# Intro to linear regression with R

Víctor Peña

We'll analyze a dataset in Sheather (2009) that has information about 150 Italian restaurants in Manhattan that were open in 2001 (some of them are closed now). The variables are:

- `Case`: case-indexing variable
- `Restaurant`: name of the restaurant
- `Price`: average price of a meal and a drink
- `Food`: average Zagat rating of the quality of the food (from 0 to 25)
- `Decor`: same as above, but with quality of the decor
- `Service`: same as above, but with quality of service
- `East`: it is equal to `East` if the restaurant is on the East Side (i.e. east of Fifth Ave) and `West` otherwise.

In our analysis, the response variable will be `Price`.

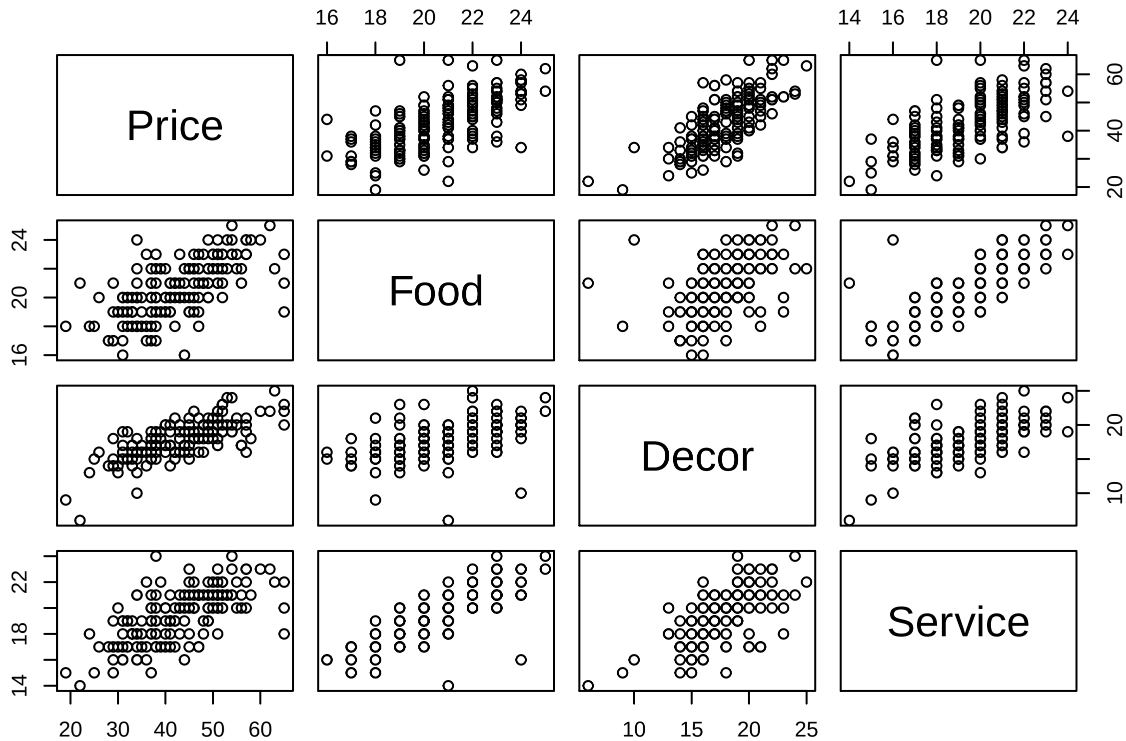
The command below reads in the dataset

```
nyc = read.csv("http://vicpena.github.io/sta9750/spring19/nyc.csv")
```

## Exploratory data analysis

The function `pairs` creates a scatterplot matrix for `numeric` variables:

```
library(tidyverse)
nycplot = nyc %>% select(-Case, -Restaurant, - East)
pairs(nycplot)
```




The dataset `nycplot` excludes the variables `Case`, `Restaurant`, and `East`.

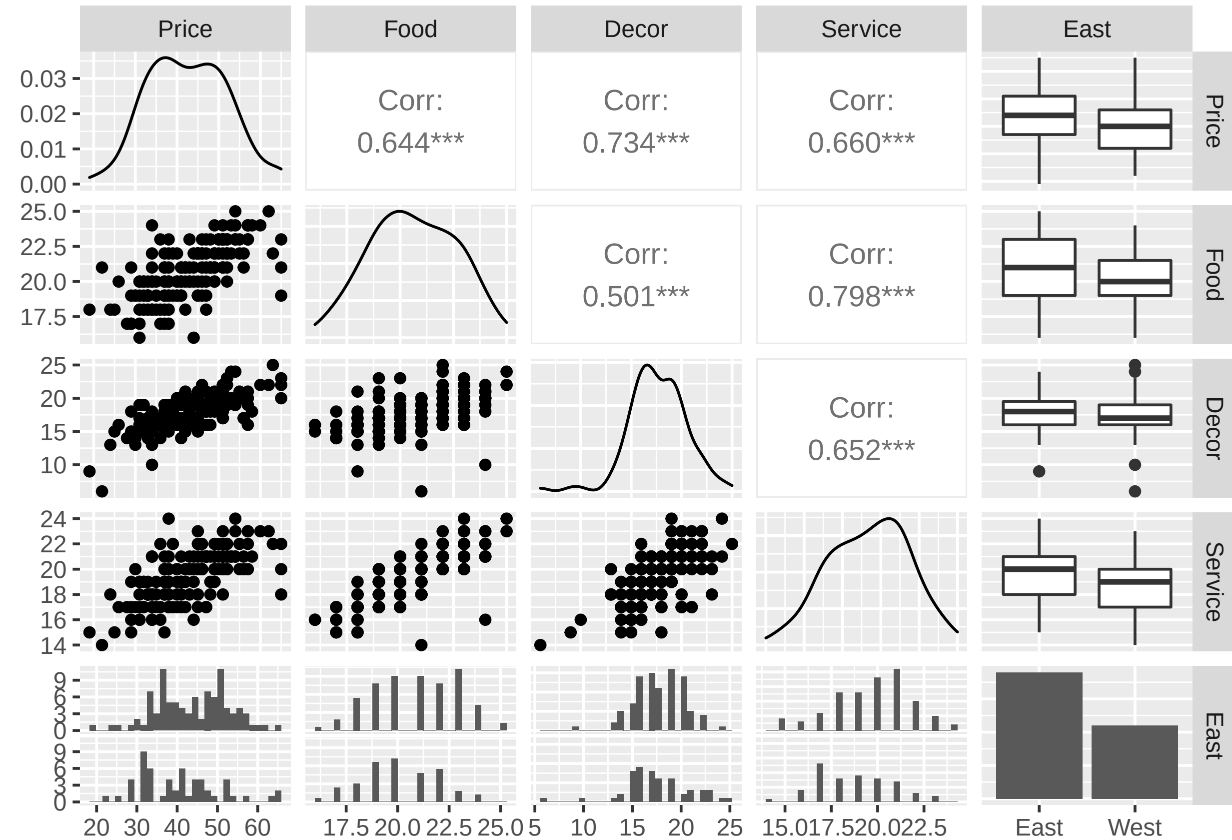
We can get quick and dirty summaries of the variables with `summary`. An advantage is that it's able to handle categorical variables, such as `East`:

```
summary(nyc)
```

```
##       Case          Restaurant             Price            Food      
##  Min.   :  2.00   Length:150         Min.   :19.00   Min.   :16.00  
##  1st Qu.: 41.50   Class :character   1st Qu.:35.25   1st Qu.:19.00  
##  Median : 83.50   Mode  :character   Median :42.00   Median :21.00  
##  Mean   : 84.17                      Mean   :42.62   Mean   :20.61  
##  3rd Qu.:124.75                      3rd Qu.:49.75   3rd Qu.:22.00  
##  Max.   :168.00                      Max.   :65.00   Max.   :25.00  
##      Decor          Service          East          
##  Min.   : 6.00   Min.   :14.00   Length:150        
##  1st Qu.:16.00   1st Qu.:18.00   Class :character  
##  Median :18.00   Median :20.00   Mode  :character  
##  Mean   :17.69   Mean   :19.39                     
##  3rd Qu.:19.00   3rd Qu.:21.00                     
##  Max.   :25.00   Max.   :24.00
```

The function `ggpairs` in `library(GGally)` produces the equivalent plot, but with `ggplot2`:

```r
library(GGally)
nycplot = nyc %>% select(-Case, -Restaurant)
ggpairs(nycplot)
```




Do you see any interesting patterns?

## Fitting regression models with the `lm` function

Fitting regression models with `R` is easy. For example, we can fit a model where the outcome is `Price` and the predictors are `Food`, `Decor`, `Service`, and `East` with the code

```r
mod = lm(Price ~ Food + Decor + Service + East, data = nyc)
```

Calling the object `mod` only gives us coefficients:

```r
mod
```

```
##
## Call:
## lm(formula = Price ~ Food + Decor + Service + East, data = nyc)
##
## Coefficients:
## (Intercept)         Food        Decor      Service     EastWest
##  -23.644163     1.634869     1.865549     0.007626    -1.613350
```

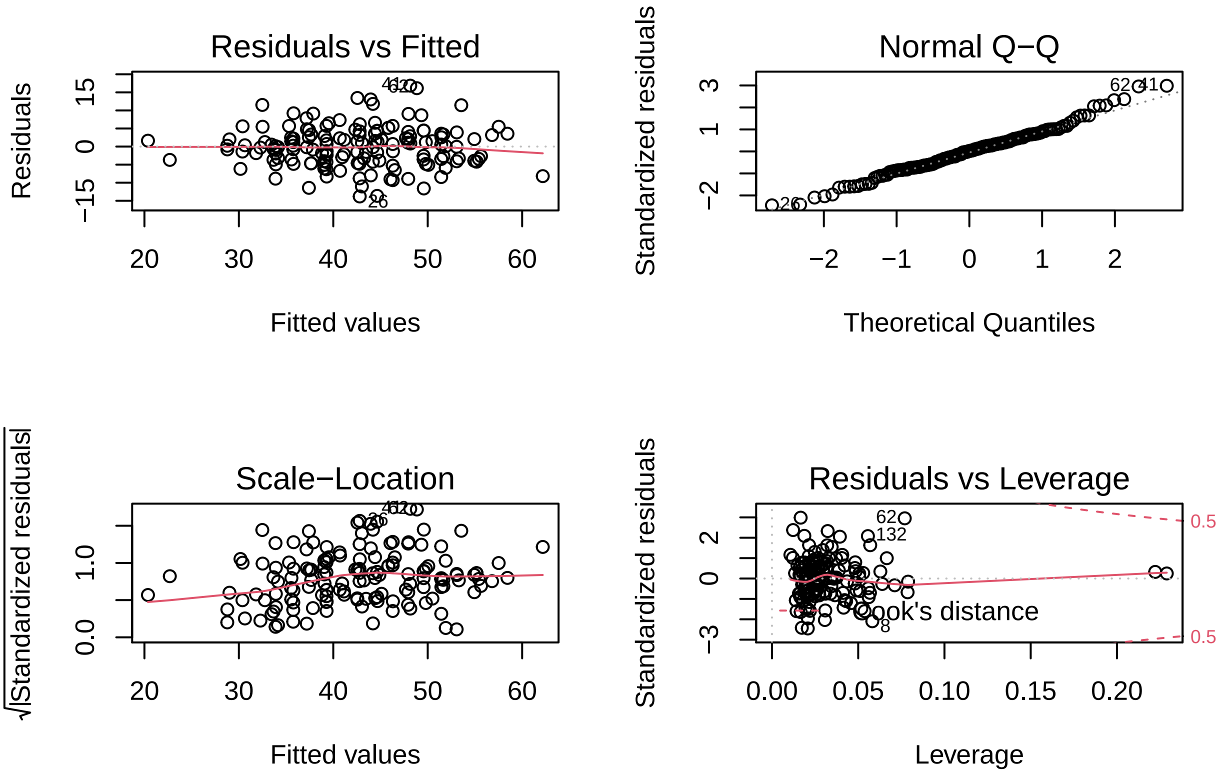
If we want $p$-values, $R^2$, and more, we can get them with `summary()`:

```
summary(mod)
```

```
##
## Call:
## lm(formula = Price ~ Food + Decor + Service + East, data = nyc)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.7995  -3.8323   0.0997   3.3449  16.8484
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -23.644163   5.079278  -4.655 7.25e-06 ***
## Food          1.634869   0.384961   4.247 3.86e-05 ***
## Decor         1.865549   0.221396   8.426 3.22e-14 ***
## Service       0.007626   0.432210   0.018    0.986
## EastWest     -1.613350   1.000385  -1.613    0.109
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.692 on 145 degrees of freedom
## Multiple R-squared:  0.6466, Adjusted R-squared:  0.6369
## F-statistic: 66.34 on 4 and 145 DF,  p-value: < 2.2e-16
```

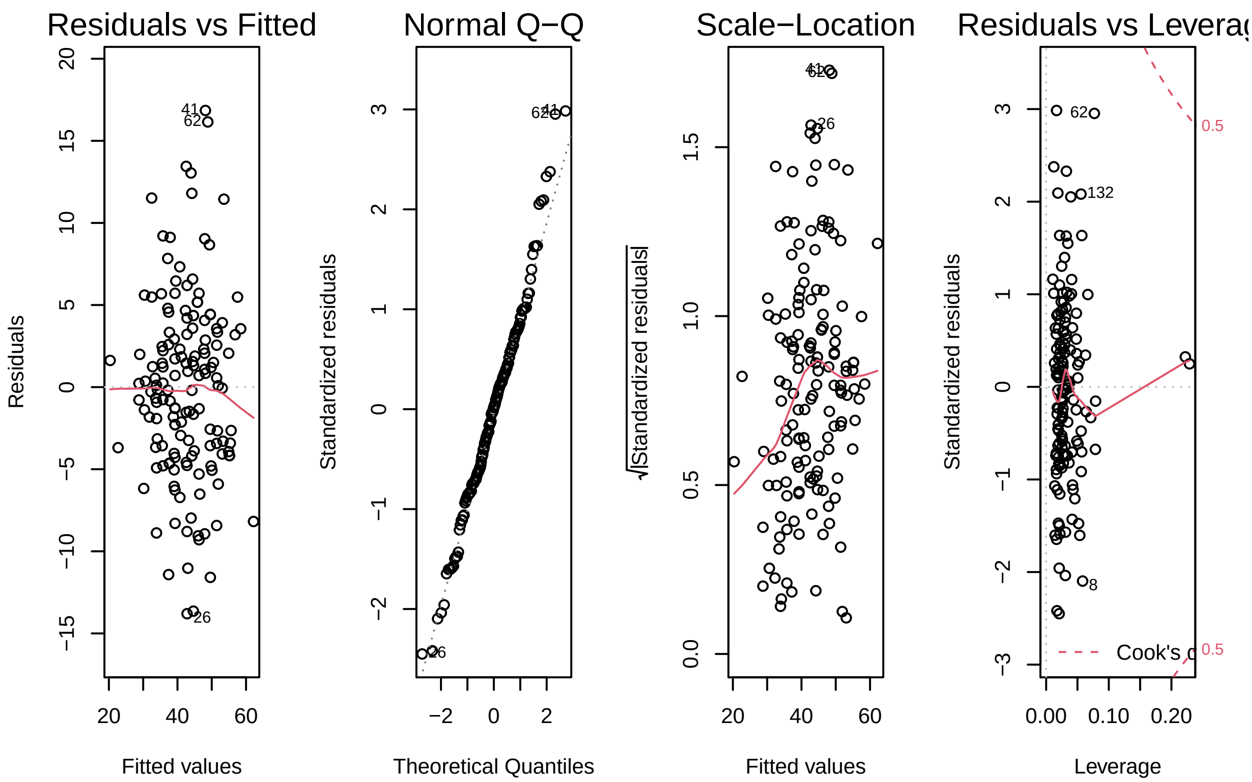
We can get diagnostic plots by `plot`ting the model. That will give us 4 diagnostic plots. We can arrange them in a figure with 2 rows and 2 columns with `par(mfrow=c(2,2))`:

```
par(mfrow=c(2,2))
plot(mod)
```




If, for some reason, we want them in 1 row and 4 columns:

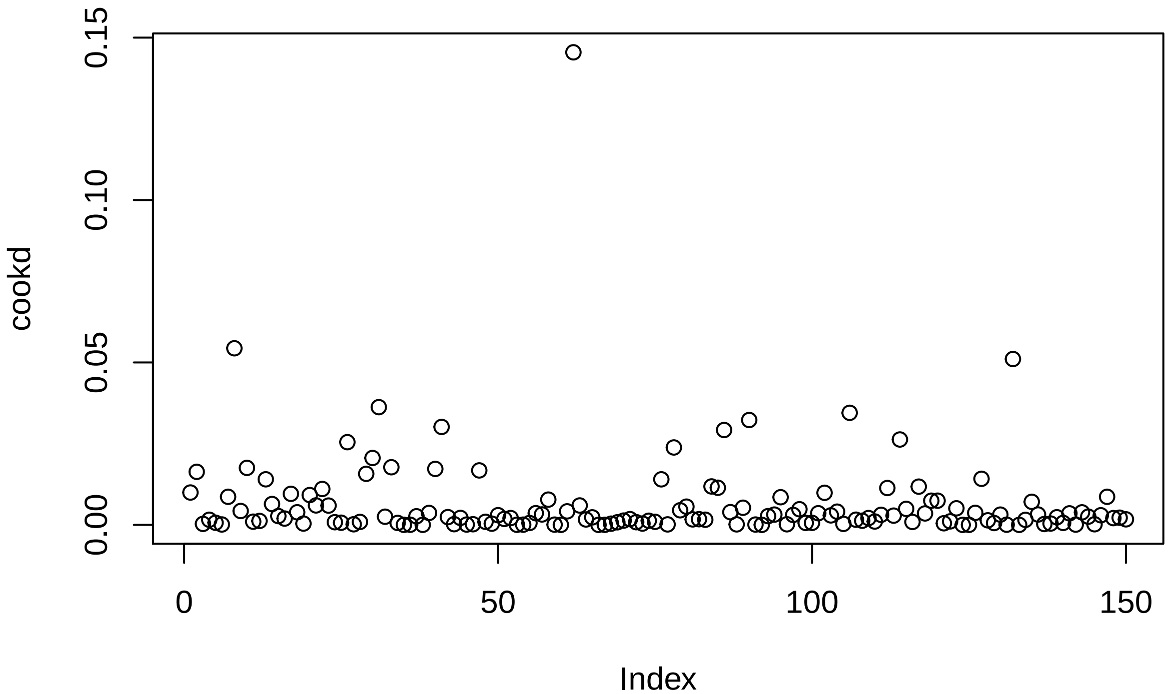
```r
par(mfrow=c(1,4))
plot(mod)
```




The instruction `par(mfrow=c(<rows>, <columns>)` isn't specific to "models". We can use it to arrange figures with multiple rows and columns of plots in `library(graphics)`. Unfortunately, it doesn't work with `ggplot2`. The analogue instruction for `ggplot2` is `grid.arrange` (see `ggplot2` handout for examples).

We can extract diagnostics from `mod`. For example, if we want to extract Cook's distances and plot them against observation number, we can use:

```
cookd = cooks.distance(mod)
plot(cookd)
```




Other useful functions are `hatvalues` (for leverages), `residuals` (for residuals), and `rstandard` (for standardized residuals).

## Automatic model selection

### Backward, forward, and stepwise

Backward selection with AIC:

```
step(mod, direction='backward')
```

```
## Start:  AIC=526.64
## Price ~ Food + Decor + Service + East
##
##           Df Sum of Sq    RSS    AIC
## - Service  1      0.01 4698.0 524.64
## <none>                 4698.0 526.64
## - East     1     84.27 4782.2 527.30
## - Food     1    584.35 5282.3 542.22
## - Decor    1   2300.45 6998.4 584.42
##
## Step:  AIC=524.64
## Price ~ Food + Decor + East
##
##         Df Sum of Sq    RSS    AIC
```

```
## <none>                4698.0 524.64
## - East    1     87.24 4785.2 525.40
## - Food    1   1166.83 5864.8 555.91
## - Decor   1   3062.26 7760.2 597.92
```

```
##
## Call:
## lm(formula = Price ~ Food + Decor + East, data = nyc)
##
## Coefficients:
## (Intercept)         Food        Decor     EastWest
##     -23.628        1.640        1.867       -1.616
```

If we want to do forward selection, we have to give a starting model and a bigger model that contains the all the variables that we might want to include in our model.

```
nullmod = lm(Price ~ 1, data = nyc) # no variables
fwd = step(nullmod,
           scope=list(upper=mod),
           direction='forward')
```

```
## Start:  AIC=674.68
## Price ~ 1
##
##            Df Sum of Sq     RSS    AIC
## + Decor     1    7157.2  6138.1 560.75
## + Service   1    5794.1  7501.3 590.83
## + Food      1    5505.8  7789.5 596.48
## + East      1     380.3 12915.0 672.33
## <none>                  13295.3 674.68
##
## Step:  AIC=560.75
## Price ~ Decor
##
##            Df Sum of Sq    RSS    AIC
## + Food      1   1352.93 4785.2 525.40
## + Service   1    763.57 5374.6 542.82
## + East      1    273.34 5864.8 555.91
## <none>                  6138.1 560.75
##
## Step:  AIC=525.4
## Price ~ Decor + Food
##
##            Df Sum of Sq    RSS    AIC
## + East      1    87.239 4698.0 524.64
## <none>                  4785.2 525.40
## + Service   1     2.980 4782.2 527.30
##
## Step:  AIC=524.64
## Price ~ Decor + Food + East
##
##            Df Sum of Sq  RSS    AIC
## <none>                  4698 524.64
## + Service   1  0.010086 4698 526.64
```

In the code above, the starting point was a model with no variables (`nullmod`) and the model that included the variables under consideration is `mod` (which contains `Food`, `Service`, `Decor`, and `East`).

We can do forward selection starting with a model that has some variable(s) already. For example, we can start with a model that has `Service` already in.

```
mod2 = lm( Price ~ Service, data = nyc)
fwd = step(mod2,
           scope=list(upper=mod),
           direction='forward')
```

```
## Start:  AIC=590.83
## Price ~ Service
##
##          Df Sum of Sq    RSS    AIC
## + Decor  1   2126.70 5374.6 542.82
## + Food   1    498.33 7002.9 582.52
## <none>               7501.3 590.83
## + East   1      7.05 7494.2 592.69
##
## Step:  AIC=542.82
## Price ~ Service + Decor
##
##         Df Sum of Sq    RSS    AIC
## + Food  1    592.34 4782.2 527.30
## + East  1     92.26 5282.3 542.22
## <none>              5374.6 542.82
##
## Step:  AIC=527.3
## Price ~ Service + Decor + Food
##
##         Df Sum of Sq    RSS    AIC
## + East  1    84.268 4698.0 526.64
## <none>              4782.2 527.30
##
## Step:  AIC=526.64
## Price ~ Service + Decor + Food + East
```

We can do stepwise regression with `direction = 'both'`. In stepwise regression, variables can get in or out of the model. We can specify the smallest and biggest model in our search with `scope`. For example, if we want to start our stepwise search with a model has `Service` as a predictor and we want to restrict our search to models that include `Service` and potentially include all the other predictors:

```
mod2 = lm( Price ~ Service, data = nyc)
fwd = step(mod2,
           scope=list(lower=mod2, upper=mod),
           direction='both')
```

```
## Start:  AIC=590.83
## Price ~ Service
##
##          Df Sum of Sq    RSS    AIC
## + Decor  1   2126.70 5374.6 542.82
```

```
## + Food    1    498.33 7002.9 582.52
## <none>                7501.3 590.83
## + East    1      7.05 7494.2 592.69
##
## Step:  AIC=542.82
## Price ~ Service + Decor
##
##         Df Sum of Sq    RSS    AIC
## + Food    1    592.34 4782.2 527.30
## + East    1     92.26 5282.3 542.22
## <none>                5374.6 542.82
## - Decor   1   2126.70 7501.3 590.83
##
## Step:  AIC=527.3
## Price ~ Service + Decor + Food
##
##         Df Sum of Sq    RSS    AIC
## + East    1     84.27 4698.0 526.64
## <none>                4782.2 527.30
## - Food    1    592.34 5374.6 542.82
## - Decor   1   2220.71 7002.9 582.52
##
## Step:  AIC=526.64
## Price ~ Service + Decor + Food + East
##
##         Df Sum of Sq    RSS    AIC
## <none>                4698.0 526.64
## - East    1     84.27 4782.2 527.30
## - Food    1    584.35 5282.3 542.22
## - Decor   1   2300.45 6998.4 584.42
```

We can change our selection criterion to BIC by adding the command `k = log(number of observations)`. For example,

```r
n = nrow(nyc)
step(mod, direction='backward', k = log(n))
```

```
## Start:  AIC=541.69
## Price ~ Food + Decor + Service + East
##
##           Df Sum of Sq    RSS    AIC
## - Service  1      0.01 4698.0 536.68
## - East     1     84.27 4782.2 539.35
## <none>                 4698.0 541.69
## - Food     1    584.35 5282.3 554.27
## - Decor    1   2300.45 6998.4 596.46
##
## Step:  AIC=536.68
## Price ~ Food + Decor + East
##
##         Df Sum of Sq    RSS    AIC
## - East    1     87.24 4785.2 534.43
## <none>                4698.0 536.68
```

```
## - Food    1   1166.83 5864.8 564.95
## - Decor   1   3062.26 7760.2 606.95
##
## Step:  AIC=534.43
## Price ~ Food + Decor
##
##          Df Sum of Sq    RSS    AIC
## <none>                4785.2 534.43
## - Food    1    1352.9 6138.1 566.77
## - Decor   1    3004.3 7789.5 602.51
```

```
##
## Call:
## lm(formula = Price ~ Food + Decor, data = nyc)
##
## Coefficients:
## (Intercept)         Food        Decor
##     -25.678        1.730        1.845
```

### All subsets selection with `library(leaps)`

If we want to find the "best" model given a set of predictors according to BIC or adjusted $R^2$, `library(leaps)` is helpful. For example, if we want to consider all models that might contain `Food`, `Decor`, `Service`, and `East`:

```r
library(leaps)
allsubs = regsubsets(Price ~ Food + Decor + Service + East, data = nyc)
```

We can see the best models with 1, 2, 3, and 4 predictors using the `summary` function:

```r
summary(allsubs)
```

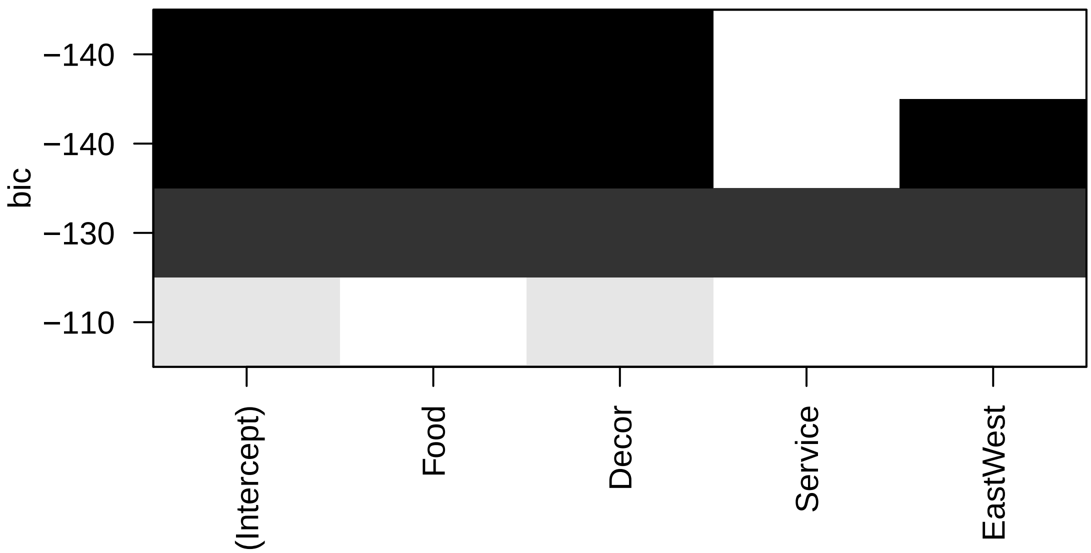
```
## Subset selection object
## Call: regsubsets.formula(Price ~ Food + Decor + Service + East, data = nyc)
## 4 Variables  (and intercept)
##          Forced in Forced out
## Food         FALSE      FALSE
## Decor        FALSE      FALSE
## Service      FALSE      FALSE
## EastWest     FALSE      FALSE
## 1 subsets of each size up to 4
## Selection Algorithm: exhaustive
##          Food Decor Service EastWest
## 1  ( 1 ) " "  "*"   " "     " "
## 2  ( 1 ) "*"  "*"   " "     " "
## 3  ( 1 ) "*"  "*"   " "     "*"
## 4  ( 1 ) "*"  "*"   "*"     "*"
```

If we restrict ourselves to all models that have, say, **exactly** 2 predictors, the "best" models according to AIC, BIC, and adjusted $R^2$ will coincide: it will be the model with 2 predictors that has the smallest residual sum of squares. The overall "best" model will be one of the 4 "best" models for a fixed number of predictors.

AIC and BIC need not agree on the overall best model, because they penalize model sizes differently (the penalty for AIC is smaller, which favors bigger models).

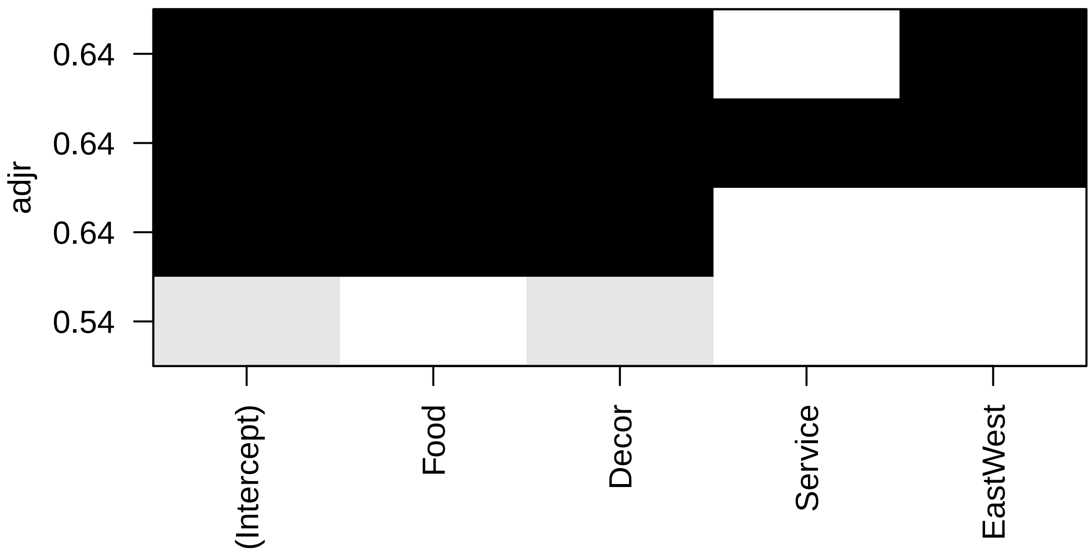
We can visualize the BICs of the "best" models (the model at the top of the plot is the best model overall):

```r
plot(allsubs)
```




We can also visualize the adjusted $R^2$s:

```r
plot(allsubs, scale = 'adjr')
```




### Prediction

The dataset `nyctest` has data for some Italian restaurants that weren't included in `nyc`. The command below reads the data for us.

```
nyctest = read.csv("http://vicpena.github.io/sta9750/spring19/nyctest.csv")
```

Let's see how well we predict the prices of the meals. We'll use the following model

```
mod = lm(Price ~ Food + Decor + East, data = nyc)
```

We can find point predictions and 99% prediction intervals as follows:

```
preds = predict(mod, newdata = nyctest, interval = 'prediction', level = 0.99)
```

Unfortunately, `predict` outputs an object of type `matrix`, but `data.frames` are more convenient for plotting (among other things). Let's convert preds into a `data.frame`:

```
preds = as.data.frame(preds)
```

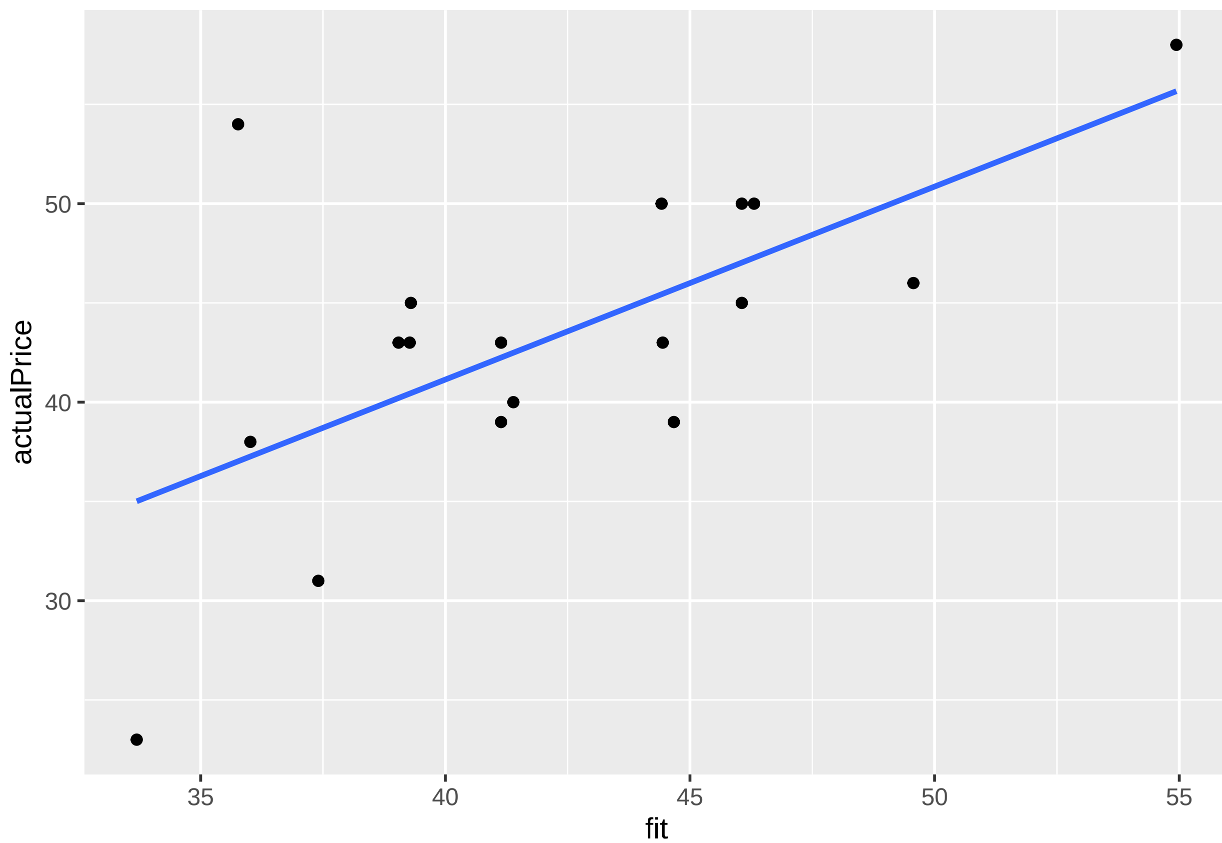
Now, let's compare the actual prices to our predictions. First, we create append the actual prices to `preds`:

```r
preds$actualPrice = nyctest$Price
preds
```

```
##         fit      lwr      upr actualPrice
## 1  44.44261 29.45164 59.43358          43
## 2  46.05904 31.15867 60.95941          45
## 3  39.04475 24.14123 53.94827          43
## 4  39.27259 24.34107 54.20410          43
## 5  54.94082 39.92882 69.95282          58
## 6  35.76544 20.77548 50.75540          54
## 7  37.40510 22.47519 52.33500          31
## 8  44.41939 29.53641 59.30237          50
## 9  49.56619 34.62748 64.50489          46
## 10 46.05904 31.15867 60.95941          50
## 11 41.14008 26.19025 56.08991          39
## 12 41.14008 26.19025 56.08991          43
## 13 39.29582 24.35479 54.23684          45
## 14 41.39114 26.39525 56.38704          40
## 15 33.69334 18.65313 48.73356          23
## 16 36.01651 21.01412 51.01890          38
## 17 44.67045 29.71621 59.62469          39
## 18 46.31011 31.32610 61.29411          50
```

Now, we can plot our predictions against the actual prices:

```r
ggplot(preds) +
    aes(x = fit, y = actualPrice) +
        geom_point() +
            geom_smooth(method = 'lm', se = FALSE)
```




We see a positive association, but the model is far from perfect.

### Interactions

Fitting interactions with `R` amounts to writing a product term in the `lm` statement.

For example, if we're working with the `hsb2` dataset in `library(openintro)` and we want to fit a model to predict math scores as a function of the score in writing, socioeconomic status, and an interaction between the two, you can use the code below:

```
library(openintro)
data(hsb2)
mod = lm(math ~ write + ses + ses*write , data = hsb2)
summary(mod)
```

```
##
## Call:
## lm(formula = math ~ write + ses + ses * write, data = hsb2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.0179  -4.5783  -0.2104   4.4228  21.5940
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)     20.336920   5.861562   3.470 0.000642 ***
## write            0.569636   0.113860   5.003 1.26e-06 ***
## sesmiddle        1.938395   7.314626   0.265 0.791289
## seshigh          2.525714   8.265754   0.306 0.760264
## write:sesmiddle  0.006858   0.140908   0.049 0.961234
## write:seshigh    0.026098   0.153401   0.170 0.865085
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.329 on 194 degrees of freedom
## Multiple R-squared:  0.4034, Adjusted R-squared:  0.388
## F-statistic: 26.24 on 5 and 194 DF,  p-value: < 2.2e-16
```

## References


- Sheather, Simon. *A modern approach to regression with R.* Springer Science & Business Media, 2009.
- Multiple and logistic regression, Datacamp course by Ben Baumer.