) " "       " "       " "       "*"       " "       " "
## 14  ( 1 ) " "       " "       " "       "*"       " "       "*"
## 15  ( 1 ) " "       " "       " "       "*"       " "       "*"
## 16  ( 1 ) " "       " "       " "       "*"       " "       "*"
## 17  ( 1 ) " "       "*"       " "       "*"       " "       "*"
## 18  ( 1 ) " "       "*"       " "       "*"       "*"       "*"
## 19  ( 1 ) " "       "*"       "*"       "*"       "*"       "*"
## 20  ( 1 ) "*"       "*"       "*"       "*"       "*"       "*"
## 21  ( 1 ) "*"       "*"       "*"       "*"       "*"       "*"
## 22  ( 1 ) "*"       "*"       "*"       "*"       "*"       "*"
##           season_code3 season_code4 is_holiday weather_code2 weather_code3
## 1  ( 1 )  " "          " "          " "        " "           " "
## 2  ( 1 )  " "          " "          " "        " "           " "
## 3  ( 1 )  " "          " "          " "        " "           " "
## 4  ( 1 )  " "          " "          " "        " "           " "
## 5  ( 1 )  " "          " "          " "        " "           "*"
## 6  ( 1 )  "*"          " "          " "        " "           "*"
## 7  ( 1 )  "*"          " "          " "        " "           "*"
## 8  ( 1 )  "*"          " "          "*"        " "           "*"
## 9  ( 1 )  "*"          " "          "*"        "*"           "*"
## 10  ( 1 ) "*"          " "          "*"        "*"           "*"
## 11  ( 1 ) "*"          " "          "*"        "*"           "*"
## 12  ( 1 ) "*"          " "          "*"        "*"           "*"
## 13  ( 1 ) "*"          " "          "*"        "*"           "*"
## 14  ( 1 ) "*"          " "          "*"        "*"           "*"
## 15  ( 1 ) "*"          "*"          "*"        "*"           "*"
## 16  ( 1 ) "*"          "*"          "*"        "*"           "*"
## 17  ( 1 ) "*"          "*"          "*"        "*"           "*"
## 18  ( 1 ) "*"          "*"          "*"        "*"           "*"
## 19  ( 1 ) "*"          "*"          "*"        "*"           "*"
## 20  ( 1 ) "*"          "*"          "*"        "*"           "*"
## 21  ( 1 ) "*"          "*"          "*"        "*"           "*"
## 22  ( 1 ) "*"          "*"          "*"        "*"           "*"
##           humidity windspeed temp duration food nightlife seedy_stuff
## 1  ( 1 )  " "      " "       " "  " "      " "  " "       " "
## 2  ( 1 )  " "      " "       " "  " "      " "  " "       " "
## 3  ( 1 )  " "      " "       " "  " "      " "  "*"       " "
## 4  ( 1 )  " "      " "       " "  " "      " "  "*"       " "
## 5  ( 1 )  " "      " "       " "  " "      " "  "*"       " "
## 6  ( 1 )  " "      " "       " "  " "      " "  "*"       " "
## 7  ( 1 )  " "      " "       " "  " "      "*"  "*"       " "
## 8  ( 1 )  " "      " "       " "  " "      "*"  "*"       " "
## 9  ( 1 )  " "      " "       " "  " "      "*"  "*"       " "
## 10  ( 1 ) " "      "*"       " "  " "      "*"  "*"       " "
## 11  ( 1 ) " "      "*"       " "  " "      "*"  "*"       " "
## 12  ( 1 ) " "      "*"       " "  " "      "*"  "*"       "*"
## 13  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
## 14  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
## 15  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
## 16  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
## 17  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
```

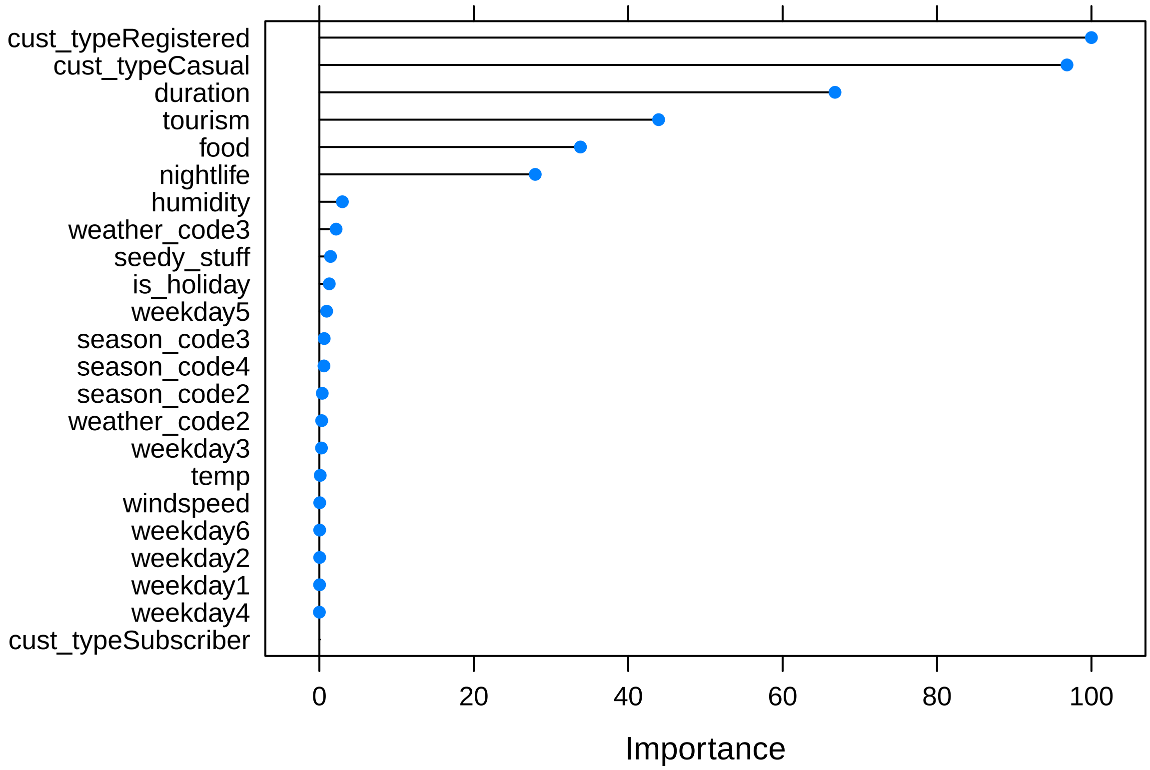
```
## 18  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
## 19  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
## 20  ( 1 ) " "      "*"       "*"  " "      "*"  "*"       "*"
## 21  ( 1 ) "*"      "*"       "*"  " "      "*"  "*"       "*"
## 22  ( 1 ) "*"      "*"       "*"  "*"      "*"  "*"       "*"
##           tourism
## 1  ( 1 )  " "
## 2  ( 1 )  "*"
## 3  ( 1 )  "*"
## 4  ( 1 )  "*"
## 5  ( 1 )  "*"
## 6  ( 1 )  "*"
## 7  ( 1 )  "*"
## 8  ( 1 )  "*"
## 9  ( 1 )  "*"
## 10  ( 1 ) "*"
## 11  ( 1 ) "*"
## 12  ( 1 ) "*"
## 13  ( 1 ) "*"
## 14  ( 1 ) "*"
## 15  ( 1 ) "*"
## 16  ( 1 ) "*"
## 17  ( 1 ) "*"
## 18  ( 1 ) "*"
## 19  ( 1 ) "*"
## 20  ( 1 ) "*"
## 21  ( 1 ) "*"
## 22  ( 1 ) "*"
```

```r
plot(forward_model)
```

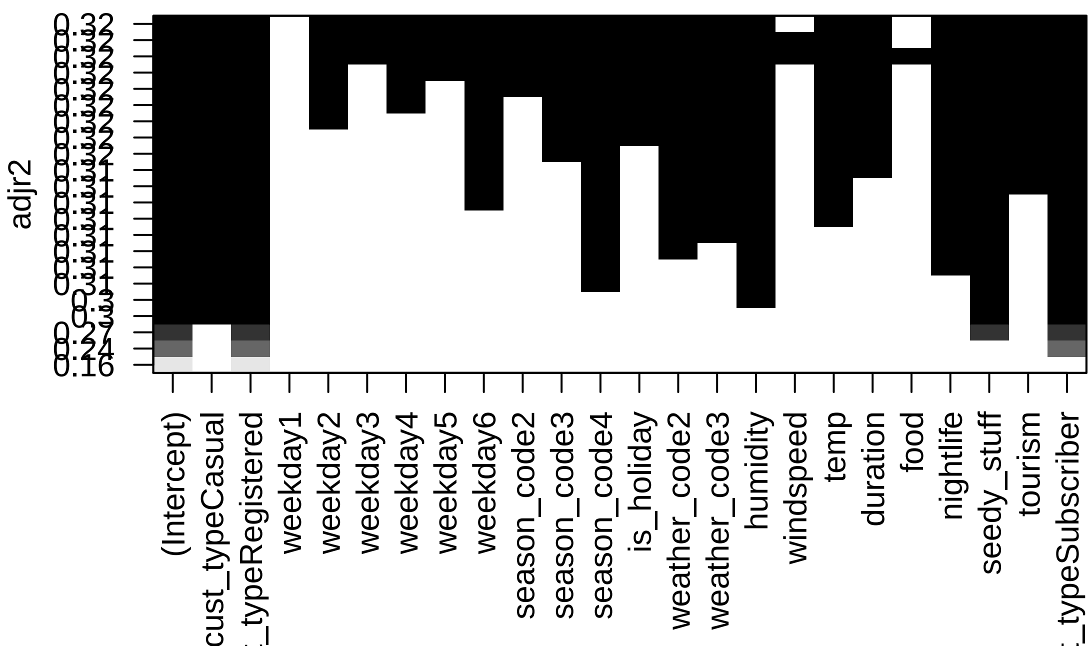



```r
plot(varImp(forward_model))
```

```
## Loading required package: pROC
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```




```r
# compare all the models
plot(forward_model$finalModel, scale = 'adjr2')
```

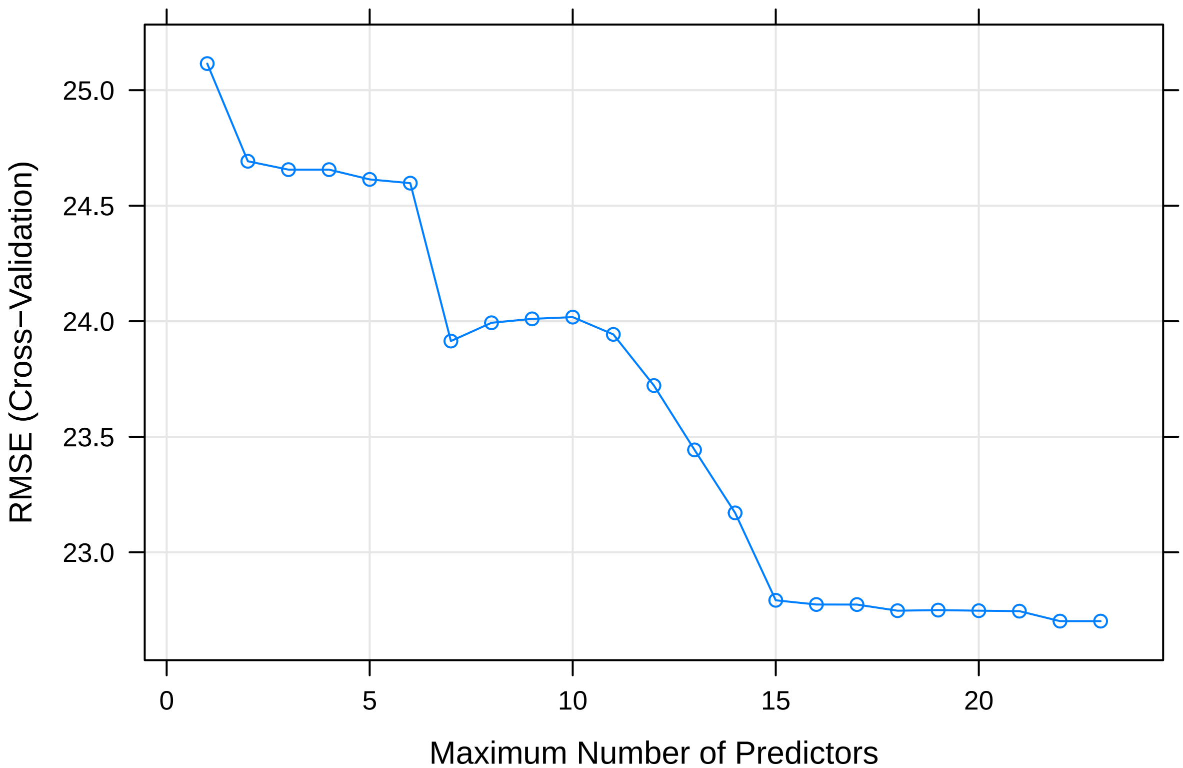



```r
# backward selection
backward_model = train(rentals ~ .,
                       data = na.omit(train),
                       method = 'leapBackward',
                       preProcess = c('center', 'scale'),
```

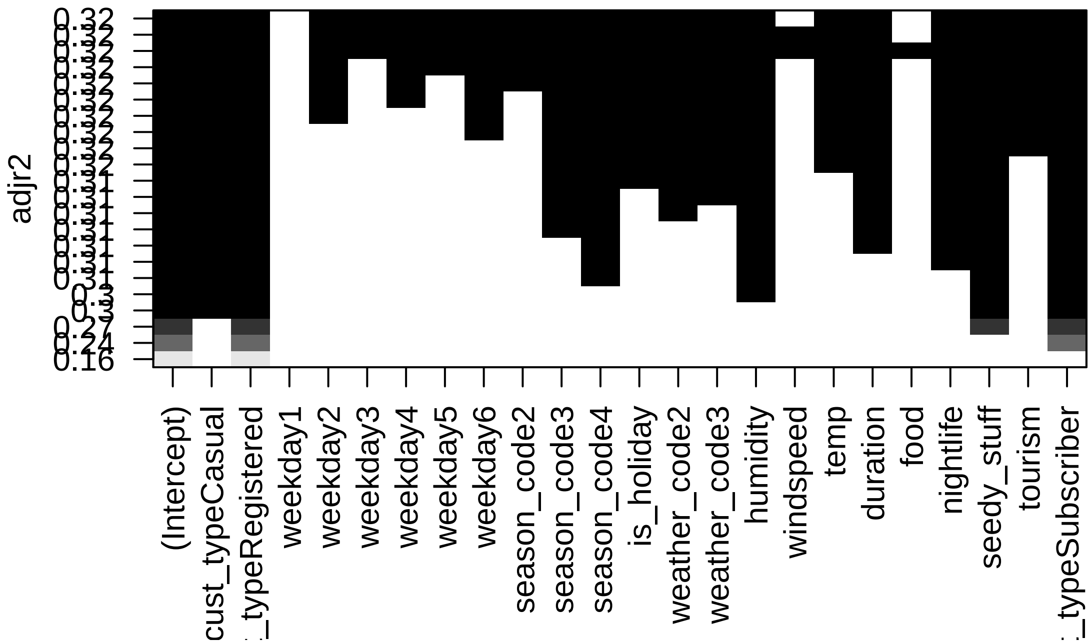
```r
                          tuneGrid = expand.grid(nvmax = 1:23),
                          trControl = trainControl(method = 'cv', number = 5))
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

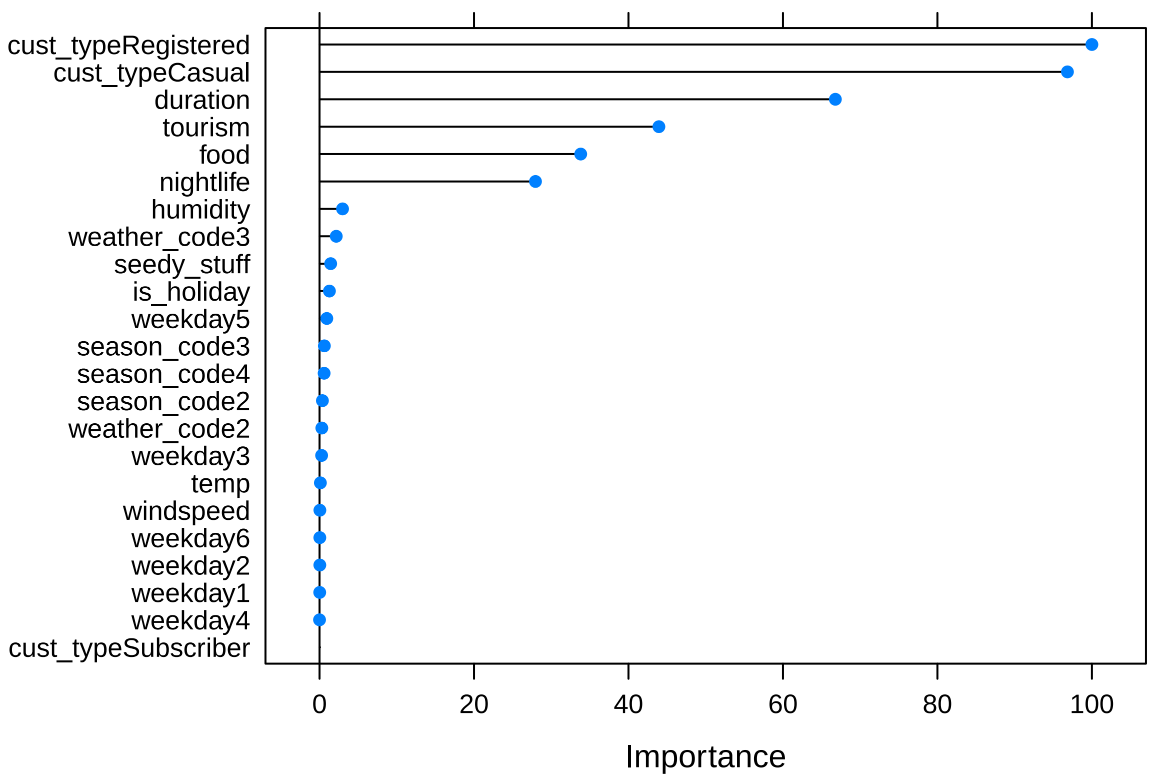
```r
plot(backward_model)
```




```r
plot(backward_model$finalModel, scale = 'adjr2')
```




```r
plot(varImp(backward_model, scale = TRUE))
```

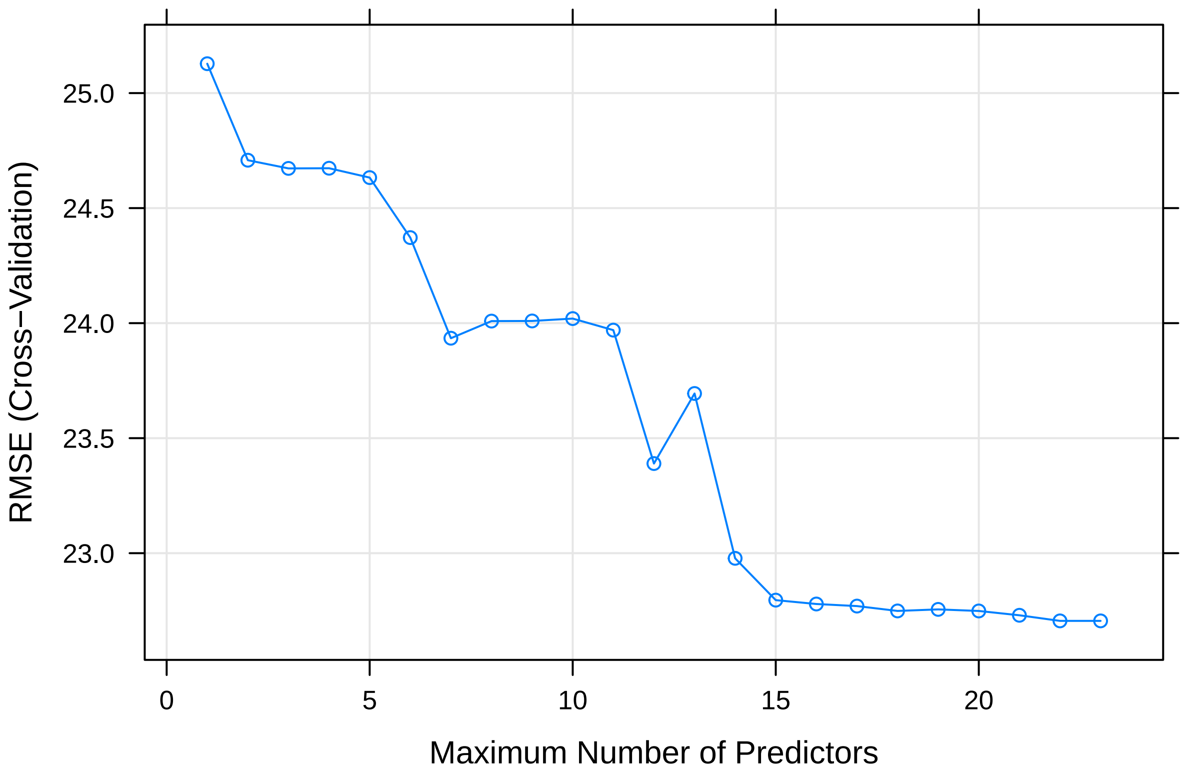



```r
# steps in both directions
hybrid_model = train(rentals ~ .,
                     data = na.omit(train),
                     method = 'leapSeq',
                     preProcess = c('center', 'scale'),
```

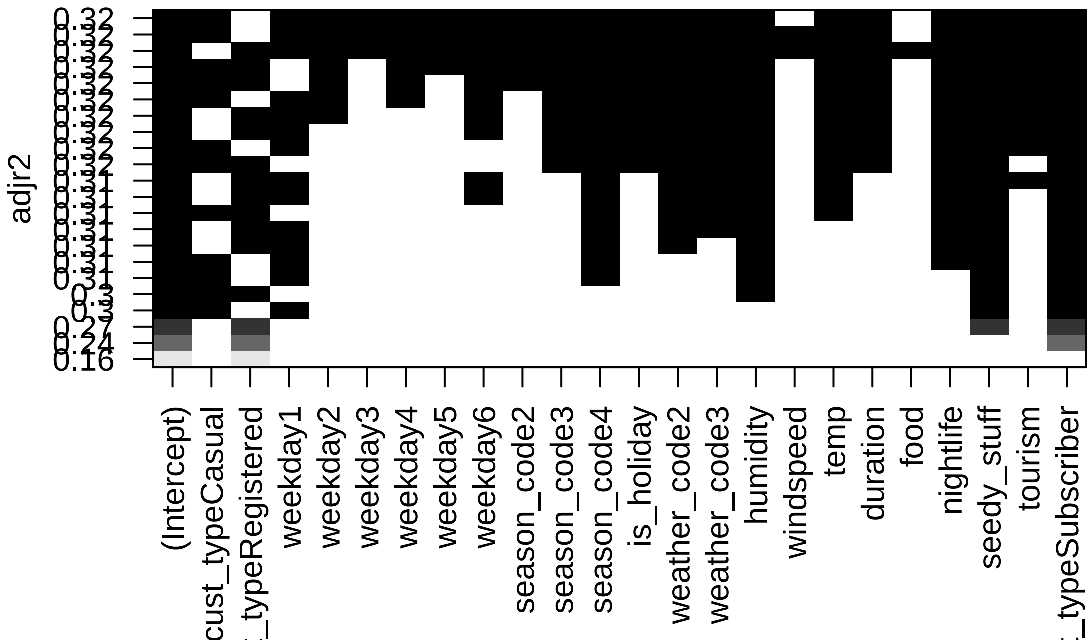
```r
                       tuneGrid = expand.grid(nvmax = 1:23),
                       trControl = trainControl(method = 'cv', number = 5))
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

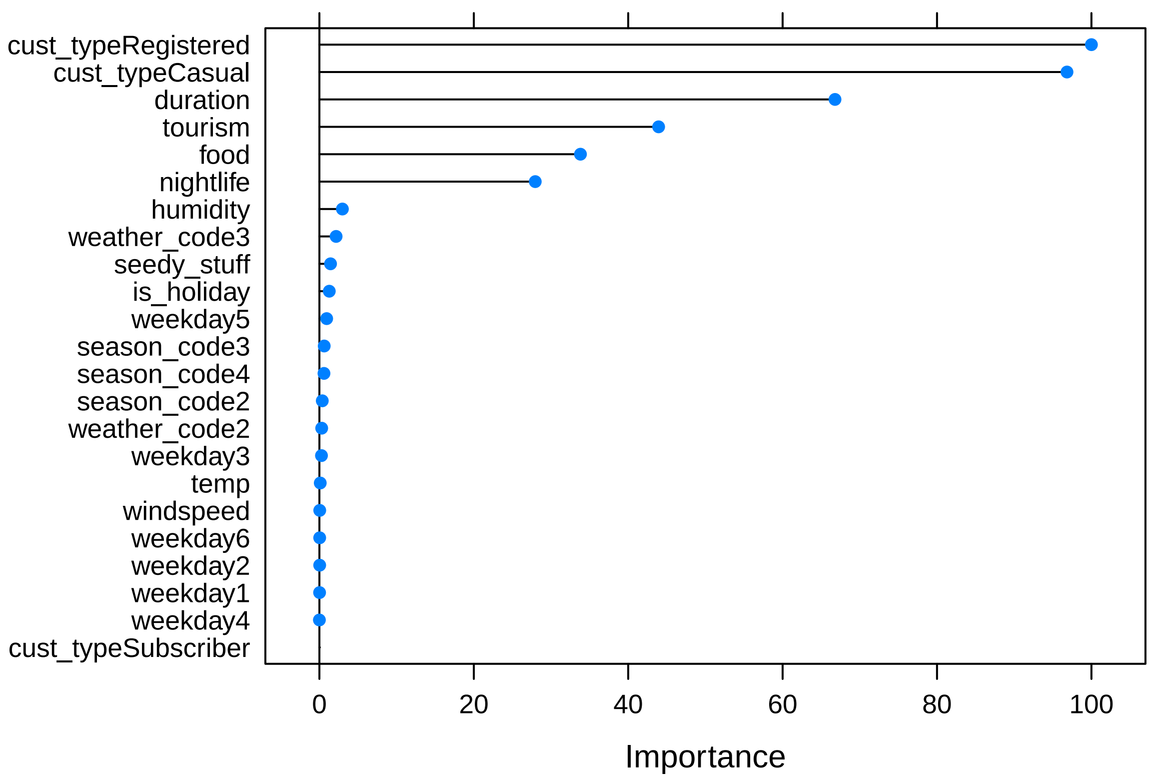
```r
plot(hybrid_model)
```




```r
plot(hybrid_model$finalModel, scale = 'adjr2')
```




```r
plot(varImp(hybrid_model))
```



Shrinkage

```r
# ridge regression
ridge_model = train(rentals ~ .,
                    data = train,
                    method = 'ridge',
                    preProcess = c('center', 'scale'),
                    tuneLength = 10,
                    # reducing the cv for speed
                    trControl = trainControl(method = 'cv', number = 5))
```

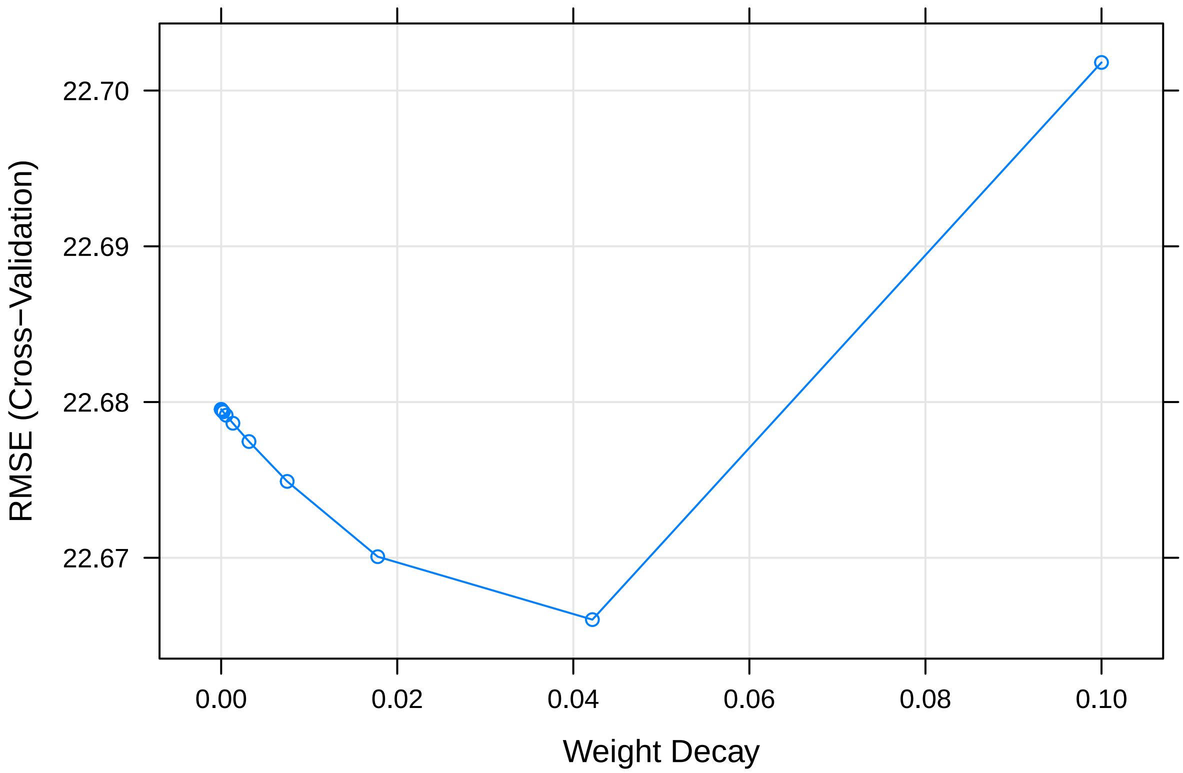
Ridge regression

```
## Loading required package: elasticnet
## Loading required package: lars
## Loaded lars 1.2
```

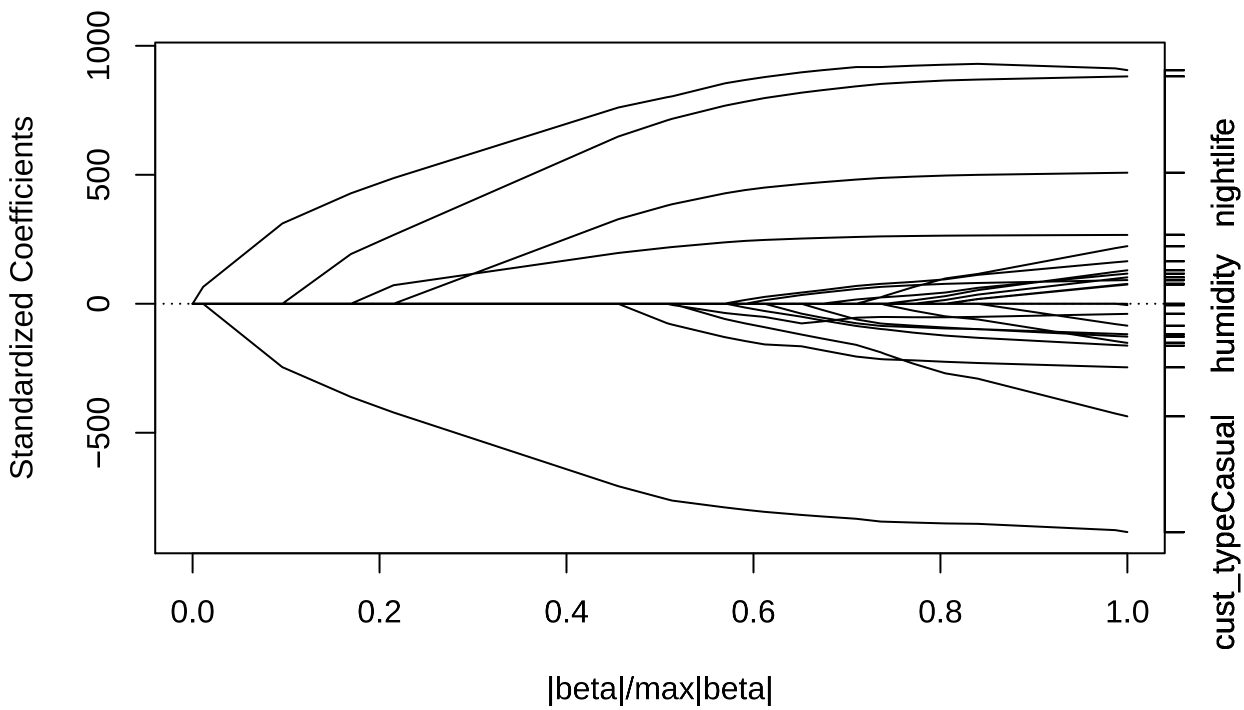
```r
print(ridge_model)
```

```
## Ridge Regression
##
## 17544 samples
##    23 predictor
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (5 fold)
##
## Summary of sample sizes: 14034, 14036, 14036, 14035, 14035
##
## Resampling results across tuning parameters:
##
##   lambda        RMSE      Rsquared   RMSE SD    Rsquared SD
##   0.0000000000  22.67953  0.3149871  0.3498054  0.01362003
##   0.0001000000  22.67946  0.3149911  0.3499419  0.01361930
##   0.0002371374  22.67937  0.3149966  0.3501295  0.01361831
##   0.0005623413  22.67915  0.3150095  0.3505753  0.01361604
##   0.0013335214  22.67864  0.3150398  0.3516381  0.01361110
##   0.0031622777  22.67746  0.3151091  0.3541830  0.01360175
##   0.0074989421  22.67490  0.3152609  0.3602675  0.01359250
##   0.0177827941  22.67007  0.3155523  0.3740618  0.01363145
##   0.0421696503  22.66603  0.3158268  0.3961864  0.01383785
##   0.1000000000  22.70180  0.3139997  0.3636063  0.01350114
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was lambda = 0.04216965.
```

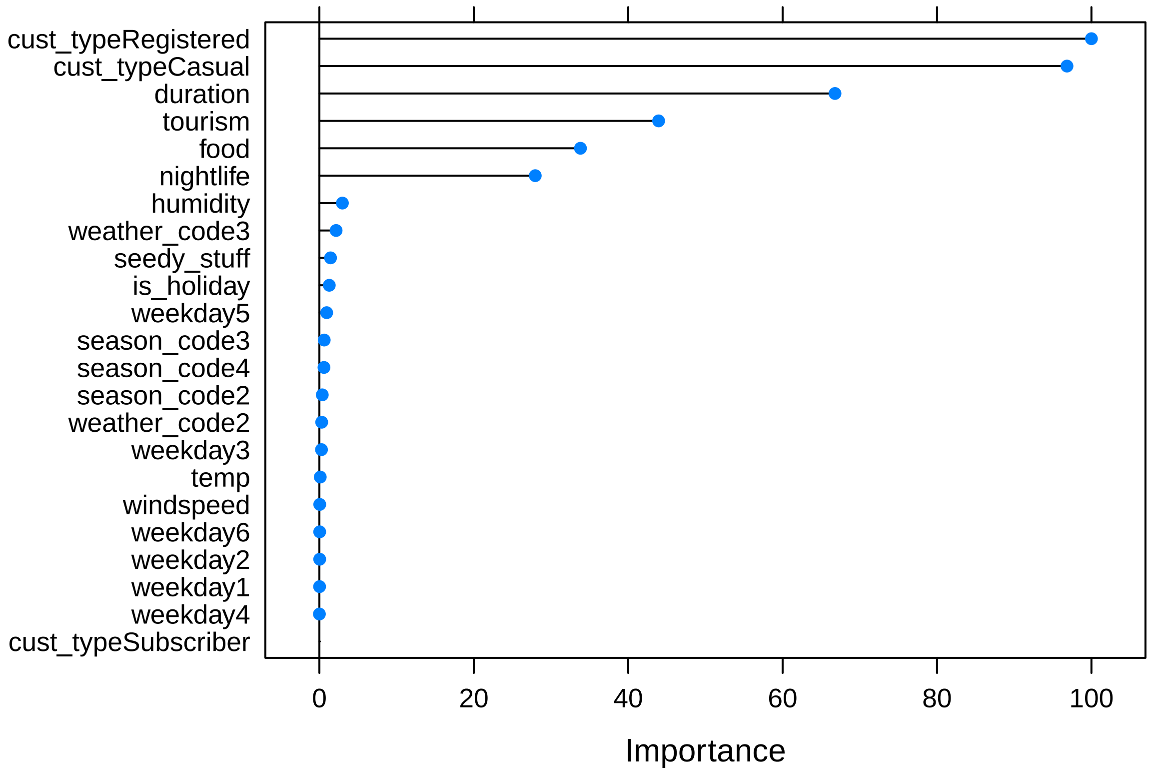
```r
plot(ridge_model)
```




```r
plot(ridge_model$finalModel)
```



```r
plot(varImp(ridge_model))
```




```r
# get the coefficients for the model
# NOTE: shrinkage methods don't have intercept terms
ridge_coefs = predict(ridge_model$finalModel, type = 'coef', mode = 'norm')$coefficients

# ridge regression with variable selection
ridge_model2 = train(rentals ~ .,
                     data = train,
                     method = 'foba',
                     preProcess = c('center', 'scale'),
                     tuneLength = 10,
                     trControl = trainControl(method = 'cv', number = 5))
```

```
## Loading required package: foba
```

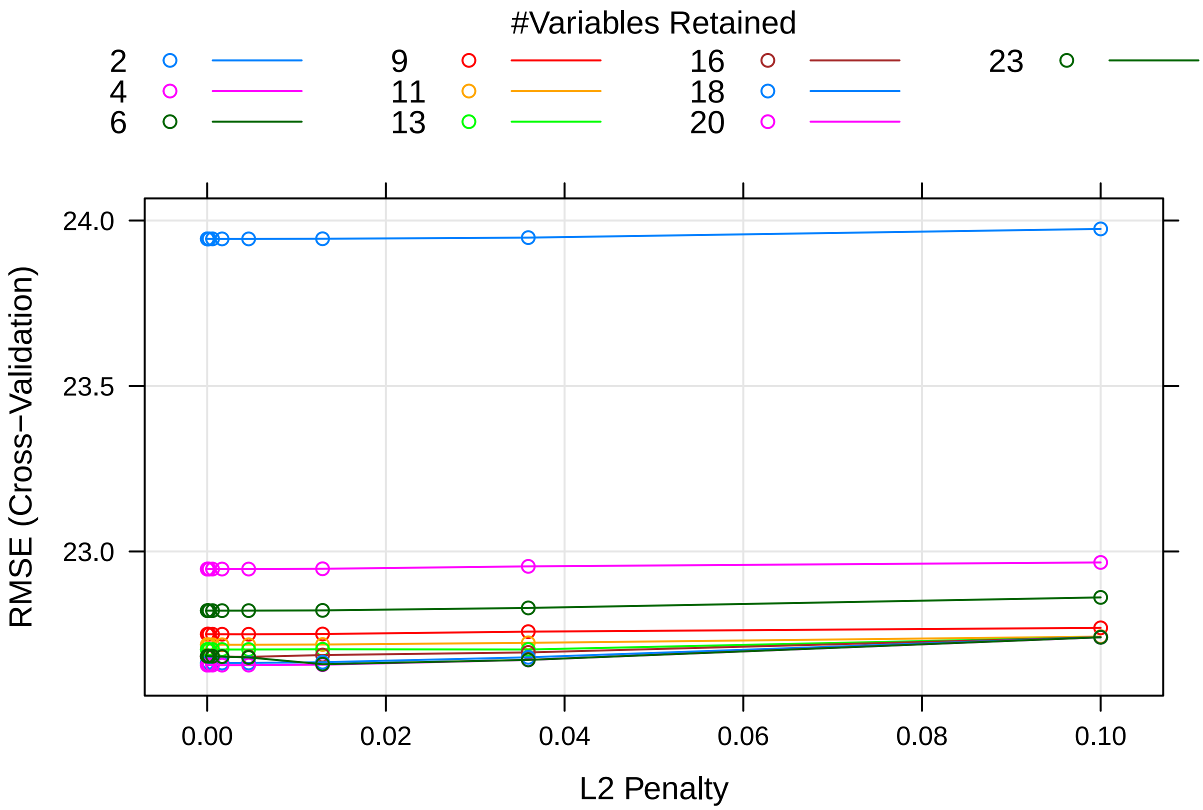
```r
print(ridge_model2)
```

```
## Ridge Regression with Variable Selection
##
## 17544 samples
##    23 predictor
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (5 fold)
##
## Summary of sample sizes: 14035, 14034, 14035, 14036, 14036
```

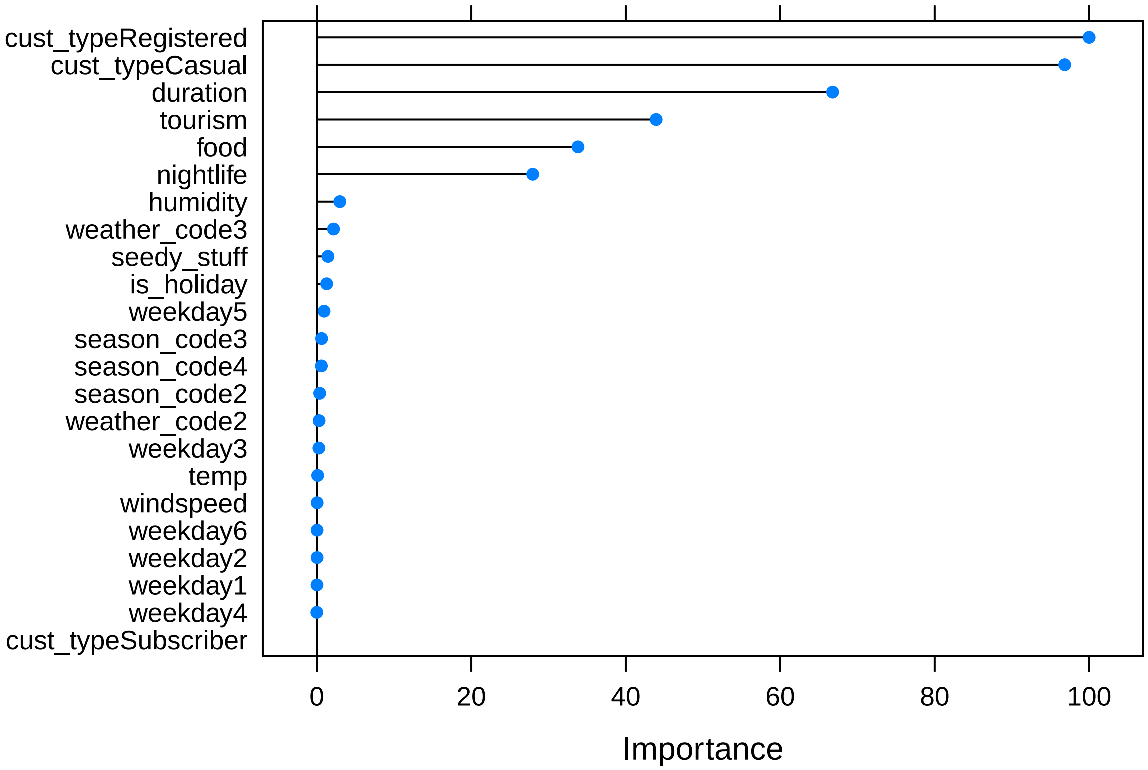
```
##
## Resampling results across tuning parameters:
##
##   lambda        k   RMSE      Rsquared   RMSE SD    Rsquared SD
##   1.000000e-05   2  23.94456  0.2365094  0.8104499  0.01567075
##   1.000000e-05   4  22.94687  0.2987498  0.7783755  0.01272099
##   1.000000e-05   6  22.82122  0.3065139  0.7393428  0.01316586
##   1.000000e-05   9  22.74995  0.3107757  0.7451701  0.01334922
##   1.000000e-05  11  22.71817  0.3127270  0.7378416  0.01356886
##   1.000000e-05  13  22.70371  0.3136189  0.7391890  0.01397096
##   1.000000e-05  16  22.68212  0.3149314  0.7576442  0.01397897
##   1.000000e-05  18  22.66295  0.3160766  0.7440202  0.01341188
##   1.000000e-05  20  22.65665  0.3164512  0.7413252  0.01325021
##   1.000000e-05  23  22.68386  0.3150051  0.7094941  0.01534515
##   2.782559e-05   2  23.94456  0.2365094  0.8104535  0.01567074
##   2.782559e-05   4  22.94687  0.2987498  0.7783823  0.01272102
##   2.782559e-05   6  22.82122  0.3065139  0.7393501  0.01316591
##   2.782559e-05   9  22.74995  0.3107757  0.7451771  0.01334925
##   2.782559e-05  11  22.71817  0.3127270  0.7378486  0.01356888
##   2.782559e-05  13  22.70371  0.3136189  0.7391960  0.01397097
##   2.782559e-05  16  22.68212  0.3149315  0.7576503  0.01397914
##   2.782559e-05  18  22.66295  0.3160766  0.7440297  0.01341219
##   2.782559e-05  20  22.65665  0.3164512  0.7413347  0.01325050
##   2.782559e-05  23  22.68384  0.3150059  0.7095204  0.01534420
##   7.742637e-05   2  23.94456  0.2365094  0.8104634  0.01567074
##   7.742637e-05   4  22.94687  0.2987498  0.7784011  0.01272109
##   7.742637e-05   6  22.82122  0.3065139  0.7393704  0.01316603
##   7.742637e-05   9  22.74994  0.3107757  0.7451968  0.01334932
##   7.742637e-05  11  22.71816  0.3127270  0.7378679  0.01356894
##   7.742637e-05  13  22.70371  0.3136188  0.7392155  0.01397100
##   7.742637e-05  16  22.68211  0.3149317  0.7576675  0.01397963
##   7.742637e-05  18  22.66295  0.3160766  0.7440559  0.01341307
##   7.742637e-05  20  22.65665  0.3164512  0.7413612  0.01325132
##   7.742637e-05  23  22.68379  0.3150080  0.7095934  0.01534157
##   2.154435e-04   2  23.94455  0.2365093  0.8104908  0.01567072
##   2.154435e-04   4  22.94686  0.2987497  0.7784534  0.01272127
##   2.154435e-04   6  22.82121  0.3065138  0.7394268  0.01316636
##   2.154435e-04   9  22.74994  0.3107757  0.7452514  0.01334953
##   2.154435e-04  11  22.71816  0.3127269  0.7379217  0.01356909
##   2.154435e-04  13  22.70370  0.3136187  0.7392698  0.01397108
##   2.154435e-04  16  22.68209  0.3149323  0.7577151  0.01398097
##   2.154435e-04  18  22.66294  0.3160765  0.7441287  0.01341549
##   2.154435e-04  20  22.65664  0.3164511  0.7414348  0.01325359
##   2.154435e-04  23  22.68367  0.3150138  0.7097965  0.01533423
##   5.994843e-04   2  23.94454  0.2365093  0.8105673  0.01567066
##   5.994843e-04   4  22.94685  0.2987496  0.7785988  0.01272179
##   5.994843e-04   6  22.82119  0.3065136  0.7395838  0.01316729
##   5.994843e-04   9  22.74992  0.3107755  0.7454031  0.01335011
##   5.994843e-04  11  22.71814  0.3127268  0.7380713  0.01356953
##   5.994843e-04  13  22.70368  0.3136183  0.7394205  0.01397130
##   5.994843e-04  16  22.68203  0.3149339  0.7578462  0.01398462
##   5.994843e-04  18  22.66292  0.3160760  0.7443301  0.01342215
##   5.994843e-04  20  22.65662  0.3164506  0.7416384  0.01325982
##   5.994843e-04  23  22.68332  0.3150298  0.7103616  0.01531374
```

```
##    1.668101e-03   2  23.94452  0.2365092  0.8107795  0.01567051
##    1.668101e-03   4  22.94683  0.2987491  0.7790021  0.01272323
##    1.668101e-03   6  22.82116  0.3065131  0.7400194  0.01316985
##    1.668101e-03   9  22.74987  0.3107750  0.7458240  0.01335171
##    1.668101e-03  11  22.71809  0.3127262  0.7384863  0.01357071
##    1.668101e-03  13  22.70366  0.3136169  0.7398385  0.01397187
##    1.668101e-03  16  22.68190  0.3149371  0.7582008  0.01399411
##    1.668101e-03  18  22.66291  0.3160735  0.7448821  0.01344004
##    1.668101e-03  20  22.65653  0.3164469  0.7422203  0.01327586
##    1.668101e-03  23  22.68239  0.3150735  0.7119221  0.01525612
##    4.641589e-03   2  23.94451  0.2365090  0.8113673  0.01567009
##    4.641589e-03   4  22.94686  0.2987473  0.7801155  0.01272718
##    4.641589e-03   6  22.82118  0.3065109  0.7412235  0.01317690
##    4.641589e-03   9  22.74985  0.3107725  0.7469845  0.01335602
##    4.641589e-03  11  22.71807  0.3127236  0.7396308  0.01357388
##    4.641589e-03  13  22.70369  0.3136120  0.7409907  0.01397318
##    4.641589e-03  16  22.68142  0.3149543  0.7595978  0.01399354
##    4.641589e-03  18  22.66313  0.3160573  0.7463585  0.01348533
##    4.641589e-03  20  22.65654  0.3164211  0.7437804  0.01332786
##    4.641589e-03  23  22.67979  0.3151991  0.7162399  0.01509888
##    1.291550e-02   2  23.94485  0.2365084  0.8129809  0.01566892
##    1.291550e-02   4  22.94767  0.2987381  0.7831458  0.01273779
##    1.291550e-02   6  22.82196  0.3065010  0.7445116  0.01319577
##    1.291550e-02   9  22.75055  0.3107582  0.7501326  0.01336696
##    1.291550e-02  11  22.71880  0.3127084  0.7427394  0.01358167
##    1.291550e-02  13  22.70456  0.3135893  0.7441153  0.01397483
##    1.291550e-02  16  22.68698  0.3146436  0.7597533  0.01375061
##    1.291550e-02  18  22.66529  0.3159642  0.7501002  0.01358421
##    1.291550e-02  20  22.65790  0.3163690  0.7473421  0.01336677
##    1.291550e-02  23  22.65932  0.3162881  0.7455381  0.01347411
##    3.593814e-02   2  23.94837  0.2365066  0.8173034  0.01566577
##    3.593814e-02   4  22.95502  0.2986825  0.7910789  0.01276449
##    3.593814e-02   6  22.82936  0.3064453  0.7532006  0.01324298
##    3.593814e-02   9  22.75768  0.3106654  0.7583139  0.01339002
##    3.593814e-02  11  22.72388  0.3126383  0.7513138  0.01362344
##    3.593814e-02  13  22.70376  0.3137103  0.7520616  0.01395254
##    3.593814e-02  16  22.69494  0.3142410  0.7597370  0.01414542
##    3.593814e-02  18  22.68007  0.3151552  0.7497092  0.01391205
##    3.593814e-02  20  22.67235  0.3156213  0.7469827  0.01406368
##    3.593814e-02  23  22.67235  0.3156213  0.7469827  0.01406368
##    1.000000e-01   2  23.97464  0.2365020  0.8281124  0.01565768
##    1.000000e-01   4  22.96686  0.2987378  0.8049871  0.01271964
##    1.000000e-01   6  22.86115  0.3050108  0.7781909  0.01263269
##    1.000000e-01   9  22.76919  0.3106199  0.7762437  0.01308251
##    1.000000e-01  11  22.74269  0.3122714  0.7776390  0.01357149
##    1.000000e-01  13  22.74080  0.3123997  0.7794152  0.01358921
##    1.000000e-01  16  22.74080  0.3123997  0.7794152  0.01358921
##    1.000000e-01  18  22.74080  0.3123997  0.7794152  0.01358921
##    1.000000e-01  20  22.74080  0.3123997  0.7794152  0.01358921
##    1.000000e-01  23  22.74080  0.3123997  0.7794152  0.01358921
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were k = 20 and lambda = 0.001668101.
```

```r
plot(ridge_model2)
```




```r
plot(varImp(ridge_model2))
```



Selection, ridge regression, and lasso are just a couple techniques at our disposal for decreasing our model size. See this page for a list of other available options to try out if you like.

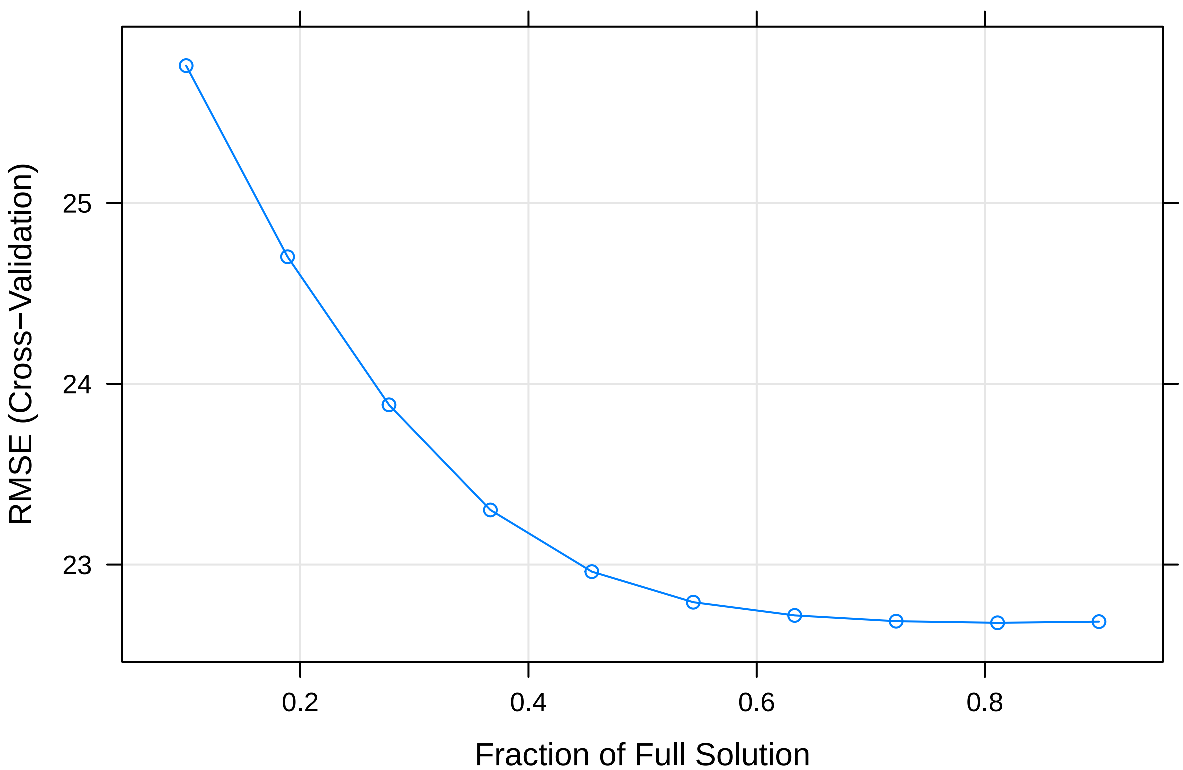
```r
lasso_model = train(rentals ~ .,
                    data = na.omit(train),
                    method = 'lasso',
                    preProc = c('scale', 'center'),
                    tuneLength = 10,
                    trControl = trainControl(method = 'cv', number = 5))

print(lasso_model)
```

Lasso

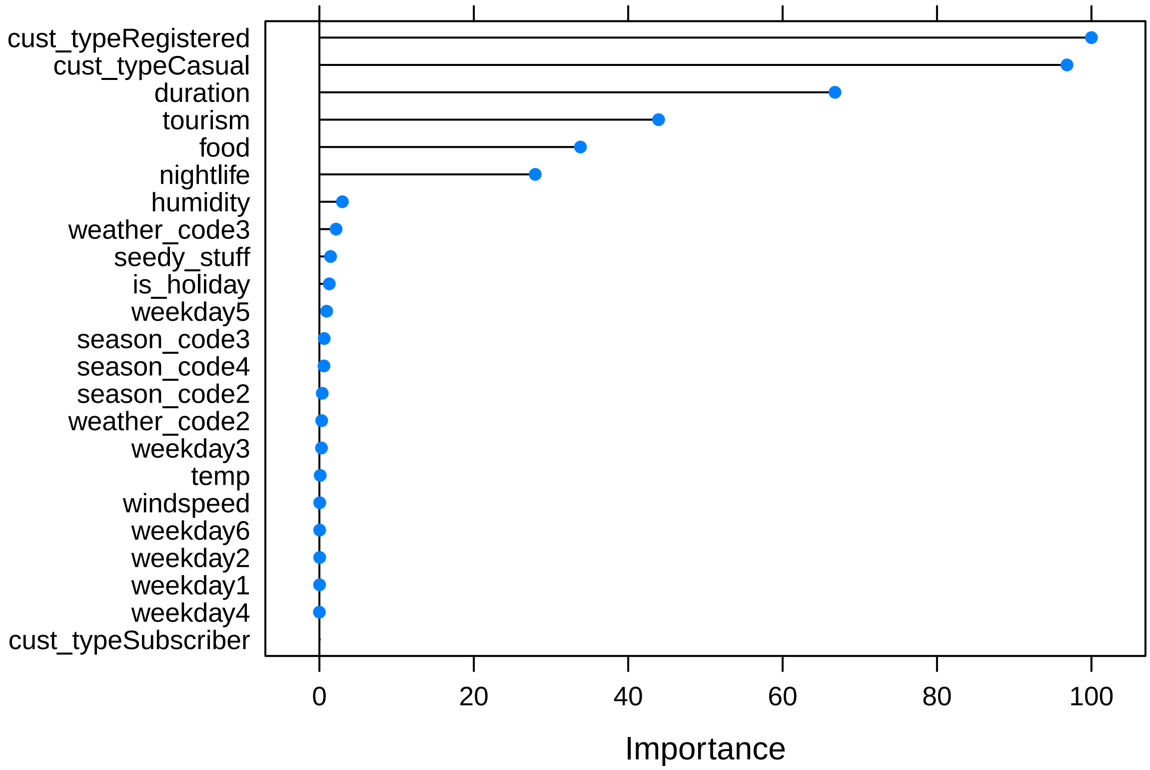
```
## The lasso
##
## 17544 samples
##    23 predictor
##
## Pre-processing: scaled, centered
## Resampling: Cross-Validated (5 fold)
##
## Summary of sample sizes: 14036, 14036, 14034, 14035, 14035
##
## Resampling results across tuning parameters:
##
```

```
##    fraction   RMSE      Rsquared   RMSE SD    Rsquared SD
##    0.1000000  25.75970  0.2068984  1.0870948  0.019556361
##    0.1888889  24.70244  0.2625006  1.1146097  0.016440291
##    0.2777778  23.88351  0.2893794  1.1147130  0.014011003
##    0.3666667  23.30187  0.2984203  1.0855549  0.012363285
##    0.4555556  22.96077  0.3046642  1.0395281  0.012220116
##    0.5444444  22.79172  0.3098546  0.9871854  0.011273764
##    0.6333333  22.71866  0.3129602  0.9474077  0.010119256
##    0.7222222  22.68669  0.3146011  0.9252657  0.009311565
##    0.8111111  22.67779  0.3150491  0.9174462  0.008395242
##    0.9000000  22.68431  0.3146268  0.9147171  0.007201117
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was fraction = 0.8111111.
```

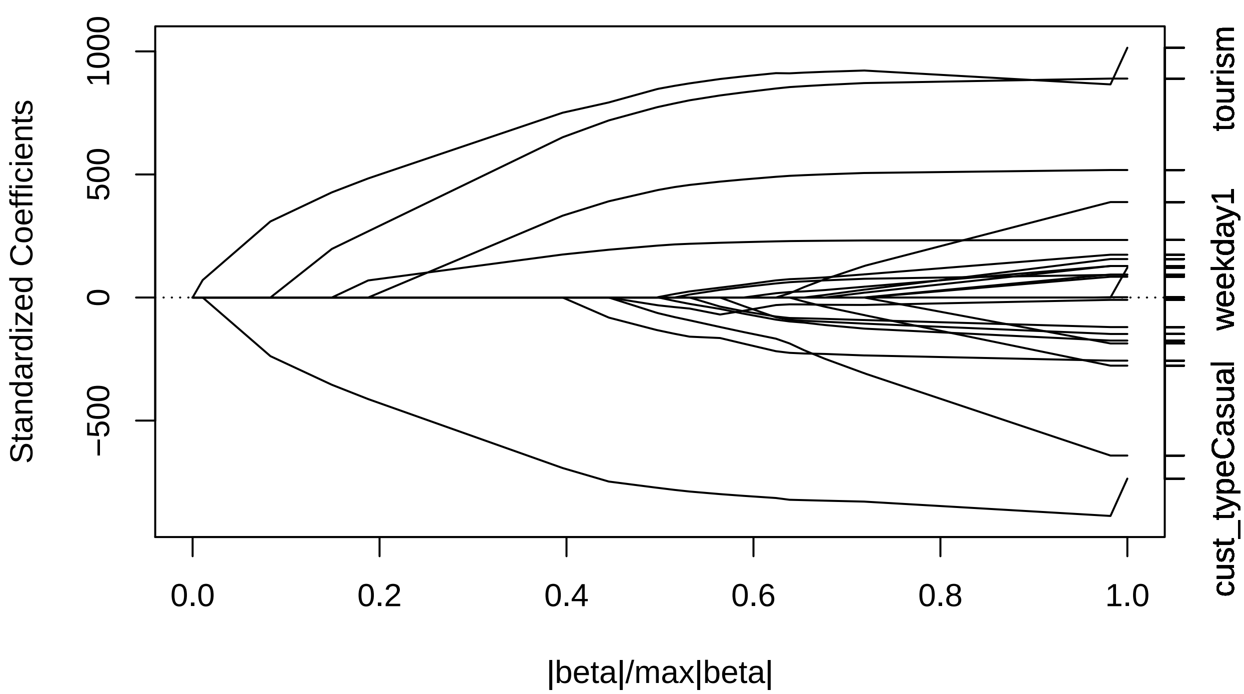
```r
plot(lasso_model)
```




```r
plot(varImp(lasso_model))
```




```r
plot(lasso_model$finalModel)
```




```r
# get the model coefficients
lasso_coefs = predict(lasso_model$finalModel, type = 'coef', mode = 'norm')$coefficients
```

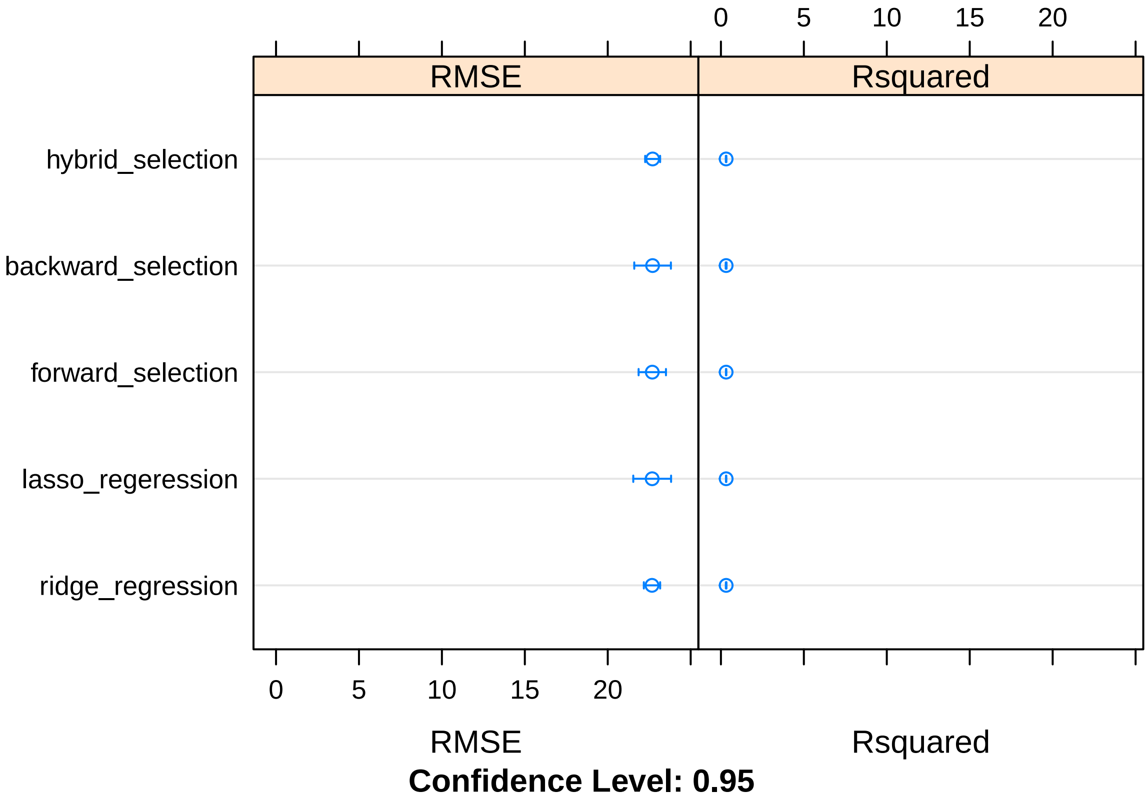
Measuring predictive accuracy

All right, now we've got a nice collection of models. Which one should we report?

```
results = resamples(list(forward_selection = forward_model,
                              backward_selection = backward_model,
                              hybrid_selection = hybrid_model,
                              ridge_regression = ridge_model,
                              lasso_regeression = lasso_model))

# compare RMSE and R-squared
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: forward_selection, backward_selection, hybrid_selection, ridge_regression, lasso_regeression
## Number of resamples: 5
##
## RMSE
##                     Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## forward_selection  21.97   22.09  22.68 22.68   23.18 23.50    0
## backward_selection 21.62   21.96  22.91 22.70   23.28 23.74    0
## hybrid_selection   22.34   22.35  22.75 22.71   22.89 23.19    0
## ridge_regression   22.05   22.64  22.65 22.67   22.86 23.12    0
## lasso_regeression  21.46   21.96  23.12 22.68   23.24 23.60    0
##
## Rsquared
##                      Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## forward_selection  0.2978  0.3157 0.3166 0.3146  0.3200 0.3231    0
## backward_selection 0.2824  0.3049 0.3263 0.3140  0.3271 0.3294    0
## hybrid_selection   0.2937  0.3112 0.3140 0.3135  0.3163 0.3325    0
## ridge_regression   0.2922  0.3163 0.3211 0.3158  0.3214 0.3281    0
## lasso_regeression  0.3075  0.3087 0.3118 0.3150  0.3201 0.3272    0
```

```
# plot results
dotplot(results)
```



Those are in-sample statistics however, so if we want to compare the model's out-of-sample prediction accuracy, we need to compute the RMSE using the test data we held out. Let's compare two models: backward selection and lasso:

```
backward_predictions = predict(backward_model, test)
sqrt(mean((backward_predictions - test$rentals)^2 , na.rm = TRUE))
```

```
## [1] 23.12121
```

```
lasso_predictions = predict(lasso_model, test)
sqrt(mean((lasso_predictions - test$rentals)^2 , na.rm = TRUE))
```

```
## [1] 23.15005
```

Project tips

Check out this list of different model selection methods and try a couple out.

- How do they work?
- Which works best?

Once you've spent some time exploring candidate models, pick one and use it in your report.