Multiple and Logistic Regression

Seth Harrison

Last compiled: Mar 08, 2021

Contents

Chapter 1

Prerequisites

This material is from the DataCamp course Multiple and Logistic Regression by Ben Baumer. Before using this material, the reader should have completed and be comfortable with the material in the DataCamp module Correlation and Regression.

Reminder to self: each `*.Rmd` file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

Chapter 2

Parallel Slopes

In this chapter you'll learn about the class of linear models called "parallel slopes models." These include one numeric and one categorical explanatory variable.

What if you have two groups?

2.1 Fitting a parallel slopes model

We use the `lm()` function to fit linear models to data. In this case, we want to understand how the price of mariokart games sold at auction varies as a function of not only the number of wheels included in the package, but also whether the item is new or used. Obviously, it is expected that you might have to pay a premium to buy these new. But how much is that premium? Can we estimate its value after *controlling for the number of wheels*?

We will fit a parallel slopes model using `lm()`. In addition to the `data` argument, `lm()` needs to know which variables you want to include in your regression model, and how you want to include them. It accomplishes this using a `formula` argument. A simple linear regression formula looks like `y ~ x`, where `y` is the name of the response variable, and `x` is the name of the explanatory variable. Here, we will simply extend this formula to include multiple explanatory variables. A parallel slopes model has the form `y ~ x + z`, where `z` is a categorical explanatory variable, and `x` is a numerical explanatory variable.

The output from `lm()` is a model object, which when printed, will show the fitted coefficients.

Exercise

- The dataset `mariokart` is already loaded for you. Explore the data using `glimpse()` or `str()`.

```r
library(openintro)
data(mariokart)
glimpse(mariokart)
```

```
Rows: 143
Columns: 12
$ id          <dbl> 150377422259, 260483376854, 320432342985, 280405224677, 17~
$ duration    <int> 3, 7, 3, 3, 1, 3, 1, 1, 3, 7, 1, 1, 1, 1, 7, 7, 3, 3, 1, 7~
$ n_bids      <int> 20, 13, 16, 18, 20, 19, 13, 15, 29, 8, 15, 15, 13, 16, 6, ~
$ cond        <fct> new, used, new, new, new, new, used, new, used, used, new,~
$ start_pr    <dbl> 0.99, 0.99, 0.99, 0.99, 0.01, 0.99, 0.01, 1.00, 0.99, 19.9~
$ ship_pr     <dbl> 4.00, 3.99, 3.50, 0.00, 0.00, 4.00, 0.00, 2.99, 4.00, 4.00~
$ total_pr    <dbl> 51.55, 37.04, 45.50, 44.00, 71.00, 45.00, 37.02, 53.99, 47~
$ ship_sp     <fct> standard, firstClass, firstClass, standard, media, standar~
$ seller_rate <int> 1580, 365, 998, 7, 820, 270144, 7284, 4858, 27, 201, 4858,~
$ stock_photo <fct> yes, yes, no, yes, yes, yes, yes, yes, yes, no, yes, yes, ~
$ wheels      <int> 1, 1, 1, 1, 2, 0, 0, 2, 1, 1, 2, 2, 2, 2, 1, 0, 1, 1, 2, 2~
$ title       <fct> "~~ Wii MARIO KART &amp; WHEEL ~ NINTENDO Wii ~ BRAND NEW ~
```

```r
# Or
# str(mariokart)
# Data munging to agree with DataCamp mario_kart
mario_kart <- mariokart %>%
  filter(total_pr < 100)
str(mario_kart)
```

```
tibble [141 x 12] (S3: tbl_df/tbl/data.frame)
 $ id         : num [1:141] 1.5e+11 2.6e+11 3.2e+11 2.8e+11 1.7e+11 ...
 $ duration   : int [1:141] 3 7 3 3 1 3 1 1 3 7 ...
 $ n_bids     : int [1:141] 20 13 16 18 20 19 13 15 29 8 ...
 $ cond       : Factor w/ 2 levels "new","used": 1 2 1 1 1 1 2 1 2 2 ...
 $ start_pr   : num [1:141] 0.99 0.99 0.99 0.99 0.01 ...
 $ ship_pr    : num [1:141] 4 3.99 3.5 0 0 4 0 2.99 4 4 ...
 $ total_pr   : num [1:141] 51.5 37 45.5 44 71 ...
 $ ship_sp    : Factor w/ 8 levels "firstClass","media",..: 6 1 1 6 2 6 6 8 5 1 ...
 $ seller_rate: int [1:141] 1580 365 998 7 820 270144 7284 4858 27 201 ...
 $ stock_photo: Factor w/ 2 levels "no","yes": 2 2 1 2 2 2 2 2 2 1 ...
 $ wheels     : int [1:141] 1 1 1 1 2 0 0 2 1 1 ...
 $ title      : Factor w/ 80 levels " Mario Kart Wii with Wii Wheel for Wii (New)",..: 80 60 22 7
```

```r
save(mario_kart,file = "./Data/mario_kart.RData")
```

- Use `lm()` to fit a parallel slopes model for total price as a function of the number of wheels and the condition of the item. Use the argument `data` to specify the dataset you're using.

```r
# fit parallel slopes
lm(total_pr ~ wheels + cond, data = mario_kart)
```

```

Call:
lm(formula = total_pr ~ wheels + cond, data = mario_kart)

Coefficients:
(Intercept)       wheels     condused  
     42.370        7.233       -5.585  
```

Reasoning about two intercepts

The `mariokart` data contains several other variables. The `total_pr`, `start_pr`, and `ship_pr` variables are numeric, while the `cond` and `stockPhoto` variables are categorical.

Which formula will result in a parallel slopes model?

- `total_pr ~ start_pr + ship_pr`
- `cond ~ start_pr + stock_photo`
- **`total_pr ~ ship_pr + stock_photo`**
- `total_pr ~ cond`

Visualizing parallel slopes models

2.2 Using geom_line() and augment()

Parallel slopes models are so-named because we can visualize these models in the data space as not one line, but two parallel lines. To do this, we'll draw two things:

- a scatterplot showing the data, with color separating the points into groups
- a line for each value of the categorical variable

Our plotting strategy is to compute the fitted values, plot these, and connect the points to form a line. The `augment()` function from the `broom` package provides an easy way to add the fitted values to our data frame, and the `geom_line()` function can then use that data frame to plot the points and connect them.

Note that this approach has the added benefit of automatically coloring the lines appropriately to match the data.

You already know how to use `ggplot()` and `geom_point()` to make the scatterplot. The only twist is that now you'll pass your `augment()`-ed model as the data argument in your `ggplot()` call. When you add your `geom_line()`, instead of letting the `y` aesthetic inherit its values from the `ggplot()` call, you can set it to the `.fitted` column of the `augment()`-ed model. This has the advantage of automatically coloring the lines for you.

Exercise

The parallel slopes model `mod` relating total price to the number of wheels and condition is already in your workspace.

```
mod <- lm(formula = total_pr ~ wheels + cond, data = mario_kart)
```

- `augment()` the model `mod` and explore the returned data frame using `glimpse()`. Notice the new variables that have been created.

```
library(broom)
augmented_mod <- augment(mod)
glimpse(augmented_mod)
```

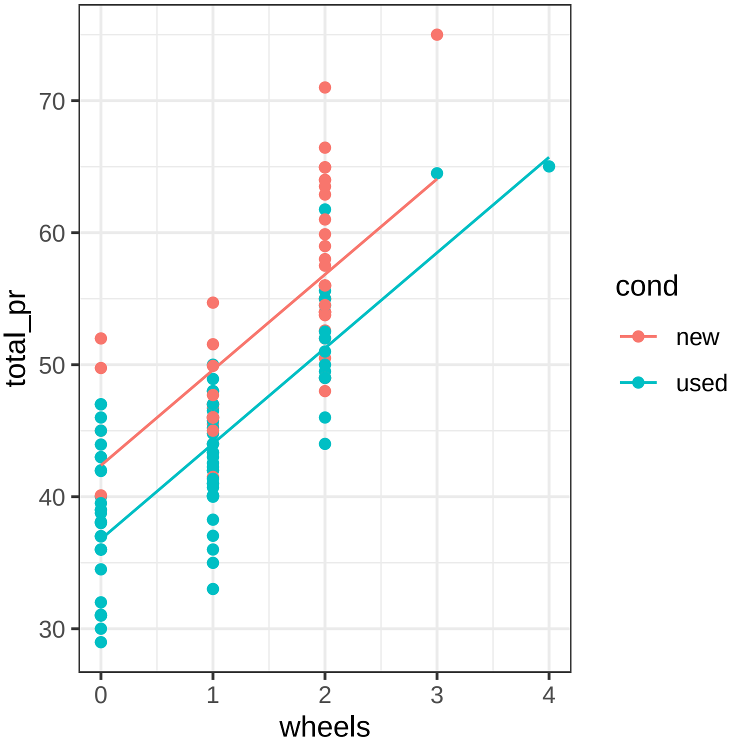
```
Rows: 141
Columns: 9
$ total_pr   <dbl> 51.55, 37.04, 45.50, 44.00, 71.00, 45.00, 37.02, 53.99, 47.~
$ wheels     <int> 1, 1, 1, 1, 2, 0, 0, 2, 1, 1, 2, 2, 2, 2, 1, 0, 1, 1, 2, 0,~
$ cond       <fct> new, used, new, new, new, new, used, new, used, used, new, ~
$ .fitted    <dbl> 49.60260, 44.01777, 49.60260, 49.60260, 56.83544, 42.36976,~
$ .resid     <dbl> 1.9473995, -6.9777674, -4.1026005, -5.6026005, 14.1645592, ~
$ .hat       <dbl> 0.02103158, 0.01250410, 0.02103158, 0.02103158, 0.01915635,~
$ .sigma      <dbl> 4.902339, 4.868399, 4.892414, 4.881308, 4.750591, 4.899816,~
$ .cooksd     <dbl> 1.161354e-03, 8.712334e-03, 5.154337e-03, 9.612441e-03, 5.5~
$ .std.resid <dbl> 0.40270893, -1.43671086, -0.84838977, -1.15857953, 2.926332~
```

- Draw the scatterplot and save it as `data_space` by passing the `augment()`-ed model to `ggplot()` and using `geom_point()`.

```r
# scatterplot, with color
data_space <- ggplot(data = augmented_mod,
                     aes(x = wheels, y = total_pr,
                         color = cond)) +
  geom_point()
```

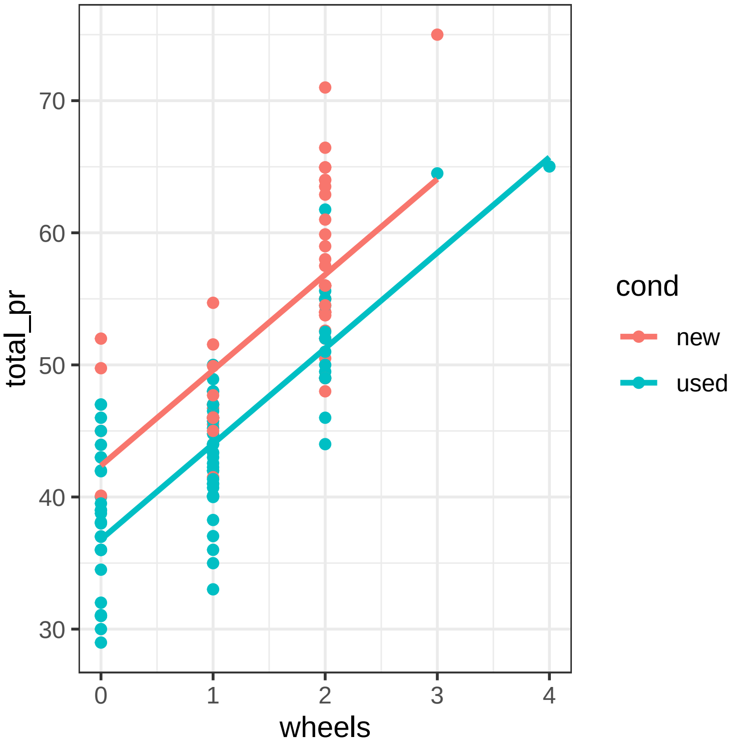
- Use `geom_line()` once to add two parallel lines corresponding to our model.

```r
# single call to geom_line()
data_space +
  geom_line(aes(x = wheels, y = .fitted)) +
  theme_bw()
```



2.2.1 An alternative approach using `geom_parallel_slopes()` from `moderndive`

```
library(moderndive)
ggplot(data = mario_kart, aes(x = wheels, y = total_pr, color = cond)) +
  geom_point() +
  geom_parallel_slopes(se = FALSE) +
  theme_bw()
```



Interpreting parallel slopes coefficients

Intercept interpretation

Recall that the `cond` variable is either `new` or `used`. Here are the fitted coefficients from your model:

```
lm(total_pr ~ wheels + cond, data = mario_kart)
```

```

Call:
lm(formula = total_pr ~ wheels + cond, data = mario_kart)

Coefficients:
(Intercept)       wheels     condused
     42.370        7.233       -5.585
```

Choose the correct interpretation of the coefficient on `condused`:

- For each additional wheel, the expected price of a used mariokart is $5.58 lower.
- **The expected price of a used mariokart is $5.58 less than that of a new one with the same number of wheels.**
- The expected price of a new mariokart is $5.58 less than that of a used one with the same number of wheels.
- The used mariokarts are always $5.58 cheaper.

Common slope interpretation

Recall the fitted coefficients from our model:

```
lm(total_pr ~ wheels + cond, data = mario_kart)
```

```

Call:
lm(formula = total_pr ~ wheels + cond, data = mario_kart)

Coefficients:
(Intercept)       wheels     condused
     42.370        7.233       -5.585
```

Choose the correct interpretation of the slope coefficient:

- **For each additional wheel, the expected price of a mariokart increases by $7.23 regardless of whether it is new or used.**
- For each additional wheel, the expected price of a new mariokart increases by $7.23.
- The expected price of a used mariokart is $5.59 less than that of a new one with the same number of wheels.
- You should always expect to pay $42.37 for a mariokart.

Three ways to describe a model

2.3 Syntax from math

The `babies` data set contains observations about the birthweight and other characteristics of children born in the San Francisco Bay area from 1960–1967.

We would like to build a model for birthweight as a function of the mother's age and whether this child was her first (`parity == 0`). Use the mathematical specification below to code the model in R.

$$birthweight = \beta_0 + \beta_1 \cdot age + \beta_2 \cdot parity + \varepsilon$$

Exercise

The birthweight variable is recorded in the column `bwt`.

- Use `lm()` to build the parallel slopes model specified above. It's not necessary to use `factor()` in this case as the variable `parity` is coded using binary numeric values.

```r
# build model
lm(bwt ~ age + parity, data = babies)
```

```
Call:
lm(formula = bwt ~ age + parity, data = babies)

Coefficients:
(Intercept)          age       parity  
  118.27782      0.06315     -1.65248  
```

2.4 Syntax from plot

This time, we'd like to build a model for birthweight as a function of the length of gestation and the mother's smoking status. Use Figure 2.1 to inform your model specification.

```
ggplot(data = babies, aes(x = gestation, y = bwt, color = factor(smoke))) +
  geom_point(alpha = 0.5) +
  theme_bw()
```

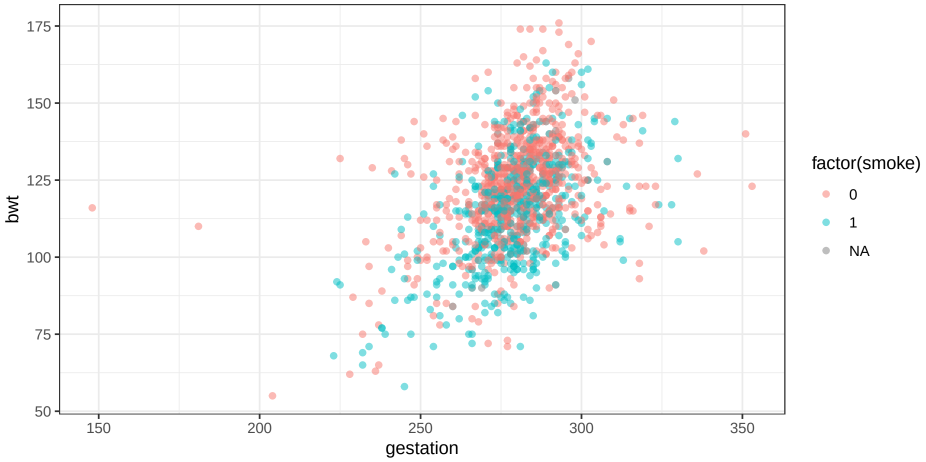


Figure 2.1: 'bwt' versus 'gestation'

Exercise

- Use `lm()` to build a parallel slopes model implied by the plot. It's not necessary to use `factor()` in this case either.

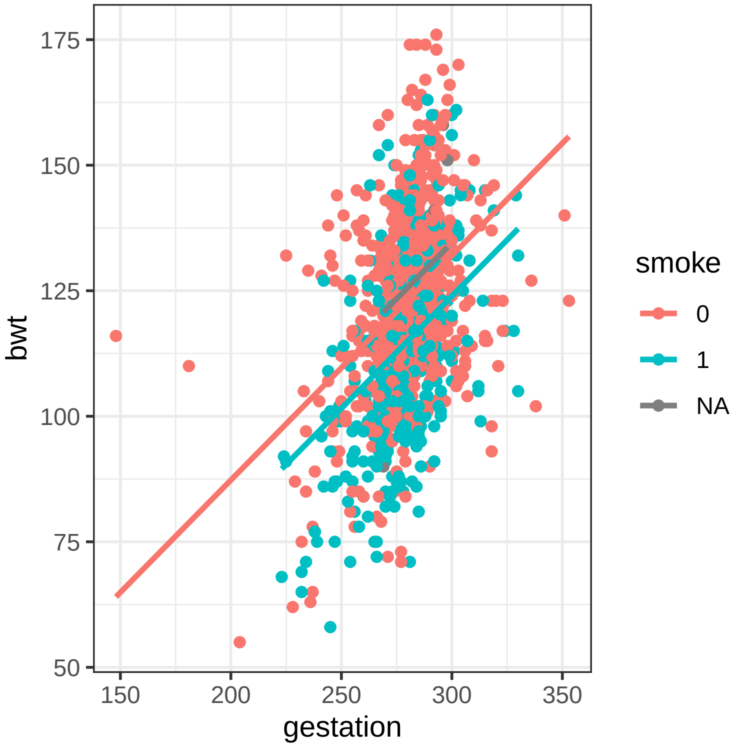
```
# build model
lm(bwt ~ gestation + smoke, data = babies)
```

```

Call:
lm(formula = bwt ~ gestation + smoke, data = babies)

Coefficients:
(Intercept)    gestation        smoke
    -0.9317       0.4429      -8.0883
```

```
babies <- babies %>%
  mutate(smoke = factor(smoke))
ggplot(data = babies, aes(x = gestation, y = bwt, color = smoke)) +
         geom_point() +
         geom_parallel_slopes(se = FALSE) +
         theme_bw()
```



Chapter 3

Evaluating and extending parallel slopes model

This chapter covers model evaluation. By looking at different properties of the model, including the adjusted R-squared, you'll learn to compare models so that you can select the best one. You'll also learn about interaction terms in linear models.

Model fit, residuals, and prediction

3.1 R-squared vs. adjusted R-squared

Two common measures of how well a model fits to data are R^2 (the coefficient of determination) and the adjusted R^2. The former measures the percentage of the variability in the response variable that is explained by the model. To compute this, we define

$$R^2 = 1 - \frac{SSE}{SST},$$

where SSE and SST are the sum of the squared residuals, and the total sum of the squares, respectively. One issue with this measure is that the SSE can only decrease as new variable are added to the model, while the SST depends only on the response variable and therefore is not affected by changes to the model.

This means that you can increase R^2 by adding any additional variable to your model—even random noise.

The adjusted R^2 includes a term that penalizes a model for each additional explanatory variable (where p is the number of explanatory variables).

$$R^2_{\text{adj}} = 1 - \frac{SSE}{SST} \cdot \frac{n-1}{n-p-1},$$

We can see both measures in the output of the `summary()` function on our model object.

Exercise

```
load("./Data/mario_kart.RData")
mod <- lm(total_pr ~ wheels + cond, data = mario_kart)
```

- Use `summary()` to compute R^2 and adjusted R^2 on the model object called `mod`.

```
# R^2 and adjusted R^2
summary(mod)
```

```

Call:
lm(formula = total_pr ~ wheels + cond, data = mario_kart)

Residuals:
     Min       1Q   Median       3Q      Max 
-11.0078  -3.0754  -0.8254   2.9822  14.1646 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  42.3698     1.0651  39.780  < 2e-16 ***
wheels        7.2328     0.5419  13.347  < 2e-16 ***
condused     -5.5848     0.9245  -6.041 1.35e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.887 on 138 degrees of freedom
Multiple R-squared:  0.7165,	Adjusted R-squared:  0.7124 
F-statistic: 174.4 on 2 and 138 DF,  p-value: < 2.2e-16
```

The R^2 value for `mod` is 0.7165182, and the R^2_{adj} value is 0.7124098.

- Use `mutate()` and `rnorm()` to add a new variable called `noise` to the `mario_kart` data set that consists of random noise. Save the new dataframe as `mario_kart_noisy`.

```
# add random noise
set.seed(34)
# add random noise
mario_kart_noisy <- mario_kart %>%
  mutate(noise = rnorm(nrow(mario_kart)))
```

- Use `lm()` to fit a model that includes `wheels`, `cond`, and the random noise term.

```
# compute new model
mod2 <- lm(total_pr ~ wheels + cond + noise, data = mario_kart_noisy)
```

- Use `summary()` to compute R^2 and adjusted R^2 on the new model object. Did the value of R^2 increase? **Yes** What about adjusted R^2? **It also increased. Adding random noise increase both R^2 and $R^2_{\textbf{adj}}$.**

```
# new R^2 and adjusted R^2
summary(mod2)
```

```

Call:
lm(formula = total_pr ~ wheels + cond + noise, data = mario_kart_noisy)

Residuals:
     Min       1Q   Median       3Q      Max 
-10.3256  -3.1692  -0.7492   2.8731  14.1293 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  42.2788     1.0659  39.664  < 2e-16 ***
wheels        7.2310     0.5410  13.367  < 2e-16 ***
condused     -5.4003     0.9354  -5.774 4.97e-08 ***
noise        -0.4930     0.4059  -1.215    0.227    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.879 on 137 degrees of freedom
Multiple R-squared:  0.7195,	Adjusted R-squared:  0.7134 
F-statistic: 117.2 on 3 and 137 DF,  p-value: < 2.2e-16
```

3.2 Prediction

Once we have fit a regression model, we can use it to make predictions for unseen observations or retrieve the fitted values. Here, we explore two methods for doing the latter.

A traditional way to return the fitted values (i.e. the $\hat{y}$'s) is to run the `predict()` function on the model object. This will return a vector of the fitted values. Note that `predict()` will take an optional `newdata` argument that will allow you to make predictions for observations that are not in the original data.

A newer alternative is the `augment()` function from the `broom` package, which returns a `data.frame` with the response variable (y), the relevant explanatory variables (the x's), the fitted value ($\hat{y}$) and some information about the residuals ($\hat{\varepsilon}$). `augment()` will also take a `newdata` argument that allows you to make predictions.

Exercise

The fitted model `mod` is already in your environment.

- Compute the fitted values of the model as a vector using `predict()`.

```r
# return a vector
VEC <- predict(mod)
head(VEC)
```

```
       1        2        3        4        5        6 
49.60260 44.01777 49.60260 49.60260 56.83544 42.36976 
```

- Compute the fitted values of the model as one column in a `data.frame` using `augment()`.

```r
# return a data frame
DF <- broom::augment(mod)
head(DF)
```

```
# A tibble: 6 x 9
  total_pr wheels cond  .fitted .resid   .hat .sigma .cooksd .std.resid
     <dbl>  <int> <fct>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>      <dbl>
1     51.6      1 new      49.6   1.95 0.0210   4.90 0.00116      0.403
2     37.0      1 used     44.0  -6.98 0.0125   4.87 0.00871     -1.44 
3     45.5      1 new      49.6  -4.10 0.0210   4.89 0.00515     -0.848
4     44        1 new      49.6  -5.60 0.0210   4.88 0.00961     -1.16 
5     71        2 new      56.8  14.2  0.0192   4.75 0.0557       2.93 
6     45        0 new      42.4   2.63 0.0475   4.90 0.00505      0.551
```

Understanding interaction

Thought experiments

Suppose that after going apple picking you have 12 apples left over. You decide to conduct an experiment to investigate how quickly they will rot under certain conditions. You place six apples in a cool spot in your basement, and leave the other six on the window sill in the kitchen. Every week, you estimate the percentage of the surface area of the apple that is rotten or moldy.

Consider the following models:

$$rot = \beta_0 + \beta_1 \cdot t + \beta_2 \cdot temp,$$

and

$$rot = \beta_0 + \beta_1 \cdot t + \beta_2 \cdot temp + \beta_3 \cdot temp \cdot t,$$

where t is time, measured in weeks, and *temp* is a binary variable indicating either cool or warm.

If you decide to keep the interaction term present in the second model, you are implicitly assuming that:

- The amount of rot will vary based on the temperature.
- The amount of rot will vary based on the temperature, after controlling for the length of time they have been left out.
- **The rate at which apples rot will vary based on the temperature.**
- Time and temperature are independent.

3.3 Fitting a model with interaction

Including an interaction term in a model is easy—we just have to tell `lm()` that we want to include that new variable. An expression of the form

```r
lm(y ~ x + z + x:z, data = mydata)
```

will do the trick. The use of the colon (`:`) here means that the interaction between `x` and `z` will be a third term in the model.

Exercise

The data frame `mario_kart` is already loaded in your workspace.

- Use `lm()` to fit a model for the price of a MarioKart as a function of its condition and the duration of the auction, with interaction.

```r
# include interaction
lm(total_pr ~ cond + duration + cond:duration, data = mario_kart)
```

```

Call:
lm(formula = total_pr ~ cond + duration + cond:duration, data = mario_kart)

Coefficients:
      (Intercept)           condused           duration  condused:duration  
           58.268            -17.122             -1.966              2.325  
```

3.4 Visualizing interaction models

Interaction allows the slope of the regression line in each group to vary. In this case, this means that the relationship between the final price and the length of the auction is moderated by the condition of each item.

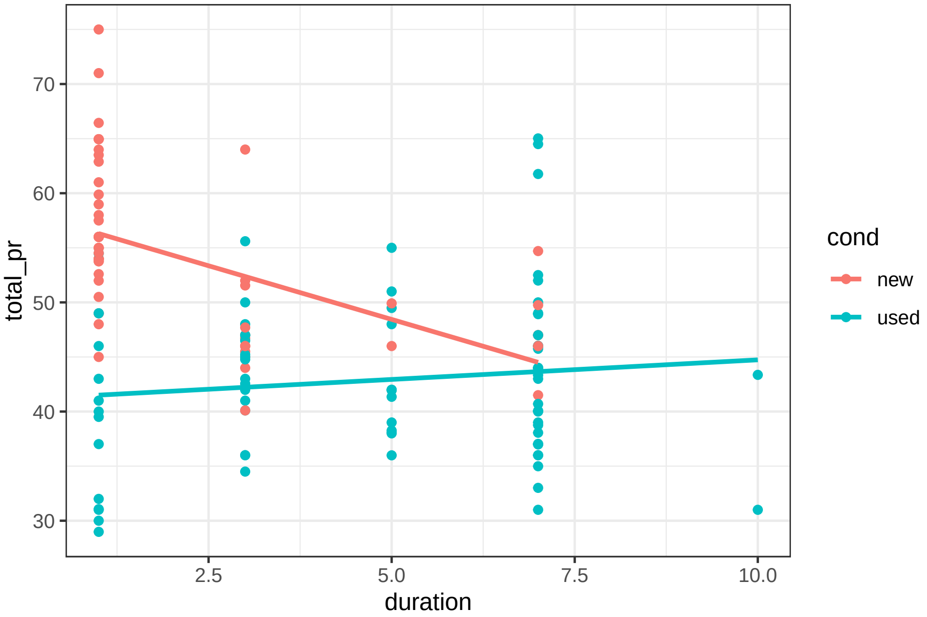
Interaction models are easy to visualize in the data space with `ggplot2` because they have the same coefficients as if the models were fit independently to each group defined by the level of the categorical variable. In this case, new and used MarioKarts each get their own regression line. To see this, we can set an aesthetic (e.g. `color`) to the categorical variable, and then add a `geom_smooth()` layer to overlay the regression line for each color.

Exercise

The dataset `mario_kart` is already loaded in your workspace.

- Use the `color` aesthetic and the `geom_smooth()` function to plot the interaction model between duration and condition in the data space. Make sure you set the `method` and `se` arguments of `geom_smooth()`.

```
# interaction plot
ggplot(data = mario_kart, aes(y = total_pr, x = duration, color = cond)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  theme_bw()
```



- How does the interaction model differ from the parallel slopes model? **Class discussion**

Simpson's paradox

3.5 Consequences of Simpson's paradox

In the simple linear regression model for average SAT score, (`total`) as a function of average teacher salary (`salary`), the fitted coefficient was -5.02 points per thousand dollars. This suggests that for every additional thousand dollars of salary for teachers in a particular state, the expected SAT score for a student from that state is about 5 points lower.

In the model that includes the percentage of students taking the SAT, the coefficient on `salary` becomes 1.84 points per thousand dollars. Choose the correct interpretation of this slope coefficient.

```
### Get Data
url <- "https://assets.datacamp.com/production/repositories/845/datasets/1a12a19d2cec83
if(!file.exists("./Data/SAT.csv")){
    download.file(url, destfile = "./Data/SAT.csv")}
SAT <- read.csv("./Data/SAT.csv")
###
lm(total ~ salary, data = SAT)
```

```

Call:
lm(formula = total ~ salary, data = SAT)

Coefficients:
(Intercept)       salary
  1.871e+03   -5.019e-03
```

```
SAT_wbin <- SAT %>%
    mutate(sat_bin = cut(sat_pct, 3))
mod <- lm(formula = total ~ salary + sat_bin, data = SAT_wbin)
mod
```

```

Call:
lm(formula = total ~ salary + sat_bin, data = SAT_wbin)

Coefficients:
     (Intercept)            salary    sat_bin(33,63]  sat_bin(63,93.1]
      1597.10773           0.00184        -191.45221        -217.73480
```

- For every additional thousand dollars of salary for teachers in a particular state, the expected SAT score for a student from that state is about 2 points lower.
- **For every additional thousand dollars of salary for teachers in a particular state, the expected SAT score for a student from that state is about 2 points higher, after controlling for the percentage of students taking the SAT.**
- The average SAT score in richer states is about 2 points higher.

3.6 Simpson's paradox in action

A mild version of Simpson's paradox can be observed in the MarioKart auction data. Consider the relationship between the final auction price and the length of the auction. It seems reasonable to assume that longer auctions would result in higher prices, since—other things being equal—a longer auction gives more bidders more time to see the auction and bid on the item.

However, a simple linear regression model reveals the opposite: longer auctions are associated with lower final prices. The problem is that all other things are not equal. In this case, the new MarioKarts—which people pay a premium for—were mostly sold in one-day auctions, while a plurality of the used MarioKarts were sold in the standard seven-day auctions.

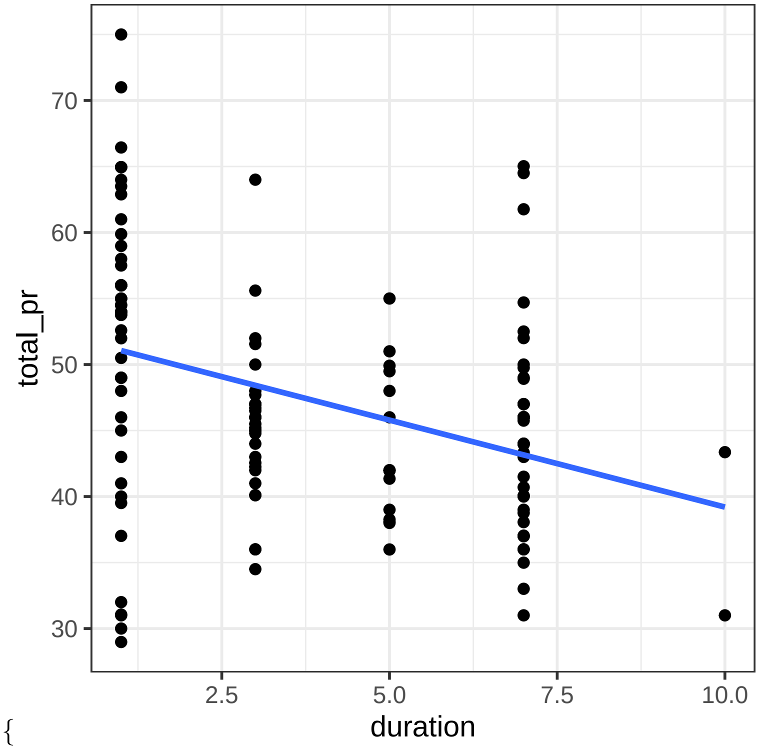
Our simple linear regression model is misleading, in that it suggests a negative relationship between final auction price and duration. However, for the used MarioKarts, the relationship is positive.

Exercise

The object `slr` is already defined for you.

```r
slr <- ggplot(mario_kart, aes(y = total_pr, x = duration)) +
  geom_point() +
  geom_smooth(method = "lm", se = 0) +
  theme_bw()
slr
```

\begin{figure}



{

}

\caption{`total_pr` versus `duration`} \end{figure}

- Fit a simple linear regression model for final auction price (`total_pr`) as a function of duration (`duration`).

```
# model with one slope
lm(total_pr ~ duration, data = mario_kart)
```

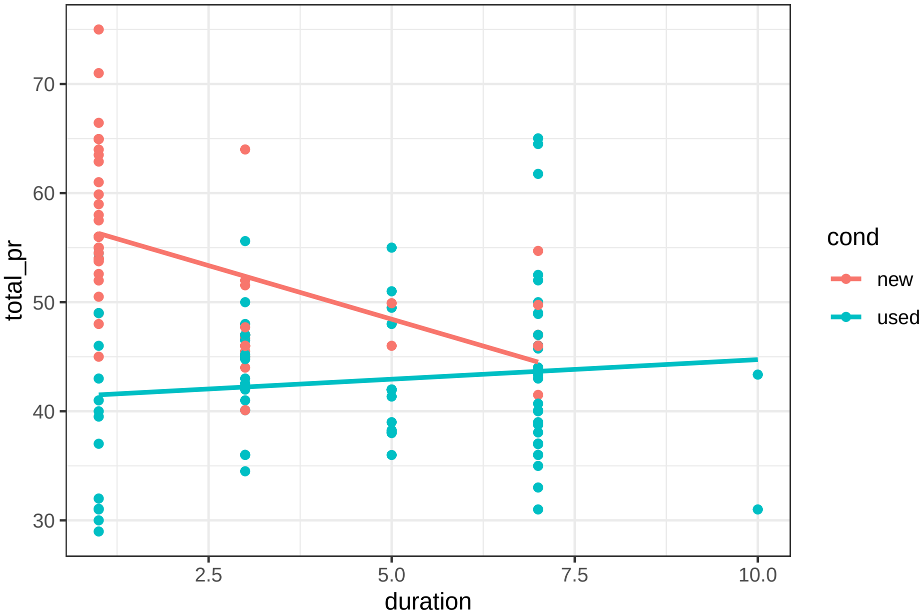
```

Call:
lm(formula = total_pr ~ duration, data = mario_kart)

Coefficients:
(Intercept)     duration
     52.374       -1.317
```

- Use `aes()` to add a color aesthetic that's mapped to the condition variable to the `slr` object, shown in Figure 3.6.

```
# plot with two slopes
slr + aes(color = cond)
```



- Which of the two groups is showing signs of Simpson's paradox? **Class discussion**

Chapter 4

Multiple Regression

This chapter will show you how to add two, three, and even more numeric explanatory variables to a linear model.

Adding a numerical explanatory variable

4.1 Fitting a MLR model

In terms of the R code, fitting a multiple linear regression model is easy: simply add variables to the model formula you specify in the `lm()` command.

In a parallel slopes model, we had two explanatory variables: one was numeric and one was categorical. Here, we will allow both explanatory variables to be numeric.

```r
load("./Data/mario_kart.RData")
str(mario_kart)
```

```
tibble [141 x 12] (S3: tbl_df/tbl/data.frame)
 $ id         : num [1:141] 1.5e+11 2.6e+11 3.2e+11 2.8e+11 1.7e+11 ...
 $ duration   : int [1:141] 3 7 3 3 1 3 1 1 3 7 ...
 $ n_bids     : int [1:141] 20 13 16 18 20 19 13 15 29 8 ...
 $ cond       : Factor w/ 2 levels "new","used": 1 2 1 1 1 1 2 1 2 2 ...
 $ start_pr   : num [1:141] 0.99 0.99 0.99 0.99 0.01 ...
 $ ship_pr    : num [1:141] 4 3.99 3.5 0 0 4 0 2.99 4 4 ...
 $ total_pr   : num [1:141] 51.5 37 45.5 44 71 ...
 $ ship_sp    : Factor w/ 8 levels "firstClass","media",..: 6 1 1 6 2 6 6 8 5 1 ...
 $ seller_rate: int [1:141] 1580 365 998 7 820 270144 7284 4858 27 201 ...
 $ stock_photo: Factor w/ 2 levels "no","yes": 2 2 1 2 2 2 2 2 2 1 ...
 $ wheels     : int [1:141] 1 1 1 1 2 0 0 2 1 1 ...
 $ title      : Factor w/ 80 levels " Mario Kart Wii with Wii Wheel for Wii (New)",..:
```

The dataset mario_kart is already loaded in your workspace.

- Fit a multiple linear regression model for total price as a function of the duration of the auction and the starting price.

```
# Fit the model using duration and start_pr
mod <- lm(total_pr ~ duration + start_pr, data = mario_kart)
mod
```

```

Call:
lm(formula = total_pr ~ duration + start_pr, data = mario_kart)

Coefficients:
(Intercept)     duration     start_pr  
     51.030       -1.508        0.233  
```

4.2 Tiling the plane

One method for visualizing a multiple linear regression model is to create a heatmap of the fitted values in the plane defined by the two explanatory variables. This heatmap will illustrate how the model output changes over different combinations of the explanatory variables.

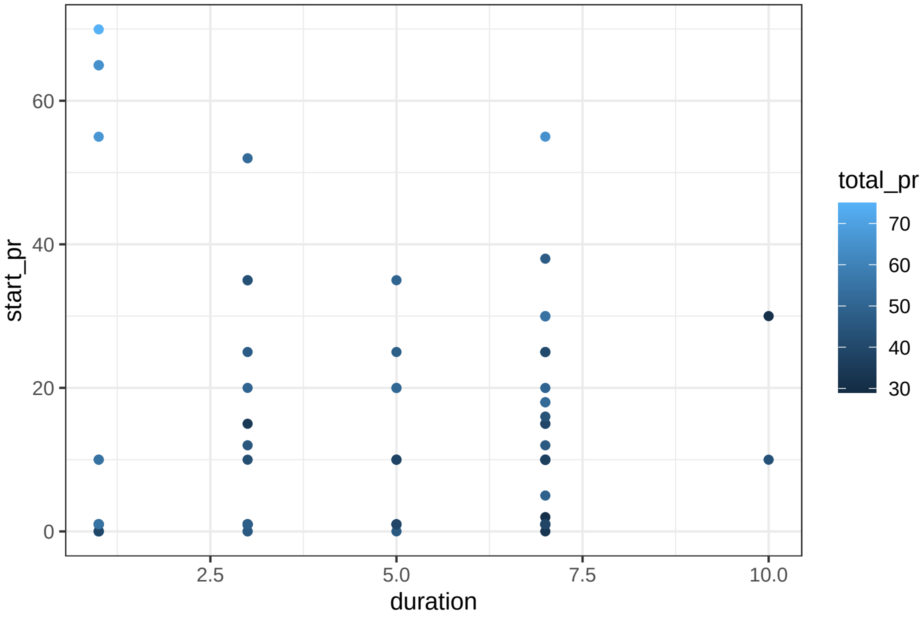
This is a multistep process:

- First, create a grid of the possible pairs of values of the explanatory variables. The grid should be over the actual range of the data present in each variable. We've done this for you and stored the result as a data frame called grid.

```
grid <- expand.grid(duration = seq(1, 10, by = 1), start_pr = seq(0.01, 69.95, by = 0.0
```

- Use augment() with the newdata argument to find the $\hat{y}$'s corresponding to the values in grid.
- Add these to the data_space plot by using the fill aesthetic and geom_tile().

```r
data_space <- ggplot(data = mario_kart,
                     aes(x = duration, y = start_pr)) +
  geom_point(aes(color = total_pr)) +
  theme_bw()
data_space
```



Exercise

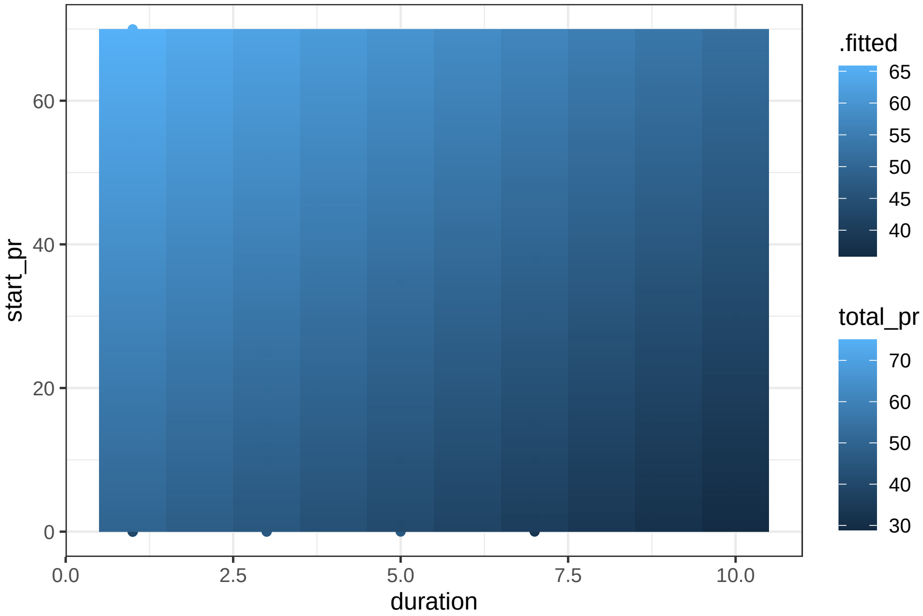
The model object `mod` is already in your workspace.

- Use `augment()` to create a `data.frame` that contains the values the model outputs for each row of `grid`.

```r
# add predictions to grid
price_hats <- broom::augment(mod, newdata = grid)
```

- Use `geom_tile` to illustrate these predicted values over the `data_space` plot. Use the `fill` aesthetic and set `alpha = 0.5`.

```r
# tile the plane
data_space +
   geom_tile(data = price_hats,
             aes(fill = .fitted), alpha = 0.5)
```



4.3 Models in 3D

An alternative way to visualize a multiple regression model with two numeric explanatory variables is as a plane in three dimensions. This is possible in R using the `plotly` package.

We have created three objects that you will need:

- `x`: a vector of unique values of `duration`
- `y`: a vector of unique values of `start_pr`
- `plane`: a matrix of the fitted values across all combinations of `x` and `y`

Much like `ggplot()`, the `plot_ly()` function will allow you to create a plot object with variables mapped to `x`, `y`, and `z` aesthetics. The `add_markers()` function is similar to `geom_point()` in that it allows you to add points to your 3D plot.

Note that `plot_ly` uses the pipe (`%>%`) operator to chain commands together.

Exercise

- Run the `plot_ly` command to draw 3D scatterplot for `total_pr` as a function of duration and `start_pr` by mapping the `z` variable to the response and the `x` and `y` variables to the explanatory variables. Duration should be on the x-axis and starting price should be on the y-axis.

```
library(plotly)
# draw the 3D scatterplot
p <- plot_ly(data = mario_kart, z = ~total_pr, x = ~duration, y = ~start_pr, opacity = 0.6) %>%
  add_markers()
p
```

- Use `add_surface()` to draw a plane through the cloud of points by setting `z = ~plane`. See wikipedia for the definition of an outer product. In what follows, we will use the R function `outer()` to compute the values of `plane`.

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathbf{T}}$$

```
summary(mod)$coef
```

```
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 51.0295070 1.17913685 43.277001 3.665684e-82
duration    -1.5081260 0.25551997 -5.902184 2.644972e-08
start_pr     0.2329542 0.04363644  5.338525 3.755647e-07
```

```
x <- seq(1, 10, length = 70)
y <- seq(0.010, 59.950, length = 70)
plane <- outer(x, y, function(a, b){summary(mod)$coef[1,1] +
    summary(mod)$coef[2,1]*a + summary(mod)$coef[3,1]*b})
# draw the plane
p %>%
  add_surface(x = ~x, y = ~y, z = ~plane, showscale = FALSE)
```

Conditional interpretation of coefficients

Coefficient magnitude

The coefficients from our model for the total auction price of MarioKarts as a function of auction duration and starting price are shown below.

```
mod
```

```

Call:
lm(formula = total_pr ~ duration + start_pr, data = mario_kart)

Coefficients:
(Intercept)     duration     start_pr  
     51.030       -1.508        0.233  
```

A colleague claims that these results imply that the duration of the auction is a more important determinant of final price than starting price, because the coefficient is larger. This interpretation is false because:

- The coefficient on duration is negative.
- Smaller coefficients are more important.
- **The coefficients have different units (dollars per day and dollars per dollar, respectively) and so they are not directly comparable.**
- The intercept coefficient is much bigger, so it is the most important one.

Practicing interpretation

Fit a multiple regression model for the total auction price of an item in the `mario_kart` data set as a function of the starting price and the duration of the auction. Compute the coefficients and choose the correct interpretation of the duration variable.

- **For each additional day the auction lasts, the expected final price declines by $1.51, after controlling for starting price.**
- For each additional dollar of starting price, the expected final price increases by $0.23, after controlling for the duration of the auction.
- The duration of the auction is a more important determinant of final price than starting price, because the coefficient is larger.
- The average auction lasts 51 days.

Adding a third (categorical) variable

4.4 Visualizing parallel planes

By including the duration, starting price, and condition variables in our model, we now have two explanatory variables and one categorical variable. Our model now takes the geometric form of two parallel planes!

The first plane corresponds to the model output when the condition of the item is `new`, while the second plane corresponds to the model output when the condition of the item is `used`. The planes have the same slopes along both the duration and starting price axes—it is the z-intercept that is different.

Once again we have stored the `x` and `y` vectors for you. Since we now have two planes, there are matrix objects `plane0` and `plane1` stored for you as well.

```
modI <- lm(total_pr ~ duration + start_pr + cond, data = mario_kart)
summary(modI)$coef
```

```
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 53.3447530  1.0804915 49.370822 3.781243e-89
duration    -0.6559841  0.2553503 -2.568957 1.127073e-02
start_pr     0.1981653  0.0382717  5.177855 7.835882e-07
condused    -8.9493214  1.3237851 -6.760403 3.635333e-10
```

```
plane0 <- outer(x, y, function(a, b){53.3447530 -0.6559841*a +
                                       0.1981653*b})
plane1 <- outer(x, y, function(a, b){53.3447530 -0.6559841*a +
                                       0.1981653*b - 8.9493214})
```

Exercise

- Use `plot_ly` to draw 3D scatterplot for `total_pr` as a function of `duration`, `start_pr`, and `cond` by mapping the `z` variable to the response and the `x` and `y` variables to the explanatory variables. Duration should be on the x-axis and starting price should be on the y-axis. Use color to represent `cond`.

```
# draw the 3D scatterplot
p <- plot_ly(data = mario_kart, z = ~total_pr, x = ~duration, y = ~start_pr, opacity = 0.6) %>%
  add_markers(color = ~cond)
p
```

- Use `add_surface()` (twice) to draw two planes through the cloud of points, one for new MarioKarts and another for used ones. Use the objects `plane0` and `plane1`.

```
# draw two planes
p %>%
  add_surface(x = ~x, y = ~y, z = ~plane0, showscale = FALSE) %>%
  add_surface(x = ~x, y = ~y, z = ~plane1, showscale = FALSE)
```

Parallel plane interpretation

The coefficients from our parallel planes model is shown below.

```
modI
```

```


Call:
lm(formula = total_pr ~ duration + start_pr + cond, data = mario_kart)

Coefficients:
(Intercept)     duration     start_pr     condused
    53.3448      -0.6560       0.1982      -8.9493
```

Choose the right interpretation of β_3 (the coefficient on `condUsed`):

- **The expected premium for new (relative to used) MarioKarts is $8.95, after controlling for the duration and starting price of the auction.**
- The expected premium for used (relative to new) MarioKarts is $8.95, after controlling for the duration and starting price of the auction.
- For each additional day the auction lasts, the expected final price declines by $8.95, after controlling for starting price and condition.

Higher dimensions

Interpretation of coefficient in a big model

This time we have thrown even more variables into our model, including the number of bids in each auction (`n_bids`) and the number of wheels. Unfortunately this makes a full visualization of our model impossible, but we can still interpret the coefficients.

```
modJ <- lm(total_pr ~ duration + start_pr + cond + wheels + n_bids,
    data = mario_kart)
modJ
```

```

Call:
lm(formula = total_pr ~ duration + start_pr + cond + wheels + 
    n_bids, data = mario_kart)

Coefficients:
(Intercept)     duration     start_pr     condused       wheels       n_bids  
    39.3741      -0.2752       0.1796      -4.7720       6.7216       0.1909  
```

Choose the correct interpretation of the coefficient on the number of wheels:

- The average number of wheels is 6.72.
- Each additional wheel costs exactly $6.72.
- Each additional wheel is associated with an increase in the expected auction price of $6.72.
- **Each additional wheel is associated with an increase in the expected auction price of $6.72, after controlling for auction duration, starting price, number of bids, and the condition of the item.**

Chapter 5

Logistic Regression

In this chapter you'll learn about using logistic regression, a generalized linear model (GLM), to predict a binary outcome and classify observations.

What is logistic regression?

5.1 Fitting a line to a binary response

When our response variable is binary, a regression model has several limitations. Among the more obvious—and logically incongruous—is that the regression line extends infinitely in either direction. This means that even though our response variable y only takes on the values 0 and 1, our fitted values $\hat{y}$ can range anywhere from $-\infty$ to ∞. This doesn't make sense.

To see this in action, we'll fit a linear regression model to data about 55 students who applied to medical school. We want to understand how their undergraduate GPA relates to the probability they will be accepted by a particular school (`Acceptance`).

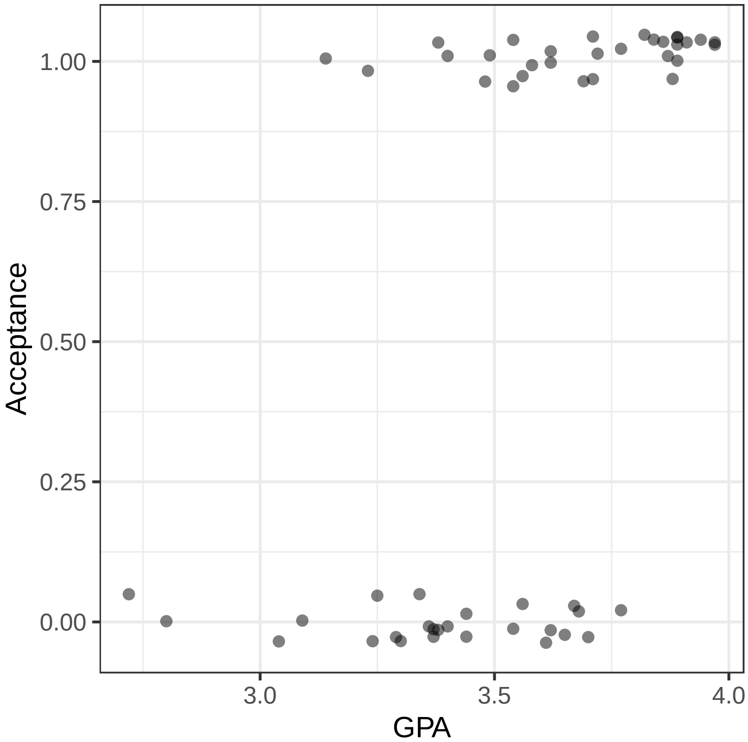
Exercise

```
library(Stat2Data)
data(MedGPA)
```

The medical school acceptance data is loaded in your workspace as `MedGPA`.

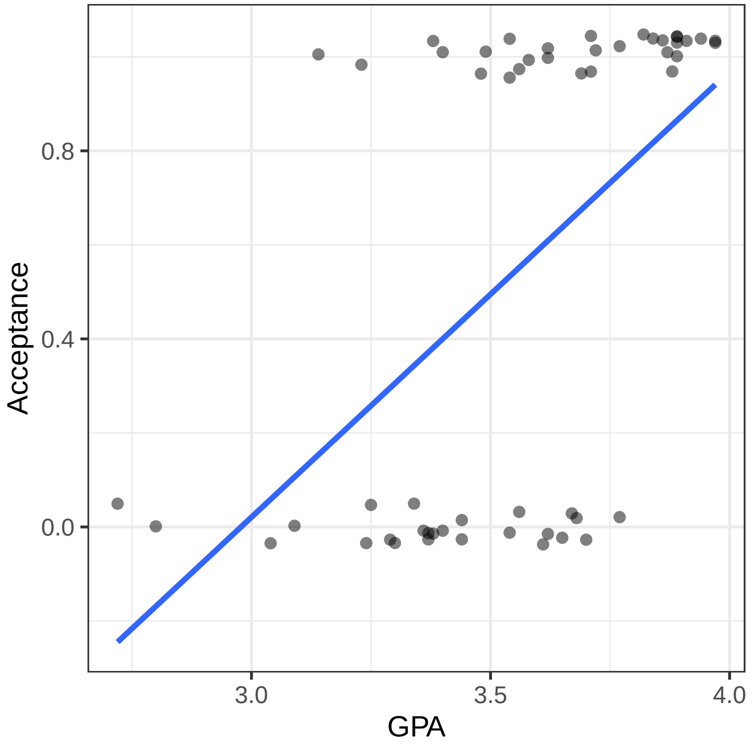
- Create a scatterplot called `data_space` for `Acceptance` as a function of `GPA`. Use `geom_jitter()` to apply a small amount of jitter to the points in the y-direction by setting `width = 0` and `height = 0.05`.

```
# scatterplot with jitter
data_space <- ggplot(data = MedGPA, aes(x = GPA, y = Acceptance)) +
  geom_jitter(width = 0, height = 0.05, alpha = 0.5) +
  theme_bw()
data_space
```



- Use `geom_smooth()` to add the simple linear regression line to `data_space`.

```
# linear regression line
data_space +
  geom_smooth(method = "lm", se = FALSE)
```



5.2 Fitting a line to a binary response (2)

In the previous exercise, we identified a major limitation to fitting a linear regression model when we have a binary response variable. However, it is not always inappropriate to do so. Note that our regression line only makes illogical predictions (i.e. $\hat{y} < 0$ or $\hat{y} > 1$) for students with very high or very low GPAs. For GPAs closer to average, the predictions seem fine.

Moreover, the alternative logistic regression model—which we will fit next—is very similar to the linear regression model for observations near the average of the explanatory variable. It just so happens that the logistic curve is very straight near its middle. Thus, in these cases a linear regression model may still be acceptable, even for a binary response.

Exercise

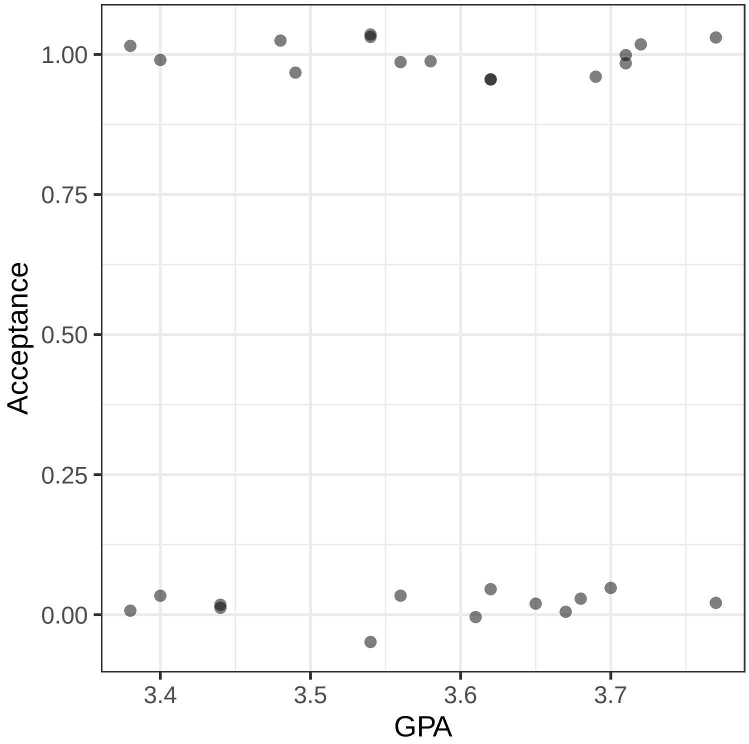
- Use `filter()` to find the subset of the observations whose GPAs are between 3.375 and 3.77, inclusive.

```
# filter
MedGPA_middle <- filter(MedGPA, GPA >= 3.375, GPA <= 3.77)
head(MedGPA_middle)
```

```
  Accept Acceptance Sex BCPM  GPA VR PS WS BS MCAT Apps
1      D          0   F 3.59 3.62 11  9  9  9   38    5
2      A          1   F 3.74 3.69 12 11  7 10   40    5
3      A          1   F 3.53 3.38  9 11  4 11   35   11
4      A          1   M 3.59 3.72 10  9  7 10   36    5
5      A          1   F 3.74 3.71  8 10  6 11   35    5
6      A          1   F 3.35 3.49 11  8  4  8   31    9
```

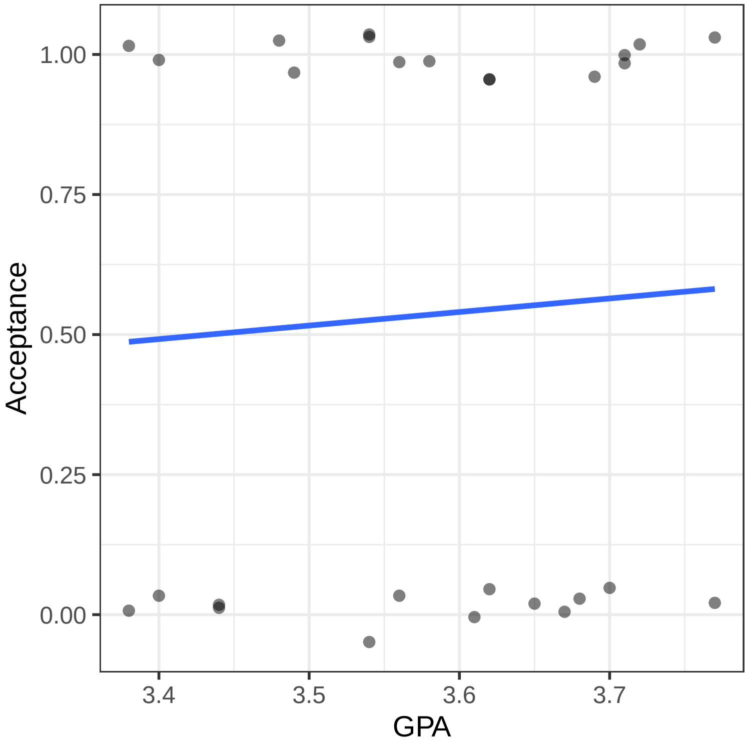
- Create a scatterplot called data_space for Acceptance as a function of GPA for only those observations. Use geom_jitter() to apply 0.05 jitter to the points in the y-direction and no jitter to the x-direction.

```
# scatterplot with jitter
data_space <- ggplot(MedGPA_middle, aes(x = GPA, y = Acceptance)) +
  geom_jitter(width = 0, height = 0.05, alpha = 0.5) +
  theme_bw()
data_space
```



- Use geom_smooth() to add only the simple linear regression line to data_space.

```
# linear regression line
data_space +
  geom_smooth(method = "lm", se = FALSE)
```



5.3 Fitting a model

Logistic regression is a special case of a broader class of generalized linear models, often known as GLMs. Specifying a logistic regression model is very similar to specify a regression model, with two important differences:

- We use the `glm()` function instead of `lm()`
- We specify the `family` argument and set it to `binomial`. This tells the GLM function that we want to fit a logistic regression model to our binary response. The terminology stems from the assumption that our binary response follows a {binomial distribution.

We still use the formula and data arguments with `glm()`.

Note that the mathematical model is now:

$$\log\left(\frac{y}{1-y}\right) = \beta_0 + \beta_1 \cdot x + \varepsilon, \tag{5.1}$$

where ε is the error term.

Exercise

- Use `glm()` to fit a logistic regression model for `Acceptance` as a function of `GPA`.

```r
# fit model
mod <- glm(Acceptance ~ GPA, data = MedGPA, family = binomial)
mod
```

```

Call:  glm(formula = Acceptance ~ GPA, family = binomial, data = MedGPA)

Coefficients:
(Intercept)          GPA
    -19.207        5.454

Degrees of Freedom: 54 Total (i.e. Null);  53 Residual
Null Deviance:      75.79
Residual Deviance: 56.84    AIC: 60.84
```

Visualizing logistic regression

5.4 Using geom_smooth()

Our logistic regression model can be visualized in the data space by overlaying the appropriate logistic curve. We can use the `geom_smooth()` function to do this. Recall that `geom_smooth()` takes a method argument that allows you to specify what type of smoother you want to see. In our case, we need to specify that we want to use the `glm()` function to do the smoothing.

However we also need to tell the `glm()` function which member of the GLM family we want to use. To do this, we will pass the `family` argument to `glm()` as a list using the `method.args` argument to `geom_smooth()`. This mechanism is common in R, and allows one function to pass a list of arguments to another function.

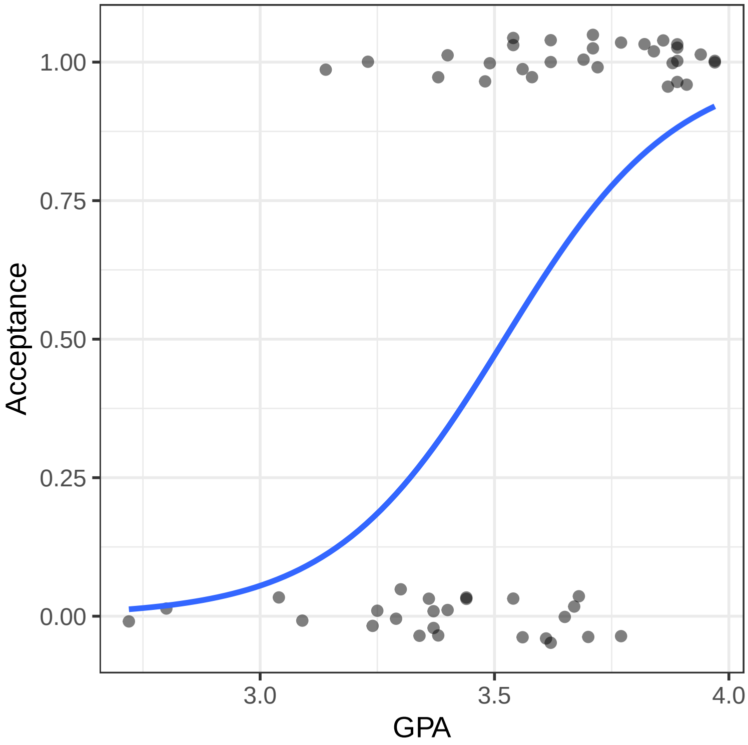
Exercise

- Create a scatterplot called `data_space` for `Acceptance` as a function of `GPA`. Use `geom_jitter()` to apply a small amount of jitter to the points in the y-direction. Set `width = 0` and `height = 0.05` in `geom_jitter()`.

```
# scatterplot with jitter
data_space <- ggplot(data = MedGPA, aes(y = Acceptance, x = GPA)) +
  geom_jitter(width = 0, height = 0.05, alpha = 0.5) +
  theme_bw()
```

- Use `geom_smooth()` to add the logistic regression line to `data_space` by specifying the `method` and `method.args` arguments to fit a logistic `glm`.

```
# add logistic curve
data_space +
  geom_smooth(method = "glm", se = FALSE, method.args = list(family = "binomial"))
```



5.5 Using bins

One of the difficulties in working with a binary response variable is understanding how it "changes." The response itself (y) is either 0 or 1, while the fitted values ($\hat{y}$)—which are interpreted as probabilities—are between 0 and 1. But if every medical school applicant is either admitted or not, what does it mean to talk about the probability of being accepted?

What we'd like is a larger sample of students, so that for each GPA value (e.g. 3.54) we had many observations (say n), and we could then take the average of those n observations to arrive at the estimated probability of acceptance. Unfortunately, since the explanatory variable is continuous, this is hopeless—it would take an infinite amount of data to make these estimates robust.

Instead, what we can do is put the observations into bins based on their GPA value. Within each bin, we can compute the proportion of accepted students, and we can visualize our model as a smooth logistic curve through those binned values.

We have created a `data.frame` called `MedGPA_binned` that aggregates the original data into separate bins for each **1/6** of GPA. It also contains the fitted values from the logistic regression model.

```
gpa_bins <- quantile(MedGPA$GPA, probs = seq(0, 1, 1/6))
gpa_bins
```

```
       0% 16.66667% 33.33333%       50% 66.66667% 83.33333%      100%
     2.72      3.30      3.44      3.58      3.70      3.87      3.97
```

```
MedGPA$bins <- cut(MedGPA$GPA, breaks = gpa_bins, include.lowest = TRUE)
head(MedGPA)
```

```
  Accept Acceptance Sex BCPM  GPA VR PS WS BS MCAT Apps       bins
1      D          0   F 3.59 3.62 11  9  9  9   38    5 (3.58,3.7]
2      A          1   M 3.75 3.84 12 13  8 12   45    3 (3.7,3.87]
3      A          1   F 3.24 3.23  9 10  5  9   33   19 [2.72,3.3]
4      A          1   F 3.74 3.69 12 11  7 10   40    5 (3.58,3.7]
5      A          1   F 3.53 3.38  9 11  4 11   35   11 (3.3,3.44]
6      A          1   M 3.59 3.72 10  9  7 10   36    5 (3.7,3.87]
```

```
MedGPA_binned <- MedGPA %>%
  group_by(bins) %>%
  summarize(mean_GPA = mean(GPA), acceptance_rate = mean(Acceptance))
MedGPA_binned
```

```
# A tibble: 6 x 3
  bins        mean_GPA acceptance_rate
  <fct>          <dbl>           <dbl>
```

```
1 [2.72,3.3]      3.11            0.2
2 (3.3,3.44]      3.39            0.2
3 (3.44,3.58]     3.54            0.75
4 (3.58,3.7]      3.65            0.333
5 (3.7,3.87]      3.79            0.889
6 (3.87,3.97]     3.91            1
```

Here we are plotting y as a function of x, where that function is

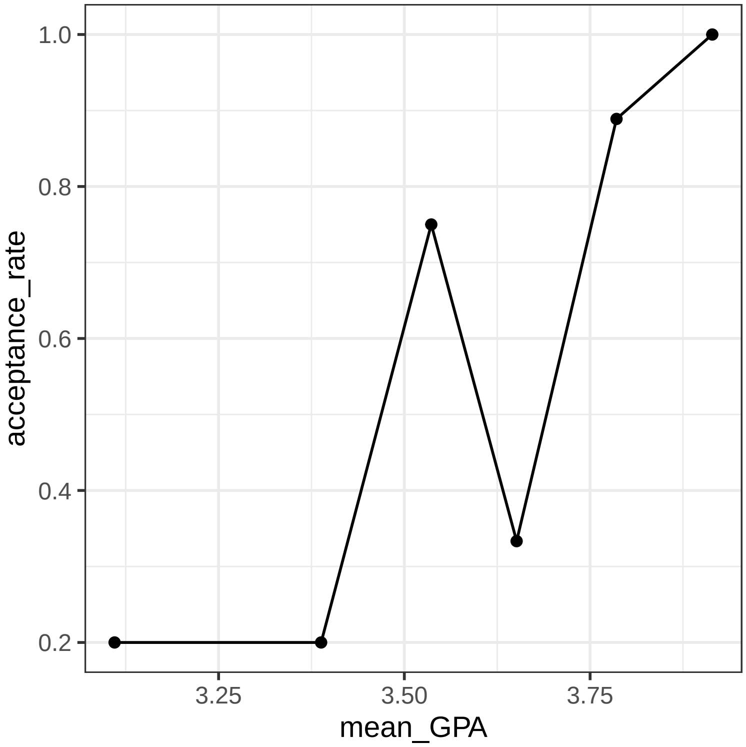
$$\hat{p}(X) = \widehat{\Pr}(Y = 1|X) = \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 x)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 x)} \tag{5.2}$$

Note that the left hand side is the expected probability y of being accepted to medical school.

Exercise

- Create a scatterplot called `data_space` for `acceptance_rate` as a function of `mean_GPA` using the binned data in `MedGPA_binned`. Use `geom_line()` to connect the points.

```
# binned points and line
data_space <- ggplot(data = MedGPA_binned, aes(x = mean_GPA, y = acceptance_rate)) +
  geom_point() +
  geom_line() +
  theme_bw()
data_space
```

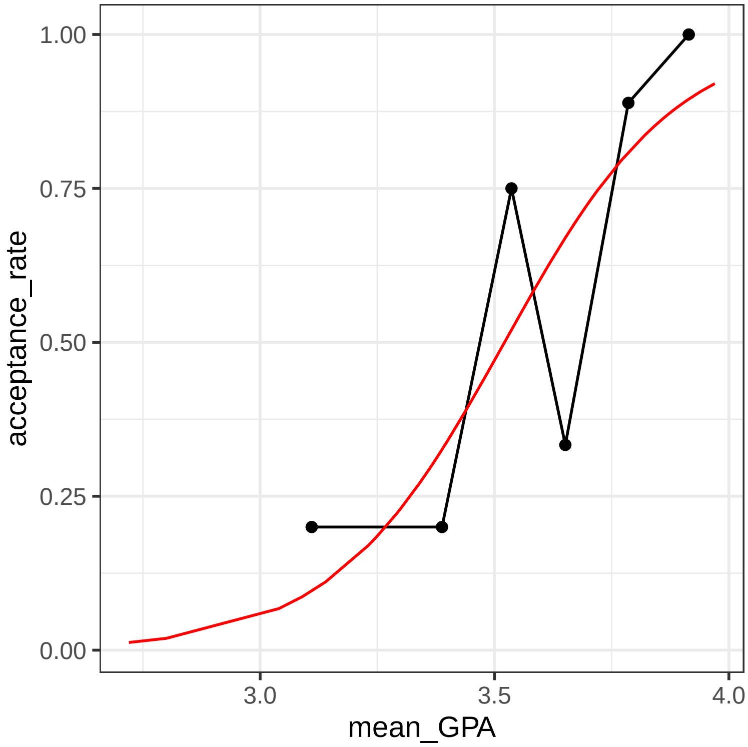


- Augment the model `mod`. Create predictions on the scale of the response variable by using the `type.predict` argument.

```
# augmented model
MedGPA_plus <- mod %>%
  augment(type.predict = "response")
```

- Use `geom_line()` to illustrate the model through the fitted values.

```
# logistic model on probability scale
data_space +
  geom_line(data = MedGPA_plus, aes(x = GPA, y = .fitted), color = "red")
```



The logistic predictions seem to follow the binned values pretty well.

5.6 Three scales approach to interpretation

5.7 Odds scale

For most people, the idea that we could estimate the probability of being admitted to medical school based on undergraduate GPA is fairly intuitive. However, thinking about how the probability changes as a function of GPA is complicated by the non-linear logistic curve. By translating the response from the probability scale to the odds scale, we make the right hand side of our equation easier to understand.

If the probability of getting accepted is y, then the odds are $y/(1-y)$. Expressions of probabilities in terms of odds are common in many situations, perhaps most notably gambling.

Here we are plotting $y/(1-y)$as a function of x, where that function is

$$\text{odds}(\hat{y}) = \frac{\hat{y}}{1-\hat{y}} = \exp(\hat{\beta}_0 + \hat{\beta}_1 \cdot x) \tag{5.3}$$

Note that the left hand side is the expected odds of being accepted to medical school. The right hand side is now a familiar exponential function of x.

The MedGPA_binned data frame contains the data for each GPA bin, while the MedGPA_plus data frame records the original observations after being augment()-ed by mod.

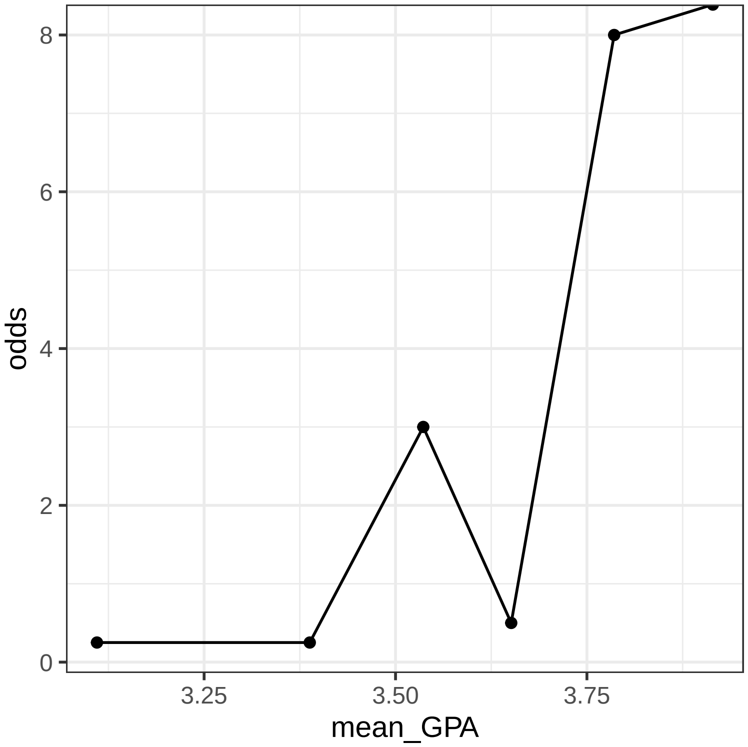
Exercise

- Add a variable called odds to MedGPA_binned that records the odds of being accepted to medical school for each bin.

```
# compute odds for bins
MedGPA_binned <- MedGPA_binned %>%
  mutate(odds = acceptance_rate / (1 - acceptance_rate))
```

- Create a scatterplot called data_space for odds as a function of mean_GPA using the binned data in MedGPA_binned. Connect the points with geom_line().

```
# plot binned odds
data_space <- ggplot(data = MedGPA_binned,
                     aes(x = mean_GPA, y = odds)) +
  geom_point() +
  geom_line() +
  theme_bw()
data_space
```

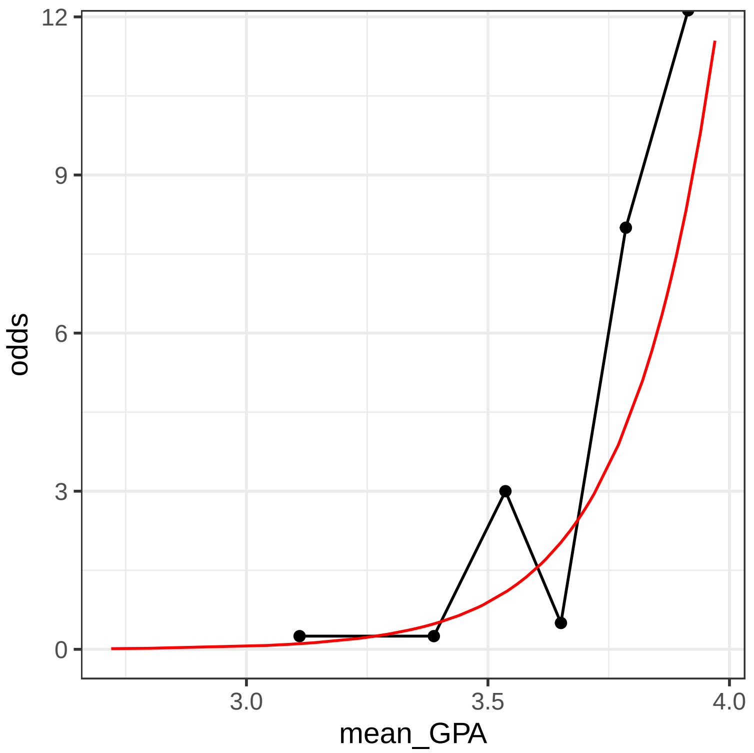


- Add a variable called `odds_hat` to `MedGPA_plus` that records the predicted odds of being accepted for each observation.

```
# compute odds for observations
MedGPA_plus <- MedGPA_plus %>%
  mutate(odds_hat = .fitted / (1 - .fitted))
```

- Use `geom_line()` to illustrate the model through the fitted values. Note that you should be plotting the $\widehat{odds}$'s.

```
# logistic model on odds scale
data_space +
  geom_line(data = MedGPA_plus, aes(x = GPA, y = odds_hat), color = "red")
```



5.8 Log-odds scale

Previously, we considered two formulations of logistic regression models:

- on the probability scale, the units are easy to interpret, but the function is non-linear, which makes it hard to understand
- on the odds scale, the units are harder (but not impossible) to interpret, and the function is exponential, which makes it harder (but not impossible) to interpret

We'll now add a third formulation:

- on the log-odds scale, the units are nearly impossible to interpret, but the function is linear, which makes it easy to understand As you can see, none of these three is uniformly superior. Most people tend to interpret the fitted values on the probability scale and the function on the log-odds scale. The interpretation of the coefficients is most commonly done on the odds scale. Recall that we interpreted our slope coefficient β_1 in linear regression as the expected change in y given a one unit change in x.

On the probability scale, the function is non-linear and so this approach won't work. On the log-odds, the function is linear, but the units are not interpretable (what does the log of the odds mean??). However, on the odds scale, a one unit change in x leads to the odds being multiplied by a factor of $\hat{\beta}_1$. To see why, we form the odds ratio:

$$\text{OR} = \frac{\text{odds}(\hat{y}|x+1)}{\text{odds}(\hat{y}|x)} = \exp \hat{\beta}_1 \tag{5.4}$$

Thus, the exponentiated coefficient β_1 tells us how the expected odds change for a one unit increase in the explanatory variable. It is tempting to interpret this as a change in the expected probability, but this is wrong and can lead to nonsensical predictions (e.g. expected probabilities greater than 1).

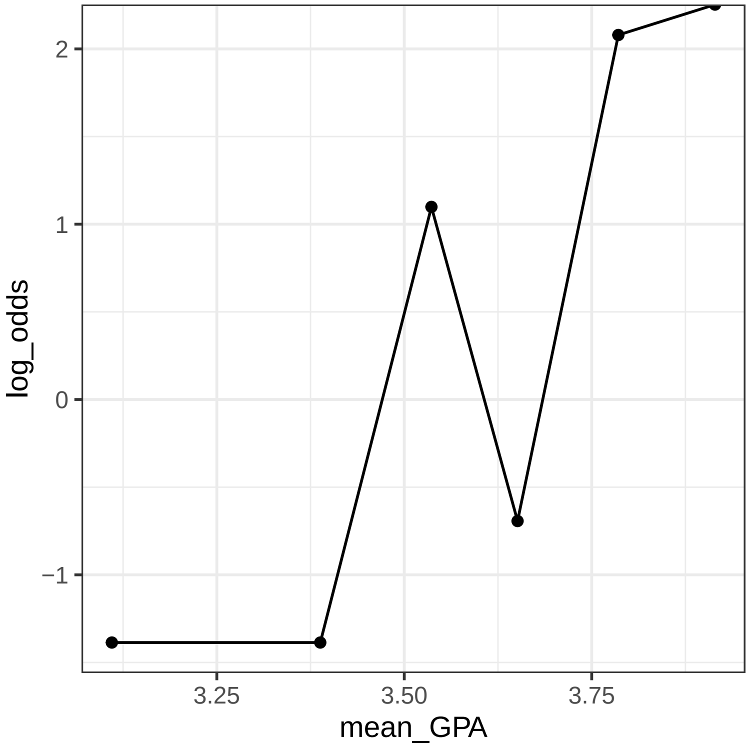
Exercise

- Add a variable called `log_odds` to `MedGPA_binned` that records the odds of being accepted for each bin. Recall that $odds(p) = p/(1-p)$.

```
# compute log odds for bins
MedGPA_binned <- MedGPA_binned %>%
      mutate(log_odds = log(acceptance_rate / (1 - acceptance_rate)))
```

- Create a scatterplot called `data_space` for `log_odds` as a function of `mean_GPA` using the binned data in `MedGPA_binned`. Use `geom_line` to connect the points.

```
# plot binned log odds
data_space <- ggplot(data = MedGPA_binned,
aes(y = log_odds, x = mean_GPA)) +
    geom_point() +
    geom_line() +
    theme_bw()
data_space
```

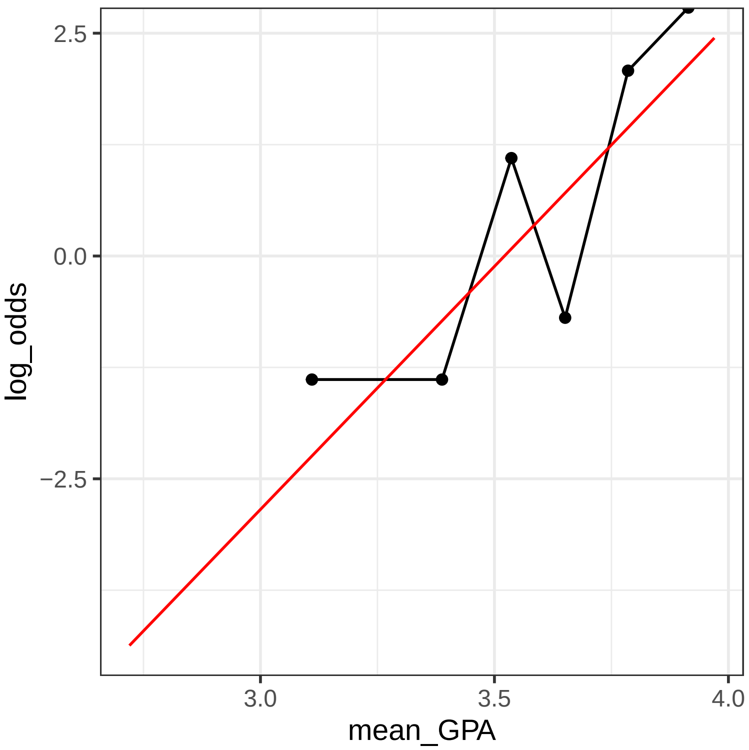


- Add a variable called `log_odds_hat` to `MedGPA_plus` that records the predicted odds of being accepted for each observation.

```
# compute log odds for observations
MedGPA_plus <- MedGPA_plus %>%
  mutate(log_odds_hat = log(.fitted / (1 - .fitted)))
```

- Use `geom_line()` to illustrate the model through the fitted values. Note that you should be plotting the $\widehat{\text{odds}}$'s.

```
# logistic model on log odds scale
data_space +
  geom_line(data = MedGPA_plus, aes(x = GPA, y = log_odds_hat), color = "red")
```



When you're on the log-odds scale, your model is a simple linear function.

Interpretation of logistic regression

The fitted coefficient $\widehat{\beta}_1$ from the medical school logistic regression model is 5.45. The exponential of this is 233.73.

Donald's GPA is 2.9, and thus the model predicts that the probability of him getting into medical school is 3.26%. The odds of Donald getting into medical school are 0.0337, or—phrased in gambling terms—29.6:1. If Donald hacks the school's registrar and changes his GPA to 3.9, then which of the following statements is **FALSE**:

- His expected odds of getting into medical school improve to 7.8833 (or about 9:8).
- His expected probability of getting into medical school improves to 88.7%.
- His expected log-odds of getting into medical school improve by 5.45.
- **His expected probability of getting into medical school improves to 7.9%.** This is a FALSE statement.

Using a logistic model

5.9 Making probabilistic predictions

Just as we did with linear regression, we can use our logistic regression model to make predictions about new observations. In this exercise, we will use the `newdata` argument to the `augment()` function from the `broom` package to make predictions about students who were not in our original data set. These predictions are sometimes called *out-of-sample*.

Following our previous discussion about scales, with logistic regression it is important that we specify on which scale we want the predicted values. Although the default is `link` – which uses the log-odds scale – we want our predictions on the probability scale, which is the scale of the `response` variable. The `type.predict` argument to `augment()` controls this behavior.

Exercise

- Create a new data frame which has one variable called `GPA` and one row, with the value 3.51.

```
# create new data frame
new_data <- data.frame(GPA = 3.51)
```

- Use `augment()` to find the expected probability of admission to medical school for a student with a GPA of 3.51.

```
# make predictions
augment(mod, newdata = new_data, type.predict = "response")
```

```
# A tibble: 1 x 2
    GPA .fitted
  <dbl>   <dbl>
1  3.51   0.484
```

By framing your prediction as a probability you can show how likely it is that this student will get admitted to medical school.

5.10 Making binary predictions

Naturally, we want to know how well our model works. Did it predict acceptance for the students who were actually accepted to medical school? Did it predict rejections for the student who were not admitted? These types of predictions are called in-sample. One common way to evaluate models with a binary response is with a confusion matrix. [Yes, that is actually what it is called!]

However, note that while our response variable is binary, our fitted values are probabilities. Thus, we have to round them somehow into binary predictions. While the probabilities convey more information, we might ultimately have to make a decision, and so this rounding is common in practice. There are many different ways to round, but for simplicity we will predict admission if the fitted probability is greater than 0.5, and rejection otherwise.

First, we'll use `augment()` to make the predictions, and then `mutate()` and `round()` to convert these probabilities into binary decisions. Then we will form the confusion matrix using the `table()` function. `table()` will compute a 2-way table when given a data frame with two categorical variables, so we will first use `select()` to grab only those variables.

You will find that this model made only 15 mistakes on these 55 observations, so it is nearly 73% accurate.

Exercise

The model object `mod` is already in your workspace.

- Create a data frame with the actual observations, and their fitted probabilities, and add a new column with the binary decision by rounding the fitted probabilities.

```
# data frame with binary predictions
tidy_mod <- augment(mod, type.predict = "response") %>%
  mutate(Acceptance_hat = round(.fitted))
```

- Compute the confusion matrix between the actual and predicted acceptance.

```
# confusion matrix
tidy_mod %>%
  select(Acceptance, Acceptance_hat) %>%
  table()
```

```
          Acceptance_hat
Acceptance  0  1
```

```
0 16  9
1  6 24
```

Chapter 6

Case Study: Italian restaurant in NYC

Explore the relationship between price and the quality of food, service, and decor for Italian restaurants in NYC.

Italian restaurants in NYC

Exploratory data analysis

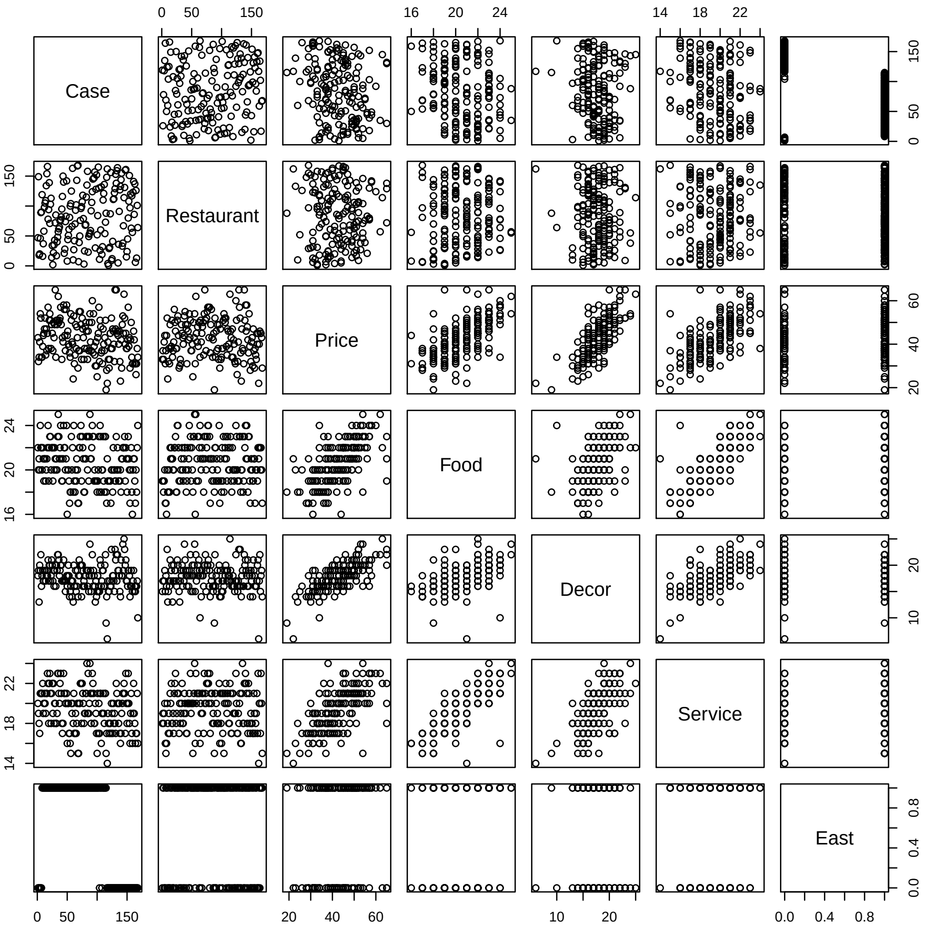
Multiple regression can be an effective technique for understanding how a response variable changes as a result of changes to more than one explanatory variable. But it is not magic – understanding the relationships among the explanatory variables is also necessary, and will help us build a better model. This process is often called exploratory data analysis (EDA) and is covered in another DataCamp course.

One quick technique for jump-starting EDA is to examine all of the pairwise scatterplots in your data. This can be achieved using the `pairs()` function. Look for variables in the `nyc` data set that are strongly correlated, as those relationships will help us check for multicollinearity later on.

Excercise

Which pairs of variables appear to be strongly correlated?

```
pairs(nyc)
```



- Case and Decor.
- Restaurant and Price.
- **Price and Food.**
- Price and East.

6.1 SLR models

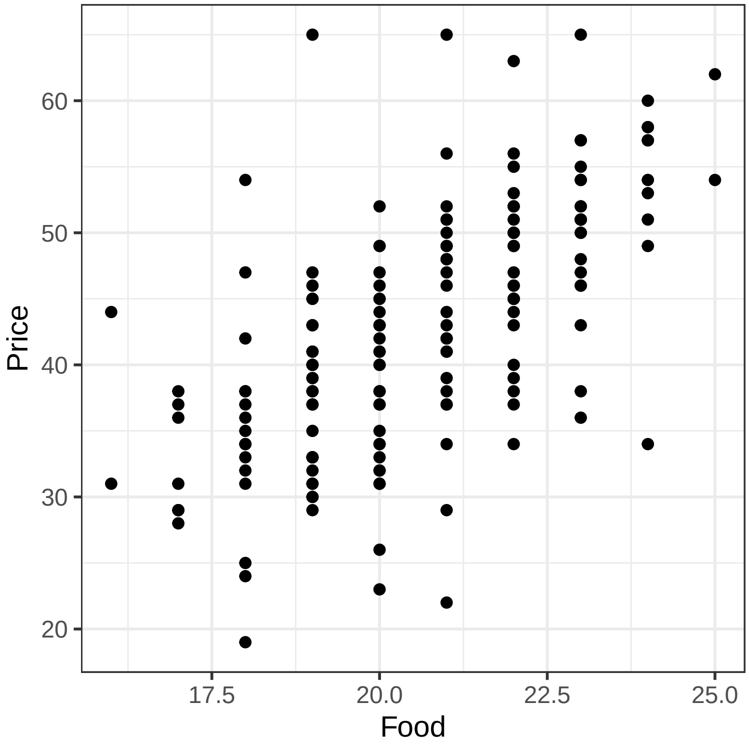
Based on your knowledge of the restaurant industry, do you think that the quality of the food in a restaurant is an important determinant of the price of a meal at that restaurant? It would be hard to imagine that it wasn't. We'll start our modeling process by plotting and fitting a model for `Price` as a function of `Food`.

On your own, interpret these coefficients and examine the fit of the model. What does the coefficient of `Food` mean in plain English? "Each additional rating point of food quality is associated with a…"

Exercise

- Use `ggplot` to make a scatter plot for `Price` as a function of `Food`.

```
# Price by Food plot
ggplot(data = nyc, aes(x = Food, y = Price)) +
  geom_point() +
  theme_bw()
```



- Use `lm()` to fit a simple linear regression model for `Price` as a function of `Food`.

```
# Price by Food model
lm(Price ~ Food, data = nyc)
```

```

Call:
lm(formula = Price ~ Food, data = nyc)

Coefficients:
(Intercept)         Food  
    -17.832        2.939  
```

What does the simple linear model say about how food quality affects price?

Incorporating another variable

6.2 Parallel lines with location

In real estate, a common mantra is that the three most important factors in determining the price of a property are “location, location, and location.” If location drives up property values and rents, then we might imagine that location would increase a restaurant’s costs, which would result in them having higher prices. In many parts of New York, the east side (east of 5th Avenue) is more developed and perhaps more expensive. [This is increasingly less true, but was more true at the time these data were collected.]

Let’s expand our model into a parallel slopes model by including the East variable in addition to Food.

Use `lm()` to fit a parallel slopes model for `Price` as a function of `Food` and `East`. Interpret the coefficients and the fit of the model. Can you explain the meaning of the coefficient on `East` in simple terms? Did the coefficient on `Food` change from the previous model? If so, why? Did it change by a lot or just a little?

Identify the statement that is *FALSE*:

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

```
Call:
lm(formula = Price ~ Food + East, data = nyc)

Coefficients:
(Intercept)         Food         East
    -17.430        2.875        1.459
```

- Each additional rating point of food quality is associated with a $2.88 increase in the expected price of meal, after controlling for location.
- The premium for an Italian restaurant in NYC associated with being on the east side of 5th Avenue is $1.46, after controlling for the quality of the food.
- **The change in the coefficient of food from $2.94 in the simple linear model to $2.88 in this model has profound practical implications for restaurant owners.**
- None of the above.

6.3 A plane in 3D

One reason that many people go to a restaurant—apart from the food—is that they don't have to cook or clean up. Many people appreciate the experience of being waited upon, and we can all agree that the quality of the service at restaurants varies widely. Are people willing to pay more for better restaurant `Service`? More interestingly, are they willing to pay more for better service, after controlling for the quality of the food?

Multiple regression gives us a way to reason about these questions. Fit the model with `Food` and `Service` and interpret the coefficients and fit. Did the coefficient on Food change from the previous model? What do the coefficients on `Food` and `Service` tell you about how these restaurants set prices?

Next, let's visually assess our model using `plotly`. The `x` and `y` vectors, as well as the `plane` matrix, have been created for you.

```r
hmod <- lm(Price ~ Food + Service, data = nyc)
summary(hmod)$coef
```

```
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) -21.158582  5.6651431 -3.734872 2.583345e-04
Food          1.495369  0.4462060  3.351297 9.971979e-04
Service       1.704101  0.4184986  4.071939 7.220788e-05
```

```
x <- seq(16, 25, length = 50)
y <- seq(14, 24, length = 50)
plane <- outer(x, y, function(a, b){-21.158582 + 1.495369*a + 1.704101*b})
```

Exercise

- Use `lm()` to fit a multiple regression model for `Price` as a function of `Food` and `Service`.

```
# fit model
lm(Price ~ Food + Service, data = nyc)
```

```

Call:
lm(formula = Price ~ Food + Service, data = nyc)

Coefficients:
(Intercept)         Food      Service
    -21.159        1.495        1.704
```

- Use `plot_ly` to draw 3D scatterplot for Price as a function of `Food` and `Service` by mapping the `z` variable to the response and the `x` and `y` variables to the explanatory variables. Place the food quality on the x-axis and service rating on the y-axis.

```
library(plotly)
# draw 3D scatterplot
p <- plot_ly(data = nyc, z = ~ Price, x = ~ Food, y = ~ Service, opacity = 0.6) %>%
  add_markers()
p
```

- Use `add_surface()` to draw a plane through the cloud of points using the object `plane`.

```
# draw a plane
p %>%
  add_surface(x = ~x, y = ~y, z = ~ plane, showscale = FALSE)
```

Is it surprising how service affects the price of a meal?

Higher dimensions

6.4 Parallel planes with location

We have explored models that included the quality of both food and service, as well as location, but we haven't put these variables all into the same model. Let's now build a parallel planes model that incorporates all three variables.

Examine the coefficients closely. Do they make sense based on what you understand about these data so far? How did the coefficients change from the previous models that you fit?

Exercise

- Use `lm()` to fit a parallel planes model for `Price` as a function of `Food`, `Service`, and `East`.

```r
# Price by Food and Service and East
lm(Price ~ Food + Service + East, data = nyc)
```

```

Call:
lm(formula = Price ~ Food + Service + East, data = nyc)

Coefficients:
(Intercept)         Food      Service         East  
   -20.8155       1.4863       1.6647       0.9649  
```

Does it seem like location has a big impact on price?

Interpretation of location coefficient

The fitted coefficients from the parallel planes model are listed below.

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

```

Call:
lm(formula = Price ~ Food + Service + East, data = nyc)

Coefficients:
(Intercept)         Food      Service         East  
   -20.8155       1.4863       1.6647       0.9649  
```

Which of the following statements is **FALSE**?

Reason about the magnitude of the `East` coefficient.

- The premium for being on the East side of 5th Avenue is just less than a dollar, after controlling for the quality of food and service.
- The impact of location is relatively small, since one additional rating point of either food or service would result in a higher expected price than moving a restaurant from the West side to the East side.
- **The expected price of a meal on the East side is about 96% of the cost of a meal on the West side, after controlling for the quality of food and service.**

6.5 Impact of location

The impact of location brings us to a modeling question: should we keep this variable in our model? In a later course, you will learn how we can conduct formal hypothesis tests to help us answer that question. In this course, we will focus on the size of the effect. Is the impact of location big or small?

One way to think about this would be in terms of the practical significance. Is the value of the coefficient large enough to make a difference to your average person? The units are in dollars so in this case this question is not hard to grasp.

Another way is to examine the impact of location in the context of the variability of the other variables. We can do this by building our parallel planes in 3D and seeing how far apart they are. Are the planes close together or far apart? Does the `East` variable clearly separate the data into two distinct groups? Or are the points all mixed up together?

```
modJ <- lm(Price ~ Food + Service + East, data = nyc)
summary(modJ)$coef
```

```
               Estimate Std. Error    t value     Pr(>|t|)
(Intercept) -20.8154761  5.6843188 -3.6619121 0.0003373782
Food          1.4862725  0.4467122  3.3271368 0.0010831115
Service       1.6646884  0.4214169  3.9502175 0.0001157434
East          0.9648814  1.1363317  0.8491195 0.3970525764
```

```
plane0 <- outer(x, y, function(a, b){-20.8154761 + 1.4862725*a + 1.6646884*b + 0.96488
plane1 <- outer(x, y, function(a, b){-20.8154761 + 1.4862725*a + 1.6646884*b})
```

Exercise

- Use plot_ly to draw 3D scatterplot for Price as a function of Food, Service, and East by mapping the z variable to the response and the x and y variables to the numeric explanatory variables. Use color to indicate the value of East. Place Food on the x-axis and Service on the y-axis.

```
library(plotly)
# draw 3D scatterplot
p <- plot_ly(data = nyc, z = ~Price, x = ~Food, y = ~Service, opacity = 0.6) %>%
  add_markers(color = ~factor(East))
p
```

- Use add_surface() (twice) to draw two planes through the cloud of points, one for restaurants on the West side and another for restaurants on the East side. Use the objects plane0 and plane1.

```
# draw two planes
p %>%
  add_surface(x = ~x, y = ~y, z = ~plane0, showscale = FALSE) %>%
  add_surface(x = ~x, y = ~y, z = ~plane1, showscale = FALSE)
```

How does this visualization relate to the model coefficients you found in the last exercise?

6.6 Full model

One variable we haven't considered is Decor. Do people, on average, pay more for a meal in a restaurant with nicer decor? If so, does it still matter after controlling for the quality of food, service, and location?

By adding a third numeric explanatory variable to our model, we lose the ability to visualize the model in even three dimensions. Our model is now a hyperplane – or rather, parallel hyperplanes – and while we won't go any further with the geometry, know that we can continue to add as many variables to our model as we want. As humans, our spatial visualization ability taps out after three numeric variables (maybe you could argue for four, but certainly no further), but neither the mathematical equation for the regression model, nor the formula specification for the model in R, is bothered by the higher dimensionality.

Use lm() to fit a parallel planes model for Price as a function of Food, Service, Decor, and East.

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

```

Call:
lm(formula = Price ~ Food + Service + Decor + East, data = nyc)

Coefficients:
(Intercept)         Food      Service        Decor         East  
 -24.023800     1.538120    -0.002727     1.910087     2.068050  
```

Notice the dramatic change in the value of the `Service` coefficient.

Which of the following interpretations is invalid?

- Since the quality of food, decor, and service were all strongly correlated, multicollinearity is the likely explanation.
- Once we control for the quality of food, decor, and location, the additional information conveyed by service is negligible.
- **Service is not an important factor in determining the price of a meal.** This is false!
- None of the above.

Wrap-up