An Introduction to Statistical Learning

Chapter 1

Notes

Introduction

In statistical learning, we often wish to predict an outcome based on a set of features. We have a *training set of data*, in which we observe the outcome and measurements for a set of objects. Using this data, we build a prediction model, or **learner**, which will enable us to predict the outcome for new, unseen objects. Predicting in the presence of the outcome variable is called **supervised learning**, which is judged based on how accurately the predictor predicts outcomes. In **unsupervised learning**, we observe only the features and have no measurements of the outcome.

Supervised learning can generally be split into two different sets of problems. The **regression problem** is the problem of predicting a continuous quantity output. The **classification problem** is the problem of predicting a discrete class label output. More generally, regression is referenced when we predict quantitative outputs, and classification is referenced when we predict qualitative outputs.

Notation

The mathematical notation takes the following form:

- Vectors of length n appear in lower case bold such as $\mathbf{y}$. Vectors that are not of length n (including scalars) appear in lower case normal font such as a. Matrices appear in capitalized bold such as $\mathbf{X}$. Random variables appear in capitalized normal font such as A (regardless of their dimensions).
- $\mathbf{X}$ denotes the input variable(s). Thus, $\mathbf{X} \in \mathbb{R}^{n \times p}$ is a matrix with n rows (or observations) and p columns (or variables).
- The jth column of matrix $\mathbf{X}$ is denoted as $\mathbf{x}_j$ such that $\mathbf{x}_j \in \mathbb{R}^n$ and

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_p \\ | & | & & | \end{bmatrix}$$

- The ith row of matrix $\mathbf{X}$ is denoted as x_i^T such that $x_i \in \mathbb{R}^p$ and

$$\mathbf{X} = \begin{bmatrix} - & x_1^T & - \\ - & x_2^T & - \\ & \vdots & \\ - & x_n^T & - \end{bmatrix}$$

- y_i denotes the ith observation of the output variable. Hence, the set of all n observations can be written in vector form as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Recall from matrix algebra that the ith row and jth column of the product of $\mathbf{A} \in \mathbb{R}^{r \times d}$ and $\mathbf{B} \in \mathbb{R}^{d \times s}$ equates to

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{d} a_{ik} b_{kj}$$

Chapter 2

Notes

Types of Variables

In statistical learning, **input variables** are used to build predictive methods and are generally denoted using the symbol X, or X_j when referring to the jth input variable. The inputs go by different names, such as **predictors**, **independent variables**, **features** or sometimes just variables. On the other hand, **output variables** are generally denoted by Y and also commonly referred to as **responses** or **dependent variables**.

In total, if we observe a quantitative response Y and p different predictors, $X_1, X_2, \ldots, X_p$, we assume that there is some relationship between Y and $X = (X_1, X_2, \ldots, X_p)$, which can be generally written as

$$Y = f(X) + \epsilon,$$

where f is some fixed (but unknown) function of X, and ϵ is a random **error term**, which is independent of X and has a zero mean. Thus, f represents the **systematic** information that X provides about Y. Since f is unknown, we often wish to estimate f. Overall, statistical learning refers to a set of approaches for estimating f in hopes of achieving accurate **prediction** and/or **inference**.

Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. In this setting, since the error term averages to zero, we can predict Y using

$$\hat{Y} = \hat{f}(X),$$

where $\hat{Y}$ represents the prediction of Y and $\hat{f}$ represents our estimate for f and is often treated as a **blackbox** since we're typically unconcerned with the form of $\hat{f}$ but rather how accurately it predicts Y. The accuracy of $\hat{Y}$ as a prediction for Y depends on two quantities, which we will refer to as the **reducible error** and the **irreducible error**. Reducible error refers to error that can be reduced by using the most appropriate statistical learning technique to estimate f. Irreducible error refers to error that is irreducible since Y is also a function of ϵ, which cannot be predicted using X. If we suppose that both $\hat{f}$ and X are fixed, then

$$\begin{aligned}
E(Y - \hat{Y})^2 &= E\left[(f(X) + \epsilon - \hat{f}(X))^2\right] \\
&= E\left[(f(X) + \epsilon - \hat{f}(X))^2\right] \\
&= E\left[f(X)^2 + \hat{f}(X)^2 + \epsilon^2 + 2\epsilon f(X) - 2\epsilon\hat{f}(X) - 2f(X)\hat{f}(X)\right] \\
&= E\left[f(X)^2 - 2f(X)\hat{f}(X) + \hat{f}(X)^2\right] + E\left[\epsilon^2\right] + E\left[2\epsilon f(X)\right] - E\left[2\epsilon\hat{f}(X)\right] \\
&= E\left[(f(X) - \hat{f}(X))^2\right] + \left(E\left[\epsilon^2\right] - E\left[\epsilon\right]^2\right) + 2f(X)E\left[\epsilon\right] - 2\hat{f}(X)E\left[\epsilon\right] \\
&= \underbrace{\left[f(X) - \hat{f}(X)\right]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}}
\end{aligned}$$

Inference

In circumstances where we're interested in understanding the association between Y and $X_1, \ldots, X_p$, we are still interested in $\hat{f}$. However, unlike the case with prediction, $\hat{f}$ should no longer be treated as a blackbox since we need to know its exact form. In general, inference allows us to attempt answering whether predictors are associated with the response and get an understanding of the nature and shape of such associations.

Estimating f

In estimating f, we use a set of observations called the **training data** to train, or teach, our method how to estimate f. Broadly speaking, most statistical learning methods can be characterized as **parametric** or **non-parametric** and have the goal of finding a function $\hat{f}$ such that $Y \approx \hat{f}(X)$.

Parametric methods involve a two-step approach in which we make an assumption about the functional form, or shape, of f and then using the training data to fit or train the model. For example, we may use a linear model in which we assume that f is linear (i.e., $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$) and then we may use **ordinary least squares (OLS)** criteria to fit the model. This methodology is called parametric because it reduces the problem of estimating f down to one of estimating a set of parameters (e.g., $\beta_0, \beta_1, \ldots, \beta_p$). Unfortunately, the parametric approach generally will lead to an estimated function that does not match the true unknown form of f. Thus, if the chosen model is too far from the true f, then our estimate will be poor. In attempts to make more flexible models, we generally need to estimate more parameters; however, this can lead to **overfitting** the data, which essentially means our estimators follow the errors, or **noise**, too closely.

Non-parametric methods, on the other hand, do not make explicit assumptions about the functional form of f. Instead they seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly. While non-parametric approaches have the potential to accurately fit a wider range of possible shapes for f, they require far more observations than what's needed for a parametric approach.

Measuring the Quality of Fit

In evaluating the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions match the observed data. In the regression setting, the most commonly-used measure is the **mean squared error (MSE)**, which can be written as

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i) \right)^2,$$

where $\hat{f}(x_i)$ is the predicted value of y_i. This definition of MSE is computed using the training data and is thus more accurately referred to as the **training MSE**. In general, we're more interested in minimizing our **test MSE**, rather than our training MSE. Thus, if we had a large number of test observations, we'd be interested in minimizing the average squared prediction error

$$\text{Ave} \left(y_0 - \hat{f}(x_0) \right)^2,$$

where (x_0, y_0) is a previously unseen test observation not used to train the prediction model. While it may seem that minimizing the test and training MSE should give similar results, in many applications, the training set MSE can be quite small, but the test MSE is often much larger. In general, the training MSE decreases monotonically with fewer **degrees of freedom**, which is a quantity that summarizes the flexibility of a curve. On the other hand, the test MSE displays a U-shape relationship with flexibility. A fundamental property of statistical learning is that as model flexibility increases, training MSE will decrease, but the test MSE may not. When a given method yields a small training MSE but a large test MSE, we are said to be overfitting the data because our statistical learning procedure is working too hard to find patterns in the training data and may be picking up some patterns that are just caused by random chance rather than by true properties of the unknown function f. In practice, there are a variety of approaches that can be used to estimate the test MSE. One important method is **cross-validation**, which is a method for estimating test MSE using the training data.

It turns out that the U-shape relationship between flexibility and test MSE is a result of the two competing properties in statistical learning methods: bias and variance. Decomposing the expected test MSE at x_0 [the "expected" is not redundant in this case], we find

$$E \left(y_0 - \hat{f}(x_0) \right)^2 = E \left[y_0^2 - 2 y_0 \hat{f}(x_0) + \hat{f}(x_0)^2 \right]$$

$$= E\left[y_0^2 - 2y_0\hat{f}(x_0) + \hat{f}(x_0)^2\right]$$

$$= E\left[(f(x_0) + \epsilon_0)^2 - 2(f(x_0) + \epsilon_0)\hat{f}(x_0) + \hat{f}(x_0)^2\right]$$

$$= E\left[f(x_0)^2 + 2\epsilon_0 f(x_0) + \epsilon_0^2 - 2f(x_0)\hat{f}(x_0) - 2\epsilon_0\hat{f}(x_0) + \hat{f}(x_0)^2\right]$$

$$= E\left[f(x_0)^2\right] + 2E\left[\epsilon_0 f(x_0)\right] - 2E\left[f(x_0)\hat{f}(x_0)\right] - 2E\left[\epsilon_0\hat{f}(x_0)\right] + E\left[\hat{f}(x_0)^2\right] + E\left[\epsilon_0^2\right]$$

$$= f(x_0)^2 + 2f(x_0)E\left[\epsilon_0\right] - 2f(x_0)E\left[\hat{f}(x_0)\right] - 2E\left[\epsilon_0\hat{f}(x_0)\right] + E\left[\hat{f}(x_0)\right]^2 + \left(E\left[\hat{f}(x_0)^2\right] - E\left[\hat{f}(x_0)\right]^2\right) + \left(E\left[\epsilon_0^2\right] - E\left[\epsilon_0\right]^2\right)$$

$$= \left(E\left[\hat{f}(x_0)\right]^2 - 2f(x_0)E\left[\hat{f}(x_0)\right] + f(x_0)^2\right) - 2E\left[\epsilon_0\hat{f}(x_0)\right] + \mathrm{Var}\left(\hat{f}(x_0)\right) + \mathrm{Var}(\epsilon_0)$$

$$= \left(\left(E\left[\hat{f}(x_0)\right] - f(x_0)\right)^2\right) - 2E\left[\epsilon_0\hat{f}(x_0)\right] + \mathrm{Var}\left(\hat{f}(x_0)\right) + \mathrm{Var}(\epsilon_0)$$

$$= \left[\mathrm{Bias}\left(\hat{f}(x_0)\right)\right]^2 - 2E\left[\epsilon_0\hat{f}(x_0)\right] + \mathrm{Var}\left(\hat{f}(x_0)\right) + \mathrm{Var}(\epsilon_0)$$

$$= \left[\mathrm{Bias}\left(\hat{f}(x_0)\right)\right]^2 + \mathrm{Var}\left(\hat{f}(x_0)\right) + \mathrm{Var}(\epsilon_0)$$

because we assume $E[\epsilon_0\hat{f}(x_0)] = E[\epsilon_0]E[\hat{f}(x_0)] = 0$ [This assumption comes from the fact that ϵ_0 did not contribute to constructing $\hat{f}$ because it is not part of the training set in addition to the assumption that x_0 is independent of ϵ_0 and the observations are independent]. It should be noted that $\mathrm{Var}(\hat{f}(x_0))$ refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. As a general rule, the more flexible the method, the greater the variance will be and the smaller the bias will be.

Many of the same concepts, such as the bias-variance trade-off, transfer over to the classification setting with a few modifications. One of these modifications is how we quantify the accuracy of our estimate $\hat{f}$. The most common approach for this process is the **training error rate**, which is the proportion of mistakes that are made if we apply our estimate $\hat{f}$ to the training observations, written out as

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

Here $\hat{y}_i$ is the predicted class label for the ith observation and $I(y_i \neq \hat{y}_i)$ is an **indicator variable** that equals 1 if $y_i \neq \hat{y}_i$ and 0 if $y_i = \hat{y}_i$. Similarly, the **test error rate** with a set of test observations (x_0, y_0) is given by

$$\mathrm{Ave}\left(I(y_0 \neq \hat{y}_0)\right)$$

Bayes Classifier

In the classification setting, it turns out the test error rate is minimized, on average, by a very simple classifier that assigns each observation to the most likely class, given its predictor values, called the **Bayes classifier**. This means we should assign a test observation with predictor vector x_0 to the class j which maximizes

$$\Pr(Y = j|X = x_0)$$

The Bayes classifier's prediction is determined by the **Bayes decision boundary** and the error rate corresponding to the Bayes classifier is called the **Bayes error rate**. Since the Bayes classifier will always choose the class for which $\Pr(Y = j|X = x_0)$ is largest, the error rate will be

$$1 - \max_j \Pr(Y = j|X = x_0)$$

and the overall Bayes error rate is given by

$$1 - E\left(\max_j \Pr(Y = j|X)\right)$$

where the expectation averages the probability over all possible values of X. The Bayes error rate is analogous to the irreducible error.

In theory, the Bayes classifier is the gold standard. However, for real data, we do not know the conditional distribution of Y given X, and so computing the Bayes classifier is impossible. One approach for estimating the conditional distribution of Y given X is the ***K*-nearest neighbors (KNN)** classifier. The KNN classifier attempts at estimating the condition probability of Y for class j by

$$\frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j),$$

where $\mathcal{N}_0$ is the *neighborhood* for x_0. Generally, we use Euclidean distance to define the neighborhood for x_0, and thus, $\mathcal{N}_0$ is the set of K points in the training data that are closest to x_0. Finally, KNN classifies the test observation x_0 to the class with the largest probability in the form above.

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Variables that are on a large scale will have a much larger effect on the distance between the observations—and hence on the KNN classifier—than variables that are on a small scale. A good way to handle this problem is to standardize the data so that all variables are given a mean of zero and a standard deviation of one.

Just as in the regression setting, there is not a strong relationship between the training error rate and the test error rate. With $K = 1$, the KNN training error rate is 0, but the test error rate may be quite high. In general, as we use more flexible classification methods, the training error rate will decline but the test error rate may not.

Exercises

Conceptual

1. For each of parts (a) through (d), indicate whether we would generally expect the performance of a flexible statistical learning method to be better or worse than an inflexible method. Justify your answer.

   (a) The sample size n is extremely large, and the number of predictors p is small.

   A flexible method should perform better because situations with large sample sizes and small numbers of predictors are not highly susceptible to large amounts of variance. Meanwhile, flexible methods will be less prone to bias than those that are inflexible.

   (b) The number of predictors p is extremely large, and the number of observations n is small.

   An inflexible method will likely perform better because situations with small n and large p (recall the curse of dimensionality) are susceptible to large amounts of variance. A flexible method is prone to the greater variance, which will likely overwhelm any gains we might receive from using flexible methods.

   (c) The relationship between the predictors and response is highly non-linear.

   We would expect a flexible method to be perform better since more flexible methods allow for better estimations of non-linear relationships.

   (d) The variance of the error terms, i.e. $\sigma^2 = \text{Var}(\epsilon)$, is extremely high.

   We would expect an inflexible method to perform better because they are less susceptible to the issue of overfitting. In this circumstance, overfitting would lead to following the large variability in ϵ.

2. Explain whether each scenario is a classification or regression problem, and indicate whether we are most interested in inference or prediction. Finally, provide n and p.

(a) We collect a set of data on the top 500 firms in the US. For each firm we record profit, number of employees, industry and the CEO salary. We are interested in understanding which factors affect CEO salary.

This is a regression problem because the output variable, CEO salary, is quantitative. We are most interested in inference because we are interested in understanding which factors affect CEO salary rather than predicting CEO salaries based on inputs. $n = 500,\ p = 3$

(b) We are considering launching a new product and wish to know whether it will be a success or a failure. We collect data on 20 similar products that were previously launched. For each product we have recorded whether it was a success or failure, price charged for the product, marketing budget, competition price, and ten other variables.

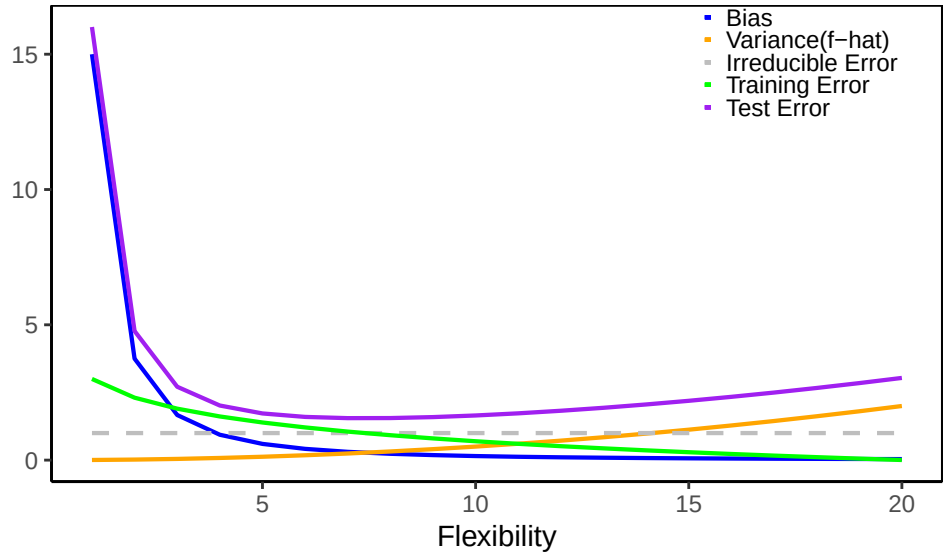
This is a classification problem because the output variable, *success* or *failure*, is qualitative (binary). We are most interested in prediction because we are interested in predicting whether the product will succeed or not. $n = 20,\ p = 13$

(c) We are interested in predicting the % change in the USD/Euro exchange rate in relation to the weekly changes in the world stock markets. Hence we collect weekly data for all of 2012. For each week we record the % change in the USD/Euro, the

This is a regression problem because the output variable, the % change in the USD/Euro exchange rate in relation to the weekly changes in the world stock markets, is quantitative. We are most interested in prediction because we are interested in predicting the % change in the USD/Euro exchange rate in relation to the weekly changes in the world stock markets. $n = 52,\ p = 3$

3. We now revisit the bias-variance decomposition.

(a) Provide a sketch of typical (squared) bias, variance, training error, test error, and Bayes (or irreducible) error curves, on a single plot, as we go from less flexible statistical learning methods towards more flexible approaches. The x-axis should represent the amount of flexibility in the method, and the y-axis should represent the values for each curve. There should be five curves. Make sure to label each one.



(b) Explain why each of the five curves has the shape displayed in part (a).

- Bias (Squared): Bias decreases monotonically with increased flexibility because more flexibility allows us to trace or follow f more precisely.
- Variance: The variance increases with increased flexibility because our method is more prone

to changes in the data set (i.e., $\hat{f}$ is subject to change more to different data when the method is more flexible).

- Bayes (Irreducible) Error: The irreducible error is represented by the horizontal dashed grey line and is not a function of the flexibility.
- Training Error: Training error decreases monotonically with increased flexibility because methods are generally designed to minimize training error, and thus, increasing the flexibility can only improve the training error.
- Test Error: With more flexible methods, the variance will increase and the bias will decrease. The relative rate of change of these two quantities determines whether the test MSE increases or decreases. As we increase the flexibility of a class of methods, the bias tends to initially decrease faster than the variance increases.

4. You will now think of some real-life applications for statistical learning.

   (a) Describe three real-life applications in which classification might be useful. Describe the response, as well as the predictors. Is the goal of each application inference or prediction? Explain your answer.

   - Predicting whether an individual is likely to commit a crime. The response is binary, which takes a value of 1 if the person is more likely to commit a crime than not and 0 otherwise. Some predictors may include age, health, criminal history, childhood factors, wealth, and location of residence. The goal is as an application of prediction as we'd like to predict whether an individual is likely to commit a crime.
   - Understanding what factors influence brand preferences. The response is the brand of choice (e.g., Nike, Adidas, or Under Armour). Some predictors may include wealth, hobbies, location of residence, and occupation. The goal is as an application of inference as we'd like to understand associations between the response and the predictors.
   - Predicting the outcome of a football game. The response is binary and indicates the winning team. Some predictors may include roster salaries, coaching salaries, team records, strengths of schedules, game location, and weather. The goal is as an application of prediction as we'd like to predict which team will win the game.

   (b) Describe three real-life applications in which regression might be useful. Describe the response, as well as the predictors. Is the goal of each application inference or prediction? Explain your answer.

   - Predicting a stock's price. The response is stock price. Some predictors may include CEO salary, return on equity, price to equity ratio, sales, profits, previous stock performance,and macroeconomic factors. The goal is as an application of prediction as we'd like to predict a future stock price.
   - Understanding what factors are associated with wages. The response is wage. Some predictors may include parents' wealth, hobbies, location of residence, education level, and experience. The goal is as an application of inference as we'd like to determine which of the predictors impact wage and how those predictors influence wage.
   - Understanding what factors influence a car's fuel economy. The response is fuel economy (such as miles per gallon). Some predictors may include the drivetrain, the number of cylinders, the weight of the car, the type of injection system, and whether the car is naturally aspirated. The goal is as an application of inference as we'd like to determine which of the predictors impact fuel economy and how those predictors influence a vehicle's fuel efficiency.

   (c) Describe three real-life applications in which cluster analysis might be useful.

   - Clustering individuals to understand similarities in behavior.
   - Filtering spam emails or fake accounts.

- Detecting faulty or poor quality products.

5. What are the advantages and disadvantages of a very flexible (versus a less flexible) approach for regression or classification? Under what circumstances might a more flexible approach be preferred to a less flexible approach? When might a less flexible approach be preferred?

   As discussed earlier, as flexibility of the method increases, the variance will increase and the bias will decrease. The relative rate of change of these two quantities determines whether the test MSE increases or decreases. As we increase the flexibility of a class of methods, the bias tends to initially decrease faster than the variance increases. Some circumstances where a flexible method would be better include cases where the sample size is large, the number of predictors is small, or the true f is non-linear. On the other hand, a less flexible approach might be preferred when dealing with a small sample size, a large number of predictors, or when there is a large amount of variance in the error term.

6. Describe the differences between a parametric and a non-parametric statistical learning approach. What are the advantages of a parametric approach to regression or classification (as opposed to a non-parametric approach)? What are its disadvantages?

   Parametric methods involve approaches in which we make an assumption about the functional form, or shape, of f while non-parametric methods make no such assumption. Parametric methods make estimating f more simple and don't require as large of samples as non-parametric methods; however, they can lead to inaccurate results when our assumption about the functional form of f fails.

7. The table below provides a training data set containing six observations, three predictors, and one qualitative response variable.

   Suppose we wish to use this data set to make a prediction for Y when $X_1 = X_2 = X_3 = 0$ using K-nearest neighbors.

   (a) Compute the Euclidean distance between each observation and the test point, $X_1 = X_2 = X_3 = 0$.

Obs.	X_1	X_2	X_3	Y	Euc. Dis.
1	0	3	0	Red	3
2	2	0	0	Red	2
3	0	1	3	Red	$\sqrt{10}$
4	0	1	2	Green	$\sqrt{5}$
5	-1	0	1	Green	$\sqrt{2}$
6	1	1	1	Red	$\sqrt{3}$

   (b) What is our prediction with $K = 1$? Why?

   Our prediction for Y is green because the 5^{th} observation is closest to the desired point.

   (c) What is our prediction with $K = 3$? Why?

   Our prediction for Y is red because the three observations closest to the desired point are observations 2, 5, and 6. Of these observations, 2/3 of them are red, so we predict red.

   (d) If the Bayes decision boundary in this problem is highly non-linear, then would we expect the best value for K to be large or small? Why?

   We would expect the best value for K to be small because smaller values for K allow for more rigid boundaries that may better account for the non-linearity.

Applied

```
head(College_dt)
```

```
   Private Apps Accept Enroll Top10perc Top25perc F.Undergrad P.Undergrad
1:     Yes 1660   1232    721        23        52        2885         537
2:     Yes 2186   1924    512        16        29        2683        1227
```

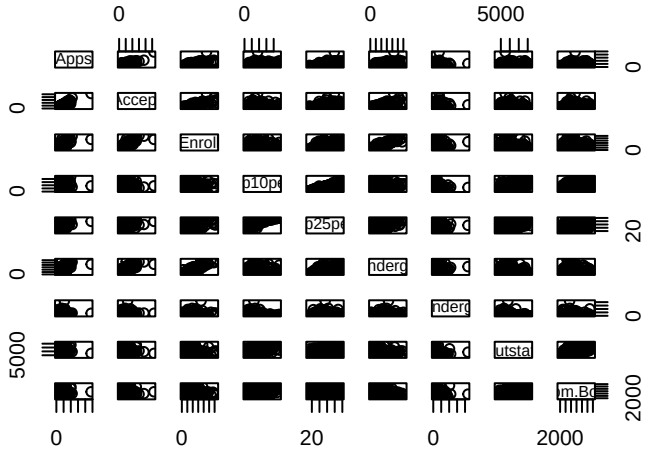
```
3:     Yes 1428   1097    336        22        50        1036          99
4:     Yes  417    349    137        60        89         510          63
5:     Yes  193    146     55        16        44         249         869
6:     Yes  587    479    158        38        62         678          41
   Outstate Room.Board Books Personal PhD Terminal S.F.Ratio perc.alumni Expend
1:     7440       3300   450     2200  70       78      18.1          12   7041
2:    12280       6450   750     1500  29       30      12.2          16  10527
3:    11250       3750   400     1165  53       66      12.9          30   8735
4:    12960       5450   450      875  92       97       7.7          37  19016
5:     7560       4120   800     1500  76       72      11.9           2  10922
6:    13500       3335   500      675  67       73       9.4          11   9727
   Grad.Rate
1:        60
2:        56
3:        54
4:        59
5:        15
6:        55
```

```
summary(College_dt)
```

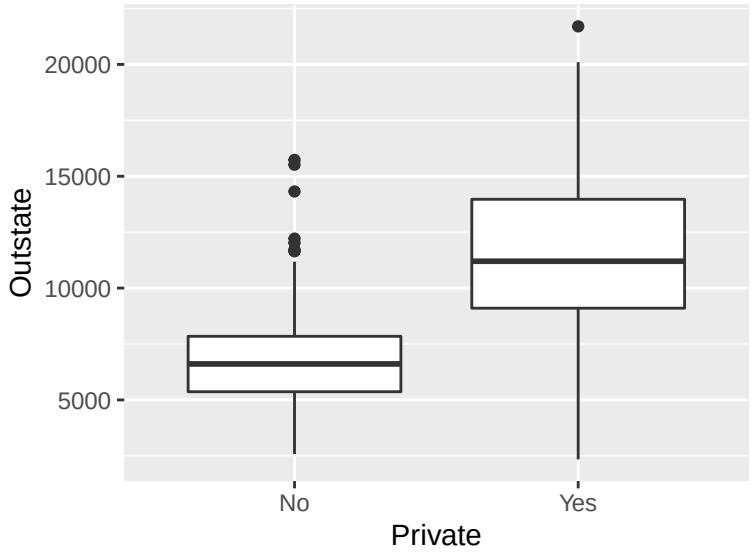
```
 Private        Apps           Accept          Enroll       Top10perc
 No :212   Min.   :   81   Min.   :   72   Min.   :  35   Min.   : 1.00
 Yes:565   1st Qu.:  776   1st Qu.:  604   1st Qu.: 242   1st Qu.:15.00
           Median : 1558   Median : 1110   Median : 434   Median :23.00
           Mean   : 3002   Mean   : 2019   Mean   : 780   Mean   :27.56
           3rd Qu.: 3624   3rd Qu.: 2424   3rd Qu.: 902   3rd Qu.:35.00
           Max.   :48094   Max.   :26330   Max.   :6392   Max.   :96.00
   Top25perc      F.Undergrad     P.Undergrad         Outstate
 Min.   :  9.0   Min.   :  139   Min.   :    1.0   Min.   : 2340
 1st Qu.: 41.0   1st Qu.:  992   1st Qu.:   95.0   1st Qu.: 7320
 Median : 54.0   Median : 1707   Median :  353.0   Median : 9990
 Mean   : 55.8   Mean   : 3700   Mean   :  855.3   Mean   :10441
 3rd Qu.: 69.0   3rd Qu.: 4005   3rd Qu.:  967.0   3rd Qu.:12925
 Max.   :100.0   Max.   :31643   Max.   :21836.0   Max.   :21700
   Room.Board       Books           Personal         PhD
 Min.   :1780   Min.   :  96.0   Min.   : 250   Min.   :  8.00
 1st Qu.:3597   1st Qu.: 470.0   1st Qu.: 850   1st Qu.: 62.00
 Median :4200   Median : 500.0   Median :1200   Median : 75.00
 Mean   :4358   Mean   : 549.4   Mean   :1341   Mean   : 72.66
 3rd Qu.:5050   3rd Qu.: 600.0   3rd Qu.:1700   3rd Qu.: 85.00
 Max.   :8124   Max.   :2340.0   Max.   :6800   Max.   :103.00
    Terminal       S.F.Ratio      perc.alumni        Expend
 Min.   : 24.0   Min.   : 2.50   Min.   : 0.00   Min.   : 3186
 1st Qu.: 71.0   1st Qu.:11.50   1st Qu.:13.00   1st Qu.: 6751
 Median : 82.0   Median :13.60   Median :21.00   Median : 8377
 Mean   : 79.7   Mean   :14.09   Mean   :22.74   Mean   : 9660
 3rd Qu.: 92.0   3rd Qu.:16.50   3rd Qu.:31.00   3rd Qu.:10830
 Max.   :100.0   Max.   :39.80   Max.   :64.00   Max.   :56233
   Grad.Rate
 Min.   : 10.00
 1st Qu.: 53.00
 Median : 65.00
 Mean   : 65.46
 3rd Qu.: 78.00
```

```
 Max.   :118.00
```

```
pairs(ss(College_dt, , 2:10))
```



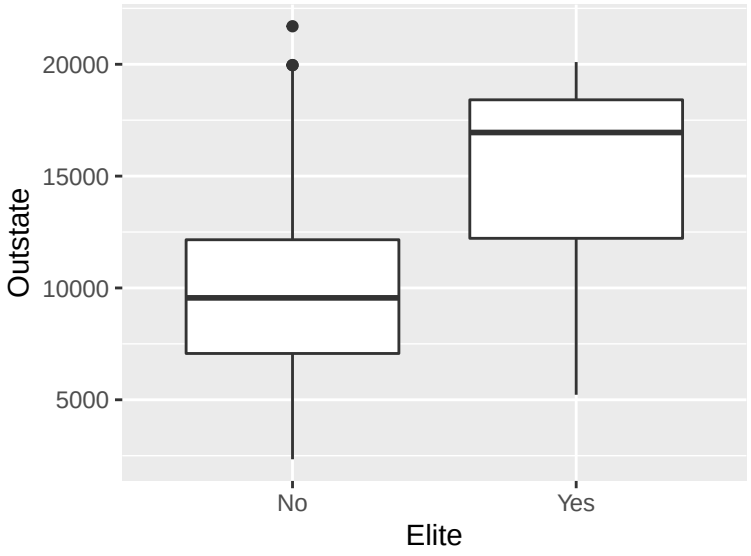

```
ggplot(College_dt, aes(Private, Outstate)) +
  geom_boxplot()
```



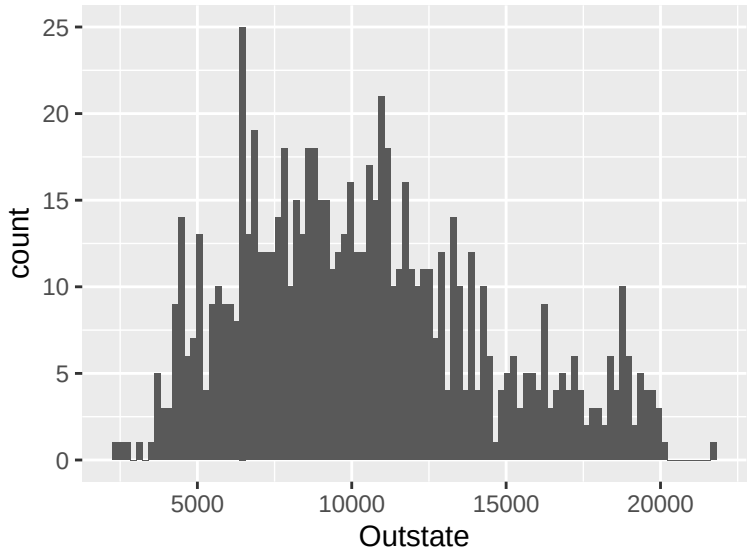

```
College_dt[, Elite := as.factor(fifelse(Top10perc > 50, "Yes", "No"))]

summary(College_dt[["Elite"]])
```

```
 No Yes
699  78
```

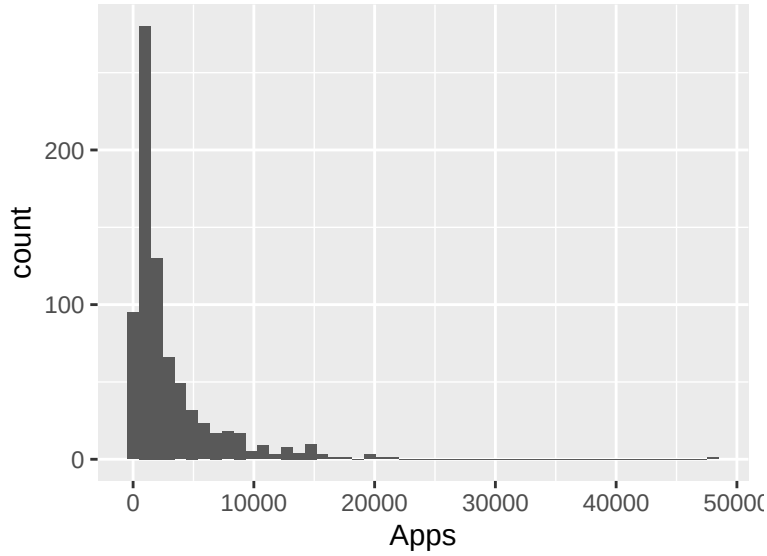
```
ggplot(College_dt, aes(Elite, Outstate)) +
  geom_boxplot()
```



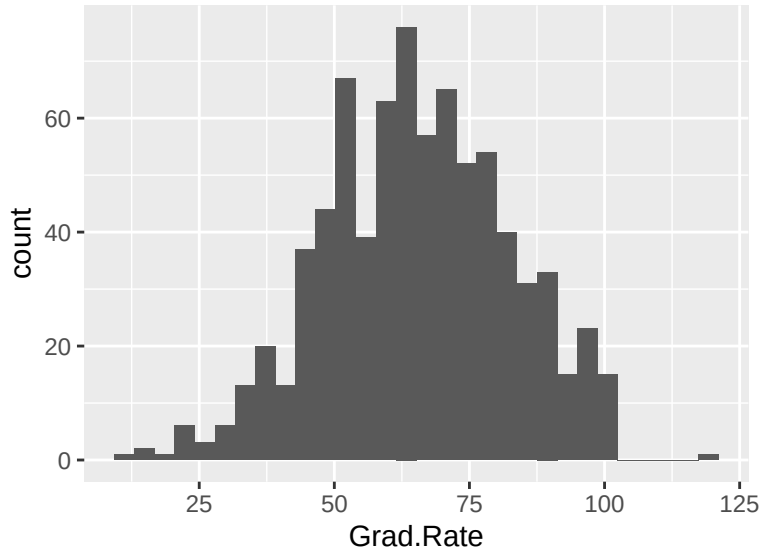

```
ggplot(College_dt, aes(Outstate)) +
  geom_histogram(bins = 100)
```




```
ggplot(College_dt, aes(Apps)) +
  geom_histogram(bins = 50)
```




```
ggplot(College_dt, aes(Grad.Rate)) +
  geom_histogram(bins = 30)
```




```
vapply(ss(Auto_dt, , c(1L, 3L, 4L, 5L, 6L)), range, numeric(2L))
```

```
      mpg displacement horsepower weight acceleration
[1,]  9.0           68         46   1613          8.0
[2,] 46.6          455        230   5140         24.8
```

```
vapply(ss(Auto_dt, , c(1L, 3L, 4L, 5L, 6L)), mean, numeric(1L))
```

```
         mpg displacement   horsepower       weight acceleration
    23.44592    194.41199    104.46939   2977.58418     15.54133
```

```
vapply(ss(Auto_dt, , c(1L, 3L, 4L, 5L, 6L)), sd, numeric(1L))
```

```
         mpg displacement   horsepower       weight acceleration
    7.805007   104.644004    38.491160   849.402560     2.758864
```

```
vapply(ss(Auto_dt, -(10:85), c(1L, 3L, 4L, 5L, 6L)), range, numeric(2L))
```

```
      mpg displacement horsepower weight acceleration
[1,] 11.0           68         46   1649          8.5
[2,] 46.6          455        230   4997         24.8
```

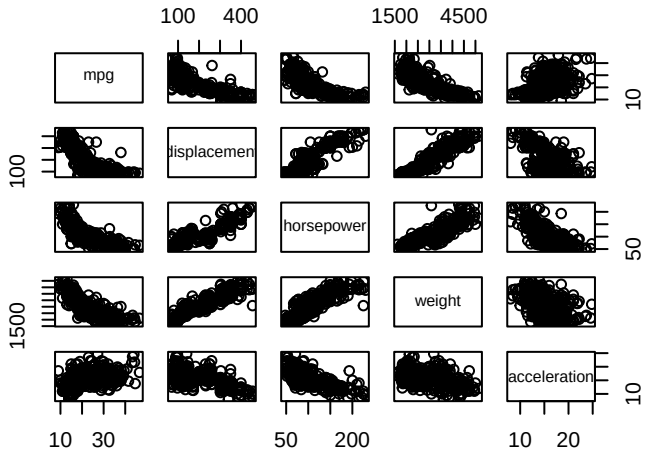
```
vapply(ss(Auto_dt, -(10:85), c(1L, 3L, 4L, 5L, 6L)), mean, numeric(1L))
```

```
         mpg displacement   horsepower       weight acceleration
    24.40443    187.24051    100.72152   2935.97152     15.72690
```

```
vapply(ss(Auto_dt, -(10:85), c(1L, 3L, 4L, 5L, 6L)), sd, numeric(1L))
```

```
         mpg displacement   horsepower       weight acceleration
    7.867283    99.678367    35.708853   811.300208     2.693721
```

```
pairs(ss(Auto_dt, , c(1L, 3L, 4L, 5L, 6L)))
```



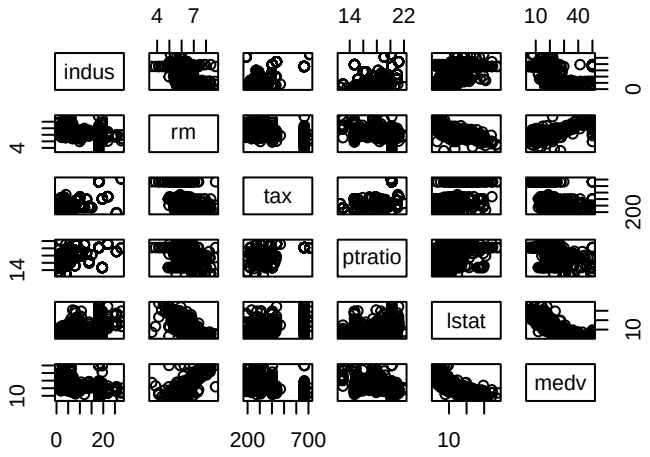

```
fnrow(Boston)
```

```
[1] 506
```

```
fncol(Boston)
```

```
[1] 13
```

```
pairs(ss(Boston, , c(3L, 6L, 10L, 11L, 12L, 13L)))
```




```
cor(Boston[["crim"]], ss(Boston, , -1L))
```

```
             zn     indus        chas       nox         rm       age        dis
[1,] -0.2004692 0.4065834 -0.05589158 0.4209717 -0.2192467 0.3527343 -0.3796701
          rad       tax   ptratio     lstat       medv
[1,] 0.6255051 0.5827643 0.2899456 0.4556215 -0.3883046
```

```
vapply(Boston, range, numeric(2L))
```

```
         crim  zn indus chas   nox    rm   age     dis rad tax ptratio lstat
[1,]  0.00632   0  0.46    0 0.385 3.561   2.9  1.1296   1 187    12.6  1.73
[2,] 88.97620 100 27.74    1 0.871 8.780 100.0 12.1265  24 711    22.0 37.97
     medv
[1,]    5
[2,]   50
```

```
fsum(Boston[["chas"]])
```

```
[1] 35
```

```
fmedian(Boston[["ptratio"]])
```

```
[1] 19.05
```

```
Boston[which.min(Boston[["medv"]]), ]
```

```
       crim zn indus chas   nox    rm age    dis rad tax ptratio lstat medv
399 38.3518  0  18.1    0 0.693 5.453 100 1.4896  24 666    20.2 30.59    5
```

```
fsum(Boston[["rm"]] > 7)
```

```
[1] 64
```

```
fsum(Boston[["rm"]] > 8)
```

```
[1] 13
```

Chapter 3

Notes

Simple Linear Regression

Simple linear regression (SLR) is a simple approach for predicting a quantitative response Y on the basis of a single predictor variable X. It assumes that there is approximately a linear relationship between X and Y such that

$$Y \approx \beta_0 + \beta_1 X$$

β_0 and β_1 are two unknown constants that represent the intercept and slope terms in the linear model. Together, β_0 and β_1 are known as the model **coefficients** or **parameters**. Using training data, we can produce estimators $\hat{\beta}_0$ and $\hat{\beta}_1$ to predict Y on the basis $X = x$. We write this as

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x,$$

where $\hat{y}$ indicates a prediction of Y at $X = x$. While there are a number of ways of estimating the coefficient estimators, the most common approach is the **least squares** criterion. Let $e_i = y_i - \hat{y}_i$ be the **residual** or difference between the observed and predicted values for the ith observation. Least squares criterion chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ such that the **residual sum of squares (RSS)** is minimized. Mathematically,

$$\text{RSS} = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

If we differentiate with respect to $\hat{\beta}_0$ and $\hat{\beta}_1$, we obtain

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Derivation

$$\frac{\partial \text{RRS}}{\partial \hat{\beta}_0} = \sum_{i=1}^{n} -2(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\rightarrow \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\rightarrow n\bar{y} - n\hat{\beta}_0 - n\hat{\beta}_1 \bar{x} = 0$$

$$\rightarrow \bar{y} = \hat{\beta}_0 + \hat{\beta}_1 \bar{x}$$

$$\rightarrow \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Note: The significance of this first order condition is that, using least squares criterion, our line of best fit runs through the sample means $\bar{y}$ and $\bar{x}$.

$$\frac{\partial \text{RRS}}{\partial \hat{\beta}_1} = \sum_{i=1}^{n} -2x_i(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\rightarrow \sum_{i=1}^{n} x_i(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\rightarrow \sum_{i=1}^{n} (x_i y_i - \hat{\beta}_0 x_i - \hat{\beta}_1 x_i^2) = 0$$

$$\rightarrow \sum_{i=1}^{n} x_i y_i = \sum_{i=1}^{n} \left(\hat{\beta}_0 x_i + \hat{\beta}_1 x_i^2 \right)$$

$$\rightarrow \sum_{i=1}^{n} x_i y_i = \sum_{i=1}^{n} \left((\bar{y} - \hat{\beta}_1 \bar{x}) x_i + \hat{\beta}_1 x_i^2 \right)$$

$$\rightarrow \sum_{i=1}^{n} x_i y_i = \sum_{i=1}^{n} \left(\bar{y} x_i - \hat{\beta}_1 \bar{x} x_i + \hat{\beta}_1 x_i^2 \right)$$

$$\rightarrow \sum_{i=1}^{n} x_i y_i = n\bar{x}\bar{y} + \hat{\beta}_1 \sum_{i=1}^{n} (x_i^2 - \bar{x} x_i)$$

$$\rightarrow \sum_{i=1}^{n} (x_i y_i) - n\bar{x}\bar{y} = \hat{\beta}_1 \sum_{i=1}^{n} x_i (x_i - \bar{x})$$

$$\rightarrow \sum_{i=1}^{n} (x_i y_i - \bar{x} y_i) = \hat{\beta}_1 \sum_{i=1}^{n} x_i (x_i - \bar{x})$$

$$\rightarrow \hat{\beta}_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

Recalling that we assume the true functional form between Y and X is $Y = f(X) + \epsilon$, we obtain the **population regression line**

$$Y = \beta_0 + \beta_1 X + \epsilon,$$

where β_0 is the intercept term (or the expected value of Y when $X = 0$) and β_1 is the slope (or the average increase in Y associated with a one-unit increase in X). Plugging our estimators into the population regression line (and omitting the error term), we obtain the **least squares line** $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$.

In general, we hope to find estimators of β_0 and β_1 that are **unbiased**. If we have an estimator $\hat{\theta}$ of a parameter θ, the bias of the estimator is defined as

$$\text{Bias}\left(\hat{\theta}\right) = E\left[\hat{\theta}\right] - \theta$$

Thus, if $\hat{\theta}$ is an unbiased estimator of θ, $\hat{\theta}$ might overestimate θ, and on the basis of another set of observations, $\hat{\theta}$ might underestimate θ. However, if we could average a huge number of estimates of θ obtained from a huge number of sets of observations, then this average would exactly equal θ.

Roughly speaking, the standard error tells us the average amount that an estimator $\hat{\theta}$ differs from the actual value of θ. Applying this framework to $\hat{\beta}_0$ and $\hat{\beta}_1$, we can compute

$$\text{SE}\left(\hat{\beta}_0\right)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \right]$$

$$\text{SE}\left(\hat{\beta}_1\right)^2 = \frac{\sigma^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2},$$

which hold under the assumptions of homoscedasticity such that $\text{Var}(\epsilon) = \text{Var}(\epsilon|X) = \sigma^2$ and random sampling.

Derivation

$$\text{SE}\left(\hat{\beta}_1\right)^2 = \text{Var}\left(\hat{\beta}_1\right)$$

$$= \text{Var}\left(\frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \right)$$

$$
\begin{aligned}
&= \text{Var}\left(\frac{\sum_{i=1}^n y_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\right) \\
&= \text{Var}\left(\frac{\sum_{i=1}^n (\beta_0+\beta_1 x_i+\epsilon_i)(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\right) \\
&= \text{Var}\left(\frac{\beta_0\sum_{i=1}^n (x_i-\bar{x}) + \beta_1\sum_{i=1}^n x_i(x_i-\bar{x}) + \sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\right) \\
&= \text{Var}\left(\frac{\beta_0\cdot 0 + \beta_1\sum_{i=1}^n (x_i-\bar{x})^2 + \sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\right) \\
&= \text{Var}\left(\beta_1 + \frac{\sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\right) \\
&= \left[\frac{1}{\sum_{i=1}^n (x_i-\bar{x})^2}\right]^2 \text{Var}\left(\sum_{i=1}^n \epsilon_i(x_i-\bar{x})\right) \\
&= \left[\frac{1}{\sum_{i=1}^n (x_i-\bar{x})^2}\right]^2 \sum_{i=1}^n \text{Var}\left(\epsilon_i(x_i-\bar{x})\right) \\
&= \left[\frac{1}{\sum_{i=1}^n (x_i-\bar{x})^2}\right]^2 \sum_{i=1}^n (x_i-\bar{x})^2 \text{Var}\left(\epsilon_i\right) \\
&= \left[\frac{1}{\sum_{i=1}^n (x_i-\bar{x})^2}\right]^2 \sum_{i=1}^n (x_i-\bar{x})^2 \sigma^2 \\
&= \frac{\sigma^2}{\left[\sum_{i=1}^n (x_i-\bar{x})^2\right]^2} \sum_{i=1}^n (x_i-\bar{x})^2 \\
&= \frac{\sigma^2}{\sum_{i=1}^n (x_i-\bar{x})^2}
\end{aligned}
$$

$$
\begin{aligned}
\text{SE}\left(\hat{\beta}_0\right)^2 &= \text{Var}\left(\hat{\beta}_0\right) \\
&= \text{Var}\left(\bar{y} - \hat{\beta}_1\bar{x}\right) \\
&= \text{Var}\left(\beta_0 + \beta_1\bar{x} + \bar{\epsilon} - \beta_1\bar{x} + \frac{\sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\bar{x}\right) \\
&= \text{Var}\left(\bar{\epsilon} + \frac{\sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\bar{x}\right) \\
&= \text{Var}\left(\bar{\epsilon}\right) + \text{Var}\left(\frac{\sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\bar{x}\right) + 2\text{Cov}\left(\bar{\epsilon}, \frac{\sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\bar{x}\right) \\
&= \frac{\sigma^2}{n} + \bar{x}^2\text{Var}\left(\frac{\sum_{i=1}^n \epsilon_i(x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\right) + \frac{2\bar{x}}{\sum_{i=1}^n (x_i-\bar{x})^2}\text{Cov}\left(\bar{\epsilon}, \sum_{i=1}^n \epsilon_i(x_i-\bar{x})\right) \\
&= \frac{\sigma^2}{n} + \frac{\bar{x}^2}{\left(\sum_{i=1}^n (x_i-\bar{x})^2\right)^2}\text{Var}\left(\sum_{i=1}^n \epsilon_i(x_i-\bar{x})\right) + \frac{2\bar{x}\sum_{i=1}^n (x_i-\bar{x})}{\sum_{i=1}^n (x_i-\bar{x})^2}\text{Cov}\left(\bar{\epsilon}, \sum_{i=1}^n \epsilon_i\right) \\
&= \frac{\sigma^2}{n} + \frac{\bar{x}^2}{\left(\sum_{i=1}^n (x_i-\bar{x})^2\right)^2}\sum_{i=1}^n \text{Var}\left(\epsilon_i(x_i-\bar{x})\right) + \frac{2\bar{x}}{n\sum_{i=1}^n (x_i-\bar{x})^2}\text{Cov}\left(\sum_{i=1}^n \epsilon_i, \sum_{i=1}^n \epsilon_i(x_i-\bar{x})\right)
\end{aligned}
$$

$$= \frac{\sigma^2}{n} + \frac{\bar{x}^2}{\left(\sum_{i=1}^n (x_i - \bar{x})^2\right)^2} \sum_{i=1}^n (x_i - \bar{x})^2 \text{Var}\left(\epsilon_i\right) + \frac{2\bar{x}}{n\sum_{i=1}^n (x_i - \bar{x})^2} \sum_{i=1}^n (x_i - \bar{x}) \text{Cov}\left(\epsilon_i, \epsilon_i\right)$$

$$= \frac{\sigma^2}{n} + \frac{\bar{x}^2}{\left(\sum_{i=1}^n (x_i - \bar{x})^2\right)^2} \sum_{i=1}^n (x_i - \bar{x})^2 \sigma^2 + \frac{2\bar{x}}{n\sum_{i=1}^n (x_i - \bar{x})^2} \sum_{i=1}^n (x_i - \bar{x}) \sigma^2$$

$$= \frac{\sigma^2}{n} + \frac{\sigma^2 \bar{x}^2}{\left(\sum_{i=1}^n (x_i - \bar{x})^2\right)^2} \sum_{i=1}^n (x_i - \bar{x})^2 + \frac{2\sigma^2 \bar{x}}{n\sum_{i=1}^n (x_i - \bar{x})^2} \sum_{i=1}^n (x_i - \bar{x})$$

$$= \frac{\sigma^2}{n} + \frac{\sigma^2 \bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} + \frac{2\sigma^2 \bar{x}}{n\sum_{i=1}^n (x_i - \bar{x})^2} \cdot 0$$

$$= \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2}\right]$$

Note: It's common to express that these formulas give the *standard deviations* of $\hat{\beta}_0$ and $\hat{\beta}_1$. The standard errors use $\hat{\sigma}^2$ in place of σ^2 since we do not observe the population standard deviation of ϵ. Additionally, these derivations are conditional in the sense that we treat X as non-random.

In practice, we replace σ with the **residual standard error** $\hat{\sigma}$, the square root of an unbiased estimator of σ^2 (however, not unbiased for σ), given by

$$\hat{\sigma} = \text{RSE} = \sqrt{\frac{\sum_{i=1}^n e_i^2}{(n-2)}} = \sqrt{\frac{\text{RSS}}{(n-2)}}$$

In total, we write

$$\hat{\text{SE}}(\hat{\beta}_0)^2 = \frac{\sum_{i=1}^n e_i^2/n(n-2) + \sum_{i=1}^n x_i^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\hat{\text{SE}}(\hat{\beta}_1)^2 = \frac{\sum_{i=1}^n e_i^2/(n-2)}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Standard errors can be used to compute **confidence intervals**. The 95% confidence interval for β_1 takes the approximate form

$$\hat{\beta}_1 \pm 2 \cdot \hat{\text{SE}}(\hat{\beta}_1)$$

and has the property that, if we take repeated samples and construct the confidence interval for each sample, 95% of the intervals will contain the true unknown value of the parameter β_1. Note that this interval only holds when the errors are Gaussian or there is a sufficient sample size to satisfy the asymptotic standard normal distribution.

Standard errors can also be used to perform **hypothesis tests** on the coefficients. The most common hypothesis test involves testing the **null hypothesis** of

$$H_0 : \beta_1 = 0$$

against the **alternative hypothesis**

$$H_a : \beta_1 \neq 0$$

To test the null hypothesis, we need to determine whether $\hat{\beta}_1$ is sufficiently far from the hypothesized value for β_1 (zero in the above example) such that we can confidently reject the null hypothesis in favor of the alternative. To determine whether we can reject the null, we compute a ***t*-statistic** given by

$$t = \frac{\hat{\beta}_1 - b_1}{\hat{\text{SE}}(\hat{\beta}_1)},$$

where b_1 denotes the hypothesized value for β_1 in the null (again, zero in the case above). In practice, we also compute ***p*-values**, which is the probability of committing a type I error, or the probability of observing the results from our analysis given the null hypothesis is true.

Assessing the Accuracy of the Model

Once we compute our estimates for the coefficients of interest and performed hypothesis testing, it is natural to want to quantify the extent to which the model fits the data. The quality of a linear regression fit is typically assessed using the residual standard error (RSE) and the $\boldsymbol{R^2}$ **statistic** or **coefficient of determination**.

Recall that the residual standard error (also commonly called the standard error of regression) is given by

$$\text{RSE} = \sqrt{\frac{\sum_{i=1}^{n} e_i^2}{(n-2)}} = \sqrt{\frac{\text{RSS}}{(n-2)}}$$

and is an estimator of the standard deviation of ϵ. Another way to think about this is that, even if we knew the true values of β_0 and β_1, any prediction of Y on the basis of X would be off by about the RSE on average. Thus, the RSE is considered a measure of the lack of fit of the model.

A more common measure for the goodness-of-fit is the R^2 since it does not depend on the units of Y. The R^2 statistic is the proportion of variance in Y explained by X and is given by the squared correlation coefficient between the actual values of Y and the predicted values $\hat{Y}$. To calculate the R^2, we use the formula(s)

$$R^2 = \frac{\text{ESS}}{\text{TSS}} = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}},$$

where RSS is the residual sum of squares, TSS is the **total sum of squares** given by $\sum_{i=1}^{n}(y_i - \bar{y})^2$, and ESS is the **explained sum of squares** given by $\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$. In total,

$$\text{TSS} = \text{ESS} + \text{RSS}$$

Derivation

$$\begin{aligned}
\text{ESS} + \text{RSS} &= \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\
&= \sum_{i=1}^{n}(\hat{y}_i^2 - 2\bar{y}\hat{y}_i + \bar{y}^2) + \sum_{i=1}^{n}(y_i^2 - 2y_i\hat{y}_i + \hat{y}_i^2) \\
&= \left[\sum_{i=1}^{n}\hat{y}_i^2 - 2\bar{y}\sum_{i=1}^{n}\hat{y}_i + \sum_{i=1}^{n}\bar{y}^2\right] + \left[\sum_{i=1}^{n}y_i^2 - 2\sum_{i=1}^{n}y_i\hat{y}_i + \sum_{i=1}^{n}\hat{y}_i^2\right] \\
&= \left[\sum_{i=1}^{n}y_i^2 - 2n\bar{y}^2 + \sum_{i=1}^{n}\bar{y}^2\right] + \left[\sum_{i=1}^{n}\hat{y}_i^2 - 2\sum_{i=1}^{n}y_i\hat{y}_i + \sum_{i=1}^{n}\hat{y}_i^2\right] \\
&= \left[\sum_{i=1}^{n}y_i^2 - 2\bar{y}\sum_{i=1}^{n}y_i + \sum_{i=1}^{n}\bar{y}^2\right] + \left[\sum_{i=1}^{n}(2\hat{y}_i^2 - 2y_i\hat{y}_i)\right] \\
&= \left[\sum_{i=1}^{n}(y_i^2 - 2\bar{y}y_i + \bar{y}^2)\right] + \left[\sum_{i=1}^{n}2\hat{y}_i(\hat{y}_i - y_i)\right] \\
&= \sum_{i=1}^{n}(y_i - \bar{y})^2 - 2\sum_{i=1}^{n}\hat{y}_i(e_i) \\
&= \text{TSS} - 2\sum_{i=1}^{n}e_i(\hat{\beta}_0 + \hat{\beta}_1 x_i) \\
&= \text{TSS} - 2\left[\hat{\beta}_0\sum_{i=1}^{n}e_i + \hat{\beta}_1\sum_{i=1}^{n}x_i e_i\right] \\
&= \text{TSS}
\end{aligned}$$

Recall that the least squares first order conditions are solved such that $\sum_{i=1}^{n} e_i = 0$ and $\sum_{i=1}^{n} x_i e_i = 0$.

Multiple Linear Regression

We can extend the framework of simple linear regression to **multiple linear regression (MLR)** in which we have more than one regressor. In general, suppose we have p distinct predictors. Then the multiple linear regression model takes the form

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon,$$

where β_j is the ceteris paribus average effect on Y of a one-unit increase in X_j. Just as with SLR, we choose our estimates of $\beta_0, \beta_1, \ldots, \beta_p$ such that we minimize

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Thus, if we form our predictions using

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$$

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p)^2$$

In matrix form, this can be rewritten as

$$\text{RSS} = e^T e = (Y - \mathbf{X}\hat{\beta})^T (Y - \mathbf{X}\hat{\beta}),$$

where $Y \in \mathbb{R}^n$ is a vector of the n observed values for the response variable, $\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$ is a matrix of the n observed values for the p predictors and n ones (for the intercept), and $\hat{\beta} \in \mathbb{R}^{(p+1)}$ is a vector of our $p+1$ estimators (i.e., $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)^T$). Differentiating with respect to $\hat{\beta}$, we obtain

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T Y)$$

Derivation

$$\begin{aligned}
\text{RSS} &= (Y - \mathbf{X}\hat{\beta})^T (Y - \mathbf{X}\hat{\beta}) \\
&= (Y^T - \hat{\beta}^T \mathbf{X}^T)(Y - \mathbf{X}\hat{\beta}) \\
&= Y^T Y - Y^T \mathbf{X}\hat{\beta} - \hat{\beta}^T \mathbf{X}^T Y + \hat{\beta}^T \mathbf{X}^T \mathbf{X}\hat{\beta} \\
\frac{\partial \text{RSS}}{\partial \hat{\beta}} &= \frac{\partial}{\partial \hat{\beta}} \left(Y^T Y - Y^T \mathbf{X}\hat{\beta} - \hat{\beta}^T \mathbf{X}^T Y + \hat{\beta}^T \mathbf{X}^T \mathbf{X}\hat{\beta} \right) = 0 \\
\frac{\partial}{\partial \hat{\beta}} \left(Y^T Y \right) - \frac{\partial}{\partial \hat{\beta}} \left(Y^T \mathbf{X}\hat{\beta} \right) - \frac{\partial}{\partial \hat{\beta}} \left(\hat{\beta}^T \mathbf{X}^T Y \right) + \frac{\partial}{\partial \hat{\beta}} \left(\hat{\beta}^T \mathbf{X}^T \mathbf{X}\hat{\beta} \right) &= 0 \\
0 - \mathbf{X}^T Y - \mathbf{X}^T Y + 2\mathbf{X}^T \mathbf{X}\hat{\beta} &= 0 \\
\mathbf{X}^T \mathbf{X}\hat{\beta} &= \mathbf{X}^T Y \\
\hat{\beta} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T Y
\end{aligned}$$

To show the derivations of the partial derivatives:

$$Y^T \mathbf{X}\hat{\beta} = \begin{bmatrix} y_1 & \cdots & y_n \end{bmatrix} \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \vdots \\ \hat{\beta}_p \end{bmatrix}$$

$$= \begin{bmatrix} y_1 & \cdots & y_n \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 + \hat{\beta}_1 x_{11} + \cdots + \hat{\beta}_p x_{1p} \\ \vdots \\ \hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \cdots + \hat{\beta}_p x_{np} \end{bmatrix}$$

$$= \left(\hat{\beta}_0 y_1 + \hat{\beta}_1 x_{11} y_1 + \cdots + \hat{\beta}_p x_{1p} y_1\right) + \cdots + \left(\hat{\beta}_0 y_n + \hat{\beta}_1 x_{n1} y_n + \cdots + \hat{\beta}_p x_{np} y_n\right)$$

$$\frac{\partial Y^T \mathbf{X}\hat{\beta}}{\partial \hat{\beta}} = \begin{bmatrix} \frac{\partial Y^T \mathbf{X}\hat{\beta}}{\partial \hat{\beta}_0} \\ \vdots \\ \frac{\partial Y^T \mathbf{X}\hat{\beta}}{\partial \hat{\beta}_p} \end{bmatrix} = \begin{bmatrix} y_1 + \cdots + y_n \\ \vdots \\ x_{1p} y_1 + \cdots + x_{np} y_n \end{bmatrix} = \mathbf{X}^T Y$$

$$\hat{\beta}^T \mathbf{X}^T Y = \begin{bmatrix} \hat{\beta}_0 & \cdots & \hat{\beta}_p \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \\ x_{11} & \cdots & x_{n1} \\ \vdots & \ddots & \vdots \\ x_{1p} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$= \begin{bmatrix} \hat{\beta}_0 & \cdots & \hat{\beta}_p \end{bmatrix} \begin{bmatrix} y_1 + \cdots + y_n \\ \vdots \\ x_{1p} y_1 + \cdots + x_{np} y_n \end{bmatrix}$$

$$= \hat{\beta}_0 (y_1 + \cdots + y_n) + \cdots + \hat{\beta}_p (x_{1p} + \cdots + x_{np} y_n)$$

$$= \left(\hat{\beta}_0 y_1 + \hat{\beta}_1 x_{11} y_1 + \cdots + \hat{\beta}_p x_{1p} y_1\right) + \cdots + \left(\hat{\beta}_0 y_n + \hat{\beta}_1 x_{n1} y_n + \cdots + \hat{\beta}_p x_{np} y_n\right)$$

$$\frac{\partial \hat{\beta}^T \mathbf{X}^T Y}{\partial \hat{\beta}} = \begin{bmatrix} \frac{\partial \hat{\beta}^T \mathbf{X}^T Y}{\partial \hat{\beta}_0} \\ \vdots \\ \frac{\partial \hat{\beta}^T \mathbf{X}^T Y}{\partial \hat{\beta}_p} \end{bmatrix} = \begin{bmatrix} y_1 + \cdots + y_n \\ \vdots \\ x_{1p} y_1 + \cdots + x_{np} y_n \end{bmatrix} = \mathbf{X}^T Y$$

$$\hat{\beta}^T \mathbf{X}^T \mathbf{X} \hat{\beta} = \begin{bmatrix} \hat{\beta}_0 & \cdots & \hat{\beta}_p \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \\ x_{11} & \cdots & x_{n1} \\ \vdots & \ddots & \vdots \\ x_{1p} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \vdots \\ \hat{\beta}_p \end{bmatrix}$$

$$= \begin{bmatrix} \hat{\beta}_0 & \cdots & \hat{\beta}_p \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \\ x_{11} & \cdots & x_{n1} \\ \vdots & \ddots & \vdots \\ x_{1p} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 + \hat{\beta}_1 x_{11} + \cdots + \hat{\beta}_p x_{1p} \\ \vdots \\ \hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \cdots + \hat{\beta}_p x_{np} \end{bmatrix}$$

$$= \begin{bmatrix} \hat{\beta}_0 + \hat{\beta}_1 x_{11} + \cdots + \hat{\beta}_p x_{1p} \\ \vdots \\ \hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \cdots + \hat{\beta}_p x_{np} \end{bmatrix}^T \begin{bmatrix} \hat{\beta}_0 + \hat{\beta}_1 x_{11} + \cdots + \hat{\beta}_p x_{1p} \\ \vdots \\ \hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \cdots + \hat{\beta}_p x_{np} \end{bmatrix}$$

$$= \left(\hat{\beta}_0 + \hat{\beta}_1 x_{11} + \cdots + \hat{\beta}_p x_{1p}\right)^2 + \cdots + \left(\hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \cdots + \hat{\beta}_p x_{np}\right)^2$$

$$\frac{\partial \hat{\beta}^T \mathbf{X}^T \mathbf{X} \hat{\beta}}{\partial \hat{\beta}} = \begin{bmatrix} \frac{\partial \hat{\beta}^T \mathbf{X}^T \mathbf{X} \hat{\beta}}{\partial \hat{\beta}_0} \\ \vdots \\ \frac{\partial \hat{\beta}^T \mathbf{X}^T \mathbf{X} \hat{\beta}}{\partial \hat{\beta}_p} \end{bmatrix}$$

$$= \begin{bmatrix} (\hat{\beta}_0 + \hat{\beta}_1 x_{11} + \cdots + \hat{\beta}_p x_{1p}) + \cdots + (\hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \cdots + \hat{\beta}_p x_{np}) \\ \vdots \\ x_{1p}(\hat{\beta}_0 + \hat{\beta}_1 x_{11} + \cdots + \hat{\beta}_p x_{1p}) + \cdots + x_{np}(\hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \cdots + \hat{\beta}_p x_{np}) \end{bmatrix} = 2\mathbf{X}^T \mathbf{X} \hat{\beta}$$

Similar to the setting of hypothesis testing in simple linear regression, we commonly want to test the null hypothesis that none of the predictors are associated with the response variable. We can write this mathematically as

$$H_0 : \beta_1 = \cdots = \beta_p = 0$$

Likewise, the alternative can be written as

$$H_a : \text{ at least one } \beta_j \text{ is non-zero.}$$

We test this hypothesis using the ***F*-statistic**, which is computed using the formula

$$F = \frac{(\text{TSS} - \text{RSS})/p}{\text{RSS}/(n-p-1)}$$

When H_0 is true and the errors ϵ_i have a normal distribution (or the sample size n is sufficiently large), the F-statistic follows an F-distribution. Thus, we reject the null in favor of the alternative hypothesis if

$$F > \mathcal{F}_{p,n-p-1},$$

where F is our calculated F-statistic and $\mathcal{F}$ represents the critical value derived from F-distribution with $p, n-p-1$ degrees of freedom at the chosen significance level.

The above setting can be applied to more general joint hypotheses. Suppose we wish to test whether the last q predictors are not associated with the response variable. The corresponding null hypothesis can be written as

$$H_0 : \beta_{p-q+1} = \cdots = \beta_p = 0$$

In this case, we fit a second model that uses all the variables except those last q (i.e., we estimate $Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_{p-q}$). Denoting the residual sum of squares for that model as RSS_0, the appropriate F-statistic is

$$F = \frac{(\text{RSS}_0 - \text{RSS})/q}{\text{RSS}/(n-p-1)}$$

When we reject H_0 in favor of H_a, we naturally want to know which of the predictors is associated with the response. The task of determining which predictors are associated with the response in order to fit a single model involving only those predictors is referred to as **variable selection**. Generally, we wish to select the best model available to us, which can be determined by various statistics including **Mallow's C_p**, **Akaike information criterion (AIC)**, **Bayesian information criterion (BIC)**, and **adjusted R^2**. Unfortunately, testing each model available to us is unfeasible as there are 2^p possibilities (not including transformations of the predictors). Thus, we use three approaches for this task:

- **Forward selection**: We begin with the **null model** (the model that contains an intercept but no predictors). We then fit p simple linear regressions and add to the null model the variable that results in the lowest RSS. We then add to that model the variable that results in the lowest RSS for the new two-variable model. This process is continued until some stopping rule is satisfied.
- **Backward selection**: We start with all variables in the model and remove the variable is the least statistically significant. The new $(p-1)$-variable model is fit, and the variable with the largest p-value is removed. This procedure continues until a stopping rule is reached. For instance, we may stop when all remaining variables have a p-value below some threshold.
- **Mixed selection**: We start with no variables in the model and, as with forward selection, add the variable that provides the best fit. We continue to add variables one-by-one. The p-values for variables can become larger as new predictors are added to the model. Hence, if at any point the p-value for one of the variables in the model rises above a certain threshold, then we remove that variable from the model. We continue to perform these forward and backward steps until all variables in the model have a sufficiently low p-value, and all variables outside the model would have a large p-value if added to the model.

Just as we did in the SLR case, we generally use R^2 and RSE to measure how well our model fits the observed data. While R^2 can still be calculated using

$$R^2 = \frac{\text{ESS}}{\text{TSS}} = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}},$$

the RSE takes its more general form

$$\text{RSE} = \sqrt{\frac{\sum_{i=1}^{n} e_i^2}{(n-p-1)}} = \sqrt{\frac{\text{RSS}}{(n-p-1)}}$$

by adjusting for the fewer degrees of freedom.

Prediction

When we would like to predict values of Y using our estimated model, there are three sorts of uncertainty associated with such predictions. The first sort of uncertainty stems from the reducible error due to the fact that $\hat{\beta}_0, \ldots, \hat{\beta}_p$ are estimates of $\beta_0, \ldots, \beta_p$. The second sort of uncertainty stems from **model bias**, which is another source of reducible error but comes from our underlying assumption that $f(X)$ is truly linear. The final sort of uncertainty comes from the error term ϵ or the irreducible error. The irreducible error makes it such that, even if we knew $\beta_0, \ldots, \beta_p$, our predictions remain imperfect. Because of these levels of uncertainty, we use **confidence intervals** and **prediction intervals** to make predictions. More specifically, we use a confidence interval to quantify the uncertainty surrounding the *average* Y given the values of X and a prediction to quantify the uncertainty surrounding a *particular* Y given the values of X. In other words, prediction intervals are always wider than confidence intervals, because they incorporate both the error in the estimate for $f(X)$ (the reducible error) and the uncertainty as to how much an individual point will differ from the population regression plane (the irreducible error).

Regression with Qualitative Predictors

Our framework for multiple regression can also be applied in the case where we have a **factor**, or qualitative predictor. To incorporate factors, we introduce an indicator or **dummy variable** that takes on one if the corresponding value evaluates to true and zero otherwise. We can also use dummy variables in the circumstance where a qualitative predictor has more than two levels. Suppose we have a qualitative predictor x with ℓ levels, then we write our regression equation as

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_{\ell-1} x_{i(\ell-1)} + \epsilon_i,$$

where $x_1, \ldots, x_{i(\ell-1)}$ are $\ell - 1$ dummy variables representing the different levels of the factor. Note that we only use $\ell - 1$ to avoid perfect collinearity. The level with no dummy variable is referred to as the **baseline**. Thus, β_0 is the average y for the baseline, and β_j is the average difference between the baseline and the jth level of the factor.

Non-Linear Models

Two of our assumptions regarding linear models up to this point are:

- The additivity assumption: The association between a predictor X_j and the response Y does not depend on the values of the other predictors.
- The linearity assumption: The change in the response Y associated with a one-unit change in X_j is constant, regardless of the value of X_j.

These assumptions can be relaxed using some common classical approaches for extending the linear model.

To relax the additivity assumption, we often use **interaction terms**. Consider the case where there are two regressors X_1 and X_2. A model constructed using an interaction term would take the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \epsilon,$$

where X_1X_2 is the interaction term between our original predictors X_1 and X_2. Interaction terms are beneficial to our model when there is an interaction effect, meaning the effect of one regressor on the response fluctuates with the relative quantity of another regressor. When using an interaction term, the **hierarchical principle** states that we should always include the variables that make up the interaction term (X_1 and X_2 in the above example), even if the p-values associated with their coefficients are not significant.

Relaxing the linearity assumption is similar to that of the additivity model. In general, we use **polynomial regression** to accommodate non-linear relationships between the regressors and the response variables. Consider the case in which we have one regressor X_1. A model constructed using a **quadratic** would take the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \epsilon,$$

Issues Pertaining to Linear Models

When we fit a linear regression model to a particular data set, many problems may occur. Most common among these are the following:

1. Non-linearity of the Response-Predictor Relationships: If the true relationship between the predictors and the response is non-linear in its parameters, all of the conclusions that we draw from the fit are suspect, and the prediction accuracy of the model can be significantly reduced. **Resiudal plots**, where we plot the residuals e_i against the fitted values $\hat{y}_i$, are a useful graphical tool for identifying non-linearity.

2. Correlation of the Error Terms: An important assumption of the linear regression model is that the error terms, $\epsilon_1, \ldots, \epsilon_n$, are uncorrelated (we also state this as there is no **serial correlation** or **autocorrelation**). This means that knowing the value for ϵ_i provides no information about ϵ_{i+1}. Generally, this assumption is satisfied under the assumption of random sampling, which is commonly used with **cross-sectional data**. However, serial correlation frequently occurs in the context of **time series** data, which consists of observations for which measurements are obtained at discrete points in time. It's often the case that observations obtained at adjacent time points will have positively correlated errors. Since the error terms are unobserved, in practice, we check for **tracking** in the residuals, where adjacent residuals have similar values.

3. Non-constant Variance of Error Terms: Another important assumption of the linear regression model is that the error terms have a constant variance such that $\text{Var}(\epsilon_i) = \text{Var}(\epsilon_i|X) = \sigma^2$ (this is also known as **homoscedasticity**). Cases in which there exist non-constant variances in the errors are said to display **heteroscedasticity**. One remedy to the presence of heteroscedasticity is using **weighted least squares**, where weights given to each observation are proportional to the variances.

4. Outliers: An **outlier** is a point for which y_i is far from the value predicted by the model $\hat{y}_i$. Typically, an outlier that does not have an unusual predictor value has little effect on the least squares fit; however, it can cause other problems such as a significant increase in the RSE. If we believe that an outlier is a result of a data collection error, one solution is to simply remove the observation. However, one should be cautious when performing such actions since an outlier may instead indicate a deficiency with the model.

5. High Leverage Points: Observations classed as **high leverage** are those with an unusual value for x_i. Generally, removing a high leverage observation has a much more substantial impact on the least squares line than removing an outlier (i.e., high leverage observations tend to have a sizable impact on the estimated regression line). In order to quantify an observation's leverage, we compute the leverage statistic given by

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i'=1}^{n}(x_{i'} - \bar{x})^2}$$

   in the SLR case, where an h_i close to one indicates an observation with high leverage.

6. Collinearity: **Collinearity** refers to the situation in which two or more predictor variables are closely related to one another. High but imperfect correlation between two or more independent variables

leads to large values for $\text{Var}\left(\hat{\beta}_j\right)$. Worrying about high degrees of correlation among the independent variables in the sample is really no different from worrying about a small sample size as both work to increase $\text{Var}(\hat{\beta}_j)$. The circumstance in which collinearity exists between three or more variables is called **multicollinearity**. Because it is possible for collinearity to exist between three or more variables even if no pair of variables has a particularly high correlation, we use the **variance inflation factor (VIF)** to assess the level of collinearity in a model. We can compute this value by using the formula

$$\text{VIF}(\hat{\beta}_j) = \frac{1}{1 - R^2_{X_j|X_{-j}}},$$

where $R^2_{X_j|X_{-j}}$ is the R^2 from regressing X_j onto all other predictors.

K-Nearest Neighbors Regression

While linear regression is a *parametric* approach to estimating f (since it assumes f is linear), one of the simplest and best-known *non-parametric* methods is ***K*-nearest neighbors regression (KNN regression)**, which is closely related to the KNN classifier. Given a value for K and a prediction point x_0, KNN regression identifies the K training observations that are closest to x_0, represented by $\mathcal{N}_0$. It then estimates $f(x_0)$ using the average of all the training responses in $\mathcal{N}_0$. Written mathematically,

$$\hat{f}(x_0) = \frac{1}{K} \sum_{x \in \mathcal{N}_0} y_i$$

The optimal value for K will depend on the bias-variance tradeoff. Small values for K provides the most flexible fit, which will tend to experience low amounts of bias and high amounts of variance due to the fact that the prediction in a given region is entirely dependent on just one observation. On the other hand, larger values for K provide a smoother and less variable fit since the prediction is an average of several points, and so changing one observation has a smaller effect. However, the smoothing may contribute additional bias by masking some of the structure in $f(X)$.

In general, a parametric approach will outperform a non-parametric approach if the parametric approach that has been selected is close to the true form of f. This is the case because, if the underlying parametric assumption is close to true, a non-parametric approach incurs a cost in variance that is not offset by a reduction in bias. In a real life situation in which the true relationship is unknown, one might suspect that KNN should be favored over linear regression because it will at worst perform slightly worse than linear regression if the true relationship is linear and may give substantially better results if the true relationship is non-linear. But in reality, even when the true relationship is highly non-linear, KNN may still provide worse results to those provided by linear regression. This is because, in higher dimensions, KNN often performs worse than linear regression, a phenomenon called the **curse of dimensionality**. Roughly speaking for a given sample size, as the number of dimensions p increases, the K observations that are nearest to a given test observation x_0 tend to be further away from x_0, leading to poorer predictions of $f(x_0)$ and hence a poor KNN fit. Thus, it's the case that parametric methods will tend to outperform non-parametric approaches when there is a small number of observations per predictor.

Lab

- The `names()` function can be used to find the names of the pieces of an object (such as an `lm` object). Although we can extract these pieces by name using `$` (e.g., `lm.fit$coefficients`), it is safer to use extractor functions (e.g., `coef(lm.fit)`).
- The command `confint()` can be used to produce confidence intervals for the coefficient estimates.
- The `predict()` function can be used to produce confidence intervals and prediction intervals for Y given a value (or set of values) for X.
- We can use `grid.arrange()` from the package `gridExtra` to group plots together (side-by-side and/or on top of each other).

Alternatively, using R's base graphics, we can use `par(mfrow = )` to achieve the same goal.

- The function `rstudent()` will return studentized residuals.
- Leverage statistics can be computed for any number of predictors using the `hatvalues()` function.
- The `vif()` function, part of the `car` package, can be used to compute variance inflation factors.
- `lm(Y ~ . - X_j)` (where the `X_j` is optional) can be used to regress Y on all of the predictors except for X_j.
- `lm.fit <- update(lm.fit, ~ ...)` can be used to update a model with a new specification.
- We can use `X_1:X_2` in the `lm()` function to include an interaction term. However, it's better to use `X_1 * X_2`, which will include X_1, X_2, and X_1X_2 in the model.
- We can use `I(X^n)` to include X^n in our model. Alternatively, we can use `poly(X, n, raw = TRUE)` to include $X, X^2, \ldots, X^{n-1}, X^n$ in our model.

Exercises

Conceptual

1. Describe the null hypotheses to which the p-values given in Table 3.4 correspond. Explain what conclusions you can draw based on these p-values. Your explanation should be phrased in terms of `sales`, `TV`, `radio`, and `newspaper`, rather than in terms of the coefficients of the linear model.

   Each p-value corresponds to the probability of committing a type I error under the null hypothesis that the corresponding predictor individually has no effect on `sales`. Thus, looking at the p-value for `TV`, we reject the null that `TV` has no effect on `sales` in favor of the alternative at every conventional significance level. The same concept applies for `radio`. On the other hand, the p-value corresponding to `newspaper` does not provide such convincing evidence. Thus, we fail to reject the null hypothesis that `newspaper` has no individual effect on `sales`.

2. Carefully explain the differences between the KNN classifier and KNN regression methods.

   The KNN classifier method is a classification method in which the response variable is qualitative Using the KNN classifier, our prediction for the response variable is that which occurs *most frequently* in the K nearest points to x_0, and thus, it takes on an already observed value. On the other hand, KNN regression is a regression problem in which the response variable is quantitative. Using the KNN regression method, our prediction for the response variable is the *average* of the K nearest points to x_0, and thus, it needn't take on an observed value.

3. Suppose we have a data set with five predictors, $X_1 =$ GPA, $X_2 =$ IQ, $X_3 =$ Level (1 for College and 0 for High School), $X_4 =$ Interaction between GPA and IQ, and $X_5 =$ Interaction between GPA and Level. The response is starting salary after graduation (in thousands of dollars). Suppose we use least squares to fit the model, and get $\hat{\beta}_0 = 50$, $\hat{\beta}_1 = 20$, $\hat{\beta}_2 = 0.07$, $\hat{\beta}_3 = 35$, $\hat{\beta}_4 = 0.01$, $\hat{\beta}_5 = -10$.

   (a) Which answer is correct, and why?

      i. For a fixed value of IQ and GPA, high school graduates earn more, on average, than college graduates.

      ii. For a fixed value of IQ and GPA, college graduates earn more, on average, than high school graduates.

      iii. For a fixed value of IQ and GPA, high school graduates earn more, on average, than college graduates provided that the GPA is high enough.

      iv. For a fixed value of IQ and GPA, college graduates earn more, on average, than high school graduates provided that the GPA is high enough.

We can write the predicted model as

$$\hat{y} = 50 + 20 \cdot \text{GPA} + 0.07 \cdot \text{IQ} + 35 \cdot \text{Level} + 0.01 \cdot \text{GPA} \times \text{IQ} - 10 \cdot \text{GPA} \times \text{Level}$$

If we set Level $= 1$, we obtain

$$\hat{y}(\text{Level} = 1) = 50 + 20 \cdot \text{GPA} + 0.07 \cdot \text{IQ} + 35 + 0.01 \cdot \text{GPA} \times \text{IQ} - 10 \cdot \text{GPA}$$

And, if we set Level $= 0$, we obtain

$$\hat{y}(\text{Level} = 0) = 50 + 20 \cdot \text{GPA} + 0.07 \cdot \text{IQ} + 0.01 \cdot \text{GPA} \times \text{IQ}$$

Thus,

$$\hat{y}(\text{Level} = 1) > \hat{y}(\text{Level} = 0)$$

$$85 + 20 \cdot \text{GPA} + 0.07 \cdot \text{IQ} + 0.01 \cdot \text{GPA} \times \text{IQ} - 10 \cdot \text{GPA} > 50 + 20 \cdot \text{GPA} + 0.07 \cdot \text{IQ} + 0.01 \cdot \text{GPA} \times \text{IQ}$$

$$35 - 10 \cdot \text{GPA} > 0$$

$$\text{GPA} < 3.5$$

Hence, we can conclude that (for our sample) iii. is correct since high school graduates earn more, on average, when GPA > 3.5.

(b) Predict the salary of a college graduate with IQ of 110 and a GPA of 4.0.

$$\hat{y} = 50 + 20(4.0) + 0.07(110) + 35(1) + 0.01(4.0)(110) - 10(4.0)(1) = 137.1$$

or \$137100.

(c) True or false: Since the coefficient for the GPA/IQ interaction term is very small, there is very little evidence of an interaction effect. Justify your answer.

False, without knowing the corresponding standard error, we cannot derive the level of evidence against the null that the interaction term between GPA and IQ has no effect on the response variable. Roughly speaking, if the corresponding standard error is less than 0.005, there actually would be strong evidence against the null.

4. I collect a set of data ($n = 100$ observations) containing a single predictor and a quantitative response. I then fit a linear regression model to the data, as well as a separate cubic regression, i.e. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$.

(a) Suppose that the true relationship between X and Y is linear, i.e. $Y = \beta_0 + \beta_1 X + \epsilon$. Consider the training residual sum of squares (RSS) for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

We would expect the cubic regression model to have a lower training RSS. Including additional regressors in a model monotonically decreases the RSS because our coefficients estimators are chosen to minimize the training RSS. Thus, a model with a subset of the predictors from another model will always have a greater RSS.

(b) Answer (a) using test rather than training RSS.

Changing the context to test RSS, we would now expect the test RSS for the linear regression model to be smaller. This is because the true relationship is linear and thus the cubic regression model will be subject to overfitting. In other words, there is no additional gain in unbiasedness from using the cubic model but additional variance is introduced.

(c) Suppose that the true relationship between X and Y is not linear, but we don't know how far it is from linear. Consider the training RSS for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

The answer to part (a) still applies here. Adding additional predictors to a model only decreases the training RSS (if the same observations are used). Thus, we would always expect the cubic training regression model to have a lower training RSS.

(d) Answer (c) using test rather than training RSS.

Changing the context to test RSS, it would be difficult to tell which should be lower without knowing the extent to which the relationship between Y and X is non-linear. If the relationship between X and Y is almost linear, we would likely expect the test RSS from the linear regression model to be lower. On the other hand, if the relationship between X and Y is very non-linear, we would likely expect the test RSS from the cubic regression model to be lower. In total, if the reduction in bias from using the cubic regression model outweighs any additional variance added to the model, the test RSS from the cubic regression model should be lower. Otherwise, the test RSS from the linear regression model should be lower.

5. Consider the fitted values that result from performing linear regression without an intercept. In this setting, the ith fitted value takes the form

$$\hat{y}_i = x_i\hat{\beta},$$

where

$$\hat{\beta} = \left(\sum_{i=1}^{n} x_i y_i\right) / \left(\sum_{i'=1}^{n} x_{i'}^2\right)$$

Show that we can write

$$\hat{y}_i = \sum_{i'=1}^{n} a_{i'} y_{i'}.$$

What is $a_{i'}$?

Note: We interpret this result by saying that the fitted values from linear regression are linear combinations of the response values.

Extending $\hat{\beta}$ makes it clearer to derive the desired relationship.

$$\begin{aligned}
\hat{\beta} &= \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i'=1}^{n} x_{i'}^2} \\
&= \frac{x_1 y_1 + x_2 y_2 + \cdots + x_n y_n}{\sum_{i'=1}^{n} x_{i'}^2} \\
&= \frac{x_1}{\sum_{i'=1}^{n} x_{i'}^2} y_1 + \frac{x_2}{\sum_{i'=1}^{n} x_{i'}^2} y_2 + \cdots + \frac{x_n}{\sum_{i'=1}^{n} x_{i'}^2} y_n \\
&a_{1'} y_{1'} + a_{2'} y_{2'} + \cdots + a_{n'} y_{n'} \\
&\sum_{i'=1}^{n} a_{i'} y_{i'},
\end{aligned}$$

where $a_{i'} = \frac{x_i}{\sum_{i'=1}^{n} x_{i'}^2}$.

6. Using (3.4), argue that in the case of simple linear regression, the least squares line always passes through the point $(\bar{x}, \bar{y})$.

From (3.4), we have $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$ and $\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$. We only need the first part to show the least squares line always passes through the point $(\bar{x}, \bar{y})$.

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$
$$\rightarrow \bar{y} = \hat{\beta}_0 + \hat{\beta}_1 \bar{x}$$

From here, it's trivial to see that our predicted value for $\hat{y}$ at the point $\bar{x}$ is $\bar{y}$.

7. It is claimed in the text that in the case of simple linear regression of Y onto X, the R^2 statistic (3.17) is equal to the square of the correlation between X and Y (3.18). Prove that this is the case. For simplicity, you may assume that $\bar{y} = \bar{x} = 0$.

$$r_{xy}^2 = \left(\frac{\hat{\sigma}_{xy}}{\sqrt{\hat{\sigma}_x^2 \hat{\sigma}_y^2}} \right)^2$$
$$= \left(\frac{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\left(\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right) \left(\frac{1}{n-1} \sum_{i=1}^n (y_i - \bar{y})^2 \right)}} \right)^2$$
$$= \frac{\left[\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \right]^2}{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}$$
$$= \left[\frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2} \right]^2 \sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2$$
$$= \left[\frac{\hat{\beta}_1}{\sum_{i=1}^n (y_i - \bar{y})^2} \right]^2 \sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2$$
$$= \frac{\sum_{i=1}^n \hat{\beta}_1^2 (x_i - \bar{x})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$
$$= \frac{\sum_{i=1}^n (\hat{\beta}_1 x_i - \hat{\beta}_1 \bar{x})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$
$$= \frac{\sum_{i=1}^n (\hat{\beta}_0 + \hat{\beta}_1 x_i - \hat{\beta}_0 - \hat{\beta}_1 \bar{x})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$
$$= \frac{\sum_{i=1}^n (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$
$$= \frac{\text{ESS}}{\text{TSS}}$$
$$= R^2$$

Applied

```
lm.fit <- lm(mpg ~ horsepower, data = Auto_dt)

summary(lm.fit)
```

```
Call:
lm(formula = mpg ~ horsepower, data = Auto_dt)

Residuals:
     Min       1Q   Median       3Q      Max
```

```
-13.5710  -3.2592  -0.3435   2.7630  16.9240

Coefficients:
             Estimate Std. Error t value            Pr(>|t|)
(Intercept) 39.935861   0.717499   55.66 <0.0000000000000002 ***
horsepower  -0.157845   0.006446  -24.49 <0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.906 on 390 degrees of freedom
Multiple R-squared:  0.6059,    Adjusted R-squared:  0.6049
F-statistic: 599.7 on 1 and 390 DF,  p-value: < 0.00000000000000022
```

```r
predict(lm.fit, data.frame(horsepower = 98))
```

```
       1
24.46708
```

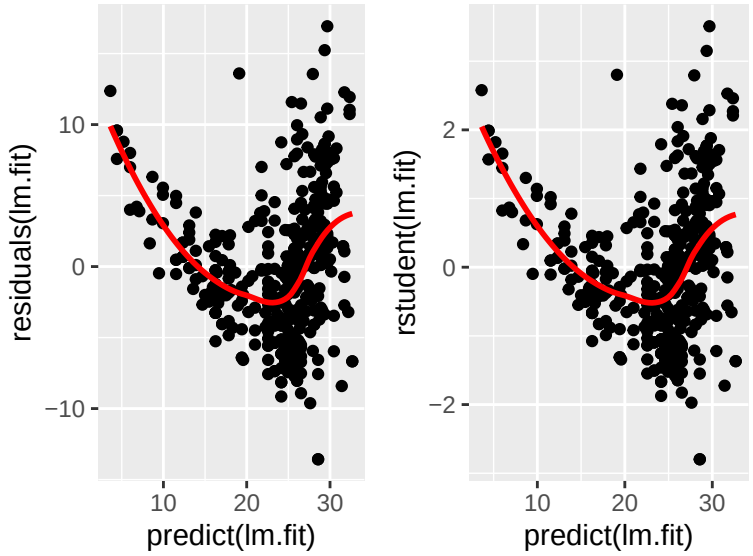
```r
predict(lm.fit, data.frame(horsepower = 98), interval = "confidence")
```

```
       fit      lwr      upr
1 24.46708 23.97308 24.96108
```

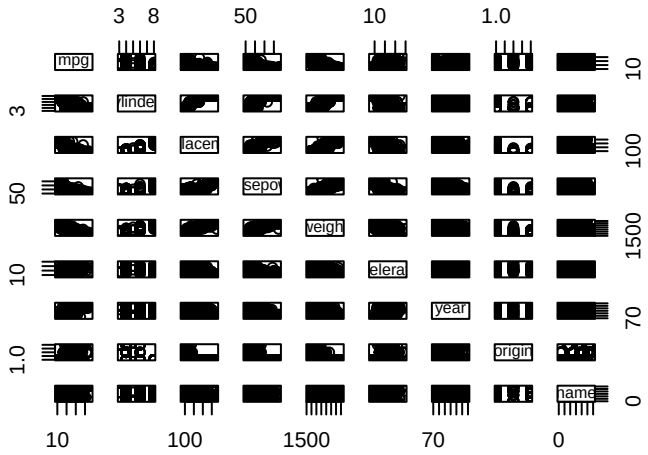
```r
predict(lm.fit, data.frame(horsepower = 98), interval = "prediction")
```

```
       fit     lwr      upr
1 24.46708 14.8094 34.12476
```

```r
ggplot(Auto_dt, aes(horsepower, mpg)) +
  geom_point(color = "blue") +
  geom_smooth(color = "red", method = "lm", formula = y ~ x, se = FALSE)

g1 <- ggplot(mapping = aes(predict(lm.fit), residuals(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")

g2 <- ggplot(mapping = aes(predict(lm.fit), rstudent(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")

grid.arrange(g1, g2, ncol = 2, newpage = FALSE)
```




```
pairs(Auto_dt)
```




```
cor(Auto_dt[, -"name"])
```

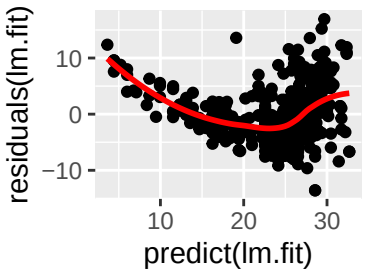
```
                    mpg  cylinders displacement horsepower     weight
mpg           1.0000000 -0.7776175   -0.8051269 -0.7784268 -0.8322442
cylinders    -0.7776175  1.0000000    0.9508233  0.8429834  0.8975273
displacement -0.8051269  0.9508233    1.0000000  0.8972570  0.9329944
horsepower   -0.7784268  0.8429834    0.8972570  1.0000000  0.8645377
weight       -0.8322442  0.8975273    0.9329944  0.8645377  1.0000000
acceleration  0.4233285 -0.5046834   -0.5438005 -0.6891955 -0.4168392
year          0.5805410 -0.3456474   -0.3698552 -0.4163615 -0.3091199
origin        0.5652088 -0.5689316   -0.6145351 -0.4551715 -0.5850054
              acceleration       year     origin
mpg              0.4233285  0.5805410  0.5652088
cylinders       -0.5046834 -0.3456474 -0.5689316
displacement    -0.5438005 -0.3698552 -0.6145351
```

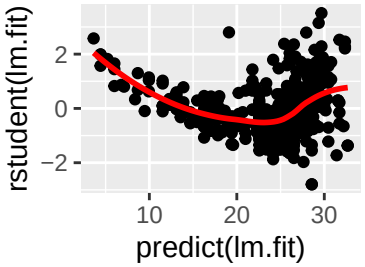
```
horsepower     -0.6891955 -0.4163615 -0.4551715
weight         -0.4168392 -0.3091199 -0.5850054
acceleration    1.0000000  0.2903161  0.2127458
year            0.2903161  1.0000000  0.1815277
origin          0.2127458  0.1815277  1.0000000
```

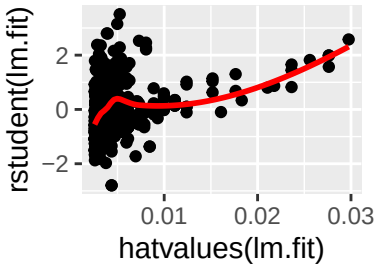
```
lm.fit <- update(lm.fit, ~ . - name)
summary(lm.fit)
```

```

Call:
lm(formula = mpg ~ horsepower, data = Auto_dt)

Residuals:
     Min       1Q   Median       3Q      Max 
-13.5710  -3.2592  -0.3435   2.7630  16.9240 

Coefficients:
             Estimate Std. Error t value            Pr(>|t|)    
(Intercept) 39.935861   0.717499   55.66 <0.0000000000000002 ***
horsepower  -0.157845   0.006446  -24.49 <0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.906 on 390 degrees of freedom
Multiple R-squared:  0.6059,    Adjusted R-squared:  0.6049 
F-statistic: 599.7 on 1 and 390 DF,  p-value: < 0.00000000000000022
```

```
g1 <- ggplot(mapping = aes(predict(lm.fit), residuals(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")
g2 <- ggplot(mapping = aes(predict(lm.fit), rstudent(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")
g3 <- ggplot(mapping = aes(hatvalues(lm.fit), rstudent(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")

grid.arrange(g1, g2, g3,ncol = 2, nrow = 2, newpage= TRUE)
```








```
round(coef(summary(lm(mpg ~ weight * year, data = Auto_dt))), 10)
```

```
                  Estimate    Std. Error   t value     Pr(>|t|)
(Intercept) -110.4519188135 12.9469686338 -8.531103 0.0000000000
weight         0.0275465227  0.0044134364  6.241514 0.0000000011
year           2.0404763960  0.1718209227 11.875599 0.0000000000
weight:year   -0.0004579323  0.0000590715 -7.752171 0.0000000000
```

```
round(coef(summary(lm(mpg ~ weight + I(weight^2), data = Auto_dt))), 10)
```

```
                Estimate   Std. Error   t value     Pr(>|t|)
(Intercept) 62.2554739733 2.9930758391 20.799832 0.0000000000
weight      -0.0184956106 0.0019720556 -9.378849 0.0000000000
I(weight^2)  0.0000016966 0.0000003059  5.545252 0.0000000543
```

```
round(coef(summary(lm(mpg ~ poly(weight, degree = 3, raw= TRUE), data = Auto_dt))), 10)
```

```
                                      Estimate    Std. Error     t value
(Intercept)                       61.6952426170 11.0430488071  5.58679434
poly(weight, degree = 3, raw = TRUE)1 -0.0179297844  0.0109148521 -1.64269604
poly(weight, degree = 3, raw = TRUE)2  0.0000015154  0.0000034504  0.43919548
poly(weight, degree = 3, raw = TRUE)3  0.0000000000  0.0000000004  0.05270974
                                        Pr(>|t|)
(Intercept)                       0.0000000436
poly(weight, degree = 3, raw = TRUE)1 0.1012559705
poly(weight, degree = 3, raw = TRUE)2 0.6607643883
poly(weight, degree = 3, raw = TRUE)3 0.9579903079
```

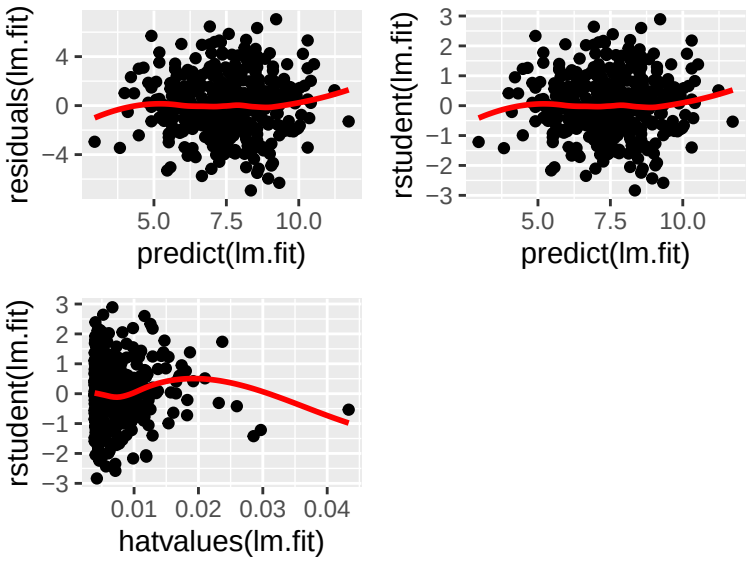
```
lm.fit <- update(lm.fit, Sales ~ Price + Urban + US, data = Carseats)

round(coef(summary(lm.fit)), 10)
```

```
               Estimate  Std. Error      t value     Pr(>|t|)
(Intercept) 13.04346894 0.651012245  20.03567373 0.0000000000
Price       -0.05445885 0.005241855 -10.38923205 0.0000000000
UrbanYes    -0.02191615 0.271650277  -0.08067781 0.9357389376
USYes        1.20057270 0.259041508   4.63467306 0.0000048602
```

```
lm.fit <- update(lm.fit, ~ . - Urban)

confint(lm.fit)
```

```
                 2.5 %      97.5 %
(Intercept) 11.79032020 14.27126531
Price       -0.06475984 -0.04419543
USYes        0.69151957  1.70776632
```

```
g1 <- ggplot(mapping = aes(predict(lm.fit), residuals(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")
g2 <- ggplot(mapping = aes(predict(lm.fit), rstudent(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")
g3 <- ggplot(mapping = aes(hatvalues(lm.fit), rstudent(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")

grid.arrange(g1, g2, g3,ncol = 2, nrow = 2, newpage= TRUE)
```




```
set.seed (1)
x <- rnorm (100)
y <- 2 * x + rnorm (100)
round(coef(summary(lm(y ~ x + 0))), 10)
```

```
  Estimate Std. Error  t value Pr(>|t|)
x 1.993876  0.1064767 18.72593        0
```

```
round(coef(summary(lm(x ~ y + 0))), 10)
```

```
   Estimate Std. Error  t value Pr(>|t|)
y 0.3911145 0.02088625 18.72593        0
```

```
round(coef(summary(lm(y ~ x))), 10)
```

```
               Estimate Std. Error    t value  Pr(>|t|)
(Intercept) -0.03769261 0.09698729 -0.3886346 0.6983896
```

```
x            1.99893961 0.10772703 18.5555993 0.0000000
```

```
round(coef(summary(lm(x ~ y))), 10)
```

```
             Estimate Std. Error    t value  Pr(>|t|)
(Intercept) 0.03880394 0.04266144  0.9095787 0.3652764
y           0.38942451 0.02098690 18.5555993 0.0000000
```

```
x <- rep(1, 100)
y <- rep(2, 100)
round(coef(summary(lm(y ~ x + 0))), 10)
```

```
  Estimate Std. Error          t value Pr(>|t|)
x        2          0 5298352502788811        0
```

```
round(coef(summary(lm(x ~ y + 0))), 10)
```

```
  Estimate Std. Error          t value Pr(>|t|)
y      0.5          0 5298352502788811        0
```

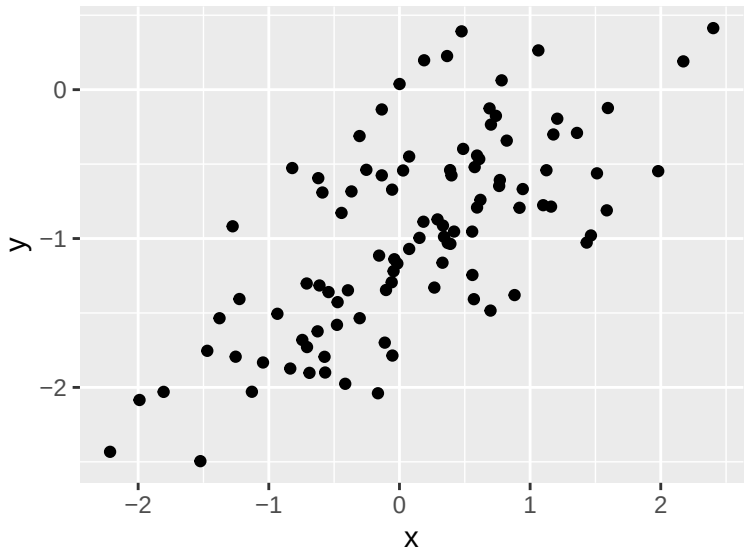
```
y <- rep(1, 100)
round(coef(summary(lm(y ~ x + 0))), 10)
```

```
  Estimate Std. Error          t value Pr(>|t|)
x        1          0 5298352502788811        0
```

```
round(coef(summary(lm(x ~ y + 0))), 10)
```

```
  Estimate Std. Error          t value Pr(>|t|)
y        1          0 5298352502788811        0
```

```
set.seed(1)
x <- rnorm(100)
eps <- rnorm(100, sd = 0.5)
y <- -1 + 0.5 * x + eps

ggplot(mapping = aes(x, y)) +
  geom_point()
```



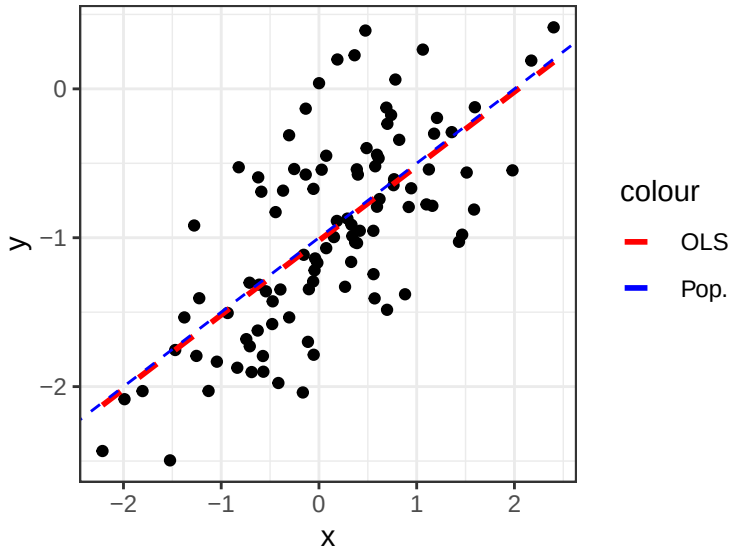
```
round(coef(summary(lm(y ~ x))), 10)
```

```
             Estimate Std. Error    t value Pr(>|t|)
(Intercept) -1.0188463 0.04849365 -21.009892        0
x            0.4994698 0.05386352   9.272878        0
```

```
confint(lm(y ~ x))
```

```
                2.5 %     97.5 %
(Intercept) -1.1150804 -0.9226122
x            0.3925794  0.6063602
```

```
ggplot(mapping = aes(x, y)) +
  geom_point() +
  geom_smooth(aes(color = "OLS"), method = "lm", formula = y ~ x, se = FALSE, linetype = "dashed") +
  geom_abline(color = "blue", slope = 0.5, intercept = -1, linetype = "dashed") +
  scale_color_manual(values = c("OLS" = "red", "Pop." = "blue")) +
  theme_bw()
```




```
round(coef(summary(lm(y ~ x + I(x^2)))), 10)
```

```
             Estimate Std. Error    t value  Pr(>|t|)
(Intercept) -0.9716425 0.05882775 -16.516738 0.0000000
x            0.5085804 0.05399135   9.419666 0.0000000
I(x^2)      -0.0594606 0.04238285  -1.402940 0.1638275
```

```
eps <- rnorm(100, sd = 0.1)
y <- -1 + 0.5 * x + eps

round(coef(summary(lm(y ~ x))), 10)
```

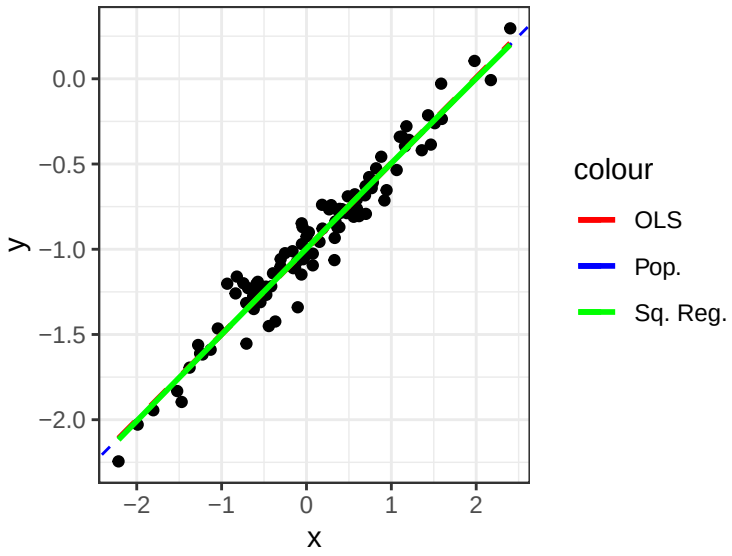
```
             Estimate Std. Error   t value Pr(>|t|)
(Intercept) -0.9972631 0.01047039 -95.24600        0
x            0.5021167 0.01162982  43.17494        0
```

```
confint(lm(y ~ x))
```

```
                2.5 %     97.5 %
```

```
(Intercept) -1.0180413 -0.9764850
x            0.4790377  0.5251957
```

```
ggplot(mapping = aes(x, y)) +
  geom_point() +
  geom_smooth(aes(color = "OLS"), method = "lm", formula = y ~ x, se = FALSE, linetype = "dashed") +
  geom_abline(color = "blue", slope = 0.5, intercept = -1, linetype = "dashed") +
  geom_smooth(aes(color = "Sq. Reg."), method = "lm", formula = y ~ x + I(x^2), se = FALSE) +
  scale_color_manual(values = c("OLS" = "red", "Pop." = "blue", "Sq. Reg." = "green")) +
  theme_bw()
```




```
round(coef(summary(lm(y ~ x + I(x^2)))), 10)
```

```
               Estimate  Std. Error    t value  Pr(>|t|)
(Intercept) -0.995886825 0.012827549 -77.636563 0.0000000
x            0.502382321 0.011772959  42.672562 0.0000000
I(x^2)      -0.001733668 0.009241695  -0.187592 0.8515884
```

```
eps <- rnorm(100, sd = 1)
y <- -1 + 0.5 * x + eps

round(coef(summary(lm(y ~ x))), 10)
```

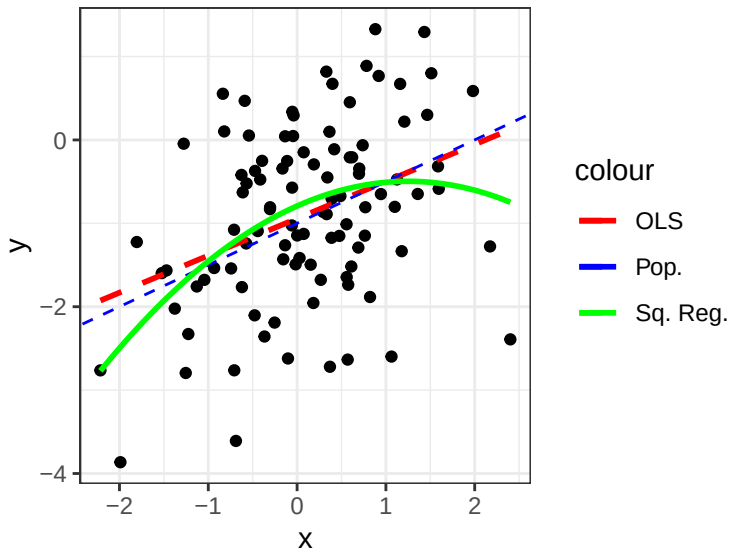
```
             Estimate Std. Error   t value     Pr(>|t|)
(Intercept) -0.9423346  0.1002815 -9.396897 0.0000000000
x            0.4443139  0.1113860  3.988957 0.0001280226
```

```
confint(lm(y ~ x))
```

```
                2.5 %     97.5 %
(Intercept) -1.1413399 -0.7433293
x            0.2232721  0.6653558
```

```
ggplot(mapping = aes(x, y)) +
  geom_point() +
  geom_smooth(aes(color = "OLS"), method = "lm", formula = y ~ x, se = FALSE, linetype = "dashed") +
  geom_abline(color = "blue", slope = 0.5, intercept = -1, linetype = "dashed") +
  geom_smooth(aes(color = "Sq. Reg."), method = "lm", formula = y ~ x + I(x^2), se = FALSE) +
```

```
  scale_color_manual(values = c("OLS" = "red", "Pop." = "blue", "Sq. Reg." = "green")) +
  theme_bw()
```



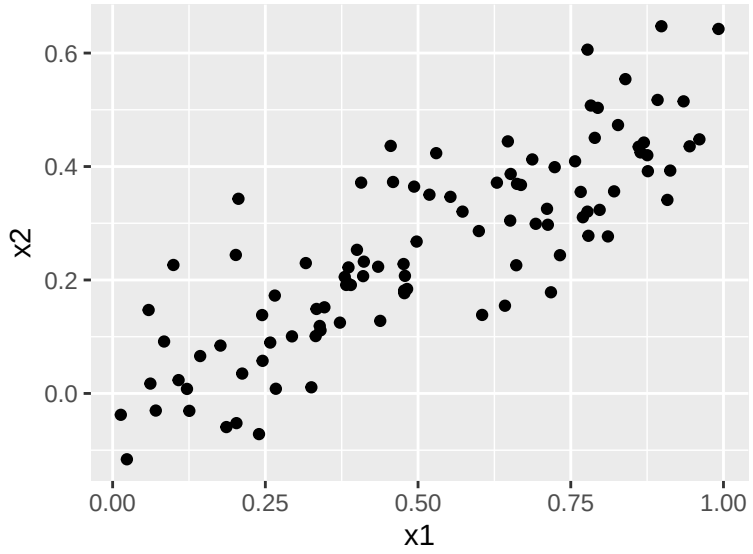

```
round(coef(summary(lm(y ~ x + I(x^2)))), 10)
```

```
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) -0.7919712 0.11994537 -6.602766 0.0000000022
x            0.4733350 0.11008431  4.299750 0.0000407456
I(x^2)      -0.1894063 0.08641546 -2.191811 0.0307861206
```

```
set.seed (1)
x1 <- runif (100)
x2 <- 0.5 * x1 + rnorm (100) / 10
y <- 2 + 2 * x1 + 0.3 * x2 + rnorm (100)

cor(x1, x2)
```

```
[1] 0.8351212
```

```
ggplot(mapping = aes(x1, x2)) +
  geom_point()
```




```
lm.fit <- update(lm.fit, y ~ x1 + x2, data = NULL)
round(coef(summary(lm.fit)), 10)
```

```
            Estimate Std. Error   t value   Pr(>|t|)
(Intercept) 2.130500  0.2318817 9.1878742 0.00000000
x1          1.439555  0.7211795 1.9961126 0.04872517
x2          1.009674  1.1337225 0.8905831 0.37535648
```

```
lm.fit <- update(lm.fit, ~ . - x2)
round(coef(summary(lm.fit)), 10)
```

```
            Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 2.112394  0.2307448 9.154676 0.0000000000
x1          1.975929  0.3962774 4.986227 0.0000026606
```

```
lm.fit <- update(lm.fit, ~ x2)
round(coef(summary(lm.fit)), 10)
```

```
            Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 2.389949  0.1949307 12.260508 0.0000000000
x2          2.899585  0.6330467  4.580365 0.0000136643
```

```
x1 <- c(x1 , 0.1)
x2 <- c(x2 , 0.8)
y <- c(y, 6)

lm.fit <- update(lm.fit, ~ x1 + x2)
round(coef(summary(lm.fit)), 10)
```

```
             Estimate Std. Error   t value    Pr(>|t|)
(Intercept) 2.2266917  0.2313578 9.6244495 0.000000000
x1          0.5394397  0.5921970 0.9109125 0.364576626
x2          2.5145694  0.8976915 2.8011508 0.006135787
```

```
lm.fit <- update(lm.fit, ~ . - x2)
round(coef(summary(lm.fit)), 10)
```

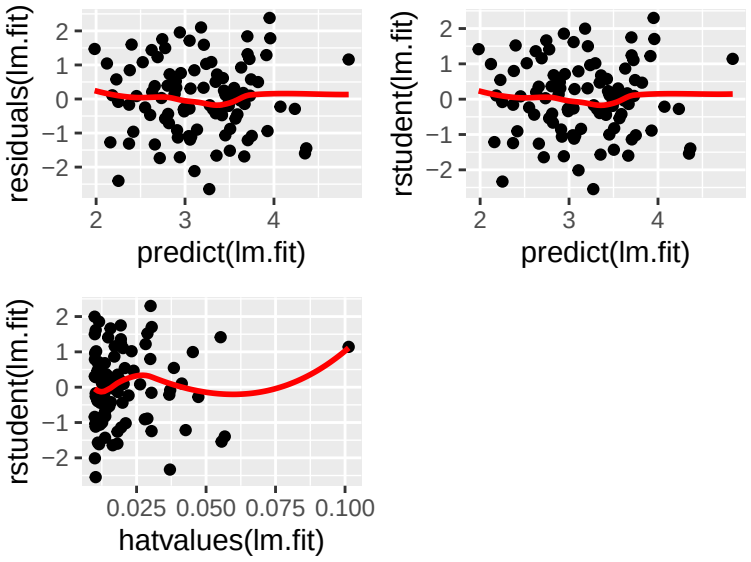
```
            Estimate Std. Error  t value     Pr(>|t|)
```

```
(Intercept) 2.256927  0.2389635 9.444654 0.0000000000
x1          1.765695  0.4123781 4.281739 0.0000429482
```

```
lm.fit <- update(lm.fit, ~ x2)
round(coef(summary(lm.fit)), 10)
```

```
            Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 2.345107  0.1912183 12.264029 0.0000000000
x2          3.119050  0.6040352  5.163689 0.0000012531
```

```
g1 <- ggplot(mapping = aes(predict(lm.fit), residuals(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")
g2 <- ggplot(mapping = aes(predict(lm.fit), rstudent(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")
g3 <- ggplot(mapping = aes(hatvalues(lm.fit), rstudent(lm.fit))) +
  geom_point() +
  geom_smooth(color = "red", se = FALSE, formula = y ~ x, method = "loess")

grid.arrange(g1, g2, g3,ncol = 2, nrow = 2, newpage= TRUE)
```




```
parallel::clusterExport(cl, "lm.fit")

lm.fit_slopes <- foreach(colname = colnames(Boston_dt[, -"crim"]), .combine = c) %dopar% {
  lm.fit <- update(lm.fit, as.formula(paste0("crim ~ ", colname)), data = Boston_dt)
  return(coef(lm.fit)[[2]])
}

print(lm.fit_slopes)
```

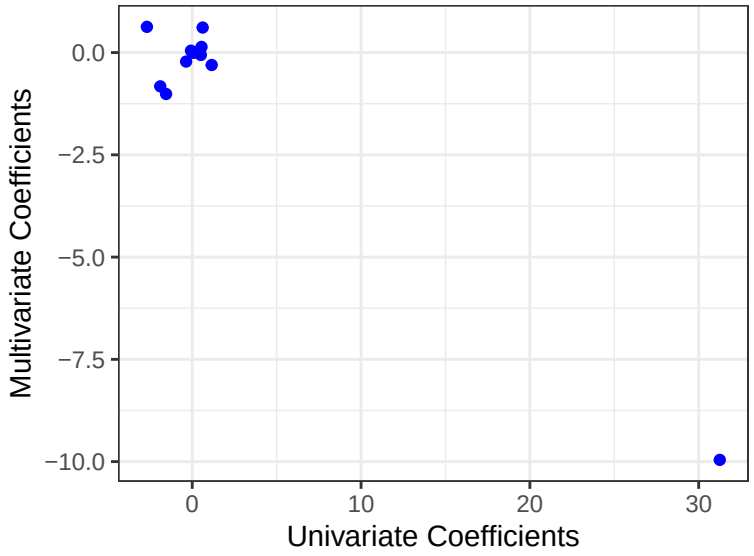
```
 [1] -0.07393498  0.50977633 -1.89277655 31.24853120 -2.68405122  0.10778623
 [7] -1.55090168  0.61791093  0.02974225  1.15198279  0.54880478 -0.36315992
```

```
lm.fit <- lm(crim ~ ., data = Boston_dt)
round(coef(summary(lm.fit)), 10)
```

```
                Estimate  Std. Error    t value     Pr(>|t|)
```

```
(Intercept) 13.7783937863 7.081825783  1.9455991 0.0522708914
zn           0.0457100386 0.018790325  2.4326370 0.0153440268
indus       -0.0583501107 0.083635091 -0.6976750 0.4857093737
chas        -0.8253775522 1.183396256 -0.6974651 0.4858405742
nox         -9.9575865471 5.289824241 -1.8824040 0.0603698592
rm           0.6289106622 0.607092390  1.0359390 0.3007384748
age         -0.0008482791 0.017948208 -0.0472626 0.9623230716
dis         -1.0122467382 0.282467566 -3.5835857 0.0003725942
rad          0.6124653115 0.087535764  6.9967438 0.0000000000
tax         -0.0037756465 0.005172346 -0.7299678 0.4657565440
ptratio     -0.3040727572 0.186359810 -1.6316434 0.1033932001
lstat        0.1388005968 0.075721250  1.8330468 0.0673984406
medv        -0.2200563590 0.059823956 -3.6783987 0.0002605302
```

```
ggplot(mapping = aes(lm.fit_slopes, unname(coef(lm.fit)[-1]))) +
  geom_point(color = "blue") +
  theme_bw() +
  labs(x = "Univariate Coefficients", y = "Multivariate Coefficients")
```




```
lm.fit_coef <- foreach(colname = colnames(Boston_dt[, -c("crim", "chas")]), .combine = rbind) %dopar% {
  return(round(coef(summary(lm(as.formula(paste0("crim ~ poly(", colname, ", 3)")), data = Boston_dt)))
}

print(lm.fit_coef)
```

```
                  Estimate Std. Error     t value     Pr(>|t|)
(Intercept)        3.613524  0.3721900   9.7088140 0.0000000000
poly(zn, 3)1     -38.749835  8.3722072  -4.6283894 0.0000046978
poly(zn, 3)2      23.939832  8.3722072   2.8594409 0.0044205069
poly(zn, 3)3     -10.071868  8.3722072  -1.2030123 0.2295386205
(Intercept)        3.613524  0.3299980  10.9501385 0.0000000000
poly(indus, 3)1   78.590819  7.4231210  10.5873014 0.0000000000
poly(indus, 3)2  -24.394796  7.4231210  -3.2863261 0.0010860571
poly(indus, 3)3  -54.129763  7.4231210  -7.2920492 0.0000000000
(Intercept)        3.613524  0.3215730  11.2370253 0.0000000000
poly(nox, 3)1     81.372015  7.2336050  11.2491649 0.0000000000
```

```
poly(nox, 3)2     -28.828594  7.2336050  -3.9853703 0.0000773675
poly(nox, 3)3     -60.361894  7.2336050  -8.3446489 0.0000000000
(Intercept)         3.613524  0.3702993   9.7583873 0.0000000000
poly(rm, 3)1      -42.379442  8.3296758  -5.0877661 0.0000005128
poly(rm, 3)2       26.576770  8.3296758   3.1906128 0.0015085455
poly(rm, 3)3       -5.510342  8.3296758  -0.6615314 0.5085751094
(Intercept)         3.613524  0.3485173  10.3682762 0.0000000000
poly(age, 3)1      68.182009  7.8397027   8.6970146 0.0000000000
poly(age, 3)2      37.484470  7.8397027   4.7813638 0.0000022912
poly(age, 3)3      21.353207  7.8397027   2.7237266 0.0066799154
(Intercept)         3.613524  0.3259240  11.0870129 0.0000000000
poly(dis, 3)1     -73.388590  7.3314790 -10.0100661 0.0000000000
poly(dis, 3)2      56.373036  7.3314790   7.6891764 0.0000000000
poly(dis, 3)3     -42.621877  7.3314790  -5.8135442 0.0000000109
(Intercept)         3.613524  0.2970690  12.1639203 0.0000000000
poly(rad, 3)1     120.907446  6.6824017  18.0934117 0.0000000000
poly(rad, 3)2      17.492299  6.6824017   2.6176664 0.0091205580
poly(rad, 3)3       4.698457  6.6824017   0.7031090 0.4823137740
(Intercept)         3.613524  0.3046845  11.8598881 0.0000000000
poly(tax, 3)1     112.645827  6.8537074  16.4357509 0.0000000000
poly(tax, 3)2      32.087251  6.8537074   4.6817364 0.0000036653
poly(tax, 3)3      -7.996811  6.8537074  -1.1667862 0.2438506811
(Intercept)         3.613524  0.3610484  10.0084186 0.0000000000
poly(ptratio, 3)1  56.045229  8.1215830   6.9007765 0.0000000000
poly(ptratio, 3)2  24.774824  8.1215830   3.0504920 0.0024054679
poly(ptratio, 3)3 -22.279737  8.1215830  -2.7432751 0.0063005136
(Intercept)         3.613524  0.3391698  10.6540249 0.0000000000
poly(lstat, 3)1    88.069666  7.6294361  11.5434044 0.0000000000
poly(lstat, 3)2    15.888164  7.6294361   2.0824821 0.0378041809
poly(lstat, 3)3   -11.574022  7.6294361  -1.5170220 0.1298905873
(Intercept)         3.613524  0.2920344  12.3736218 0.0000000000
poly(medv, 3)1    -75.057605  6.5691520 -11.4257678 0.0000000000
poly(medv, 3)2     88.086211  6.5691520  13.4090687 0.0000000000
poly(medv, 3)3    -48.033435  6.5691520  -7.3119688 0.0000000000
```

Chapter 4

Notes

An Introduction to Classification

It's often the case that we wish to predict a qualitative or **categorical** response variable, a process called **classification**. A classification technique is called a **classifier** (since it involves assigning the observation to a category, or class). Some classifiers include: logistic regression, linear discriminant analysis, quadratic discriminant analysis, naive Bayes, and K-nearest neighbors.

Generally speaking, we do not use regression methods for classification problems for two reasons:

1. A regression method cannot accommodate a qualitative response with more than two classes because such qualitative responses generally lack cardinality.
2. A regression method will not (always) provide meaningful estimates of $\Pr(Y|X)$, even with just two classes (some probabilities may fall out of the [0,1] range).

Thus, we prefer using classification methods that are truly suited for qualitative response values.

Simple Logistic Regression

Rather than modeling the response Y directly, **logistic regression** models the probability that Y belongs to a particular category. Since logistic regression is well-suited for the case of a binary qualitative response, this can be written mathematically as

$$p(X) = \Pr(Y = 1|X)$$

To model $p(X)$ in logistic regression, we use the **logistic function**,

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

The logistic function always produces an S-shaped curve. Thus, regardless of the value of X, we will obtain a sensible prediction between 0 and 1. From the logistic function, we can also derive the **odds**

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0+\beta_1 X},$$

which can take on any value between 0 (an indication of a low $\Pr(Y = 1|X)$) and ∞ (an indication of a high $\Pr(Y = 1|X)$). Thus, if we know a particular subset of the sample has an odds of c, then

$$\frac{p(X)}{1 - p(X)} = c,$$

which can be manipulated to show

$$p(X) = \frac{c}{c + 1}$$

Returning to the logistic function, if we take the (natural) logarithm of both sides, we obtain

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X,$$

where the left-hand side is called the **log odds** or **logit**, which is linear in X. Thus, increasing X by one-unit increases the log odds by β_1 (or equivalently, it multiplies the odds by e^{β_1}). However, β_1 does not correspond to the change in $p(X)$ for a one-unit increase in X. Instead, the change in $p(X)$ depends on the current value of X. Nevertheless, if $\beta_1 > 0$, $p(X)$ increases with X, and if $\beta_1 < 0$, $p(X)$ decreases with X.

To fit a logistic regression model, we use a method called **maximum likelihood**, where we choose $\hat{\beta}_0$ and $\hat{\beta}_1$ to maximize the **likelihood function**

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$$

After computing estimates for β_0 and β_1, predicting $\Pr(Y = 1|X)$ is derived from

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0+\hat{\beta}_1 X}}$$

Multiple Logistic Regression

The same framework from simple logistic regression can be applied to multiple logistic regression. Considering the case of p predictors where

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}},$$

we again use the maximum likelihood method to derive estimators for $\beta_0, \beta_1, \ldots, \beta_p$.

Derivation of Maximum Likelihood Estimators (MLE)

Recall the likelihood function

$$L(\beta) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

In general, since maximizing both relative to β results in the same estimators, it's easier to work with the **log-likelihood function**

$$\begin{aligned}
\ell(\beta) &= \log\left(\prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}\right) \\
&= \sum_{i=1}^{n} \left[\log\left(p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}\right)\right] \\
&= \sum_{i=1}^{n} y_i \log(p(x_i)) + (1 - y_i)\log(1 - p(x_i)) \\
&= \sum_{i=1}^{n} y_i \log\left(\frac{1}{1 + e^{-z_i}}\right) + (1 - y_i)\log\left(\frac{e^{-z_i}}{1 + e^{-z_i}}\right),
\end{aligned}$$

where $z_i = \sum_{j=0}^{p} \beta_j x_{ij}$.

$$\begin{aligned}
&= \sum_{i=1}^{n} y_i \left[\log\left(\frac{1}{1 + e^{-z_i}}\right) - \log\left(\frac{e^{-z_i}}{1 + e^{-z_i}}\right)\right] + \log\left(\frac{e^{-z_i}}{1 + e^{-z_i}}\right) \\
&= \sum_{i=1}^{n} y_i \left[-\log\left(1 + e^{-z_i}\right) - \log\left(e^{-z_i}\right) + \log\left(1 + e^{-z_i}\right)\right] + \log\left(\frac{e^{-z_i}}{1 + e^{-z_i}}\right) \\
&= \sum_{i=1}^{n} y_i \left[-\log(e^{-z_i})\right] + \log\left(\frac{1}{1 + e^{z_i}}\right) \\
&= \sum_{i=1}^{n} y_i z_i - \log\left(1 + e^{z_i}\right)
\end{aligned}$$

$$\frac{\partial \ell(\beta)}{\partial \beta} = \frac{\partial}{\partial \beta}\left[\sum_{i=1}^{n} y_i z_i - \log(1 + e^{z_i})\right] = \vec{0}$$

$$= \begin{bmatrix} \sum_{i=1}^{n} \frac{\partial}{\partial \beta_0}\left[y_i \sum_{j=0}^{p} (\beta_j x_{ij}) - \log(1 + e^{\sum_{j=0}^{p} \beta_j x_{ij}})\right] \\ \vdots \\ \sum_{i=1}^{n} \frac{\partial}{\partial \beta_p}\left[y_i \sum_{j=0}^{p} (\beta_j x_{ij}) - \log(1 + e^{\sum_{j=0}^{p} \beta_j x_{ij}})\right] \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{n} \left(y_i - \frac{1}{1+e^{-\sum_{j=0}^{p} \beta_j x_{ij}}} \right) \\ \vdots \\ \sum_{i=1}^{n} \left(x_{ip} y_i - \frac{x_{ip}}{1+e^{-\sum_{j=0}^{p} \beta_j x_{ij}}} \right) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{n} \left(y_i - \frac{1}{1+e^{-z_i}} \right) \\ \vdots \\ \sum_{i=1}^{n} x_{ip} \left(y_i - \frac{1}{1+e^{-z_i}} \right) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{n} \left(y_i - p(x_i) \right) \\ \vdots \\ \sum_{i=1}^{n} x_{ip} \left(y_i - p(x_i) \right) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{n} \left(y_i - p(x_i) \right) \\ \vdots \\ \sum_{i=1}^{n} x_{ip} \left(y_i - p(x_i) \right) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} (\mathbf{y} - \mathbf{p}) \\ \vdots \\ \mathbf{x}_p^T (\mathbf{y} - \mathbf{p}) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \mathbf{X}^T (\mathbf{y} - \mathbf{p}) = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

Unfortunately, we cannot solve for this equality analytically. Instead, we turn to the Newton Raphson numerical iteration method to approximate the equality. From the Newton Raphson Method

$$\nabla_\beta \ell(\beta_{(k+1)}) = \nabla_\beta \ell(\beta_{(k)}) + (\beta_{(k+1)} - \beta_{(k)}) \nabla_{\beta\beta} \ell(\beta_{(k)}) = 0$$

$$\beta_{(k+1)} \nabla_{\beta\beta} \ell(\beta_{(k)}) = -\nabla_\beta \ell(\beta_{(k)}) + \beta_{(k)} \nabla_{\beta\beta} \ell(\beta_{(k)})$$

$$\beta_{(k+1)} = \beta_{(k)} - \frac{\nabla_\beta \ell(\beta_{(k)})}{\nabla_{\beta\beta} \ell(\beta_{(k)})}$$

We solved for $\nabla_\beta \ell(\beta)$ as

$$\nabla_\beta \ell(\beta) = \begin{bmatrix} \sum_{i=1}^{n} \left(y_i - p(x_i) \right) \\ \vdots \\ \sum_{i=1}^{n} x_{ip} \left(y_i - p(x_i) \right) \end{bmatrix}$$

Thus, we can solve for the Hermitian of the log-likelihood.

$$\nabla_{\beta\beta} \ell(\beta) = \begin{bmatrix} \frac{\partial^2 \ell(\beta)}{\partial \beta_0^2} & \cdots & \frac{\partial^2 \ell(\beta)}{\partial \beta_0 \beta_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 \ell(\beta)}{\partial \beta_p \beta_0} & \cdots & \frac{\partial^2 \ell(\beta)}{\partial \beta_p^2} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial}{\partial \beta_0} \left(\sum_{i=1}^{n} \left(y_i - p(x_i) \right) \right) & \cdots & \frac{\partial}{\partial \beta_p} \left(\sum_{i=1}^{n} \left(y_i - p(x_i) \right) \right) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial \beta_0} \left(\sum_{i=1}^{n} x_{ip} \left(y_i - p(x_i) \right) \right) & \cdots & \frac{\partial}{\partial \beta_p} \left(\sum_{i=1}^{n} x_{ip} \left(y_i - p(x_i) \right) \right) \end{bmatrix}$$

$$
\begin{aligned}
&= \begin{bmatrix}
\sum_{i=1}^{n} \frac{e^{-\sum_{j=0}^{p} \beta_j x_{ij}}}{\left(1+e^{-\sum_{j=0}^{p} \beta_j x_{ij}}\right)^2} & \cdots & \sum_{i=1}^{n} \frac{x_{ip} e^{-\sum_{j=0}^{p} \beta_j x_{ij}}}{\left(1+e^{-\sum_{j=0}^{p} \beta_j x_{ij}}\right)^2} \\
\vdots & \ddots & \vdots \\
\sum_{i=1}^{n} \frac{x_{ip} e^{-\sum_{j=0}^{p} \beta_j x_{ij}}}{\left(1+e^{-\sum_{j=0}^{p} \beta_j x_{ij}}\right)^2} & \cdots & \sum_{i=1}^{n} \frac{x_{ip}^2 e^{-\sum_{j=0}^{p} \beta_j x_{ij}}}{\left(1+e^{-\sum_{j=0}^{p} \beta_j x_{ij}}\right)^2}
\end{bmatrix} \\
&= \begin{bmatrix}
\sum_{i=1}^{n} p(x_i)\left(1-p(x_i)\right) & \cdots & \sum_{i=1}^{n} x_{ip} p(x_i)\left(1-p(x_i)\right) \\
\vdots & \ddots & \vdots \\
\sum_{i=1}^{n} x_{ip} p(x_i)\left(1-p(x_i)\right) & \cdots & \sum_{i=1}^{n} x_{ip}^2 p(x_i)\left(1-p(x_i)\right)
\end{bmatrix} \\
&= \mathbf{X}^T \mathbf{W} \mathbf{X},
\end{aligned}
$$

where $\mathbf{W} \in \mathbb{R}^{n \times n} = \operatorname{diag}\{p(x_i)\left(1-p(x_i)\right), \ldots, p(x_i)\left(1-p(x_i)\right)\}$

Finally, the steps to estimate β using the Newton-Raphson iteration method are:

1. Input the initial estimate $\beta_{(0)}$.
2. To obtain estimates on the $(k+1)$th iteration, calculate

$$\beta_{(k+1)} = \beta_{(k)} + (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{p})$$

3. The iteration is continued until $\beta_{(k+1)} \approx \beta_{(k)}$.

Multinomial Logistic Regression

In cases in which we wish to use logistic regression with non-binary responses (response variables that have more than two classes), we use **multinomial logisitic regression**. To begin, we choose the Kth class to serve as our **baseline**. Then, the multinomial logistic function takes the form

$$\Pr(Y = k | X = x) = \frac{e^{\beta_{k0} + \beta_{k1} x_1 + \cdots + \beta_{kp} x_p}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p}}$$

for $k = 1, \ldots, K-1$, and

$$\Pr(Y = K | X = x) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p}}$$

Thus, we can obtain the log odds or logit between any pair of classes by

$$
\begin{aligned}
\log\left(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)}\right) &= \log\left(\frac{e^{\beta_{k0} + \beta_{k1} x_1 + \cdots + \beta_{kp} x_p}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p}}\right) - \log\left(\frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p}}\right) \\
&= \log\left(e^{\beta_{k0} + \beta_{k1} x_1 + \cdots + \beta_{kp} x_p}\right) - \log\left(1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p}\right) + \log\left(1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p}\right) \\
&= \beta_{k0} + \beta_{k1} x_1 + \cdots + \beta_{kp} x_p
\end{aligned}
$$

An alternative coding for multinomial logistic regression, known as **softmax** coding, treats all K classes symmetrically and assumes that, for $k = 1, \ldots, K$,

$$\Pr(Y = k | X = x) = \frac{e^{\beta_{k0} + \beta_{k1} x_1 + \ldots \beta_{kp} x_p}}{1 + \sum_{l=1}^{K} e^{\beta_{l0} + \beta_{l1} x_1 + \ldots \beta_{lp} x_p}}$$

Bayes' Theorem

For $K \geq 2$, let π_k be the **prior** probability that a randomly chosen observation comes from the kth class (i.e. $\Pr(Y = k)$) and $f_k(X) \equiv \Pr(X|Y = k)$ denote the density function of X for an observation that comes from the kth class. Then **Bayes' theorem** states

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)},$$

which is the **posterior** probability that an observation $X = x$ belongs to the kth class (this is also denoted $p_k(x)$). Using a random sample, estimating π_k is simple as we just compute the fraction of the training observations from the kth class. Unfortunately, estimating the density function $f_k(x)$ is much more difficult. We know that the Bayes classifier, which classifies an observation x to the class for which $p_k(x)$ is largest, has the lowest possible error rate out of all classifiers. Therefore, if we can find a way to estimate $f_k(x)$, then we can plug it into Bayes' theorem in order to approximate the Bayes classifier. Three classifiers that use different estimates of $f_k(x)$ to approximate the Bayes classifier include: linear discriminant analysis, quadratic discriminant analysis, and naive Bayes.

Linear Discriminant Analysis

For a moment, assume we have one predictor and that $f_k(x)$ is **normal** or **Gaussian**. Mathematically, we write this as

$$f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2},$$

where μ_k and σ_k are parameters for the kth class. If we assume a common standard deviation among the k classes (i.e., $\sigma_k = \sigma$), we can plug the previous equation into the formula for Bayes' theorem and obtain

$$p_k(x) = \frac{\pi_k \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{l=1}^{K} \pi_l \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_l}{\sigma}\right)^2}}$$

[Recall that $\pi_k = \Pr(Y = k) \neq \pi \approx 3.1415927$]. Taking the natural logarithm of the previous expression,

$$\log\left(p_k(x)\right) = \log\left(\frac{\pi_k \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{l=1}^{K} \pi_l \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_l}{\sigma}\right)^2}}\right)$$

Since we're concerned with maximizing this value by assigning it to the kth class and the denominator does not vary with k, we write

$$\log\left(\pi_k \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}\right) = \log(\pi_k) - \log(\sigma\sqrt{2\pi}) + \log\left(e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}\right)$$

$$= \log(\pi_k) - \log(\sigma\sqrt{2\pi}) - \frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2$$

$$= \log(\pi_k) - \log(\sigma) - \log(\sqrt{2\pi}) - \frac{1}{2}\left(\frac{x^2 - 2x\mu_k + \mu_k^2}{\sigma^2}\right)$$

$$= \log(\pi_k) - \log(\sigma) - \log(\sqrt{2\pi}) - \frac{x^2}{2\sigma^2} + \frac{x\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2}$$

Since $\log(\sqrt{2\pi})$, $\log(\sigma)$, and $x^2/2\sigma^2$ do not vary with k (recall we are treating x as fixed), we obtain

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k),$$

which is maximized when the above equation for $p_k(x)$ is maximized (with respect to the class k). This equation tells us which class the Bayes classifier assigns an observation to under our assumptions of a normally distributed x given $y = k$ and a common standard deviation for the K classes. Thus, if $K = 2$, then the Bayes decision boundary is given by the points where $\delta_1(x) = \delta_2(x)$. Unfortunately, in almost all real life situations, we do not observe population parameters, and thus, we are unable to calculate the Bayes classifier.

Using the **linear discriminant analysis (LDA)** method, we approximate the parameters using their estimators. Thus, the LDA classifier uses the following **discriminant functions**

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k),$$

where if n is the total number of observations in the training data and n_k is the number of observations belonging to the kth class in the training data,

$$\hat{\pi}_k = \frac{n_k}{n}$$

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

We can extend the LDA classifier to the case of multiple predictors. Now, we will assume that $X = (X_1, X_2, \ldots, X_p)$ is drawn from a **multivariate Gaussian** (or **multivariate normal**) distribution, with a class-specific multivariate mean vector and a common covariance matrix. Mathematically, to indicate a p-dimensional random variable X has a multivariate Gaussian distribution, we write $X \sim \mathcal{N}(\mu, \mathbf{\Sigma})$, where $E(X) = \mu$ is the mean of X ($\mu = [E(X_1), \ldots, E(X_p)]$), and $\text{Cov}(X) = \mathbf{\Sigma}$ is the $p \times p$ covariance matrix of X. Formally, the multivariate Gaussian density is defined as

$$f(x) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)}$$

Thus, the LDA classifier assumes that the observations in the kth class are drawn from a multivariate Gaussian distribution $\mathcal{N}(\mu_k, \mathbf{\Sigma})$, where μ_k is a class-specific mean vector, and $\mathbf{\Sigma}$ is a shared covariance matrix between the K classes. Taking the numerator of Bayes' theorem and applying a similar logic to our derivation of the Bayes classifier from the case where $p = 1$

$$\log\left(\pi_k \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \mathbf{\Sigma}^{-1}(x-\mu_k)}\right) = \log(\pi_k) - \log\left((2\pi)^{p/2}\right) - \log\left(|\mathbf{\Sigma}|^{1/2}\right) + \log\left(e^{-\frac{1}{2}(x-\mu_k)^T \mathbf{\Sigma}^{-1}(x-\mu_k)}\right)$$

$$= \log(\pi_k) - \log\left((2\pi)^{p/2}\right) - \log\left(|\mathbf{\Sigma}|^{1/2}\right) - \frac{1}{2}(x-\mu_k)^T \mathbf{\Sigma}^{-1}(x-\mu_k)$$

Again, since $\log(\sqrt{2\pi})$, $\log(\sigma)$, and $x^2/2\sigma^2$ do not vary with k (recall we are treating x as fixed), we obtain

$$\log(\pi_k) - \frac{1}{2}(x^T - \mu_k^T)(\mathbf{\Sigma}^{-1}x - \mathbf{\Sigma}^{-1}\mu_k)$$

$$= \log(\pi_k) - \frac{1}{2}x^T \mathbf{\Sigma}^{-1} x + \frac{1}{2}x^T \mathbf{\Sigma}^{-1}\mu_k + \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1} x - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k$$

Removing the term that doesn't vary with k ($\frac{1}{2}x^T \mathbf{\Sigma}^{-1} x$) and realizing that $x^T \mathbf{\Sigma}^{-1}\mu_k = \mu_k^T \mathbf{\Sigma}^{-1} x$ (since scalars are symmetric), we obtain

$$\delta_k(x) = x^T \mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \log(\pi_k)$$

Once again, as we did in the case where $p = 1$, we derive the discriminant functions ($\hat{\delta}_k$) by using the estimators for $\pi_1, \ldots, \pi_K$, $\mu_1, \ldots, \mu_K$, and $\mathbf{\Sigma}$.

Recall that the Bayes classifier chooses the class k that maximizes the posterior probability and δ_k. Likewise, linear discriminant analysis chooses the class k that maximizes $\hat{\delta}_k$. In some circumstances, we may be particularly concerned with minimizing incorrect predictions for a certain class (such as minimizing the number of false negatives a test gives). In these cases, we can simply adjust the threshold to better meet our goals. Generally, if we move the threshold to be more stringent toward our goal, we can expect the overall test error rate to increase but the test error rate regarding our interest to decrease.

Quadratic Discriminant Analysis

Another approach for estimating the Bayes classifier is **quadratic discriminant analysis (QDA)**. Like LDA, the QDA classifier results from assuming that the observations from each class are drawn from a Gaussian distribution and plugging estimates for the parameters into Bayes' theorem in order to perform prediction. However, unlike LDA, QDA assumes that each class has its own covariance matrix. In other words, an observation from the kth class is of the form $X \sim \mathcal{N}(\mu_k, \mathbf{\Sigma}_k)$, where $\mathbf{\Sigma}_k$ is a covariance matrix for the kth class. Under this assumption, we can take the numerator from Bayes' theorem and apply a similar logic to that used for the derivation of the Bayes classifier in the case of LDA

$$\log\left(\pi_k \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k)}\right) = \log(\pi_k) - \log\left((2\pi)^{p/2}\right) - \log\left(|\mathbf{\Sigma}_k|^{1/2}\right) + \log\left(e^{-\frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k)}\right)$$

$$= \log(\pi_k) - \log\left((2\pi)^{p/2}\right) - \log\left(|\mathbf{\Sigma}_k|^{1/2}\right) - \frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k)$$

$$= \log(\pi_k) - \log\left((2\pi)^{p/2}\right) - \frac{1}{2}\log\left(|\mathbf{\Sigma}_k|\right) - \frac{1}{2}(x^T - \mu_k^T)(\mathbf{\Sigma}_k^{-1}x - \mathbf{\Sigma}_k^{-1}\mu_k)$$

$$= \log(\pi_k) - \log\left((2\pi)^{p/2}\right) - \frac{1}{2}\log\left(|\mathbf{\Sigma}_k|\right) - \frac{1}{2}(x^T\mathbf{\Sigma}_k^{-1}x - x^T\mathbf{\Sigma}_k^{-1}\mu_k - \mu_k^T\mathbf{\Sigma}_k^{-1}x + \mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k)$$

Removing the term that doesn't vary with k ($\log\left((2\pi)^{p/2}\right)$), we obtain

$$\delta_k(x) = -\frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}x + \frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}\mu_k + \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}x - \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\log\left(|\mathbf{\Sigma}_k|\right) + \log(\pi_k)$$

$$\delta_k(x) = -\frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}x + x^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\log\left(|\mathbf{\Sigma}_k|\right) + \log(\pi_k)$$

Once again, as we did in the case of LDA, we derive the discriminant functions ($\hat{\delta}_k$) by using the estimators for $\pi_1, \ldots, \pi_K$, $\mu_1, \ldots, \mu_K$, and $\mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_K$.

Choosing between LDA and QDA is another matter of the bias-variance trade-off. LDA assumes that the K classes share a common covariance matrix, which makes the number of required estimates smaller than that of QDA (much smaller in high dimensional cases). LDA is a much less flexible classifier than QDA and, thus, has lower variance. This can lead to improved prediction performance if LDA's assumption about a common covariance matrix is fairly accurate. On the other hand, if the assumption about a common covariance matrix is badly off, we should prefer QDA since LDA can suffer from biasedness. Generally speaking, LDA tends to be preferred if there are relatively few training observations (i.e., reducing variance is crucial), while QDA is preferred if the training set is very large (i.e., variance is no longer a great concern) or the assumption of a common covariance matrix is clearly unrealistic.

Naive Bayes

The naive Bayes classifier takes a different approach to those of LDA and QDA. Instead of assuming $f_1(x), \ldots, f_K(x)$ belong to a particular family of distributions, naive Bayes classifier assumes the p predictors are independent within a given class. Mathematically, this means for $k = 1, \ldots, K$,

$$f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times f_{kp}(x_p),$$

which greatly simplifies estimating $f_k(x)$ since we now assume there is no association between the p predictors. In most settings, we don't really believe the p predictors are independent within each class; however, naive Bayes often leads to decent results, especially in settings where n is not large enough relative to p for us to effectively estimate the joint distribution of the predictors within each class. This is the case because, while the naive Bayes assumption introduces some bias, it reduces variance. Thus, it works well as a result of the bias-variance trade-off.

Plugging our assumption into Bayes' theorem, we obtain

$$\Pr(Y = k|X = x) = \frac{\pi_k \times f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times f_{kp}(x_p)}{\sum_{l=1}^{K} \pi_l \times f_{l1}(x_1) \times f_{l2}(x_2) \times \cdots \times f_{lp}(x_p)}$$

Hence, it remains to obtain estimators for $f_{k1}, \ldots, f_{kp}$ using the training data. If X_j is qualitative, estimating f_{kj} is identical to how we estimate π_k as we just take the relative proportions of X_j within class k. If X_j is quantitative, there are two common approaches.

1. Assume $X_j|Y = k \sim \mathcal{N}(\mu_{jk}, \sigma_{jk}^2)$. In other words, assume each predictor is drawn from a univariate Gaussian distribution within each class. This is similar to QDA, but now we assume $\mathbf{\Sigma}_k$ is a diagonal matrix.
2. A non-parametric approach to estimating f_{kj} is making a histogram for the observations of the jth predictor within each class and using the proportion of the training observations in the kth class that belong to the same histogram bin as x_j. We may also use a **kernel density estimator**, which is essentially a smoothed version of a histogram.

Generally speaking, we expect to see a greater pay-off to using naive Bayes relative to LDA or QDA when p is large or n is small such that reducing the variance is very important.

Classifier Performance

- Logistic Regression: Comparatively, logistic regression may be preferred when n is small and/or p is large. It performs best when the Bayes decision boundary is linear in x. Logistic regression will compete with LDA in these circumstances but is preferred when the predictors are not normally distributed (within each class).
- Linear Discriminant Analysis: Much like logistic regression, LDA may be preferred when n is small and/or p is large. It also performs best when the Bayes decision boundary is linear in x. LDA will compete with logistic regression in these circumstances but is preferred when the predictors are normally distributed (within each class).
- Quadratic Discriminant Analysis: Generally speaking, QDA may be preferred when n is large and/or p is small. It'll tend to perform better than LDA and logistic regression when the Bayes decision boundary is non-linear in the predictors. However, QDA suffers greatly when the assumption regarding the normality of the predictors is broken.
- Naive Bayes: Generally speaking, naive Bayes may be preferred when n is small and/or p is large. It performs better than LDA and QDA when reducing variance is a key issue. It tends to perform best when variance is an issue but the Bayes decision boundary is non-linear in the predictors. It should be noted that naive Bayes suffers when the assumption regarding the independence between the predictors is broken.
- K-Nearest Neighbors (Cross-Validated): KNN is a non-parametric method. Thus, it generally requires a relatively large n and small p. KNN tends to perform best when the Bayes decision boundary is a complicated non-linear function of the predictors. Generally speaking, KNN will be used when the gains in unbiasedness outweigh any additional variance introduced by using this method.

Poisson Regression

In some cases, we deal with response variables that are neither qualitative nor quantitative. Instead, these variables take on **counts**, or non-negative integer values. While linear regression can be used in such situations,

linear regression is not ideal as it will give negative and non-integer predictions. Additionally, because of the mean-variance relationship with a count response variable (when the expected value is low, the variance also tends to be lower), the general assumption of homoscedastic errors does not hold. Using a natural logarithm transformation of the response can overcome much of the heteroscedasticity and obviate the possibility of negative predictions; however, such transformation make interpretation more difficult and cannot be used when the response can take on a value of zero. It turns out that **Poisson regression** is a better alternative.

Poisson regression relies on the **Poisson distribution**. If a random variable $Y \in \mathbb{N}$ follows the Poisson distribution, then

$$\Pr(Y = k) = \frac{e^{-\lambda}\lambda^k}{k!} \text{ for } k = 0, 1, 2 \ldots,$$

where $\lambda = E(Y) = \text{Var}(Y)$. In practical situations, rather than modeling Y as a Poisson distribution with a fixed mean for all values of the predictors, we would like λ to be a function of the predictors. Thus, we consider the following model for the mean

$$\log\left(\lambda(X_1, \ldots, X_p)\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

or equivalently

$$\lambda(X_1, \ldots, X_p) = e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}$$

In total, the Poisson regression model takes the form

$$\Pr(Y = k|X) = \frac{\exp\left(-e^{\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p}\right) e^{(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)k}}{k!}$$

Thus, we must obtain estimators of the coefficients $\beta_0, \ldots, \beta_p$, which we obtain using the same maximum likelihood approach used for logistic regression. In this setting, we maximize the likelihood function

$$L(\beta_0, \ldots, \beta_p) = \prod_{i=1}^{n} \frac{e^{-\lambda(x_i)}\lambda(x_i)^{y_i}}{y_i!},$$

where $\lambda(x_i) = e^{\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}}$.

Lab

- The `glm()` function can be used to fit many types of generalized linear models, including logistic regression. The syntax of the `glm()` function is similar to that of `lm()`, except that we must pass a `family` argument. For linear regression, use `family = gaussian`, for logistic regression use `family = binomial`, and for Poisson regression use `family = poisson`.
- With a fitted logistic regression model use the `predict(..., type = "response")` to get out probabilities of the form $P(Y = 1|X)$. Use the `contrasts()` command to gather the class codings `R` specified.
- Use the `lda() function` from the `MASS` library to fit an LDA model. The syntax is identical to that of the `lm()` function.
- Similarly, use the `qda() function` from the `MASS` library to fit a QDA model.
- Using the same syntax to that of the `lm()`, `lda()`, and `qda()` functions, use `naiveBayes()` from the `e1071` library to fit a naive Bayes model. By default, this implementation of the naive Bayes classifier models each quantitative feature using a Gaussian distribution, but a kernel density method can be used to estimate the distributions.
- To implement a KNN model, use the `knn()` function from the `class` library. This function requires four arguments: the training data, the test data, class labels, and a value for K.
- The `scale()` function standardizes data so that all variables are given a mean of zero and a standard deviation of one. Alternatively, we can use `gknn()` from the `e1071` library to implement a standardized KNN model using similar syntax to that of `naiveBayes()`.

Exercises

Conceptual

1. Using a little bit of algebra, prove that (4.2) is equivalent to (4.3). In other words, the logistic function representation and logit representation for the logistic regression model are equivalent.

$$\begin{aligned}\frac{p(X)}{1-p(X)} &= \frac{\frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}}{1-\frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}} \\ &= \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}-e^{\beta_0+\beta_1 x}} \\ &= e^{\beta_0+\beta_1 x}\end{aligned}$$

2. It was stated in the text that classifying an observation to the class for which (4.17) is largest is equivalent to classifying an observation to the class for which (4.18) is largest. Prove that this is the case. In other words, under the assumption that the observations in the kth class are drawn from a $\mathcal{N}(\mu_k, \sigma^2)$ distribution, the Bayes classifier assigns an observation to the class for which the discriminant function is maximized.

$$\begin{aligned}\log\left(\pi_k \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}\right) &= \log(\pi_k) - \log(\sigma\sqrt{2\pi}) + \log\left(e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}\right) \\ &= \log(\pi_k) - \log(\sigma\sqrt{2\pi}) - \frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2 \\ &= \log(\pi_k) - \log(\sigma) - \log(\sqrt{2\pi}) - \frac{1}{2}\left(\frac{x^2-2x\mu_k+\mu_k^2}{\sigma^2}\right) \\ &= \log(\pi_k) - \log(\sigma) - \log(\sqrt{2\pi}) - \frac{x^2}{2\sigma^2} + \frac{x\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2}\end{aligned}$$

   Since $\log(\sqrt{2\pi})$, $\log(\sigma)$, and $x^2/2\sigma^2$ do not vary with k (recall we are treating x as fixed), we obtain

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

3. This problem relates to the QDA model, in which the observations within each class are drawn from a normal distribution with a class-specific mean vector and a class specific covariance matrix. We consider the simple case where $p = 1$; i.e. there is only one feature.

   Suppose that we have K classes, and that if an observation belongs to the kth class then X comes from a one-dimensional normal distribution, $X \sim (\mu_k, \sigma_k^2)$. Recall that the density function for the one-dimensional normal distribution is given in (4.16). Prove that in this case, the Bayes classifier is not linear. Argue that it is in fact quadratic.

$$\begin{aligned}\log\left(\pi_k \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2}\right) &= \log(\pi_k) - \log(\sigma_k\sqrt{2\pi}) + \log\left(e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2}\right) \\ &= \log(\pi_k) - \log(\sigma_k\sqrt{2\pi}) - \frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2 \\ &= \log(\pi_k) - \log(\sigma_k) - \log(\sqrt{2\pi}) - \frac{1}{2}\left(\frac{x^2-2x\mu_k+\mu_k^2}{\sigma_k^2}\right) \\ &= \log(\pi_k) - \log(\sigma_k) - \log(\sqrt{2\pi}) - \frac{x^2}{2\sigma_k^2} + \frac{x\mu_k}{\sigma_k^2} - \frac{\mu_k^2}{2\sigma_k^2}\end{aligned}$$

Since $\log(\sqrt{2\pi})$ does not vary with k, we obtain

$$\delta_k(x) = x^2 \cdot -\frac{1}{2\sigma_k^2} + x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) - \log(\sigma_k)$$

4. When the number of features p is large, there tends to be a deterioration in the performance of KNN and other *local* approaches that perform prediction using only observations that are near the test observation for which a prediction must be made. This phenomenon is known as the *curse of dimensionality*, and it ties into the fact that non-parametric approaches often perform poorly when p is large. We will now investigate this curse.

   (a) Suppose that we have a set of observations, each with measurements on $p = 1$ feature, X. We assume that X is uniformly (evenly) distributed on $[0, 1]$. Associated with each observation is a response value. Suppose that we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation. For instance, in order to predict the response for a test observation with $X = 0.6$, we will use observations in the range $[0.55, 0.65]$. On average, what fraction of the available observations will we use to make the prediction?

   Considering X is uniformly distributed and we wish to use observations within 10% of X, on average, 1/10 of the available observations will be available for the prediction.

   I've disregarded $X \notin [0.05, 0.95]$ here and throughout the rest of the problem.

   (b) Now suppose that we have a set of observations, each with measurements on $p = 2$ features, X_1 and X_2. We assume that (X_1, X_2) are uniformly distributed on $[0, 1] \times [0, 1]$. We wish to predict a test observation's response using only observations that are within 10% of the range of X_1 and within 10% of the range of X_2 closest to that test observation. For instance, in order to predict the response for a test observation with $X_1 = 0.6$ and $X_2 = 0.35$, we will use observations in the range $[0.55, 0.65]$ for X_1 and in the range $[0.3, 0.4]$ for X_2. On average, what fraction of the available observations will we use to make the prediction?

   We can pictures this problem geometrically as a unit square with X_1 on the horizontal axis and X_2 on the vertical axis. In part (a), we found that 1/10 of the available observations can be used on average if we're interested in observations within 10% of X_1. Thus, we can imagine a vertical strip of that unit square that takes up one-tenth of the area. The same idea can be applied with X_2, where there is now a horizontal strip of the unit square that takes up one-tenth of the area. Thus, we're left with 1/100 of the unit square where those two strips overlap one another.

   (c) Now suppose that we have a set of observations on $p = 100$ features. Again the observations are uniformly distributed on each feature, and again each feature ranges in value from 0 to 1. We wish to predict a test observation's response using observations within the 10% of each feature's range that is closest to that test observation. What fraction of the available observations will we use to make the prediction?

   The same framework to that applied in parts (a) and (b) can be applied to this problem. It should be evident that, with uniformly distributed features where we wish to use observations within 10% of the test observation, we will have $(1/10)^p$ of the observations available to us on average. Thus, with $p = 100$, in the same scenario, we can expect to have $\frac{1}{10^{100}}$ of the observations available to us on average.

   (d) Using your answers to parts (a)-(c), argue that a drawback of KNN when p is large is that there are very few training observations "near" any given test observation.

   As stated in part (c), with uniformly distributed features where we wish to use observations within 10% of the test observation, we will have $(1/10)^p$ of the observations available to us on average. Thus, when choosing K, there will either be a sacrifice in the number of observations available to us or the distance between the observations must become larger.

(e) Now suppose that we wish to make a prediction for a test observation by creating a p-dimensional hypercube centered around the test observation that contains, on average, 10% of the training observations. For $p = 1, 2,$ and 100, what is the length of each side of the hypercube? Comment on your answer.

Note: A hypercube is a generalization of a cube to an arbitrary number of dimensions. When $p = 1$, a hypercube is simply a line segment, when $p = 2$ it is a square, and when $p = 100$ it is a 100-dimensional cube.

Assuming each feature is uniformly distributed on $[a, b]$, the length of each side of the hypercube is $\frac{1}{10(b-a)}$.

5. We now examine the differences between LDA and QDA.

(a) If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?

It's likely that QDA performs better on the training set due to its greater flexibility. However, we would certainly expect LDA to perform better on the test set because there is no loss in biasedness (the decision boundary is truly linear) but there are large gains in reduced variance to that of QDA.

(b) If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?

Once again, it's likely that QDA performs better on the training set due to its greater flexibility. However, performance of the two models in the test set depends on several factors. LDA will suffer from unbiasedness in comparison to QDA, but QDA will suffer from greater variance. Thus, if gains in unbiasedness from using QDA exceed any additional variance introduced, then QDA should perform better on the test set. Otherwise, we would expect LDA to perform better.

(c) In general, as the sample size n increases, do we expect the test prediction accuracy of QDA relative to LDA to improve, decline, or be unchanged? Why?

Generally, as n increases, we expect the test prediction accuracy of QDA to improve in comparison to that of LDA. This is the circumstance because greater sample sizes work to reduce variance. Thus, QDA suffers less from additional variance but may provide gains in unbiasedness when compared to LDA.

(d) True or False: Even if the Bayes decision boundary for a given problem is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is flexible enough to model a linear decision boundary. Justify your answer.

False. As stated in part (a), we expect LDA to perform better on the test set because there is no loss in biasedness (the decision boundary is truly linear), but there are large gains in reduced variance.

6. Suppose we collect data for a group of students in a statistics class with variables $X_1 =$ hours studied, $X_2 =$ undergrad GPA, and $Y =$ receive an A. We fit a logistic regression and produce estimated coefficient, $\hat{\beta}_0 = -6, \hat{\beta}_1 = 0.05, \hat{\beta}_2 = 1$.

(a) Estimate the probability that a student who studies for 40 h and has an undergrad GPA of 3.5 gets an A in the class.

$$\hat{p}(X) = \frac{1}{1 + e^{-(-6+0.05\cdot 40+1\cdot 3.5)}} \approx 0.3775$$

(b) How many hours would the student in part (a) need to study to have a 50% chance of getting an A in the class?

$$\frac{\hat{p}(X)}{1-\hat{p}(X)} = e^{-6+0.05\cdot X_1+1\cdot 3.5} = 1$$
$$\ln(e^{-6+0.05\cdot X_1+1\cdot 3.5}) = \ln(1)$$
$$-6 + 0.05 \cdot X_1 + 1 \cdot 3.5 = 0$$
$$X_1 = \frac{2.5}{0.05} = 50$$

7. Suppose that we wish to predict whether a given stock will issue a dividend this year ("Yes" or "No") based on X, last year's percent profit. We examine a large number of companies and discover that the mean value of X for companies that issued a dividend was $\bar{X} = 10$, while the mean for those that didn't was $\bar{X} = 0$. In addition, the variance of X for these two sets of companies was $\hat{\sigma}^2 = 36$. Finally, 80% of companies issued dividends. Assuming that X follows a normal distribution, predict the probability that a company will issue a dividend this year given that its percentage profit was $X = 4$ last year.

$$\widehat{\Pr}(Y = 1|X = 4) = \frac{\hat{\pi}_1\hat{f}_1(4)}{\sum_{l=0}^{1}\hat{\pi}_l\hat{f}_l(4)}$$
$$= \frac{0.8\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(4-\bar{X})^2/2\sigma^2}\right)}{0.8\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(4-\bar{X})^2/2\sigma^2}\right) + 0.2\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(4-\bar{X})^2/2\sigma^2}\right)}$$
$$= \frac{0.8\left(e^{-(4-10)^2/72}\right)}{0.8\left(e^{-(4-10)^2/72}\right) + 0.2\left(e^{-(4-0)^2/72}\right)}$$
$$\approx 0.7518525$$

8. Suppose that we take a data set, divide it into equally-sized training and test sets, and then try out two different classification procedures. First we use logistic regression and get an error rate of 20% on the training data and 30% on the test data. Next we use 1-nearest neighbors (i.e. $K = 1$) and get an average error rate (averaged over both test and training data sets) of 18%. Based on these results, which method should we prefer to use for classification of new observations? Why?

   When $K = 1$, the training error rate for KNN is 0%. Thus, if KNN gives an average error rate (over both the training and test data) of 18% with $K = 1$, then the test error rate must be 36%. Assuming we're interested in minimizing the test error rate since we're interested in accurate predictions for new observations, we should prefer the logistic regression model over the KNN model.

9. This problem has to do with *odds*.

   (a) On average, what fraction of people with an odds of 0.37 of defaulting on their credit card payment will in fact default?

$$\frac{p(X)}{1-p(X)} = 0.37$$
$$p(X) = 0.37 - 0.37p(X)$$
$$p(X) = \frac{0.37}{1.37} \approx 0.2701$$

   (b) Suppose that an individual has a 16% chance of defaulting on her credit card payment. What are the odds that she will default?

$$\frac{p(X)}{1-p(X)} = \frac{0.16}{0.84} \approx 0.1905$$

10. Equation 4.32 derived an expression for $\log(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)})$ in the setting where $p > 1$, so that the mean for the kth class, μ_k, is a p-dimensional vector, and the shared covariance matrix $\mathbf{\Sigma}$ is a $p \times p$ matrix. However, in the setting with $p = 1$, (4.32) takes a simpler form, since the means $\mu_1, \ldots, \mu_k$ and the variance σ^2 are scalars. In this simpler setting, repeat the calculation in (4.32), and provide expressions for a_k and b_{kj} in terms of $\pi_k, \pi_K, \mu_k, \mu_K$, and σ^2.

$$\begin{aligned}
\log\left(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)}\right) &= \log\left(\frac{\pi_k f_k(x)}{\pi_K f_K(x)}\right) \\
&= \log\left(\frac{\pi_k \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{\pi_K \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_K)^2\right)}\right) \\
&= \log\left(\frac{\pi_k}{\pi_K}\right) - \frac{1}{2\sigma^2}(x-\mu_k)^2 + \frac{1}{2\sigma^2}(x-\mu_K)^2 \\
&= \log\left(\frac{\pi_k}{\pi_K}\right) - \frac{1}{2\sigma^2}\left(x^2 - 2\mu_k x + \mu_k^2 - x^2 + 2\mu_K x - \mu_K^2\right) \\
&= \log\left(\frac{\pi_k}{\pi_K}\right) + \frac{1}{2\sigma^2}\left(\mu_K^2 - \mu_k^2\right) + \frac{1}{\sigma^2}(\mu_k - \mu_K)x \\
&= a_k + b_k x,
\end{aligned}$$

where $a_k = \log\left(\frac{\pi_k}{\pi_K}\right) + \frac{1}{2\sigma^2}\left(\mu_K^2 - \mu_k^2\right)$ and $b_k = \frac{1}{\sigma^2}(\mu_k - \mu_K)$.

11. Work out the detailed form of a_k, b_{kj}, and b_{kjl} in (4.33). Your answer should involve $\pi_k, \pi_K, \mu_k, \mu_K, \mathbf{\Sigma}_k$, and $\mathbf{\Sigma}_K$.

$$\begin{aligned}
\log\left(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)}\right) &= \log\left(\frac{\pi_k f_k(x)}{\pi_K f_K(x)}\right) \\
&= \log\left(\frac{\pi_k \exp\left(-\frac{1}{2}(x-\mu_k)^T \mathbf{\Sigma}_k^{-1}(x-\mu_k)\right)}{\pi_K \exp\left(-\frac{1}{2}(x-\mu_K)^T \mathbf{\Sigma}_K^{-1}(x-\mu_k)\right)}\right) \\
&= \log\left(\frac{\pi_k}{\pi_K}\right) - \frac{1}{2}(x-\mu_k)^T \mathbf{\Sigma}_k^{-1}(x-\mu_k) + \frac{1}{2}(x-\mu_K)^T \mathbf{\Sigma}_K^{-1}(x-\mu_k) \\
&= \log\left(\frac{\pi_k}{\pi_K}\right) - \frac{1}{2}(x^T - \mu_k^T)(\mathbf{\Sigma}_k^{-1}x - \mathbf{\Sigma}_k^{-1}\mu_k) + \frac{1}{2}(x^T - \mu_K^T)(\mathbf{\Sigma}_K^{-1}x - \mathbf{\Sigma}_K^{-1}\mu_k) \\
&= \log\left(\frac{\pi_k}{\pi_K}\right) - \frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}x + \frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}\mu_k + \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}x - \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k + \frac{1}{2}x^T\mathbf{\Sigma}_K^{-1}x - \frac{1}{2}x^T\mathbf{\Sigma}_K^{-1}\mu_K - \frac{1}{2}\mu_K^T\mathbf{\Sigma}_K^{-1}x + \frac{1}{2}\mu_K^T\mathbf{\Sigma}_K^{-1}\mu_K \\
&= \log\left(\frac{\pi_k}{\pi_K}\right) + \frac{1}{2}\left(\mu_K^T\mathbf{\Sigma}_K^{-1}\mu_K - \mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k\right) + x^T\left(\mathbf{\Sigma}_k^{-1}\mu_k - \mathbf{\Sigma}_K^{-1}\mu_K\right) + \frac{1}{2}x^T\left(\mathbf{\Sigma}_K^{-1} - \mathbf{\Sigma}_k^{-1}\right)x
\end{aligned}$$

12. Suppose that you wish to classify an observation $X \in \mathbb{R}$ into `apples` and `oranges`. You fit a logistic regression model and find that

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

Your friend fits a logistic regression model to the same data using the `softmax` formulation in (4.13), and finds that

$$\widehat{\Pr}(Y = \texttt{orange}|X = x) = \frac{\exp(\hat{\alpha}_{\texttt{orange0}} + \hat{\alpha}_{\texttt{orange1}} x)}{\exp(\hat{\alpha}_{\texttt{orange0}} + \hat{\alpha}_{\texttt{orange1}} x) + \exp(\hat{\alpha}_{\texttt{apple0}} + \hat{\alpha}_{\texttt{apple1}} x)}$$

.

(a) What is the log odds of `orange` versus `apple` in your model?

$$\ln\left(\frac{\widehat{\Pr}(Y=\texttt{orange}|X=x)}{\widehat{\Pr}(Y=\texttt{apple}|X=x)}\right)=\ln\left(\frac{\widehat{\Pr}(Y=\texttt{orange}|X=x)}{1-\widehat{\Pr}(Y=\texttt{orange}|X=x)}\right)$$

$$=\ln\left(\frac{\frac{\exp(\hat{\beta}_0+\hat{\beta}_1x)}{1+\exp(\hat{\beta}_0+\hat{\beta}_1x)}}{1-\frac{\exp(\hat{\beta}_0+\hat{\beta}_1x)}{1+\exp(\hat{\beta}_0+\hat{\beta}_1x)}}\right)$$

$$=\ln\left(\exp(\hat{\beta}_0+\hat{\beta}_1x)\right)$$

$$=\hat{\beta}_0+\hat{\beta}_1x$$

(b) What is the log odds of orange versus apple in your friend's model?

$$\ln\left(\frac{\widehat{\Pr}(Y=\texttt{orange}|X=x)}{\widehat{\Pr}(Y=\texttt{apple}|X=x)}\right)=\ln\left(\frac{\frac{\exp(\hat{\alpha}_{\texttt{orange0}}+\hat{\alpha}_{\texttt{orange1}}x)}{\exp(\hat{\alpha}_{\texttt{orange0}}+\hat{\alpha}_{\texttt{orange1}}x)+\exp(\hat{\alpha}_{\texttt{apple0}}+\hat{\alpha}_{\texttt{apple1}}x)}}{\frac{\exp(\hat{\alpha}_{\texttt{apple0}}+\hat{\alpha}_{\texttt{apple1}}x)}{\exp(\hat{\alpha}_{\texttt{orange0}}+\hat{\alpha}_{\texttt{orange1}}x)+\exp(\hat{\alpha}_{\texttt{apple0}}+\hat{\alpha}_{\texttt{apple1}}x)}}\right)$$

$$=\ln\left(\frac{\exp(\hat{\alpha}_{\texttt{orange0}}+\hat{\alpha}_{\texttt{orange1}}x)}{\exp(\hat{\alpha}_{\texttt{apple0}}+\hat{\alpha}_{\texttt{apple1}}x)}\right)$$

$$=(\hat{\alpha}_{\texttt{orange0}}-\hat{\alpha}_{\texttt{apple0}})+(\hat{\alpha}_{\texttt{orange1}}-\hat{\alpha}_{\texttt{apple1}})x$$

(c) Suppose that in your model, $\hat{\beta}_0=2$ and $\hat{\beta}_1=-1$. What are the coefficient estimates in your friend's model? Be as specific as possible.

$\hat{\alpha}_{\texttt{orange0}}-\hat{\alpha}_{\texttt{apple0}}=2$ and $\hat{\alpha}_{\texttt{orange1}}-\hat{\alpha}_{\texttt{apple1}}=-1$.

(d) Now suppose that you and your friend fit the same two models on a different data set. This time, your friend gets the coefficient estimates $\hat{\alpha}_{\texttt{orange0}}=1.2, \hat{\alpha}_{\texttt{orange1}}=-2, \hat{\alpha}_{\texttt{apple0}}=3, \hat{\alpha}_{\texttt{apple1}}=0.6$. What are the coefficient estimates in your model?

$\hat{\beta}_0=-1.8$ and $\hat{\beta}_1=-2.6$.

(e) Finally, suppose you apply both models from (d) to a data set with 2,000 test observations. What fraction of the time do you expect the predicted class labels from your model to agree with those from your friend's model? Explain your answer.

I would expect the predicted class labels to always be the same between the two models. From the text: "The softmax coding is equivalent to the coding just described in the sense that the fitted values, log odds between any pair of classes, and other key model outputs will remain the same, regardless of coding."

Applied

```
table(Weekly_dt[["Year"]])
```

```
1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
  47   52   52   52   52   52   53   52   52   52   52   52   52   52   52   52
2006 2007 2008 2009 2010
  52   53   52   52   52
```

```
sapply(ss(Weekly_dt, , 2:8), summary)
```

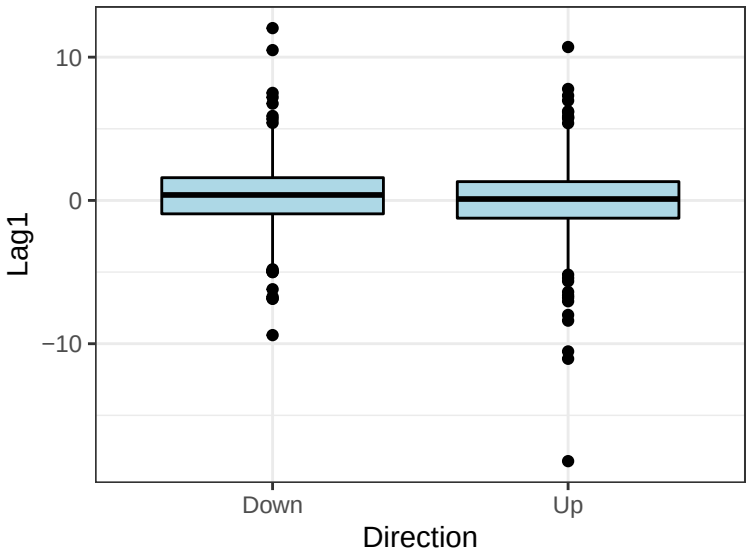
```
              Lag1       Lag2        Lag3        Lag4        Lag5   Volume
Min.    -18.1950000 -18.195000 -18.1950000 -18.1950000 -18.1950000 0.087465
1st Qu.  -1.1540000  -1.154000  -1.1580000  -1.1580000  -1.1660000 0.332022
Median    0.2410000   0.241000   0.2410000   0.2380000   0.2340000 1.002680
Mean      0.1505849   0.151079   0.1472048   0.1458182   0.1398926 1.574618
3rd Qu.   1.4050000   1.409000   1.4090000   1.4090000   1.4050000 2.053727
Max.     12.0260000  12.026000  12.0260000  12.0260000  12.0260000 9.328214
             Today
Min.    -18.195000
1st Qu.  -1.154000
Median    0.241000
Mean      0.149899
3rd Qu.   1.405000
Max.     12.026000
```

```
table(Weekly_dt[["Direction"]])
```

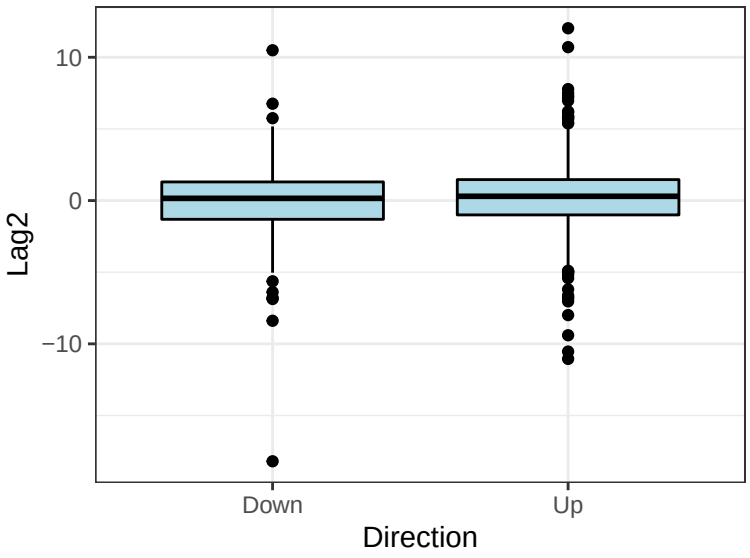
```
Down   Up
 484  605
```

```
lapply(colnames(ss(Weekly_dt, , 2:8)),
       function(variable)
           ggplot(Weekly_dt, aes_string("Direction", variable)) +
           geom_boxplot(color = "black", fill = "lightblue") +
           theme_bw())
```

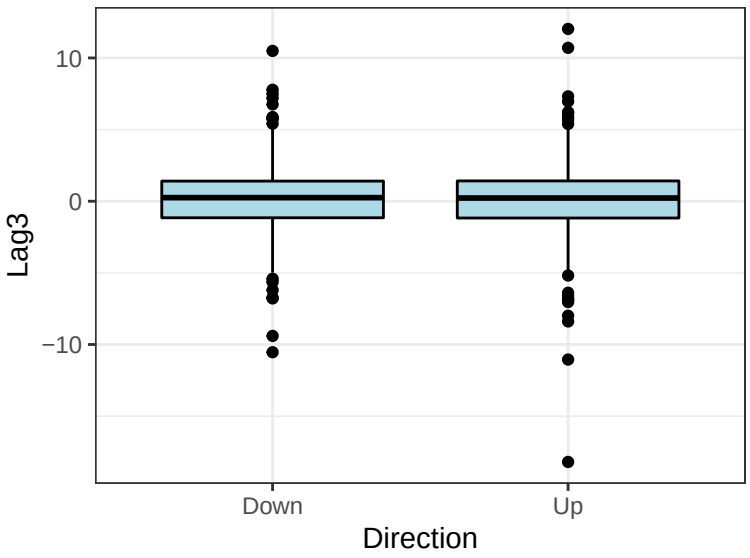
```
[[1]]
```



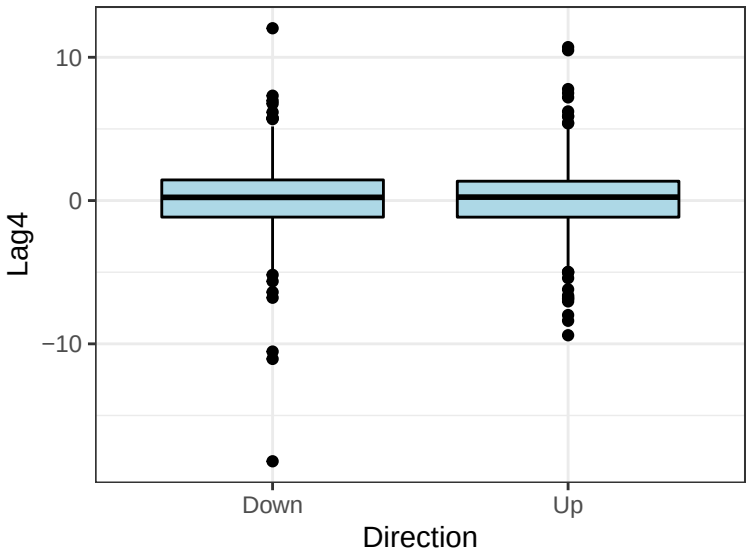

```
[[2]]
```



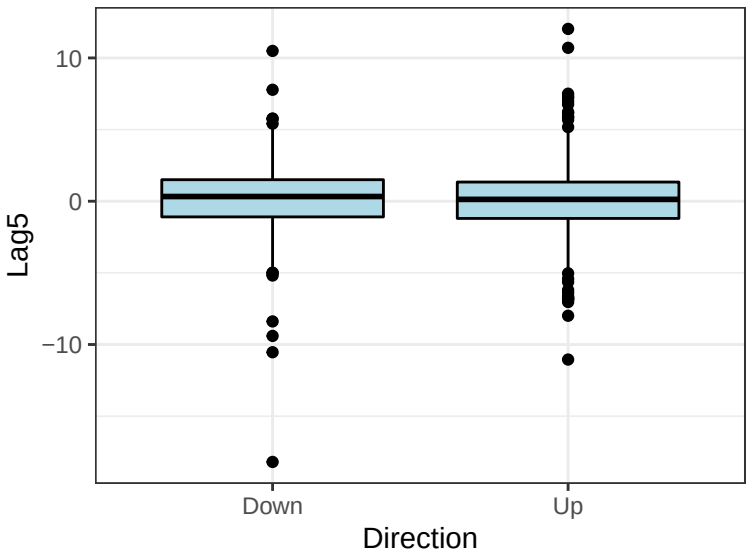

```
[[3]]
```



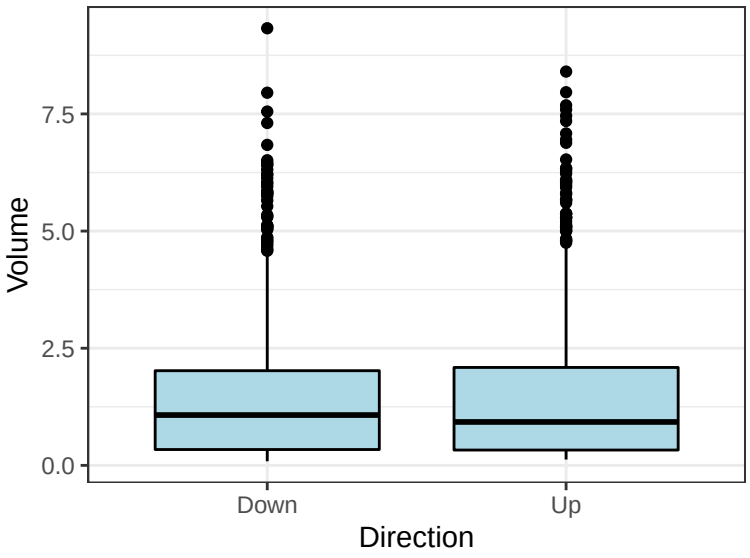

```
[[4]]
```



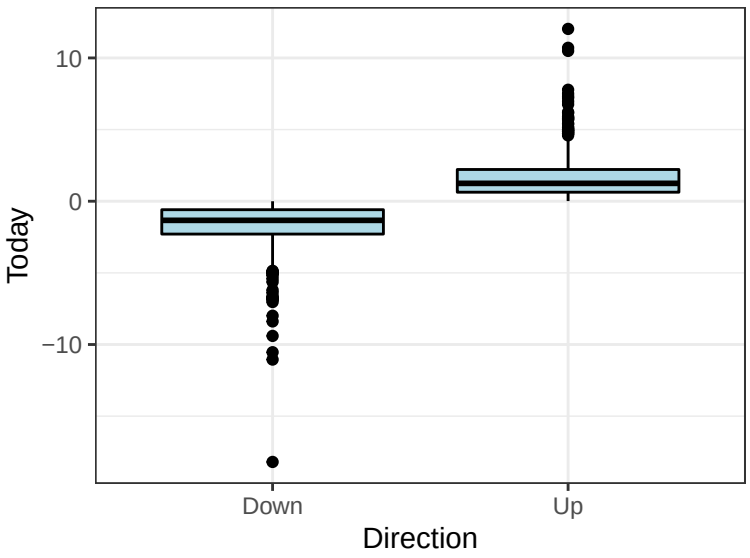

```
[[5]]
```




```
[[6]]
```




```
[[7]]
```




```
logistic.model <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume, data = Weekly_dt, family =

summary(logistic.model)
```

```
Call:
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
    Volume, family = binomial, data = Weekly_dt)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6949  -1.2565   0.9913   1.0849   1.4579

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.26686    0.08593   3.106   0.0019 **
```

```
Lag1        -0.04127    0.02641  -1.563   0.1181
Lag2         0.05844    0.02686   2.175   0.0296 *
Lag3        -0.01606    0.02666  -0.602   0.5469
Lag4        -0.02779    0.02646  -1.050   0.2937
Lag5        -0.01447    0.02638  -0.549   0.5833
Volume      -0.02274    0.03690  -0.616   0.5377
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1496.2  on 1088  degrees of freedom
Residual deviance: 1486.4  on 1082  degrees of freedom
AIC: 1500.4

Number of Fisher Scoring iterations: 4
```

```
table(fifelse(predict(logistic.model, type = "response") > 0.5, "Up", "Down"), Weekly_dt[["Direction"]])
```

```
      Down  Up
  Down   54  48
  Up    430 557
```

```
logistic.model <- glm(Direction ~ Lag2, data = Weekly_dt[Year < 2009, ], family = binomial)

table(fifelse(predict(logistic.model, newdata = Weekly_dt[Year > 2008, ], type = "response") > 0.5, "Up
```

```
      Down Up
  Down    9  5
  Up     34 56
```

```
lda.model <- lda(Direction ~ Lag2, data = Weekly_dt[Year < 2009, ])

table(predict(lda.model, newdata = Weekly_dt[Year > 2008, ], type = "response")$class, Weekly_dt[Year >
```

```
      Down Up
  Down    9  5
  Up     34 56
```

```
qda.model <- qda(Direction ~ Lag2, data = Weekly_dt[Year < 2009, ])

table(predict(qda.model, newdata = Weekly_dt[Year > 2008, ], type = "response")$class, Weekly_dt[Year >
```

```
      Down Up
  Down    0  0
  Up     43 61
```

```
naiveBayes.model <- naiveBayes(Direction ~ Lag2, data = Weekly_dt[Year < 2009, ])

table(predict(naiveBayes.model, newdata = Weekly_dt[Year > 2008, ], type = "class"), Weekly_dt[Year > 2
```

```
      Down Up
  Down    0  0
```

```
  Up     43 61
```

```
knn.model <- gknn(Direction ~ Lag2, data = Weekly_dt[Year < 2009, ], k = 1)

table(predict(knn.model, newdata = Weekly_dt[Year > 2008, ], type = "class"), Weekly_dt[Year > 2008, Di
```

```
       Down Up
  Down   21 29
  Up     22 32
```

```
knn.model <- update(knn.model, k = 5)

table(predict(knn.model, newdata = Weekly_dt[Year > 2008, ], type = "class"), Weekly_dt[Year > 2008, Di
```

```
       Down Up
  Down   15 22
  Up     28 39
```

```
knn.model <- update(knn.model, k = 10)

table(predict(knn.model, newdata = Weekly_dt[Year > 2008, ], type = "class"), Weekly_dt[Year > 2008, Di
```

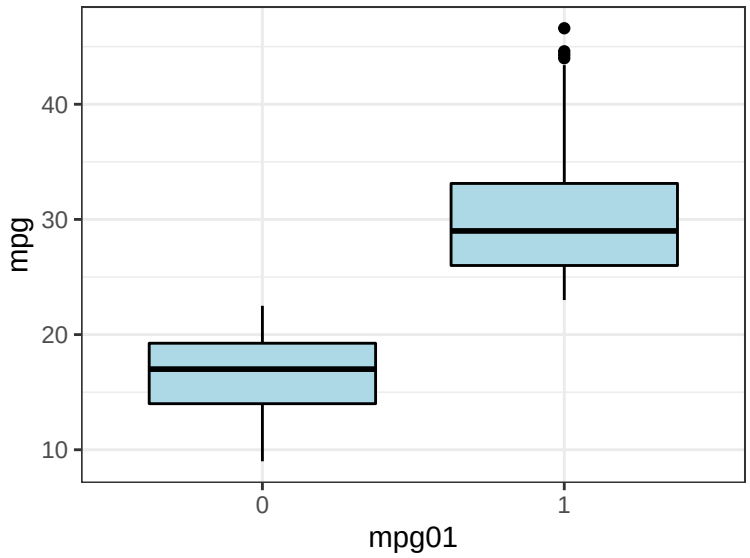
```
       Down Up
  Down   18 22
  Up     25 39
```

```
knn.model <- update(knn.model, k = 15)

table(predict(knn.model, newdata = Weekly_dt[Year > 2008, ], type = "class"), Weekly_dt[Year > 2008, Di
```

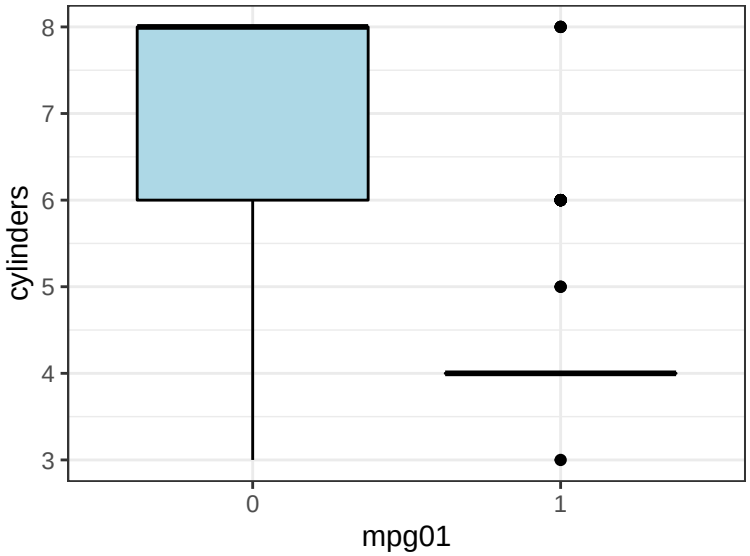
```
       Down Up
  Down   20 20
  Up     23 41
```

```
Auto_dt[, mpg01 := as.factor(as.integer(Auto_dt[["mpg"]] > fmedian(Auto_dt[["mpg"]])))]

lapply(colnames(Auto_dt[, .SD, .SDcols = is.numeric]),
       function(variable)
           ggplot(Auto_dt, aes_string("mpg01", variable)) +
           geom_boxplot(color = "black", fill = "lightblue") +
           theme_bw())
```

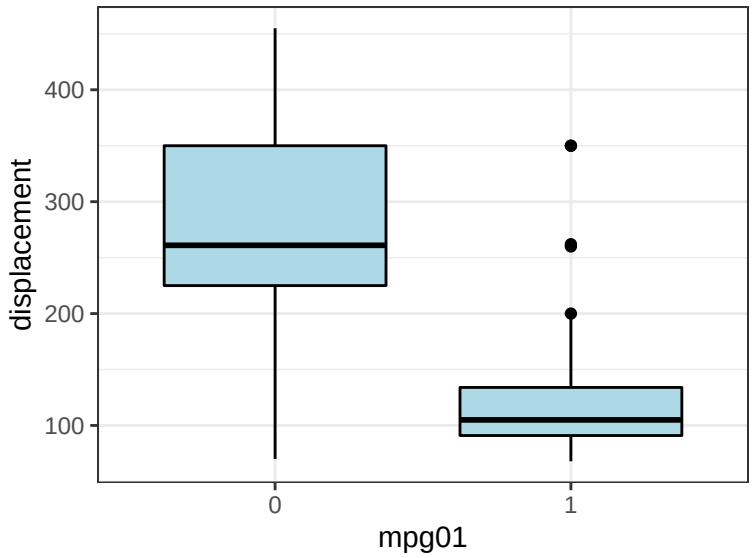
```
[[1]]
```



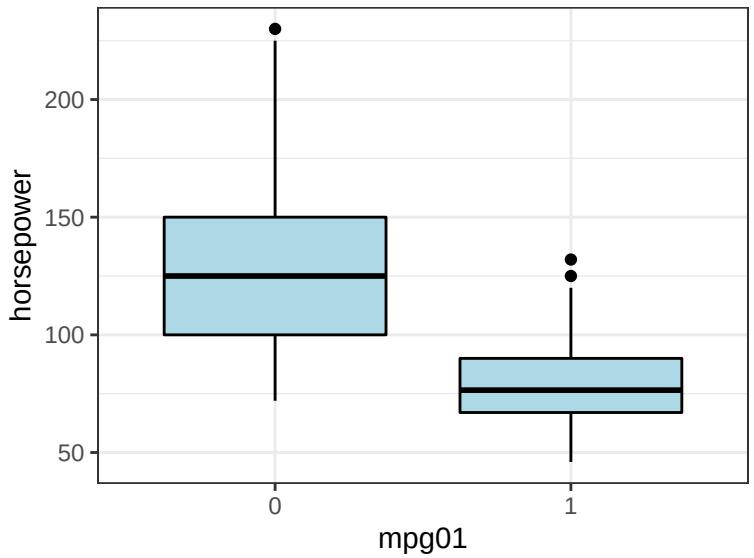
[[2]]



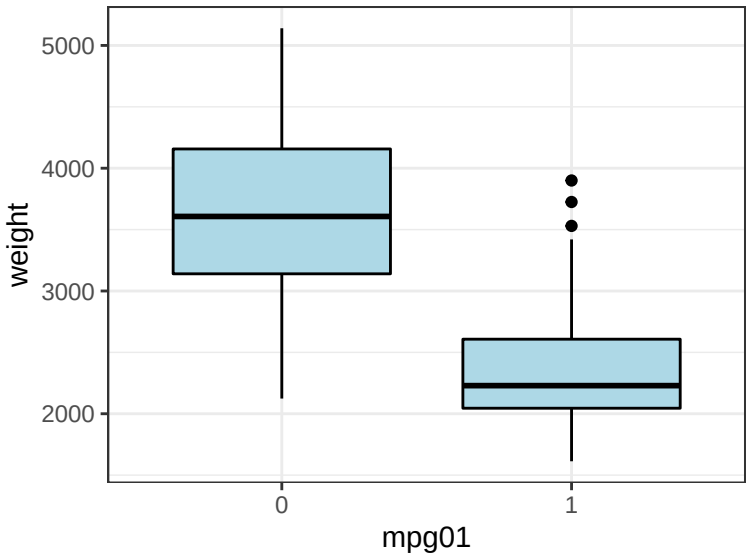
[[3]]


displacement
400
300
200
100
0
1
mpg01

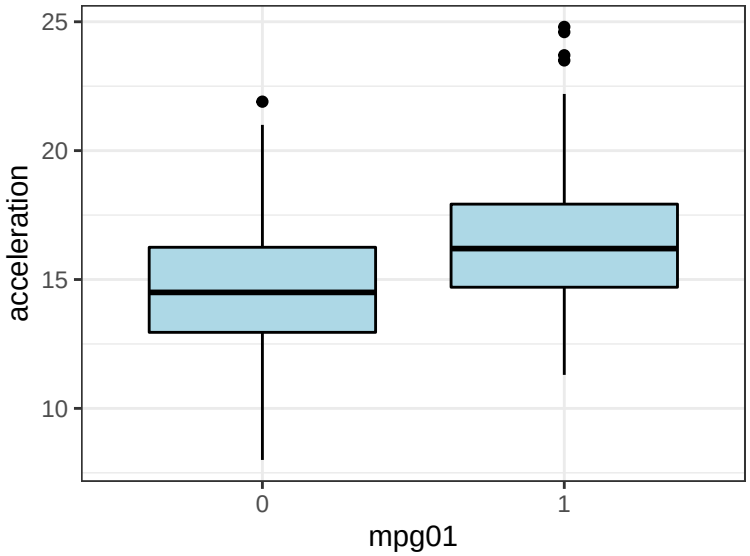
```
[[4]]
```


horsepower
200
150
100
50
0
1
mpg01

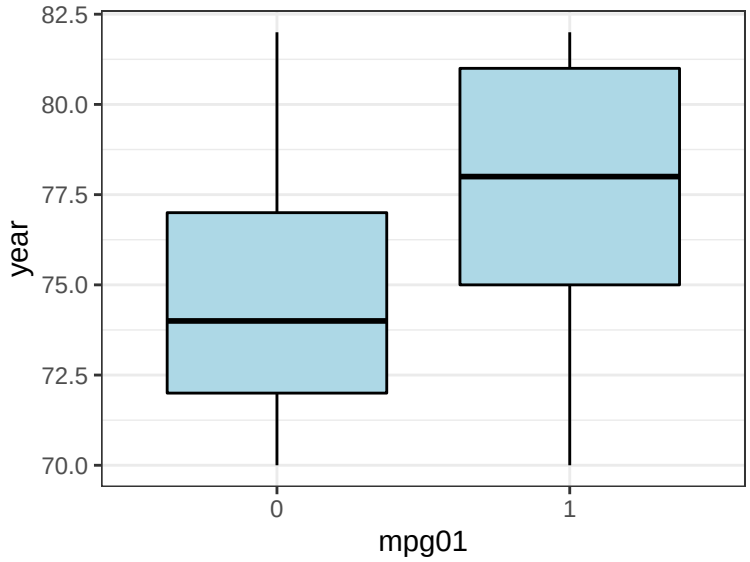
```
[[5]]
```


5000
4000
3000
2000
weight
0
1
mpg01

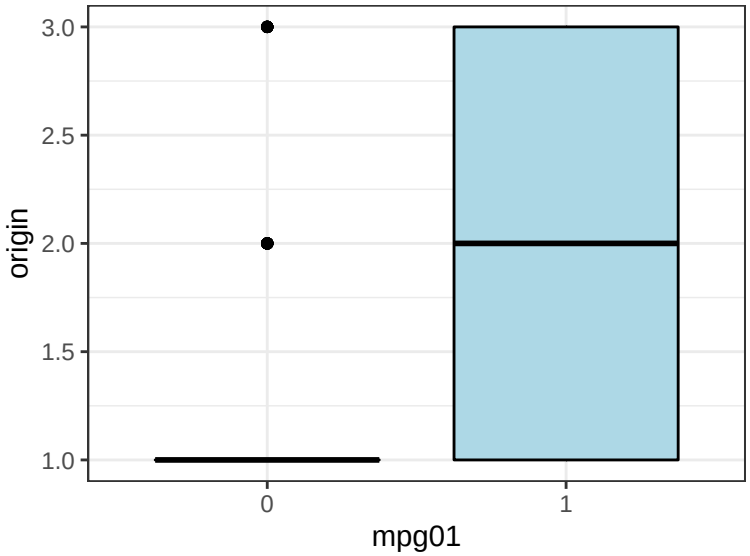
```
[[6]]
```


25
20
15
10
acceleration
0
1
mpg01

```
[[7]]
```



[[8]]




```
set.seed(123)

training_ind <- sample(seq_row(Auto_dt), size = floor(0.75 * fnrow(Auto_dt)))

lda.model <- lda(mpg01 ~ cylinders + displacement + horsepower + weight, data = Auto_dt, subset = traini

mean(Auto_dt[-training_ind, mpg01] != predict(lda.model, newdata = Auto_dt[-training_ind,], type = "resp
```

```
[1] 0.1122449
```

```
qda.model <- qda(mpg01 ~ cylinders + displacement + horsepower + weight, data = Auto_dt, subset = traini

mean(Auto_dt[-training_ind, mpg01] != predict(qda.model, newdata = Auto_dt[-training_ind,], type = "resp
```

```
[1] 0.1020408
```

```
logistic.model <- glm(mpg01 ~ cylinders + displacement + horsepower + weight, data = Auto_dt, subset = t

mean(Auto_dt[-training_ind, mpg01] != predict(logistic.model, newdata = Auto_dt[-training_ind,], type =
```

```
[1] 1
```

```
best_error <- 1

for(K in seq_row(Auto_dt[-training_ind, ])){
  curr_error <- mean(knn(Auto_dt[training_ind, c("cylinders", "displacement", "horsepower", "weight")],
  if(curr_error < best_error){
    best_K <- K
    best_error <- curr_error
  }
}
best_K
```

```
[1] 12
```

```
best_error
```

```
[1] 0.09183673
```

```
Power <- function(){
  print(2^3)
}

Power()
```

```
[1] 8
```

```
Power2 <- function(x, a){
  print(x^a)
}

Power2(3, 8)
```

```
[1] 6561
```

```
Power2(10, 3)
```

```
[1] 1000
```

```
Power2(8, 17)
```

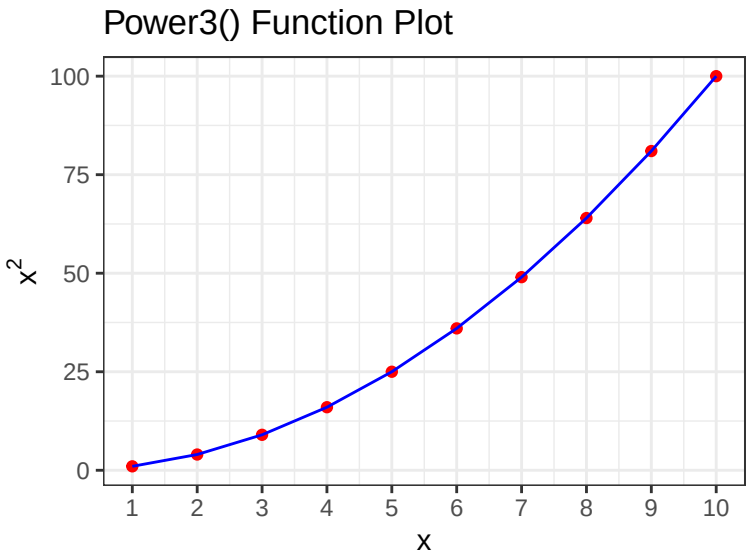
```
[1] 2251799813685248
```

```
Power2(131, 3)
```

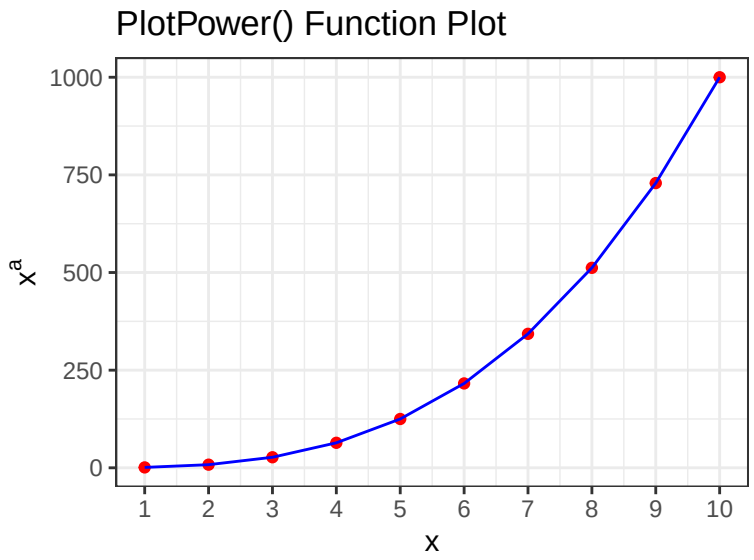
```
[1] 2248091
```

```
Power3 <- function(x, a){
  return(x^a)
}

x <- seq_len(10L)
y <- Power3(x, 2)

ggplot(mapping = aes(x,y)) +
  geom_point(color = "red") +
```

```
  geom_line(color = "blue") +
  theme_bw() +
  labs(title = "Power3() Function Plot",
       x = "x",
       y = expression(x^2)) +
  scale_x_continuous(breaks = seq_len(10L))
```




```
PlotPower <- function(x, a){
  ggplot(mapping = aes(x, x^a)) +
  geom_point(color = "red") +
  geom_line(color = "blue") +
  theme_bw() +
  labs(title = "PlotPower() Function Plot",
       x = "x",
       y = expression(x^a)) +
  scale_x_continuous(breaks = x)
}

PlotPower(seq_len(10L), 3)
```


PlotPower() Function Plot
1000
750
500
250
0
x^a
1
2
3
4
5
6
7
8
9
10
x

Chapter 5

Notes

Resampling Methods

Resampling methods involve repeatedly drawing samples from a training set and refitting a model of interest in order to obtain additional information about the fitted model. Two of the most common resampling methods are *cross-validation* and the *bootstrap*.

Cross-Validation

One technique for estimating a test error rate is the **validation set approach** in which we randomly divide the available set of observations into two parts, a **training set** and a **validation** (or **hold-out**) **set**. The mode is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set. The resulting validation set error rate provides an estimate of the test error rate. Unfortunately, the validation set approach suffers from high degrees of variability in the test error rate. Another shortcoming to the validation set approach is that only a subset of the observations are used to fit the model. Since statistical methods tend to perform worse when trained on fewer observations, this suggests that the validation set error rate may tend to overestimate the test error rate for the model fit on the entire data set.

Cross-validation is a refinement of the validation set approach that addresses these two issues and can be used to estimate the test error associated with a given statistical learning method in order to evaluate its performance or to select the appropriate level of flexibility. **Leave-one-out cross validation** (**LOOCV**) is closely related to the validation set approach, but it attempts to overcome the previously discussed drawbacks. LOOCV splits the observations into a training set of $n-1$ observations and a single test observation. This process is repeated for each n observations and provides an estimate for the test MSE through the equation

$$\text{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\text{MSE}_i = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where MSE_i is the squared error between the ith actual and predicted response values. This method of estimating the test error rate significantly alleviates both drawbacks of the validation set approach. First, it has far less bias since $n-1$ observations are used to train the model instead of $n/2$. Second, variability that's introduced by randomly dividing the observations into two sets is removed since there is no longer any randomness in the training/validation set splits. While it can be computationally expensive to implement this approach by having to fit n different models, this obstacle is overcome using the formula

$$\text{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1-h_i}\right)^2$$

in the case of least squares regression, where $\hat{y}_i$ is the ith fitted value from the model fit with all n observations and h_i is the previously defined leverage, which equals $\frac{1}{n} + \frac{(x_i-\bar{x})^2}{\sum_{i'=1}^{n}(x_{i'}-\bar{x})^2}$. Note that this is the same as the original MSE but the ith residual is divided by $0 < 1-h_i < 1$. Hence, high-leverage points are appropriately inflated.

A generalization of LOOCV is ***k*-fold CV**. This approach involves randomly dividing the set of observations into k roughly equal folds—or groups. One fold is treated as a validation fold, while the other $k-1$ folds are used as training folds. This process is repeated k times and the MSE estimate is given by

$$\text{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\text{MSE}_i$$

It's easy to see that LOOCV is a special case of k-fold CV in which $k = n$. In practice, k is generally set to 5 or 10. While adding some randomness back that LOOCV had removed, k-fold CV (with $k < n$) is much less computationally expensive, while often producing very similar but more accurate results to those of LOOCV.

Using LOOCV and k-fold CV, it's generally impossible to tell whether their test error estimates over or underestimate the true test error. For the purpose of model performance, this is an unfortunate reality. On the other hand, for the purpose of model specification, this is of little importance. Since the minimum points for the actual and estimated test error rates tend to be relatively close to each other, these techniques provide valuable guidance in selecting the most accurate model specification.

The Bootstrap

The **bootstrap** is a statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method. For an example, suppose there are two investments that yield returns X and Y in which we want to invest fractions of our fixed investment α and $1-\alpha$ respectively. If we want to minimize the variance of this portfolio, we calculate the following:

$$\text{Var}\left(\alpha X + (1-\alpha)Y\right) = \alpha^2\sigma_X^2 + 2\alpha(1-\alpha)\sigma_{XY} + (1-\alpha)^2\sigma_Y^2$$

$$\rightarrow \frac{\partial \text{Var}\left(\alpha X + (1-\alpha)Y\right)}{\partial \alpha} = 2\alpha\sigma_X^2 + 2\sigma_{XY} - 4\alpha\sigma_{XY} - 2\sigma_Y^2 + 2\alpha\sigma_Y^2 = 0$$

$$\alpha\sigma_X^2 - 2\alpha\sigma_{XY} + \alpha\sigma_Y^2 = \sigma_Y^2 - \sigma_{XY}$$

$$\alpha^* = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 - 2\sigma_{XY} + \sigma_Y^2}$$

Since we don't actually know the population parameters $\sigma_X^2, \sigma_{XY}, \sigma_Y^2$, we can use method of moments estimation to calculate

$$\hat{\alpha}^* = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 - 2\hat{\sigma}_{XY} + \hat{\sigma}_Y^2}$$

It's natural to want to know the variability related to $\hat{\alpha}^*$. Using the bootstrap method, we can estimate this uncertainty by repeatedly resampling n observations *with replacement* from the original data set of n observations. Thus, it's possible that an observation i is selected multiple times and an observation j is not selected for each bootstrap data set. This process is repeated B times, for some large value B, in order to produce B bootstrap data sets and B corresponding α estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$. Hence, we can compute the standard error of these estimates using

$$\text{SE}_B(\hat{\alpha}^*) = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}\left(\hat{\alpha}^{*i} - \frac{1}{B}\sum_{i'=1}^{B}\hat{\alpha}^{*i'}\right)^2},$$

which serves as an estimate for the standard error of $\hat{\alpha}^*$ from the original data set.

Lab

- The command `sample(n, m)` generates `m` integers ranging from 1 to `n`, which can be used as indices for splitting data into the training and test observations. The argument `replace = TRUE` will allow for replaced sampling.
- After creating a `glm` object (here `glm.fit`), we can use the command `cv.glm(data, glm.fit)` from the `boot` library to perform LOOCV. The `delta` vector contains the cross-validation results. Similarly we can use the command `cv.glm(data, glm.fit, K = k)` to perform k-fold CV.
- The command `boot(data, function, R = B)` from the `boot` library automates bootstrapping for B estimates. Here `function` is a function that defines the estimator and returns estimates.

Exercises

Conceptual

1. Using basic statistical properties of the variance, as well as single-variable calculus, derive (5.6). In other words, prove that α given by (5.6) does indeed minimize $\text{Var}\left(\alpha X + (1-\alpha)Y\right)$.

$$\text{Var}\left(\alpha X + (1-\alpha)Y\right) = \alpha^2\sigma_X^2 + 2\alpha(1-\alpha)\sigma_{XY} + (1-\alpha)^2\sigma_Y^2$$

$$\rightarrow \frac{\partial \text{Var}\left(\alpha X + (1-\alpha)Y\right)}{\partial \alpha} = 2\alpha\sigma_X^2 + 2\sigma_{XY} - 4\alpha\sigma_{XY} - 2\sigma_Y^2 + 2\alpha\sigma_Y^2 = 0$$

$$\alpha\sigma_X^2 - 2\alpha\sigma_{XY} + \alpha\sigma_Y^2 = \sigma_Y^2 - \sigma_{XY}$$

$$\alpha^* = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 - 2\sigma_{XY} + \sigma_Y^2}$$

2. We will now derive the probability that a given observation is part of a bootstrap sample. Suppose that we obtain a bootstrap sample from a set of n observations.

   (a) What is the probability that the first bootstrap observation is *not* the jth observation from the original sample? Justify your answer.

   There are n observations from the original data set which are all equally likely to be selected as the first bootstrap observation. Thus, the probability that the first bootstrap observation is not the jth observation is $(n-1)/n$ or $1 - 1/n$.

   (b) What is the probability that the second bootstrap observation is *not* the jth observation from the original sample?

   Since we replace the observations after selection using the bootstrap, this is the same answer as that from part (a) $1 - 1/n$.

   (c) Argue that the probability that the jth observation is *not* in the bootstrap sample is $(1 - 1/n)^n$.

   We can extend our answers from parts (a) and (b) to a more general case. Since we replace the observations after selection using the bootstrap, the probability that the ith bootstrap observation is not the jth observation is $1 - 1/n$. Assuming that the observations are drawn independent of one another, then the probability of the jth observation not being included in the bootstrap sample is $\prod_{i=1}^{n}(1 - \frac{1}{n}) = (1 - 1/n)^n$.

   (d) When $n = 5$, what is the probability that the jth observation is in the bootstrap sample?

$$1 - \left(1 - \frac{1}{5}\right)^5 = 0.67232$$

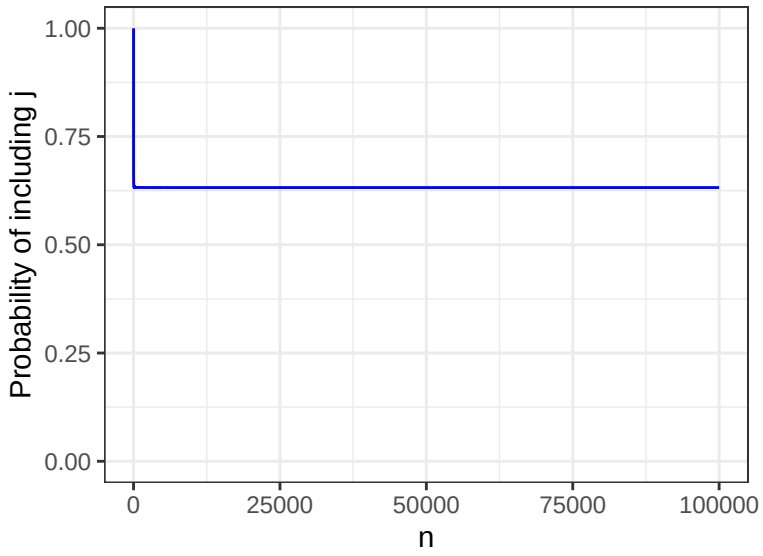
   (e) When $n = 100$, what is the probability that the jth observation is in the bootstrap sample?

$$1 - \left(1 - \frac{1}{100}\right)^{100} = 0.6339677$$

   (f) When $n = 10,000$, what is the probability that the jth observation is in the bootstrap sample?

$$1 - \left(1 - \frac{1}{10000}\right)^{10000} = 0.632139$$

   (g) Create a plot that displays, for each integer value of n from 1 to 100,000, the probability that the jth observation is in the bootstrap sample. Comment on what you observe.



The probability of including the jth observation when $n = 1$ is 1 since there's only one observation to choose from. Following that, there is a sharp decline where the probabilities hover around 0.6321416 for all other values of n.

(h) We will now investigate numerically the probability that a bootstrap sample of size $n = 100$ contains the jth observation. Here $j = 4$. We repeatedly create bootstrap samples, and each time we record whether or not the fourth observation is contained in the bootstrap sample.

```
set.seed(1)

store <- foreach(i = seq_len(10000L), .combine = c) %dopar% {
    return(sum(sample(seq_len(100L), replace = TRUE) == 4) > 0)
}

mean(store)
```

```
[1] 0.6361
```

Comment on the results obtained.

The results are close those found in part (g). In part (g), we found that the expected probability of the 4th observation being included is 63.3967659%. Here, we find the simulated samples included the 4th observation 63.61% of the time, a fairly minute difference of 0.2132341%.

3. We now review k-fold cross-validation.

(a) Explain how k-fold cross-validation is implemented.

k-fold cross-validation is a generalization of LOOCV. To start, we set a value for $k \in [1, n]$. We randomly divide the observations into k roughly equally sized folds. Then we iterate through these folds, treating one as a test fold while the other $k - 1$ folds are used to train the model. Finally, we average over the test error rates to estimate the true test error rate.

(b) What are the advantages and disadvantages of k-fold cross-validation relative to:

i. The validation set approach?

Recall the primary downsides of the validation set approach: biasedness, randomness, and smaller training set sizes. k-fold CV improves on all of these shortcomings. While the validation set approach relies on a single randomly selected test set to estimate the test error, k-fold CV

iterates over all of the folds and averages their respective test error rates to estimate the true test error rate. This results in less variability, less biasedness, and, for $k > 2$, the training sets are larger. One downside of k-fold CV is that it's more computationally expensive than the validation set approach.

ii. LOOCV?

Recall the primary downsides of LOOCV: very computationally expensive and large variance. Assuming $k < n$, k-fold CV improves on both of these shortcomings. While LOOCV requires fitting n models (disregarding the linear regression model case), k-fold require $n - k$ fewer fits. Additionally, k-fold CV is less variable than LOOCV. This is a result of the fact that the n different training sets are nearly identical and thus highly correlated. One downside of k-fold CV is that it introduces additional bias in comparison to LOOCV; however, this is generally made up for by the reduction in variance.

4. Suppose that we use some statistical learning method to make a prediction for the response Y for a particular value of the predictor X. Carefully describe how we might estimate the standard deviation of our prediction.

We can use the bootstrap to estimate the standard deviation of our prediction. To start, we randomly select, with replacement, n observations from our original data set. We use this boot strap data set to fit our model and use this model to make a prediction for the response Y for a particular value of the predictor X. This process is repeated B, for some large $B \in \mathbb{Z}^+$. Finally, we can use the formula

$$\mathrm{SE}_B(\hat{Y}) = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}\left(\hat{Y}_i - \frac{1}{B}\sum_{i'=1}^{B}\hat{Y}_{i'}\right)^2}$$

to estimate the standard deviation of our prediction.

Applied

```
set.seed(7)

logistic.model <- glm(default ~ income + balance, data = Default_dt, family = binomial)

num_obs <- fnrow(Default_dt)
training_ind <- sample(num_obs, num_obs / 2)
test_ind <- seq_len(num_obs)[-training_ind]

logistic.model_training <- glm(default ~ income + balance, Default_dt, family = binomial, subset = trai
val_set <- fifelse(predict.glm(logistic.model_training, newdata = Default_dt[test_ind, ], type = "respo

fsum(Default_dt[test_ind, default] != val_set) / length(val_set)
```

```
[1] 0.0242
```

```
set.seed(100)

parallel::clusterExport(cl, "num_obs")

test_err_estimates <- foreach(i = seq_len(3L), .combine = c) %dopar% {

  training_ind <- sample(num_obs, num_obs / 2)
  test_ind <- seq_len(num_obs)[-training_ind]

  logistic.model_training <- glm(formula = default ~ income + balance, data = Default_dt, family = bino
```

```
  val_set <- ifelse(predict.glm(logistic.model_training, newdata = Default_dt[test_ind, ], type = "respo

  return(sum(Default_dt[test_ind, "default"] != val_set) / length(val_set))
}

logistic.model <- glm(default ~ income + balance + student, Default_dt, family = binomial)

set.seed(123)
training_ind <- sample(num_obs, num_obs / 2)
test_ind <- seq_len(num_obs)[-training_ind]

logistic.model_training <- glm(default ~ income + balance + student, Default_dt, family = binomial, subs
val_set <- fifelse(predict.glm(logistic.model_training, newdata = Default_dt[test_ind, ], type = "respon

fsum(Default_dt[test_ind, default] != val_set) / length(val_set)
```

```
[1] 0.0272
```

```
set.seed(123)

summary(glm(default ~ income + balance, Default_dt, family = binomial))
```

```
Call:
glm(formula = default ~ income + balance, family = binomial,
    data = Default_dt)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4725  -0.1444  -0.0574  -0.0211   3.7245

Coefficients:
                 Estimate    Std. Error z value             Pr(>|z|)
(Intercept) -11.540468437   0.434756357 -26.545 < 0.0000000000000002 ***
income        0.000020809   0.000004985   4.174            0.0000299 ***
balance       0.005647103   0.000227373  24.836 < 0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2920.6  on 9999  degrees of freedom
Residual deviance: 1579.0  on 9997  degrees of freedom
AIC: 1585

Number of Fisher Scoring iterations: 8
```

```
boot.fn <- function(data = Default_dt, indices){
  coef(glm(default ~ income + balance, data, family = binomial, subset = indices))
}

boot(Default_dt, boot.fn, R = 100)
```

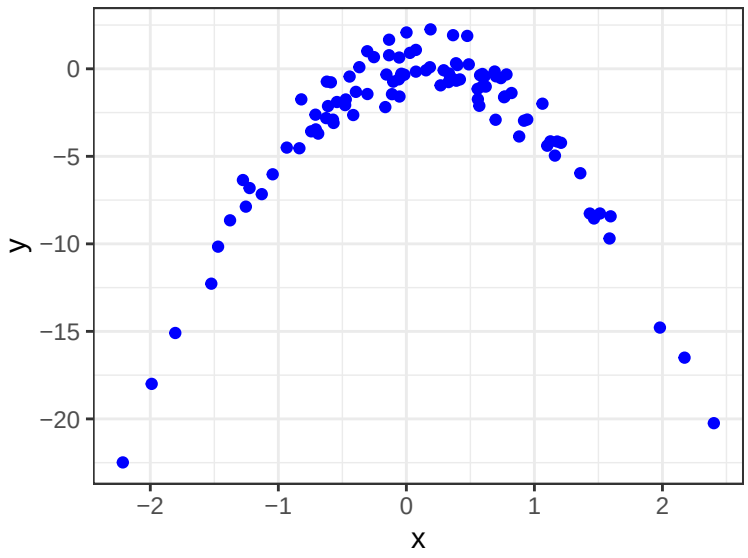
```
ORDINARY NONPARAMETRIC BOOTSTRAP
```

```
Call:
boot(data = Default_dt, statistic = boot.fn, R = 100)


Bootstrap Statistics :
           original           bias       std. error
t1* -11.54046843668 -0.0106485874475 0.421850034301
t2*   0.00002080898  0.0000001483646 0.000005124683
t3*   0.00564710294  0.0000028364069 0.000215008249
```

```
logistic.model <- glm((as.integer(Direction) - 1) ~ Lag1 + Lag2, data = Weekly_dt[-1L,])

test_errors <- foreach(i = seq_row(Weekly_dt), .combine = c) %do% {
  return(fifelse(predict.glm(logistic.model, newdata = Weekly[i, ], type = "response") > 0.5, "Up", "Do
}

mean(test_errors)
```

```
[1] 0.446281
```

```
set.seed(1)
x <- rnorm(100)
y <- x - 2 * 2 * x^2 + rnorm(100)

ggplot(mapping = aes(x, y)) +
  geom_point(color = "blue") +
  theme_bw()
```




```
set.seed(123)
df <- data.table(x = x, y = y)
cv.glm(df, glmfit = glm(y ~ x, data = df))$delta
```

```
[1] 25.02897 25.01624
```

```
cv.glm(df, glmfit = glm(y ~ poly(x, 2, raw = TRUE), data = df))$delta
```

```
[1] 0.9374236 0.9371789
```

```
cv.glm(df, glmfit = glm(y ~ poly(x, 3, raw = TRUE), data = df))$delta
```

```
[1] 0.9566218 0.9562538
```

```
cv.glm(df, glmfit = glm(y ~ poly(x, 4, raw = TRUE), data = df))$delta
```

```
[1] 0.9539049 0.9534453
```

```
mu_hat <- fmean(Boston_dt[["medv"]])
n <- fnrow(Boston_dt)
fsd(Boston_dt[["medv"]]) / sqrt(n)
```

```
[1] 0.4088611
```

```
set.seed(123)

sample_mean <- function(x, indices){
  fmean(x[indices])
}

boot_se <- boot(Boston_dt[["medv"]], sample_mean, R = 100)
boot_se
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Boston_dt[["medv"]], statistic = sample_mean, R = 100)


Bootstrap Statistics :
    original      bias    std. error
t1* 22.53281 -0.01811462   0.3333941
```

```
boot.ci(boot_se, type = "norm")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 100 bootstrap replicates

CALL :
boot.ci(boot.out = boot_se, type = "norm")

Intervals :
Level      Normal
95%   (21.9, 23.2 )
Calculations and Intervals on Original Scale
```

```
t.test(Boston[["medv"]])
```

```
    One Sample t-test

data:  Boston[["medv"]]
t = 55.111, df = 505, p-value < 0.00000000000000022
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 21.72953 23.33608
sample estimates:
```

```
mean of x
 22.53281
```

```
sample_median <- function(x, indices){
  fmedian(x[indices])
}

boot_se <- boot(Boston_dt[["medv"]], sample_mean, R = 100)
boot_se
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Boston_dt[["medv"]], statistic = sample_mean, R = 100)


Bootstrap Statistics :
    original     bias    std. error
t1* 22.53281 0.03110079     0.38905
```

```
boot.ci(boot_se, type = "norm")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 100 bootstrap replicates

CALL :
boot.ci(boot.out = boot_se, type = "norm")

Intervals :
Level      Normal
95%   (21.74, 23.26 )
Calculations and Intervals on Original Scale
```

Chapter 6

Notes

Subset Selection

In the regression setting, the standard linear model is given by

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

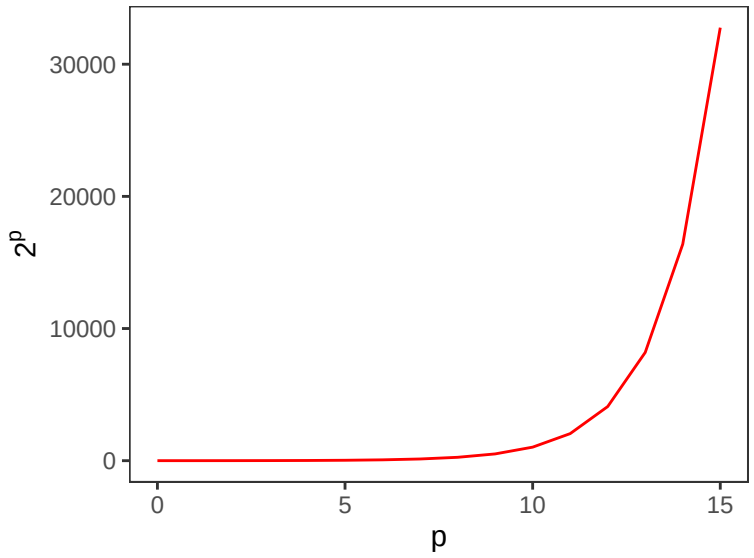
Generally, these models are fit using least squares; however, in some scenarios, other procedures can produce better results in terms of prediction accuracy and model interpretability. Three important classes of methods include subset selection, shrinkage, and dimension reduction.

Subset selection involves identifying a subset of the p predictors that we believe to be related to the response. We then fit a model using least squares on the reduced set of variables. To perform **best subset selection**, we fit a separate least squares regression for each possible combination of the p predictors. Thus, we fit all p models that contain one predictor, all $\binom{p}{2} = \frac{p(p-1)}{2}$ models with 2 predictors, all $\binom{p}{3}$ models with 3 predictors, and so on. Ultimately, without a transformation of the p predictors, we end up with 2^p possibilities, and our goal is to select the best model. The following procedure depicts how the best model is chosen:

1. Start with the null model $\mathcal{M}_0$, which contains no predictors. Thus, it simply predicts the sample mean for each prediction.
2. For $k = 1, \ldots, p$ fit all $\binom{p}{k}$ models with k predictors and select the model with the lowest RSS/greatest R^2. Denote these models using $\mathcal{M}_k$.
3. Select a single best model among $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$ using the cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

This procedure can be expanded beyond the case of linear regression to other types of models such as logistic regression. In these circumstances, instead of using RSS, we use **deviance**, which is negative two times the maximum log-likelihood and plays the role of RSS in these other models.

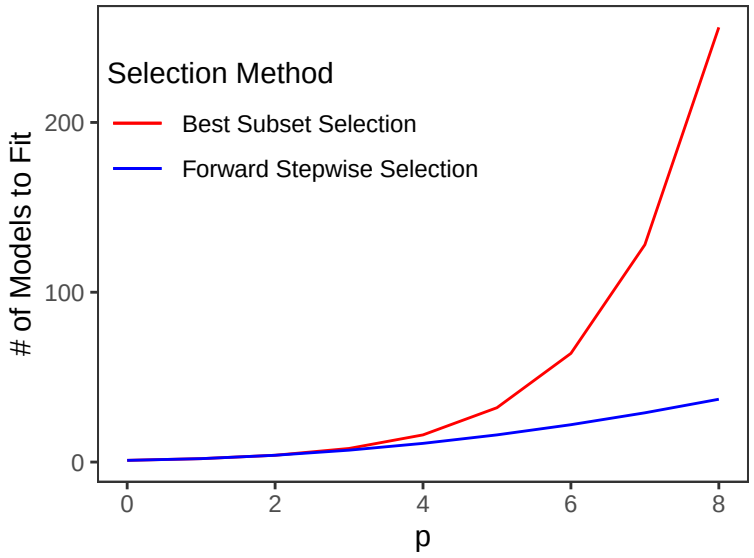
Unfortunately, best subset selection is computationally limited. The graph below shows the number of models that need to be fit as a function of p. It's evident that best subset selection is computationally infeasible for large values of p.



Due to this computational limitation and the possibility of overfitting in large search spaces, *stepwise* methods are an attractive alternative to best subset selection. **Forward stepwise selection** is one such method that proceeds as follows:

1. Start with the null $\mathcal{M}_0$, which contains no predictors.
2. For $k = 1, \ldots, p$, consider all $p - k$ models that augment the model $\mathcal{M}_{k-1}$ with the one additional predictor that improves the model (in terms of the R^2) the most.
3. Compare among $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$ using the cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

This method is significantly less computationally expensive than best subset selection as we only have to fit $p - k$ models for the kth iteration. In total, we fit $1 + \frac{p(p+1)}{2}$ models. The graph below displays the difference in computation size between best subset selection and forward stepwise selection.



Another method of subset selection that improves upon best subset selection's demanding computational nature is **backward stepwise selection**, which proceeds as follows:

1. Begin with the model that contains all p predictors, $\mathcal{M}_p$.
2. For $k = p - 1, \ldots, 0$, fit k models and select the model, $\mathcal{M}_k$, that maximizes R^2 and contains all but one of the predictors in $\mathcal{M}_{k+1}$.
3. Compare among $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$ using the cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Both forward stepwise and backward stepwise selection require fitting $1 + \frac{p(p+1)}{2}$ models. However, one shortcoming of both methods is their possibility of "missing" the best of the 2^p models fit with best subset selection. For example, consider the case of using forward stepwise selection where $p = 3$. For both best subset selection and forward stepwise selection the model $\mathcal{M}_1$ is the same. For the purpose of illustration, say both elect X_1. It may be the case that best subset selection identifies $\mathcal{M}_2$ as that which uses X_2 and X_3, while forward stepwise selection must still use X_1. If it turns out that $\mathcal{M}_2$ outperforms $\mathcal{M}_1$ and $\mathcal{M}_3$, then forward stepwise selection will miss the best model. The same concept holds for backward stepwise selection.

Methods for Estimating the Test Error Rate

We've already seen methods of using cross validation to directly estimate the test error. In addition to these methods, there are *indirect* methods for estimating the test error. These methods involve adjusting the training error to account for the bias due to overfitting.

One such method is using the $\boldsymbol{C_p}$ statistic. For a fitted least squares model with d predictors, the C_p estimate

of the test MSE is given by

$$C_p = \frac{1}{n}\left(\text{RSS} + 2d\hat{\sigma}_\epsilon^2\right)$$

Essentially, the C_p statistic adjusts the training error by adding the term $2d\hat{\sigma}_\epsilon^2$. Hence, if adding an additional regressor does not decrease the RSS enough to offset the increase in $2d\hat{\sigma}_\epsilon^2$, then we will not use the additional regressor. It should be noted that $\hat{\sigma}_\epsilon^2$ is typically estimated using the model with all p predictors.

Two additional approaches are the **Akaike information criterion (AIC)** statistic and the **Bayesian information criterion (BIC)**. These criteria are defined for a large class of models fit by maximum likelihood. In the case of d predictors with Gaussian errors, these statistics can be found using the formulas

$$\text{AIC} = \frac{1}{n\hat{\sigma}_\epsilon^2}\left(\text{RSS} + 2d\hat{\sigma}_\epsilon^2\right)$$

$$\text{BIC} = \frac{1}{n\hat{\sigma}_\epsilon^2}\left(\text{RSS} + \log(n)d\hat{\sigma}_\epsilon^2\right)$$

Much like the C_p statistic, AIC and BIC adjust the training error by adding the terms $2d\hat{\sigma}_\epsilon^2$ and $\log(n)d\hat{\sigma}_\epsilon^2$. Note that for $n > 7$, $\log(n) > 2$, so BIC penalizes/adjusts the training error more than AIC and C_p. This generally results in the selection of fewer predictors in the model.

One final approach for indirectly estimating the test error is using the **adjusted R^2** (denoted $\bar{R}^2$). Recall from earlier that R^2 is computed by $1 - \text{RSS/TSS}$ and is an estimator of the squared population correlation between $\hat{y}$ and y (ρ^2) given by $1 - \sigma_\epsilon^2/\sigma_y^2$. As we showed in chapter 3, R^2 estimates this value by estimating σ_ϵ^2 as RSS/n and σ_y^2 as TSS/n, which we know are both biased estimators of the true parameters since they do not employ Bessel's correction. $\bar{R}^2$ instead uses the unbiased estimators of σ_ϵ^2 $[\text{RSS}/(n-d-1)]$ and σ_y^2 $[\text{TSS}/(n-1)]$. Thus, the adjusted R^2 for a least squares model with d predictors can be found using the formula

$$\bar{R}^2 = 1 - \frac{\text{RSS}/(n-d-1)}{\text{TSS}/(n-1)}$$

Unlike R^2, $\bar{R}^2$ pays a price for the inclusion of an unnecessary variable in a model.

C_p, AIC, and BIC all have rigorous theoretical justifications while adjusted R^2 does not despite in intuitive nature and widespread popularity. All of the formulas provided here are for the case of linear models fit by least squares but can be extended to more general types of models.

Ridge Regression

Shrinkage methods are an alternative to subset selection methods where we fit a model containing all p predictors using a technique that *shrinks* the coefficient estimates toward zero. This methodology can lead to improvements over least squares as shrinking the coefficient estimates can significantly reduce their variances.

One such shrinkage method is called **ridge regression** where we fit our coefficient estimates $\hat{\beta}^R$ by minimizing

$$\sum_{i=1}^{n}\left(y_i - \hat{\beta}_0^R - \sum_{j=1}^{p}\hat{\beta}_j^R x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\left(\hat{\beta}_j^R\right)^2 = \text{RSS} + \lambda\sum_{j=1}^{p}\left(\hat{\beta}_j^R\right)^2,$$

where $\lambda \geq 0$ is a **tuning parameter** to be fit separately. The above quantity contains two fitting criteria; the RSS, which is used to find coefficient estimates that fit the data well, and a **shrinkage penalty**, which serves the purpose of shrinking the coefficient estimates toward zero. Unlike least squares, where multiplying X_{ij} by a constant c results in scaling β_j by $1/c$, ridge regression is sensitive to different scales. Thus, it's best to apply ridge regression after standardizing the predictors.

Ridge regression does well compared to least squares when minimizing variance is a key issue. As λ increases, the flexibility of the model decreases leading to a decrease in variance and an increase in bias. Additionally, ridge regression is computationally advantageous when compared to the requirements of best subset selection. As it turns out, simultaneously fitting the model for *all* values of λ is computationally similar to fitting a model using least squares.

The Lasso

Ridge regression has one obvious drawback in that it suffers from interpretability since all p predictors are included in the model. Increasing the value of λ will tend to reduce the magnitude of the coefficients but will not result in exclusion of any of the variables. The **lasso** is an alternative to ridge regression that overcomes this disadvantage. Similar to ridge regression, the lasso coefficients $\hat{\beta}^L$ minimize the quantity

$$\sum_{i=1}^{n}\left(y_i - \hat{\beta}_0^L - \sum_{j=1}^{p}\hat{\beta}_j^L x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\left|\hat{\beta}^L\right| = \text{RSS} + \lambda||\hat{\beta}^L||_1$$

This quantity is very similar to that used for ridge regression, however it uses the ℓ_1 (pronounced "ell 1") norm for the shrinkage penalty instead of the squared ℓ_2 norm. The effect of using the ℓ_1 norm instead is that some of the coefficient estimates are forced to exactly zero. Thus, we say that the lasso yields **sparse** models, meaning that the models involve only a subset of the variables. As a result, the lasso can result in models that are easier to interpret.

Much like ridge regression, the lasso should involve standardized predictors and can lead to improvements of least squares by reducing variance for additional bias. Unlike ridge regression, the lasso performs variable selection. Choosing between ridge regression and the lasso is a question to be solved using cross validation. In general, the lasso will outperform ridge regression when only a relatively small subset of the p predictors have very small or exactly zero coefficients. On the other hand, ridge regression will tend to perform better than the lasso when all or most of the p predictors should be included in the model.

Selecting the Tuning Parameter

Both the lasso and ridge regression are heavily dependent on the value chosen for the tuning parameter λ (or equivalently, the value for the constraint s). Selecting a value for λ is another task for cross-validation. Generally, we start out by selecting a grid of values for λ. Then, we compute the cross-validation errors for each of these values for λ and select the value for which the CV error is smallest. Finally, we re-fit the model using all of the available observations and the selected value of λ.

Dimension Reduction Methods

In addition to shrinkage and subset selection methods, another common approach for controlling the variance of our estimators is **dimension reduction**. If we let $Z_1, Z_2, \ldots, Z_M$ represent $M < p$ linear combinations of our p predictors such that

$$Z_m = \sum_{j=1}^{p}\phi_{jm}X_j,$$

then we can fit a linear regression model

$$y_i = \theta_0 + \sum_{m=1}^{M}\theta_m z_{im} + \epsilon_i$$

using OLS. If the constants $\phi_{1m}, \phi_{2m}, \ldots, \phi_{pm}$ are chosen wisely, dimension reduction methods can outperform regular OLS. Combining the two above equations, we find that

$$\sum_{m=1}^{M}\theta_m z_{im} = \sum_{m=1}^{M}\theta_m\sum_{j=1}^{p}\phi_{jm}x_{ij} = \sum_{m=1}^{M}\sum_{j=1}^{p}\theta_m\phi_{jm}x_{ij}$$

$$= \sum_{j=1}^{p}\sum_{m=1}^{M}\theta_m\phi_{jm}x_{ij} = \sum_{j=1}^{p}\beta_j x_{ij},$$

where $\beta_j = \sum_{m=1}^{M}\phi_{jm}\theta_m$. Thus, dimension reduction is a special case of least squares, and it turns out that if $M = p$, then no dimension reduction occurs and we obtain the same results as we would using OLS.

Dimension reduction requires two main steps. First, we obtain the transformed predictors $Z_1, \ldots, Z_m$, which we follow with fitting our model using these M predictors. We already know how to fit the model using these M predictors, but it remains to decide on how to obtain the transformed predictors. Two common approaches for achieving this task are principal components and partial least squares.

Principal Component Analysis for Regression

Principal component analysis (**PCA**) is a popular approach for deriving a smaller set of features from a large set of variables. The first principal component direction of the data is that along which the predictors vary the *most*. That is we choose the set of values for $\phi_{11}, \phi_{21}, \ldots, \phi_{p1}$ by solving

$$\max_{\phi_{11},\phi_{21},\ldots,\phi_{p1}} \text{Var}\left(\sum_{j=1}^{p} \phi_{j1} \times (X_j - \bar{X}_j)\right) \text{ s.t. } \sum_{j=1}^{p} \phi_{j1}^2 = 1$$

After solving for $\phi_{11}, \phi_{21}, \ldots, \phi_{p1}$, we solve the **principal component scores** $z_{11}, \ldots, z_{n1}$ using the formula

$$z_{i1} = \sum_{j=1}^{p} \phi_{j1}(x_{ij} - \bar{x}_j)$$