Multiple and Logistic Regression

Your Name Here

Last compiled: Aug 06, 2019

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.

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()`.

```
library(openintro)
data(marioKart)
glimpse(marioKart)
```

```
Observations: 143
Variables: 12
$ ID         <dbl> 150377422259, 260483376854, 320432342985, 280405224...
$ duration   <int> 3, 7, 3, 3, 1, 3, 1, 1, 3, 7, 1, 1, 1, 1, 7, 7, 3, ...
$ nBids      <int> 20, 13, 16, 18, 20, 19, 13, 15, 29, 8, 15, 15, 13, ...
$ cond       <fct> new, used, new, new, new, new, used, new, used, use...
$ startPr    <dbl> 0.99, 0.99, 0.99, 0.99, 0.01, 0.99, 0.01, 1.00, 0.9...
```

```
$ shipPr     <dbl> 4.00, 3.99, 3.50, 0.00, 0.00, 4.00, 0.00, 2.99, 4.0...
$ totalPr    <dbl> 51.55, 37.04, 45.50, 44.00, 71.00, 45.00, 37.02, 53...
$ shipSp     <fct> standard, firstClass, firstClass, standard, media, ...
$ sellerRate <int> 1580, 365, 998, 7, 820, 270144, 7284, 4858, 27, 201...
$ stockPhoto <fct> yes, yes, no, yes, yes, yes, yes, yes, yes, no, yes...
$ wheels     <int> 1, 1, 1, 1, 2, 0, 0, 2, 1, 1, 2, 2, 2, 2, 1, 0, 1, ...
$ title      <fct> "~~ Wii MARIO KART &amp; WHEEL ~ NINTENDO Wii ~ BRA...
```

```
# Or
# str(marioKart)
# Data munging to agree with DataCamp mario_kart
mario_kart <- marioKart %>%
  filter(totalPr < 100)
str(mario_kart)
```

```
'data.frame':   141 obs. of  12 variables:
 $ ID        : num  1.5e+11 2.6e+11 3.2e+11 2.8e+11 1.7e+11 ...
 $ duration  : int  3 7 3 3 1 3 1 1 3 7 ...
 $ nBids     : int  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 ...
 $ startPr   : num  0.99 0.99 0.99 0.99 0.01 ...
 $ shipPr    : num  4 3.99 3.5 0 0 4 0 2.99 4 4 ...
 $ totalPr   : num  51.5 37 45.5 44 71 ...
 $ shipSp    : Factor w/ 8 levels "firstClass","media",..: 6 1 1 6 2 6 6 8 5 1 ...
 $ sellerRate: int  1580 365 998 7 820 270144 7284 4858 27 201 ...
 $ stockPhoto: Factor w/ 2 levels "no","yes": 2 2 1 2 2 2 2 2 2 1 ...
 $ wheels    : int  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 4 19 3
```

```
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.

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

```

Call:
lm(formula = totalPr ~ 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 `totalPr`, `startPr`, and `shipPr` variables are numeric, while the `cond` and `stockPhoto` variables are categorical.

Which formula will result in a parallel slopes model?

- `totalPr ~ startPr + shipPr`
- `cond ~ startPr + stockPhoto`

- `totalPr ~ shipPr + stockPhoto`
- `totalPr ~ cond`

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 = totalPr ~ 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)
```

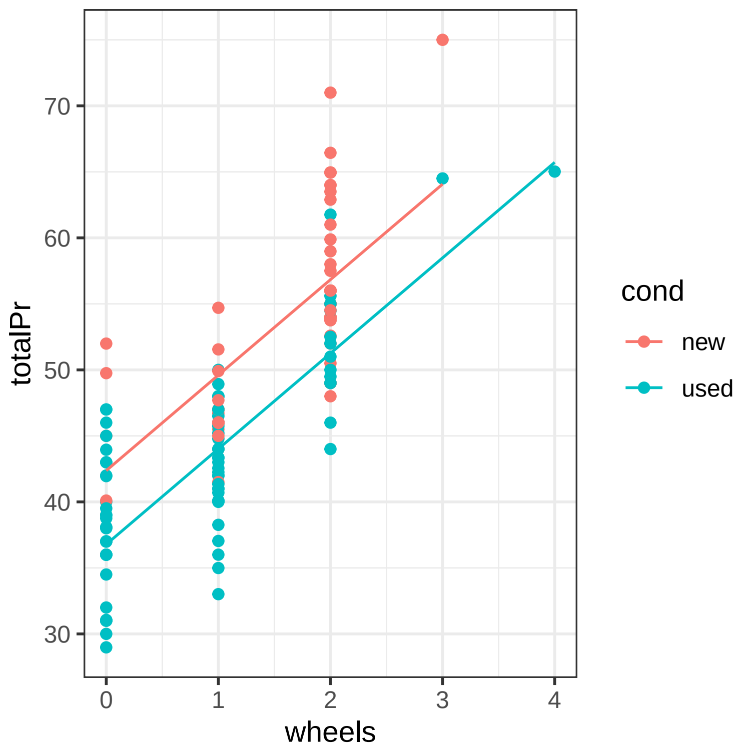
```
Observations: 141
Variables: 10
$ totalPr    <dbl> 51.55, 37.04, 45.50, 44.00, 71.00, 45.00, 37.02, 53...
$ wheels     <int> 1, 1, 1, 1, 2, 0, 0, 2, 1, 1, 2, 2, 2, 2, 1, 0, 1, ...
$ cond       <fct> new, used, new, new, new, new, used, new, used, use...
$ .fitted    <dbl> 49.60260, 44.01777, 49.60260, 49.60260, 56.83544, 4...
$ .se.fit    <dbl> 0.7087865, 0.5465195, 0.7087865, 0.7087865, 0.67645...
$ .resid     <dbl> 1.9473995, -6.9777674, -4.1026005, -5.6026005, 14.1...
$ .hat       <dbl> 0.02103158, 0.01250410, 0.02103158, 0.02103158, 0.0...
$ .sigma     <dbl> 4.902339, 4.868399, 4.892414, 4.881308, 4.750591, 4...
$ .cooksd    <dbl> 1.161354e-03, 8.712334e-03, 5.154337e-03, 9.612441e...
$ .std.resid <dbl> 0.40270893, -1.43671086, -0.84838977, -1.15857953, ...
```

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

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

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

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



Intercept interpretation

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

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

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

```

Call:
lm(formula = totalPr ~ 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.

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.

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

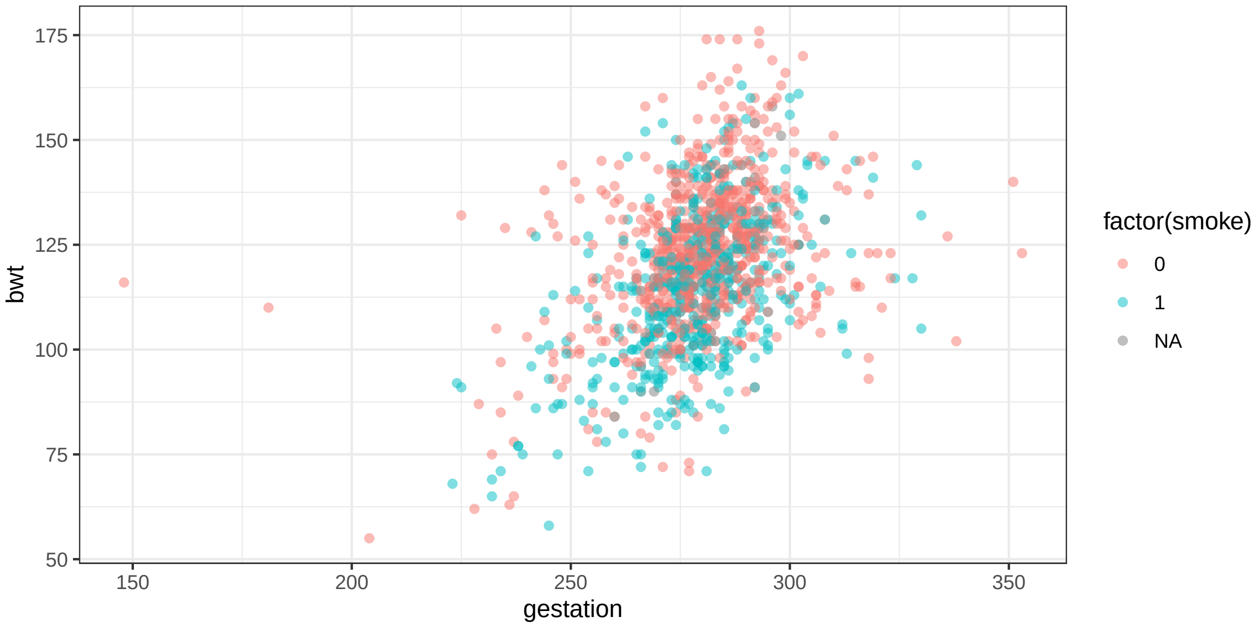


Figure 2.1: ‘bwt‘ versus ‘gestation‘

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.

```
# 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()
```

Exercise

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

```r
# 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
```

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.

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(totalPr ~ wheels + cond, data = mario_kart)
```

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

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

```

Call:
lm(formula = totalPr ~ 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`.

```r
# 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.

```r
# compute new model
mod2 <- lm(totalPr ~ 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_{adj}.**

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

```

Call:
lm(formula = totalPr ~ 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()`.

```
# 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()`.

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

```
# A tibble: 6 x 10
  totalPr wheels cond  .fitted .se.fit .resid   .hat .sigma .cooksd
    <dbl>  <int> <fct>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
1    51.6      1 new      49.6   0.709   1.95 0.0210   4.90 0.00116
2    37.0      1 used     44.0   0.547  -6.98 0.0125   4.87 0.00871
3    45.5      1 new      49.6   0.709  -4.10 0.0210   4.89 0.00515
4    44        1 new      49.6   0.709  -5.60 0.0210   4.88 0.00961
5    71        2 new      56.8   0.676  14.2  0.0192   4.75 0.0557
6    45        0 new      42.4   1.07    2.63 0.0475   4.90 0.00505
# ... with 1 more variable: .std.resid <dbl>
```

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

```
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.

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

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

Coefficients:
      (Intercept)           condused           duration
           58.268            -17.122             -1.966
condused:duration
            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.

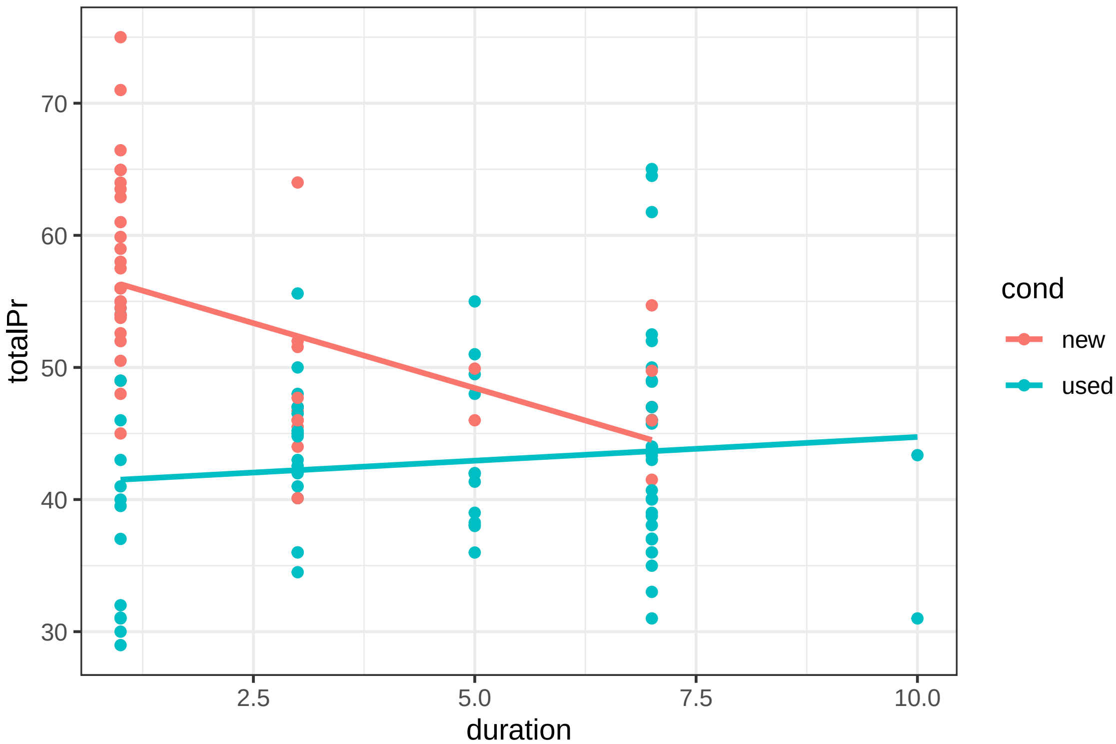
Interaction models are easy to visualize in the data space with `ggplot2` because they have the same coefficients as if the models were fit independently to each group defined by the level of the categorical variable. In this case, new and used MarioKarts each get their own regression line. To see this, 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()`.

```r
# interaction plot
ggplot(data = mario_kart, aes(y = totalPr, 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**

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.

```
SAT <- read.csv("https://assets.datacamp.com/production/repositories/845/datasets/1a12a19d2cec83ca0b5864
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 = totalPr, x = duration)) +
  geom_point() +
  geom_smooth(method = "lm", se = 0) +
  theme_bw()
slr
```

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

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

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

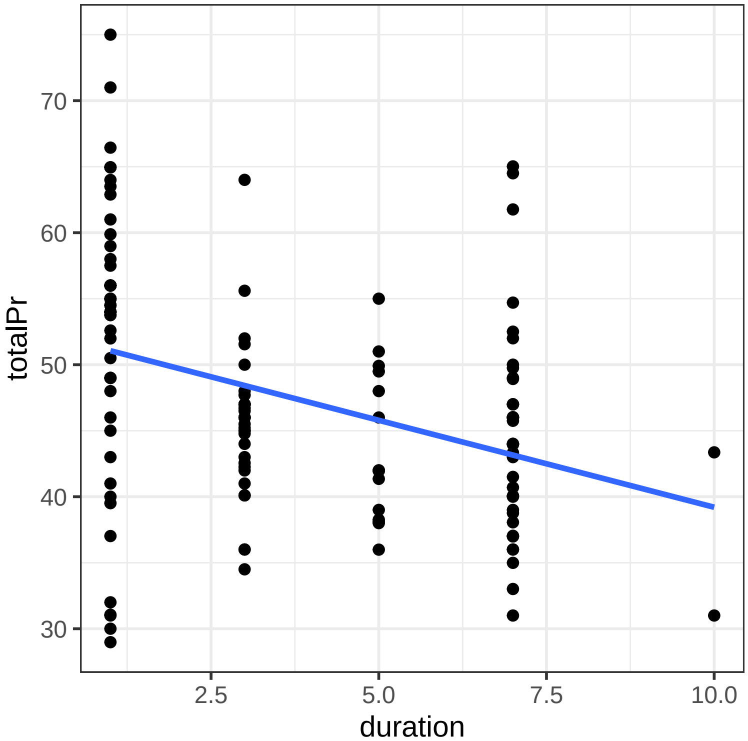
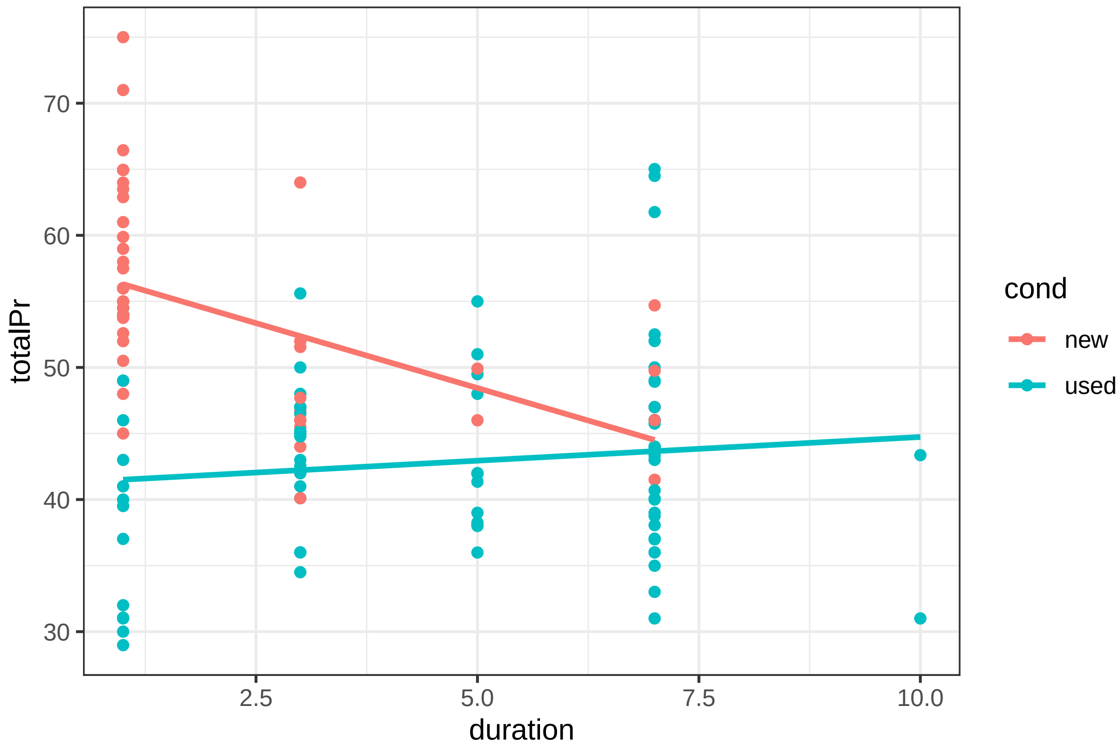


Figure 3.1: ‘totalPr‘ versus ‘duration‘

```
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 3.1.

```
# 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.

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.

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

```
'data.frame':	141 obs. of  12 variables:
 $ ID        : num  1.5e+11 2.6e+11 3.2e+11 2.8e+11 1.7e+11 ...
 $ duration  : int  3 7 3 3 1 3 1 1 3 7 ...
 $ nBids     : int  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 ...
 $ startPr   : num  0.99 0.99 0.99 0.99 0.01 ...
 $ shipPr    : num  4 3.99 3.5 0 0 4 0 2.99 4 4 ...
 $ totalPr   : num  51.5 37 45.5 44 71 ...
 $ shipSp    : Factor w/ 8 levels "firstClass","media",..: 6 1 1 6 2 6 6 8 5 1 ...
 $ sellerRate: int  1580 365 998 7 820 270144 7284 4858 27 201 ...
 $ stockPhoto: Factor w/ 2 levels "no","yes": 2 2 1 2 2 2 2 2 2 1 ...
 $ wheels    : int  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 4 19 34
```

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 startPr
mod <- lm(totalPr ~ duration + startPr, data = mario_kart)
mod
```

```
Call:
lm(formula = totalPr ~ duration + startPr, data = mario_kart)

Coefficients:
(Intercept)     duration      startPr
     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.

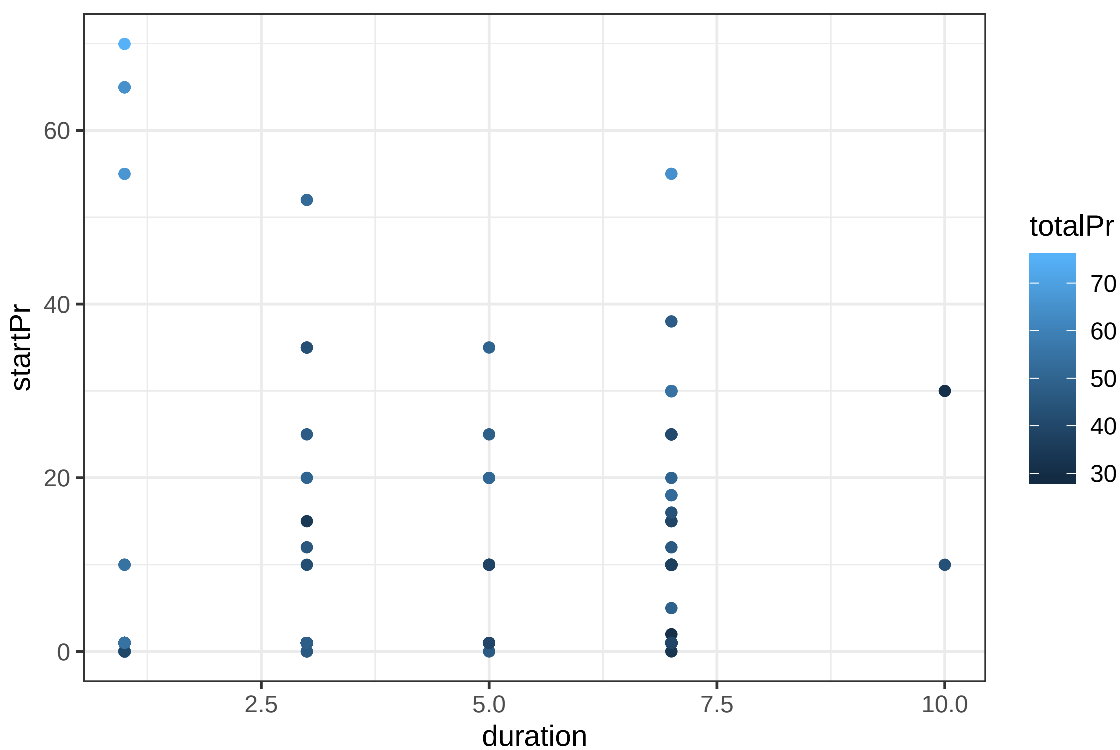
This is a multistep process:

- 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), startPr = seq(0.01, 69.95, by = 0.01))
```

- 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()`.

```
data_space <- ggplot(data = mario_kart,
                     aes(x = duration, y = startPr)) +
  geom_point(aes(color = totalPr)) +
  theme_bw()
data_space
```



Exercise

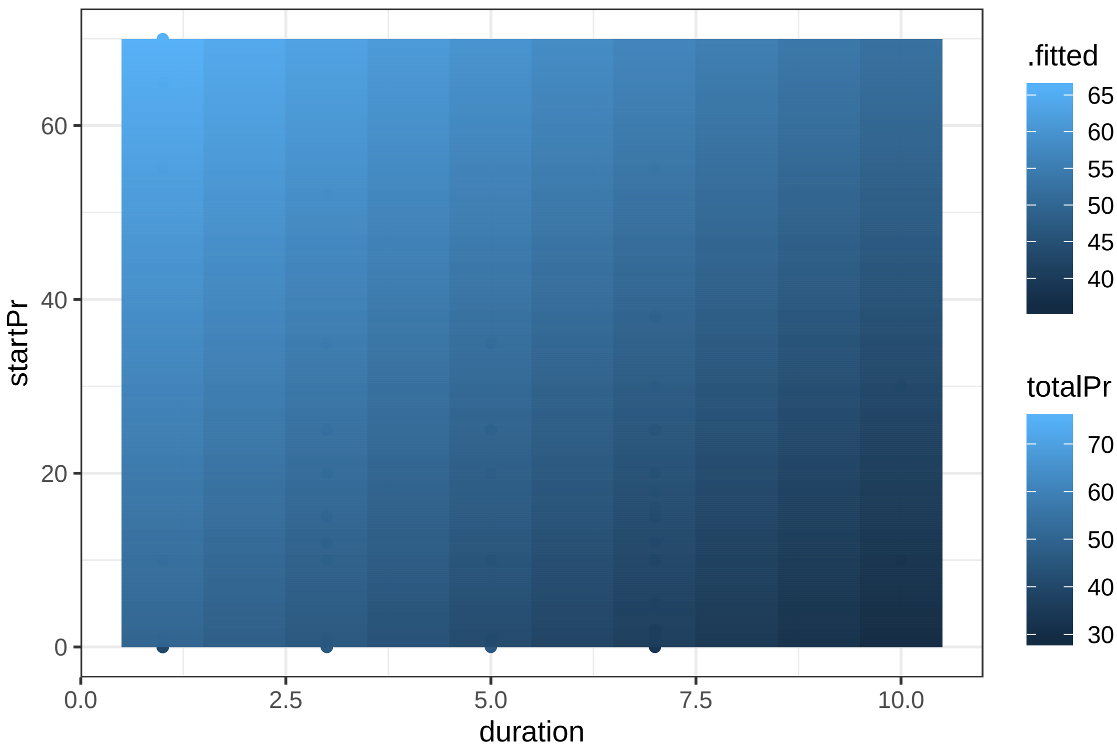
The model object `mod` is already in your workspace.

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

```
# 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`.

```
# 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 `startPr`

- `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 `totalPr` as a function of duration and `startPr` 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 = ~totalPr, x = ~duration, y = ~startPr, opacity = 0.6) %>%
  add_markers()
p
```

WebGL is not supported by your browser - visit https://get.webgl.org for more info

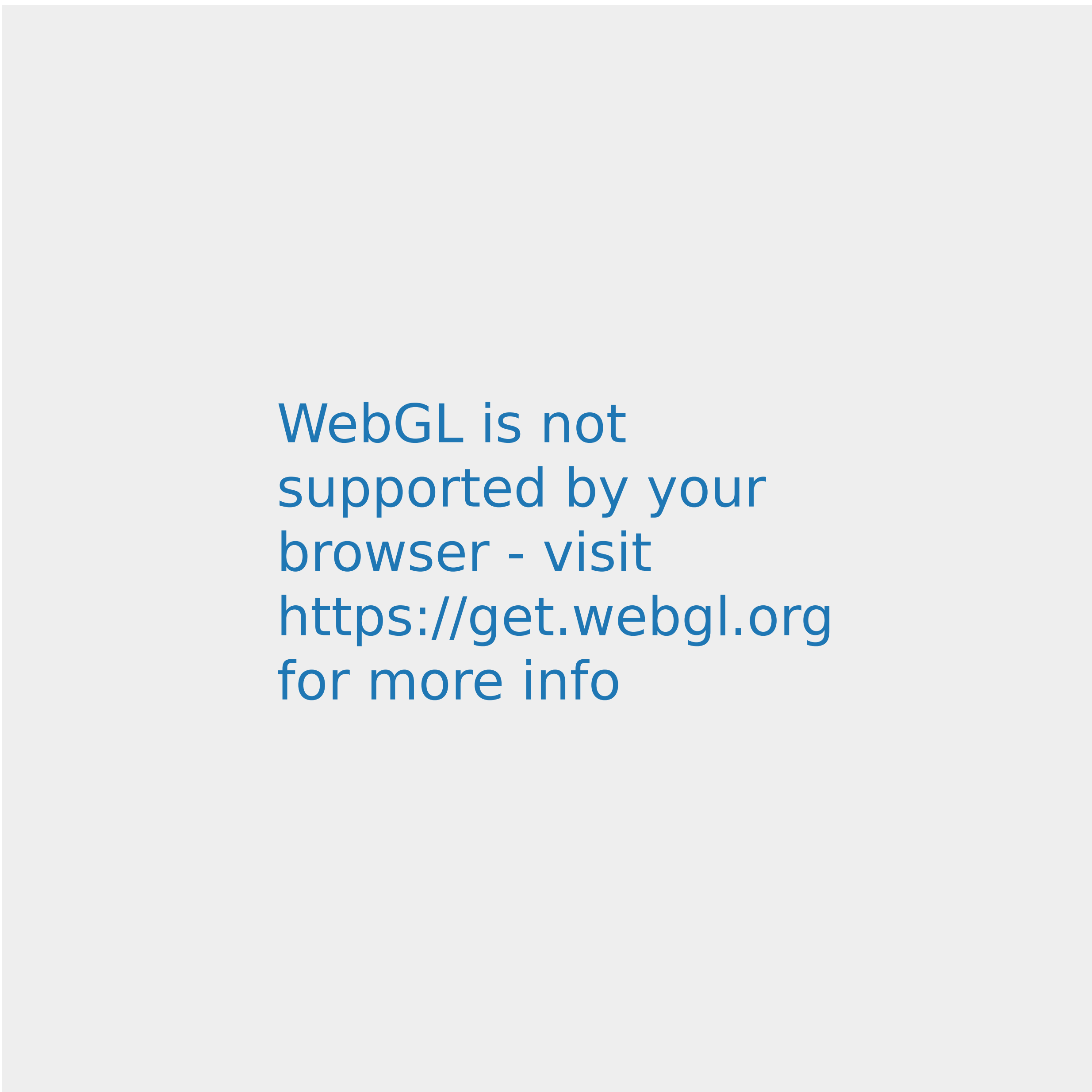
- Use `add_surface()` to draw a plane through the cloud of points by setting `z = ~plane`. See wikipedia for the definition of an outer product. In what follows, we will use the R function `outer()` to compute the values of `plane`.

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

```
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
startPr      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)
```



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 = totalPr ~ duration + startPr, data = mario_kart)

Coefficients:
(Intercept)     duration      startPr
     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.

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(totalPr ~ duration + startPr + cond, data = mario_kart)
summary(modI)$coef
```

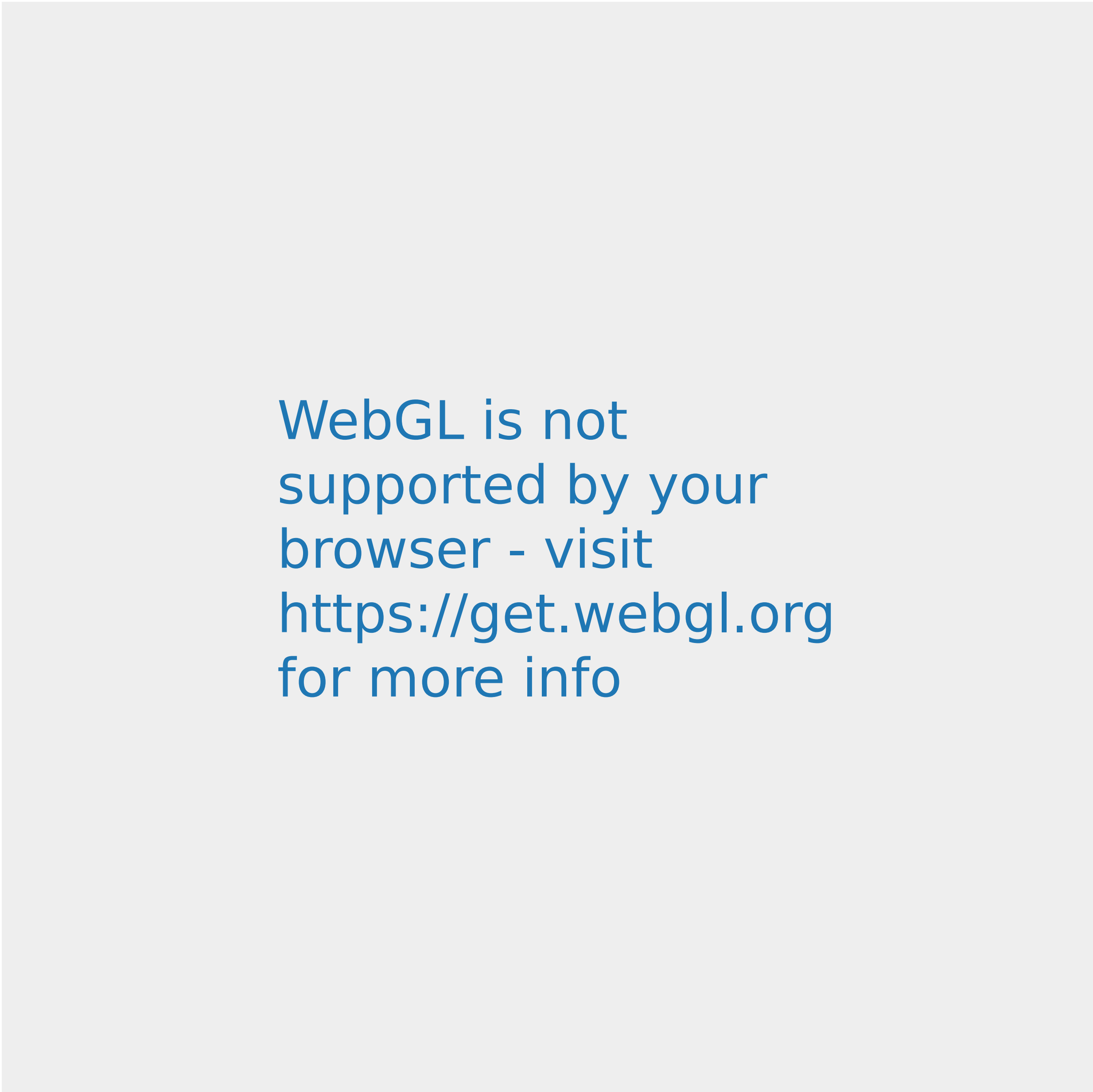
```
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 53.3447530  1.0804915 49.370822 3.781243e-89
duration    -0.6559841  0.2553503 -2.568957 1.127073e-02
startPr      0.1981653  0.0382717  5.177855 7.835882e-07
condused    -8.9493214  1.3237851 -6.760403 3.635333e-10
```

```
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

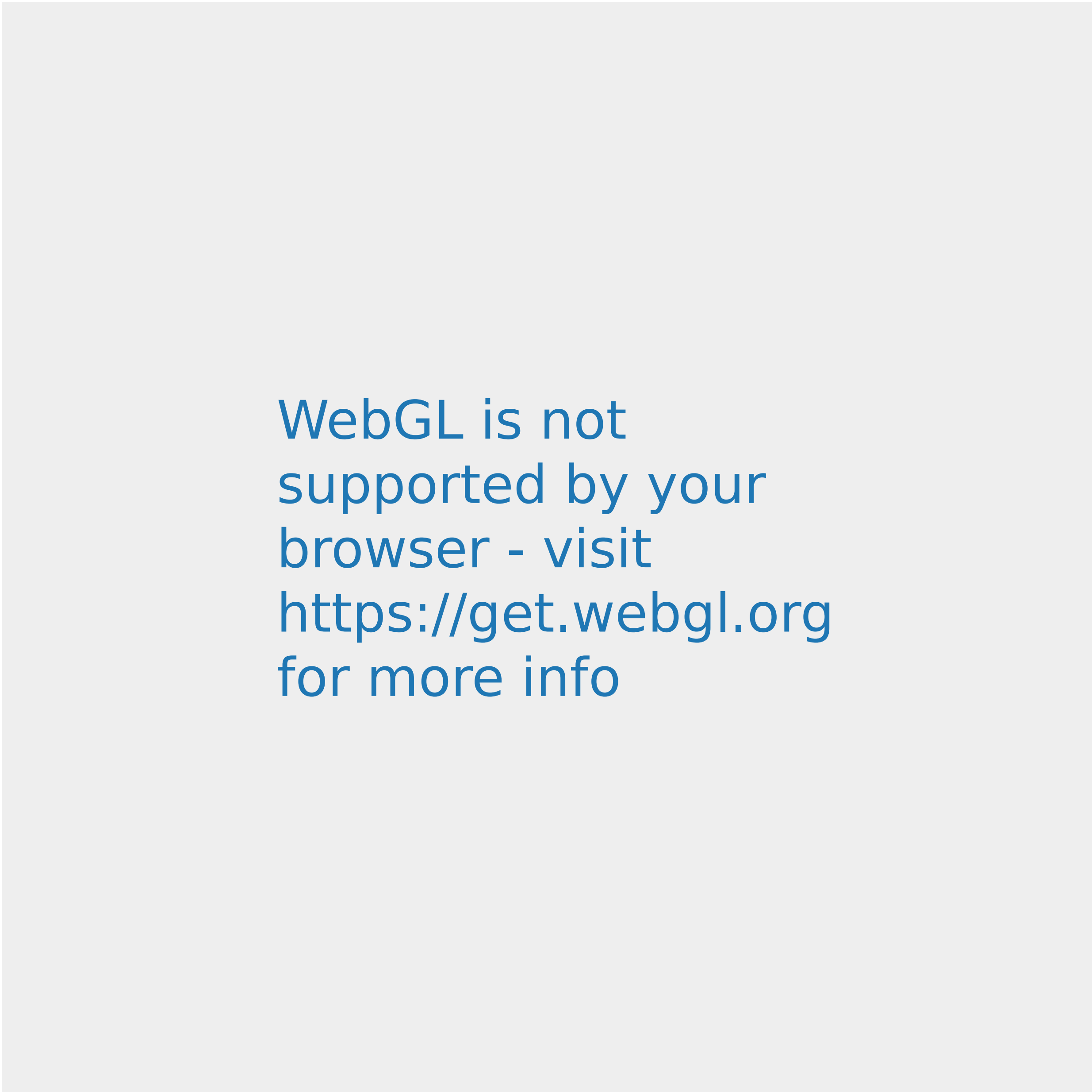
- Use `plot_ly` to draw 3D scatterplot for `totalPr` as a function of `duration`, `startPr`, and `cond` by mapping the `z` variable to the response and the `x` and `y` variables to the explanatory variables. Duration should be on the x-axis and starting price should be on the y-axis. Use color to represent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

```
# draw the 3D scatterplot
p <- plot_ly(data = mario_kart, z = ~totalPr, x = ~duration, y = ~startPr, 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 = totalPr ~ duration + startPr + cond, data = mario_kart)

Coefficients:
(Intercept)     duration      startPr     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.

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 (`nBids`) and the number of wheels. Unfortunately this makes a full visualization of our model impossible, but we can still interpret the coefficients.

```r
modJ <- lm(totalPr ~ duration + startPr + cond + wheels + nBids,
    data = mario_kart)
modJ
```

```

Call:
lm(formula = totalPr ~ duration + startPr + cond + wheels + nBids,
    data = mario_kart)

Coefficients:
(Intercept)     duration      startPr     condused       wheels
    39.3741      -0.2752       0.1796      -4.7720       6.7216
      nBids
     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

Applications

Chapter 6

Final Words