CHAPTER 9
Capstone: R for Data Analytics

In this chapter, we'll apply what we've learned about data analysis and visualization in R to explore and test relationships in the familiar *mpg* dataset. You'll learn a couple of new R techniques here, including how to conduct a t-test and linear regression. We'll begin by calling up the necessary packages, reading in *mpg.csv* from the *mpg* subfolder of the book repository's *datasets* folder, and selecting the columns of interest. We've not used `tidymodels` so far in this book, so you may need to install it.

```
library(tidyverse)
library(psych)
library(tidymodels)

# Read in the data, select only the columns we need
mpg <- read_csv('datasets/mpg/mpg.csv') %>%
  select(mpg, weight, horsepower, origin, cylinders)

#> -- Column specification --------------------------------------------------------
#> cols(
#>  mpg = col_double(),
#>  cylinders = col_double(),
#>  displacement = col_double(),
#>  horsepower = col_double(),
#>  weight = col_double(),
#>  acceleration = col_double(),
#>  model.year = col_double(),
#>  origin = col_character(),
#>  car.name = col_character()
#> )

head(mpg)
#> # A tibble: 6 x 5
#>     mpg weight horsepower origin cylinders
#>   <dbl>  <dbl>      <dbl> <chr>      <dbl>
#> 1    18   3504        130 USA            8
```

```
#> 2    15   3693        165 USA            8
#> 3    18   3436        150 USA            8
#> 4    16   3433        150 USA            8
#> 5    17   3449        140 USA            8
#> 6    15   4341        198 USA            8
```

Exploratory Data Analysis

Descriptive statistics are a good place to start when exploring data. We'll do so with the `describe()` function from `psych`:

```
describe(mpg)
#>            vars   n    mean     sd  median trimmed    mad  min
#> mpg           1 392   23.45   7.81   22.75   22.99   8.60    9
#> weight        2 392 2977.58 849.40 2803.50 2916.94 948.12 1613
#> horsepower    3 392  104.47  38.49   93.50   99.82  28.91   46
#> origin*       4 392    2.42   0.81    3.00    2.53   0.00    1
#> cylinders     5 392    5.47   1.71    4.00    5.35   0.00    3
#>               max  range  skew kurtosis    se
#> mpg          46.6   37.6  0.45    -0.54  0.39
#> weight     5140.0 3527.0  0.52    -0.83 42.90
#> horsepower  230.0  184.0  1.08     0.65  1.94
#> origin*       3.0    2.0 -0.91    -0.86  0.04
#> cylinders     8.0    5.0  0.50    -1.40  0.09
```

Because *origin* is a categorical variable, we should be careful to interpret its descriptive statistics. (In fact, `psych` uses `*` to signal this warning.) We are, however, safe to analyze its one-way frequency table, which we'll do using a new `dplyr` function, `count()`:

```
mpg %>%
  count(origin)
#> # A tibble: 3 x 2
#>   origin     n
#>   <chr>  <int>
#> 1 Asia      79
#> 2 Europe    68
#> 3 USA      245
```

We learn from the resulting count column *n* that while the majority of observations are American cars, the observations of Asian and European cars are still likely to be representative samples of their subpopulations.

Let's further break these counts down by `cylinders` to derive a two-way frequency table. I will combine `count()` with `pivot_wider()` to display `cylinders` along the columns:

```
mpg %>%
  count(origin, cylinders) %>%
  pivot_wider(values_from = n, names_from = cylinders)
#> # A tibble: 3 x 6
```

```
#>   origin   `3`   `4`   `6`   `5`   `8`
#>   <chr>  <int> <int> <int> <int> <int>
#> 1 Asia       4    69     6    NA    NA
#> 2 Europe    NA    61     4     3    NA
#> 3 USA       NA    69    73    NA   103
```

Remember that `NA` indicates a missing value in R, in this case because no observations were found for some of these cross-sections.

Not many cars have three- or five-cylinder engines, and *only* American cars have eight cylinders. It's common when analyzing data to have *imbalanced* datasets where there is a disproportionate number of observations in some levels. Special techniques are often needed to model such data. To learn more about working with imbalanced data, check out *Practical Statistics for Data Scientists*, 2nd edition by Peter Bruce et al. (O'Reilly).

We can also find the descriptive statistics for each level of *origin*. First, we'll use `select()` to choose the variables of interest, then we can use `psych`'s `describeBy()` function, setting `groupBy` to `origin`:

```
mpg %>%
  select(mpg, origin) %>%
  describeBy(group = 'origin')

#>  Descriptive statistics by group
#> origin: Asia
        vars  n  mean   sd median trimmed  mad min  max range
#> mpg        1 79 30.45 6.09   31.6   30.47 6.52  18 46.6  28.6
#> origin*    2 79  1.00 0.00    1.0    1.00 0.00   1  1.0   0.0
        skew kurtosis   se
#> mpg     0.01    -0.39 0.69
#> origin*  NaN      NaN 0.00

#> origin: Europe
        vars  n mean   sd median trimmed  mad  min  max range
#> mpg        1 68 27.6 6.58     26    27.1 5.78 16.2 44.3  28.1
#> origin*    2 68  1.0 0.00      1     1.0 0.00  1.0  1.0   0.0
        skew kurtosis  se
#> mpg     0.73     0.31 0.8
#> origin*  NaN      NaN 0.0

#> origin: USA
        vars   n  mean   sd median trimmed  mad min max range
#> mpg        1 245 20.03 6.44   18.5   19.37 6.67   9  39    30
#> origin*    2 245  1.00 0.00    1.0    1.00 0.00   1   1     0
        skew kurtosis   se
#> mpg     0.83     0.03 0.41
#> origin*  NaN      NaN 0.00
```

Let's learn more about the potential relationship between *origin* and *mpg*. We'll get started by visualizing the distribution of *mpg* with a histogram, which is shown in Figure 9-1:

```
ggplot(data = mpg, aes(x = mpg)) +
  geom_histogram()
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

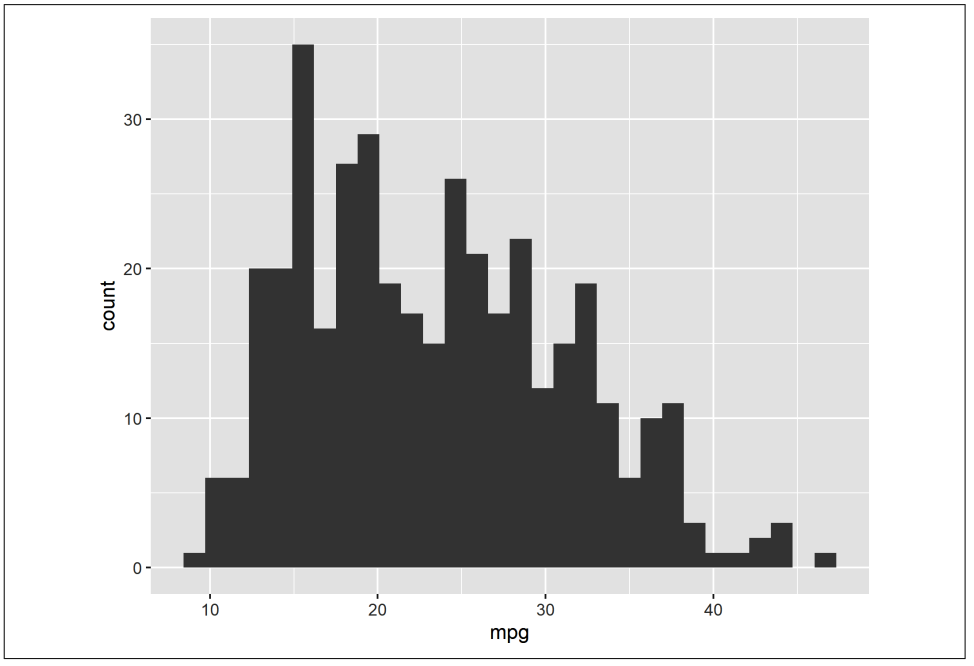

Figure 9-1. Distribution of mpg

We can now hone in on visualizing the distribution of *mpg* by *origin*. Overlaying all three levels of *origin* on one histogram could get cluttered, so a boxplot like what's shown in Figure 9-2 may be a better fit:

```
ggplot(data = mpg, aes(x = origin, y = mpg)) +
  geom_boxplot()
```

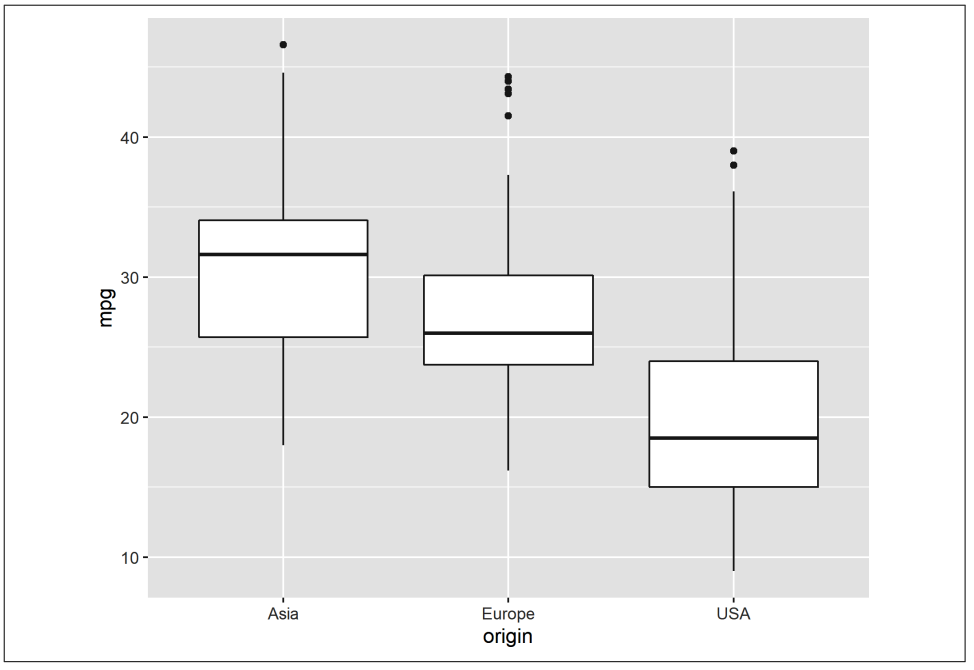

Figure 9-2. Distribution of mpg by origin

If we'd rather visualize these as histograms, and not make a mess, we can do so in R with a *facet* plot. Use `facet_wrap()` to split the `ggplot2` plot into subplots, or *facets*. We'll start with a ~, or tilde operator, followed by the variable name. When you see the tilde used in R, think of it as the word "by." For example, here we are faceting a histogram by `origin`, which results in the histograms shown in Figure 9-3:

```
# Histogram of mpg, facted by origin
ggplot(data = mpg, aes(x = mpg)) +
  geom_histogram() +
  facet_grid(~ origin)
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

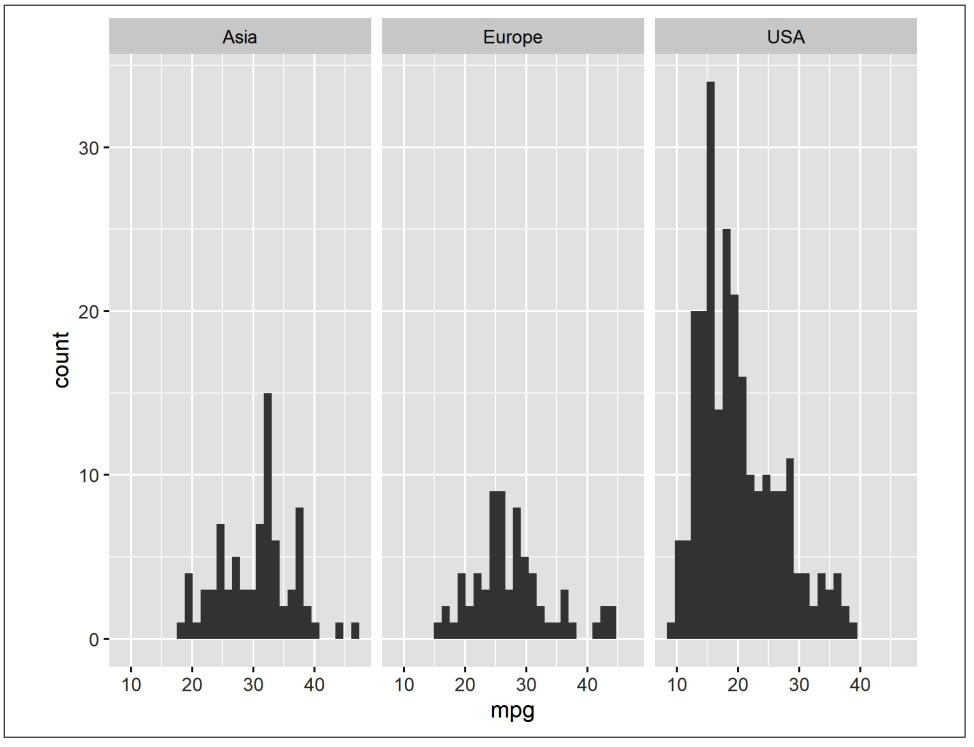

Figure 9-3. Distribution of mpg by origin

Hypothesis Testing

You could continue to explore the data using these methods, but let's move into hypothesis testing. In particular, I would like to know whether there is a significant difference in mileage between American and European cars. Let's create a new data frame containing just these observations; we'll use it to conduct a t-test.

```
mpg_filtered <- filter(mpg, origin=='USA' | origin=='Europe')
```

Testing Relationships Across Multiple Groups

We could indeed use hypothesis testing to look for a difference in mileage across American, European, and Asian cars; this is a different statistical test called *analysis of variance*, or ANOVA. It's worth exploring next on your analytics journey.

Independent Samples t-test

R includes a `t.test()` function out of the box: we need to specify where our data comes from with the `data` argument, and we'll also need to specify what *formula* to test. To do that, we'll set the relationship between independent and dependent variables with the ~ operator. The dependent variable comes in front of the ~, with independent variables following. Again, you interpret this notation as analyzing the effect of `mpg` "by" `origin`.

```
# Dependent variable ~ ("by") independent variable
t.test(mpg ~ origin, data = mpg_filtered)
#>  Welch Two Sample t-test
#>
#>     data:  mpg by origin
#>     t = 8.4311, df = 105.32, p-value = 1.93e-13
#>     alternative hypothesis: true difference in means is not equal to 0
#>     95 percent confidence interval:
#>     5.789361 9.349583
#>     sample estimates:
#>     mean in group Europe    mean in group USA
#>                 27.60294             20.03347
```

Isn't it great that R even explicitly states what our alternative hypothesis is, *and* includes the confidence interval along with the p-value? (You can tell this program was built for statistical analysis.) Based on the p-value, we will reject the null; there does appear to be evidence of a difference in means.

Let's now turn our attention to relationships between continuous variables. First, we'll use the `cor()` function from base R to print a correlation matrix. We'll do this only for the continuous variables in *mpg*:

```
select(mpg, mpg:horsepower) %>%
  cor()
#>                    mpg     weight horsepower
#> mpg         1.0000000 -0.8322442 -0.7784268
#> weight     -0.8322442  1.0000000  0.8645377
#> horsepower -0.7784268  0.8645377  1.0000000
```

We can use `ggplot2` to visualize, for example, the relationship between weight and mileage, as in Figure 9-4:

```
ggplot(data = mpg, aes(x = weight,y = mpg)) +
  geom_point() + xlab('weight (pounds)') +
  ylab('mileage (mpg)') + ggtitle('Relationship between weight and mileage')
```

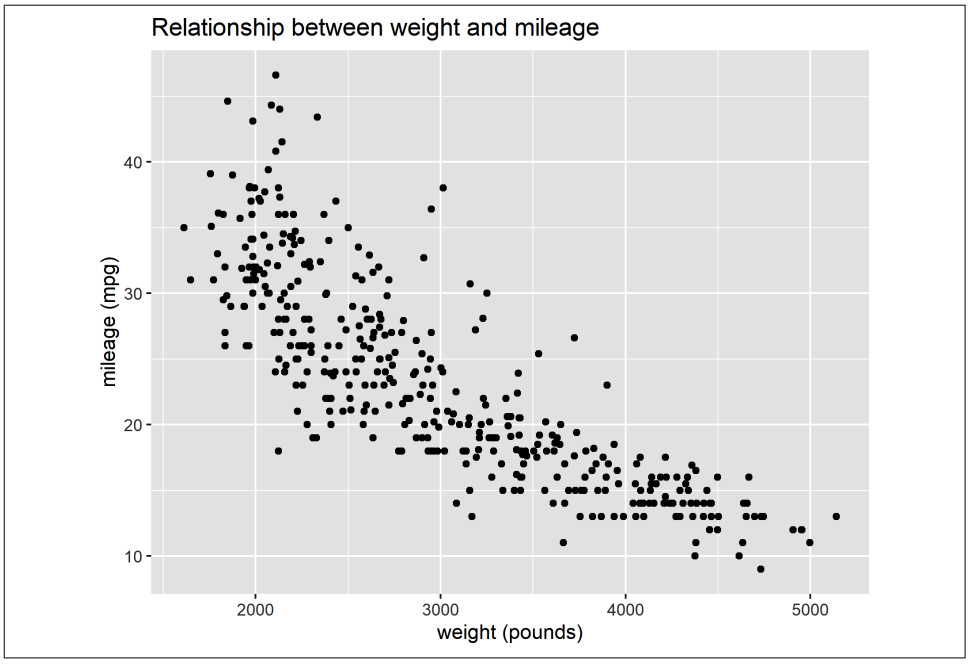

Figure 9-4. Scatterplot of weight by mpg

Alternatively, we could use the `pairs()` function from base R to produce a pairplot of all combinations of variables, laid out similarly to a correlation matrix. Figure 9-5 is a pairplot of selected variables from *mpg*:

```
select(mpg, mpg:horsepower) %>%
  pairs()
```

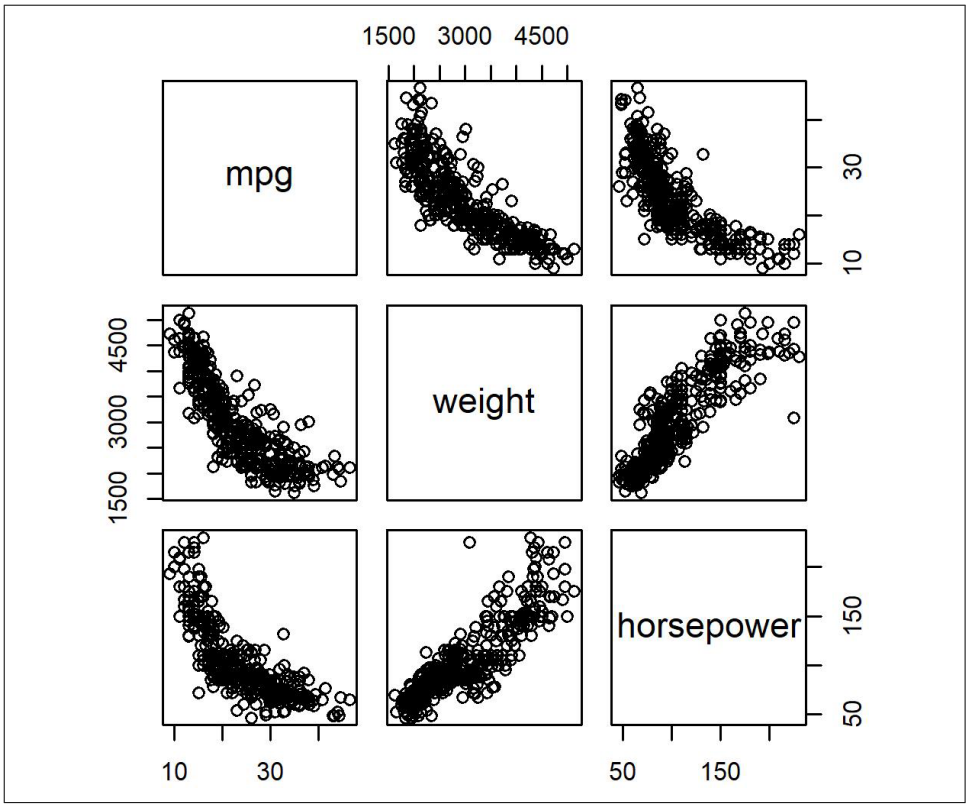

Figure 9-5. Pairplot

Linear Regression

We're ready now for linear regression, using base R's `lm()` function (this is short for *linear model*). Similar to `t.test()`, we will specify a dataset and a formula. Linear regression returns a fair amount more output than a t-test, so it's common to assign the results to a new object in R first, then explore its various elements separately. In particular, the `summary()` function provides a helpful overview of the regression model:

```
mpg_regression <- lm(mpg ~ weight, data = mpg)
summary(mpg_regression)

#>     Call:
#>     lm(formula = mpg ~ weight, data = mpg)
#>
#>     Residuals:
#>         Min       1Q   Median       3Q      Max
#>     -11.9736  -2.7556  -0.3358   2.1379  16.5194
#>
```

```
#>     Coefficients:
#>                   Estimate Std. Error t value Pr(>|t|)
#>     (Intercept) 46.216524   0.798673   57.87   <2e-16 ***
#>     weight      -0.007647   0.000258  -29.64   <2e-16 ***
#>     ---
#>     Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#>
#>     Residual standard error: 4.333 on 390 degrees of freedom
#>     Multiple R-squared:  0.6926,     Adjusted R-squared:  0.6918
#>     F-statistic: 878.8 on 1 and 390 DF,  p-value: < 2.2e-16
```

This output should look familiar. Here you'll see the coefficients, p-values, and R-squared, among other figures. Again, there does appear to be a significant influence of weight on mileage.

Last but not least, we can fit this regression line over the scatterplot by including `geom_smooth()` in our `ggplot()` function, setting `method` to `lm`. This results in Figure 9-6:

```
ggplot(data = mpg, aes(x = weight, y = mpg)) +
  geom_point() + xlab('weight (pounds)') +
  ylab('mileage (mpg)') + ggtitle('Relationship between weight and mileage') +
  geom_smooth(method = lm)
#> `geom_smooth()` using formula 'y ~ x'
```

Figure 9-6. Scatterplot with fit regression line of weight by mpg

Confidence Intervals and Linear Regression

Take a look at the shaded area along the fit line in Figure 9-6. This is the confidence interval of the regression slope, indicating with 95% confidence where we believe the true population estimate might be for each value of *x*.

Train/Test Split and Validation

Chapter 5 briefly reviewed how machine learning relates to working with data more broadly. A technique popularized by machine learning that you may encounter in your data analytics work is the *train/test split*. The idea here is to *train* the model on a subset of your data, then *test* it on another subset. This provides assurance that the model doesn't just work on one particular sampling of observations, but can generalize to the wider population. Data scientists are often especially interested in how well the model does at making predictions on the testing data.

Let's split our *mpg* dataset in R, train the linear regression model on part of the data, and then test it on the remainder. To do so, we'll use the `tidymodels` package. While not part of the `tidyverse`, this package is built along the same principles and thus works well with it.

You may remember in Chapter 2 that, because we were using random numbers, the results you saw in your workbook were different than what was documented in the book. Because we'll again be splitting our dataset randomly here, we could encounter that same problem. To avoid that, we can set the *seed* of R's random number generator, which results in the same series of random numbers being generated each time. This can be done with the `set.seed()` function. You can set it to any number; `1234` is common:

```
set.seed(1234)
```

To begin the split, we can use the aptly named `initial_split()` function; from there, we'll subset our data into training and testing datasets with the `training()` and `testing()` functions, respectively.

```
mpg_split <- initial_split(mpg)
mpg_train <- training(mpg_split)
mpg_test <- testing(mpg_split)
```

By default, `tidymodels` splits the data's observations into two groups at random: 75% of the observations went to the training group, the remainder to the test. We can confirm that with the `dim()` function from base R to get the number of rows and columns in each dataset, respectively:

```
dim(mpg_train)
#> [1] 294   5
dim(mpg_test)
#> [1] 98  5
```

At 294 and 98 observations, our training and testing sample sizes should be sufficiently large for reflective statistical inference. While it's not often a consideration for the massive datasets used in machine learning, adequate sample size can be a limitation when splitting data.

It's possible to split the data into other proportions than 75/25, to use special techniques for splitting the data, and so forth. For more information, check the `tidymodels` documentation; until you become more comfortable with regression analysis, the defaults are fine.

To build our training model, we'll first *specify* what type of model it is with the `linear_reg()` function, then *fit* it. The inputs of the `fit()` function should look familiar to you, except this time we are using the training subset of *mpg* only.

```
# Specify what kind of model this is
lm_spec <- linear_reg()

# Fit the model to the data
lm_fit <- lm_spec %>%
  fit(mpg ~ weight, data = mpg_train)
#> Warning message:
#> Engine set to `lm`.
```

You will see from your console output that the `lm()` function from base R, which you've used before, was used as the *engine* to fit the model.

We can get the coefficients and p-values of our training model with the `tidy()` function, and its performance metrics (such as R-squared) with `glance()`.

```
tidy(lm_fit)
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic   p.value
#>   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
#> 1 (Intercept) 47.3      0.894         52.9 1.37e-151
#> 2 weight      -0.00795  0.000290     -27.5 6.84e- 83
#>
glance(lm_fit)
#> # A tibble: 1 x 12
#>   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC
#>       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl>
#> 1     0.721         0.720  4.23      754. 6.84e-83     1  -840. 1687.
#> # ... with 4 more variables: BIC <dbl>, deviance <dbl>,
#> #   df.residual <int>, nobs <int>
```

This is great, but what we *really* want to know is how well this model performs when we apply it to a new dataset; this is where the test split comes in. To make predictions

on `mpg_test`, we'll use the `predict()` function. I will also use `bind_cols()` to add the column of predicted Y-values to the data frame. This column by default will be called `.pred`.

```
mpg_results <- predict(lm_fit, new_data = mpg_test) %>%
  bind_cols(mpg_test)

mpg_results
#> # A tibble: 98 x 6
#>    .pred   mpg weight horsepower origin cylinders
#>    <dbl> <dbl>  <dbl>      <dbl> <chr>      <dbl>
#>  1  20.0    16   3433        150 USA            8
#>  2  16.7    15   3850        190 USA            8
#>  3  25.2    18   2774         97 USA            6
#>  4  30.3    27   2130         88 Asia           4
#>  5  28.0    24   2430         90 Europe         4
#>  6  21.0    19   3302         88 USA            6
#>  7  14.2    14   4154        153 USA            8
#>  8  14.7    14   4096        150 USA            8
#>  9  29.6    23   2220         86 USA            4
#> 10  29.2    24   2278         95 Asia           4
#> # ... with 88 more rows
```

Now that we've applied the model to this new data, let's evaluate its performance. We can, for example, find its R-squared with the `rsq()` function. From our `mpg_results` data frame, we'll need to specify which column contains the actual Y values with the `truth` argument, and which are predictions with the `estimate` column.

```
rsq(data = mpg_results, truth = mpg, estimate = .pred)
#> # A tibble: 1 x 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 rsq     standard       0.606
```

At an R-squared of 60.6%, the model derived from the training dataset explains a fair amount of variability in the testing data.

Another common evaluation metric is the root mean square error (RMSE). You learned about the concept of *residuals* in Chapter 4 as the difference between actual and predicted values; RMSE is the standard deviation of the residuals and thus an estimate of how spread errors tend to be. The `rmse()` function returns the RMSE.

```
rmse(data = mpg_results, truth = mpg, estimate = .pred)
#> # A tibble: 1 x 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 rmse    standard        4.65
```

Because it's relative to the scale of the dependent variable, there's no one-size-fits-all way to evaluate RMSE, but between two competing models using the same data, a smaller RMSE is preferred.

`tidymodels` makes numerous techniques available for fitting and evaluating models in R. We've looked at a regression model, which takes a continuous dependent variable, but it's also possible to build *classification* models, where the dependent variable is categorical. This package is a relative newcomer to R, so there is somewhat less literature available, but expect more to come as the package grows in popularity.

Conclusion

There is, of course, much more you could do to explore and test the relationships in this and other datasets, but the steps we've taken here serve as a solid opening. Earlier, you were able to conduct and interpret this work in Excel, and now you've leaped into doing it in R.

Exercises

Take a moment to try your hand at analyzing a familiar dataset with familiar steps, now using R. At the end of Chapter 4, you practiced analyzing data from the `ais` dataset in the book repository (*https://oreil.ly/egOx1*). This data is available in the R package `DAAG`; try installing and loading it from there (it is available as the object `ais`). Do the following:

1. Visualize the distribution of red blood cell count (*rcc*) by sex (*sex*).
2. Is there a significant difference in red blood cell count between the two groups of sex?
3. Produce a correlation matrix of the relevant variables in this dataset.
4. Visualize the relationship of height (*ht*) and weight (*wt*).
5. Regress *ht* on *wt*. Find the equation of the fit regression line. Is there a significant relationship? What percentage of the variance in *ht* is explained by *wt*?
6. Split your regression model into training and testing subsets. What is the R-squared and RMSE on your test model?

PART III

From Excel to Python