Data Science Machine Learning

The textbook for the Data Science course series is freely available online.

Learning Objectives

- The basics of machine learning
- How to perform cross-validation to avoid overtraining
- Several popular machine learning algorithms
- How to build a recommendation system
- What regularization is and why it is useful

Course Overview

There are six major sections in this course: introduction to machine learning; machine learning basics; linear regression for prediction, smoothing, and working with matrices; distance, knn, cross validation, and generative models; classification with more than two classes and the caret package; and model fitting and recommendation systems.

Introduction to Machine Learning

In this section, you'll be introduced to some of the terminology and concepts you'll need going forward.

Machine Learning Basics

In this section, you'll learn how to start building a machine learning algorithm using training and test data sets and the importance of conditional probabilities for machine learning.

Linear Regression for Prediction, Smoothing, and Working with Matrices

In this section, you'll learn why linear regression is a useful baseline approach but is often insufficiently flexible for more complex analyses, how to smooth noisy data, and how to use matrices for machine learning.

Distance, Knn, Cross Validation, and Generative Models

In this section, you'll learn different types of discriminative and generative approaches for machine learning algorithms.

Classification with More than Two Classes and the Caret Package

In this section, you'll learn how to overcome the curse of dimensionality using methods that adapt to higher dimensions and how to use the caret package to implement many different machine learning algorithms.

Model Fitting and Recommendation Systems

In this section, you'll learn how to apply the machine learning algorithms you have learned.

Section 1 - Introduction to Machine Learning Overview

In the **Introduction to Machine Learning** section, you will be introduced to machine learning.

After completing this section, you will be able to:

- Explain the difference between the **outcome** and the **features**.
- Explain when to use **classification** and when to use **prediction**.
- Explain the importance of **prevalence**.
- Explain the difference between **sensitivity** and **specificity**.

This section has one part: **introduction to machine learning**.

Notation

There is a link to the relevant section of the textbook: Notation

Key points

- $X_1, ..., X_p$ denote the features, Y denotes the outcomes, and $\hat{Y}$ denotes the predictions.
- Machine learning prediction tasks can be divided into **categorical** and **continuous** outcomes. We refer to these as **classification** and **prediction**, respectively.

An Example

There is a link to the relevant section of the textbook: An Example

Key points

- Y_i = an outcome for observation or index i.
- We use boldface for $\mathbf{X_i}$ to distinguish the vector of predictors from the individual predictors $X_{i,1}, ..., X_{i,784}$.
- When referring to an arbitrary set of features and outcomes, we drop the index i and use Y and bold $\mathbf{X}$.
- Uppercase is used to refer to variables because we think of predictors as random variables.
- Lowercase is used to denote observed values. For example, $\mathbf{X} = \mathbf{x}$.

Comprehension Check - Introduction to Machine Learning

1. True or False: A key feature of machine learning is that the algorithms are built with data.

☒ A. True
☐ B. False

2. True or False: In machine learning, we build algorithms that take feature values (X) and train a model using known outcomes (Y) that is then used to predict outcomes when presented with features without known outcomes.

☒ A. True
☐ B. False

Section 2 - Machine Learning Basics Overview

In the **Machine Learning Basics** section, you will learn the basics of machine learning.

After completing this section, you will be able to:

- Start to use the **caret** package.
- Construct and interpret a **confusion matrix**.
- Use **conditional probabilities** in the context of machine learning.

This section has two parts: **basics of evaluating machine learning algorithms** and **conditional probabilities**.

Caret package, training and test sets, and overall accuracy

There is a link to the relevant sections of the textbook: Training and test sets and Overall accuracy

Key points

- Note: the `set.seed()` function is used to obtain reproducible results. If you have R 3.6 or later, please use the `sample.kind = "Rounding"` argument whenever you set the seed for this course.
- To mimic the ultimate evaluation process, we randomly split our data into two — a training set and a test set — and act as if we don't know the outcome of the test set. We develop algorithms using only the training set; the test set is used only for evaluation.
- The `createDataPartition()` function from the **caret** package can be used to generate indexes for randomly splitting data.
- Note: contrary to what the documentation says, this course will use the argument p as the percentage of data that goes to testing. The indexes made from `createDataPartition()` should be used to create the test set. Indexes should be created on the outcome and not a predictor.
- The simplest evaluation metric for categorical outcomes is overall accuracy: the proportion of cases that were correctly predicted in the test set.

Code

```
if(!require(tidyverse)) install.packages("tidyverse")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(dslabs)) install.packages("dslabs")
```

```
## Loading required package: dslabs
```

```r
library(tidyverse)
library(caret)
library(dslabs)
data(heights)

# define the outcome and predictors
y <- heights$sex
x <- heights$height

# generate training and test sets
set.seed(2, sample.kind = "Rounding") # if using R 3.5 or earlier, remove the sample.kind argument
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
test_set <- heights[test_index, ]
train_set <- heights[-test_index, ]

# guess the outcome
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE)
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE) %>%
  factor(levels = levels(test_set$sex))

# compute accuracy
mean(y_hat == test_set$sex)
```

```
## [1] 0.5238095
```

```r
heights %>% group_by(sex) %>% summarize(mean(height), sd(height))
```

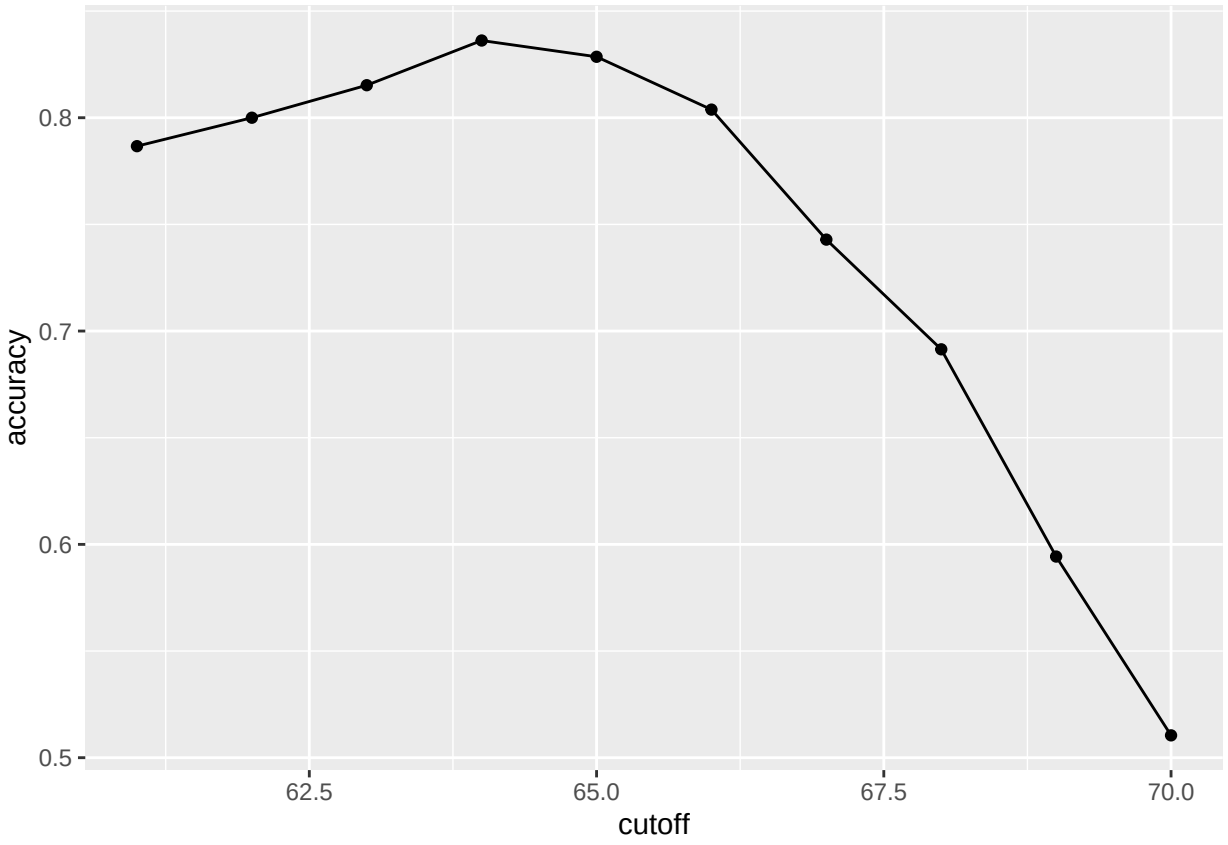
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 3
##   sex    `mean(height)` `sd(height)`
##   <fct>           <dbl>        <dbl>
## 1 Female           64.9         3.76
## 2 Male             69.3         3.61
```

```
y_hat <- ifelse(x > 62, "Male", "Female") %>% factor(levels = levels(test_set$sex))
mean(y == y_hat)
```

```
## [1] 0.7933333
```

```
# examine the accuracy of 10 cutoffs
cutoff <- seq(61, 70)
accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  mean(y_hat == train_set$sex)
})
data.frame(cutoff, accuracy) %>%
  ggplot(aes(cutoff, accuracy)) +
  geom_point() +
  geom_line()
```




```
max(accuracy)
```

```
## [1] 0.8361905
```

```
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff
```

```
## [1] 64
```

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
y_hat <- factor(y_hat)
mean(y_hat == test_set$sex)
```

```
## [1] 0.8171429
```

Comprehension Check - Basics of Evaluating Machine Learning Algorithms

1. For each of the following, indicate whether the outcome is continuous or categorical.

- Digit reader - categorical
- Height - continuous

- Spam filter - categorical
- Stock prices - continuous
- Sex - categorical

2. How many features are available to us for prediction in the `mnist` digits dataset?

You can download the `mnist` dataset using the `read_mnist()` function from the **dslabs** package.

```
mnist <- read_mnist()
ncol(mnist$train$images)
```

```
## [1] 784
```

Confusion matrix

There is a link to the relevant section of the textbook: Confusion Matrix

Key points

- Overall accuracy can sometimes be a deceptive measure because of unbalanced classes.
- A general improvement to using overall accuracy is to study sensitivity and specificity separately. **Sensitivity**, also known as the true positive rate or recall, is the proportion of actual positive outcomes correctly identified as such. **Specificity**, also known as the true negative rate, is the proportion of actual negative outcomes that are correctly identified as such.
- A confusion matrix tabulates each combination of prediction and actual value. You can create a confusion matrix in R using the `table()` function or the `confusionMatrix()` function from the **caret** package.

Code

```r
# tabulate each combination of prediction and actual value
table(predicted = y_hat, actual = test_set$sex)
```

```
##          actual
## predicted Female Male
##    Female     50   27
##    Male       69  379
```

```r
test_set %>%
  mutate(y_hat = y_hat) %>%
  group_by(sex) %>%
  summarize(accuracy = mean(y_hat == sex))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   sex    accuracy
##   <fct>     <dbl>
## 1 Female    0.420
## 2 Male      0.933
```

```r
prev <- mean(y == "Male")

confusionMatrix(data = y_hat, reference = test_set$sex)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Female Male
##     Female     50   27
##     Male       69  379
##
##                Accuracy : 0.8171
##                  95% CI : (0.7814, 0.8493)
##     No Information Rate : 0.7733
##     P-Value [Acc > NIR] : 0.008354
##
##                   Kappa : 0.4041
##
##  Mcnemar's Test P-Value : 2.857e-05
##
##             Sensitivity : 0.42017
##             Specificity : 0.93350
##          Pos Pred Value : 0.64935
##          Neg Pred Value : 0.84598
##              Prevalence : 0.22667
##          Detection Rate : 0.09524
##    Detection Prevalence : 0.14667
##       Balanced Accuracy : 0.67683
##
##        'Positive' Class : Female
##
```

Balanced accuracy and F1 score

There is a link to the relevant section of the textbook: Balanced accuracy and F1 Score

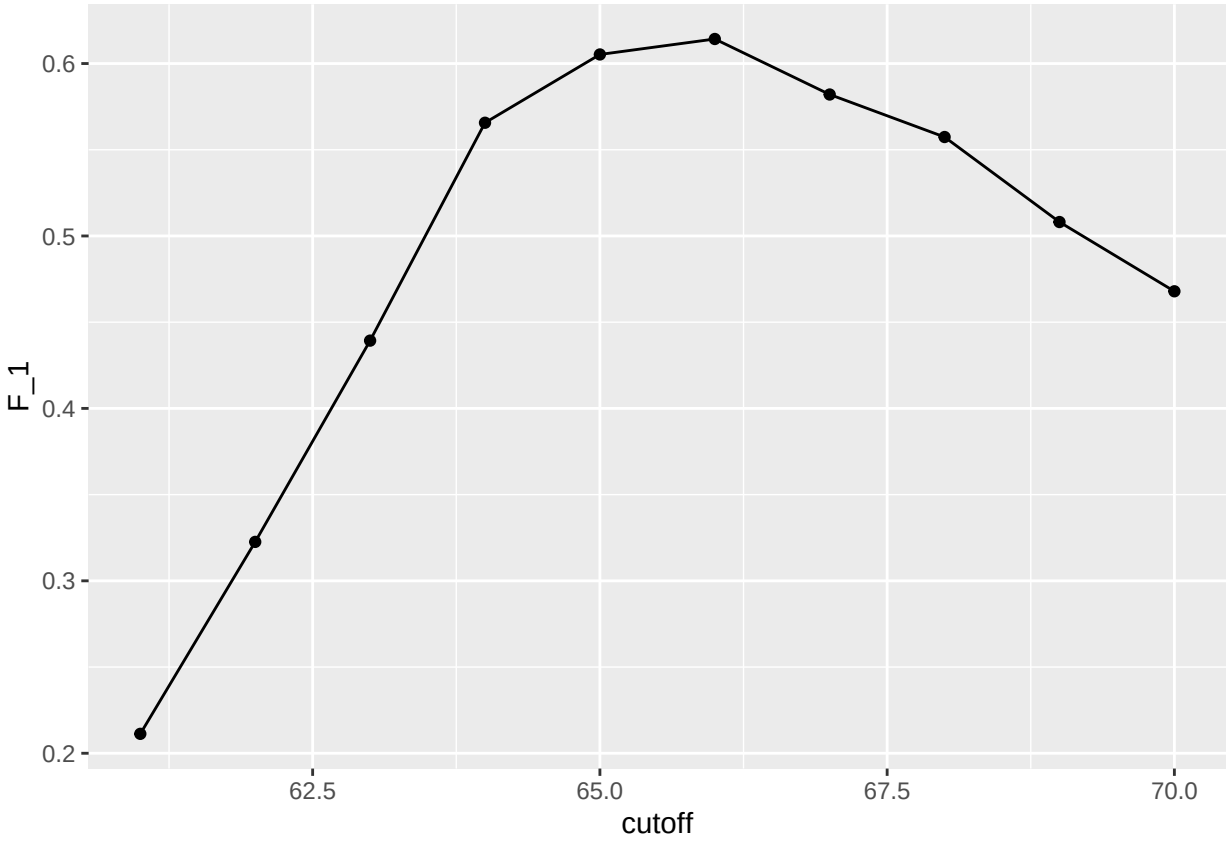
Key points

- For optimization purposes, sometimes it is more useful to have a one number summary than studying both specificity and sensitivity. One preferred metric is **balanced accuracy**. Because specificity and sensitivity are rates, it is more appropriate to compute the *harmonic* average. In fact, the **F1-score**, a widely used one-number summary, is the harmonic average of precision and recall.
- Depending on the context, some type of errors are more costly than others. The **F1-score** can be adapted to weigh specificity and sensitivity differently.
- You can compute the **F1-score** using the `F_meas()` function in the **caret** package.

Code

```
# maximize F-score
cutoff <- seq(61, 70)
F_1 <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  F_meas(data = y_hat, reference = factor(train_set$sex))
})

data.frame(cutoff, F_1) %>%
  ggplot(aes(cutoff, F_1)) +
  geom_point() +
  geom_line()
```



```
max(F_1)
```

```
## [1] 0.6142322
```

```
best_cutoff <- cutoff[which.max(F_1)]
best_cutoff
```

```
## [1] 66
```

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
sensitivity(data = y_hat, reference = test_set$sex)
```

```
## [1] 0.6806723
```

```
specificity(data = y_hat, reference = test_set$sex)
```

```
## [1] 0.8349754
```

Prevalence matters in practice

There is a link to the relevant section of the textbook: Prevalence matters in practice

Key points

- A machine learning algorithm with very high sensitivity and specificity may not be useful in practice when prevalence is close to either 0 or 1. For example, if you develop an algorithm for disease diagnosis with very high sensitivity, but the prevalence of the disease is pretty low, then the precision of your algorithm is probably very low based on Bayes' theorem.

ROC and precision-recall curves

There is a link to the relevant section of the textbook: ROC and precision-recall curves

Key points

- A very common approach to evaluating accuracy and F1-score is to compare them graphically by plotting both. A widely used plot that does this is the **receiver operating characteristic (ROC) curve**. The ROC curve plots sensitivity (TPR) versus 1 - specificity or the false positive rate (FPR).
- However, ROC curves have one weakness and it is that neither of the measures plotted depend on prevalence. In cases in which prevalence matters, we may instead make a **precision-recall plot**, which has a similar idea with ROC curve.

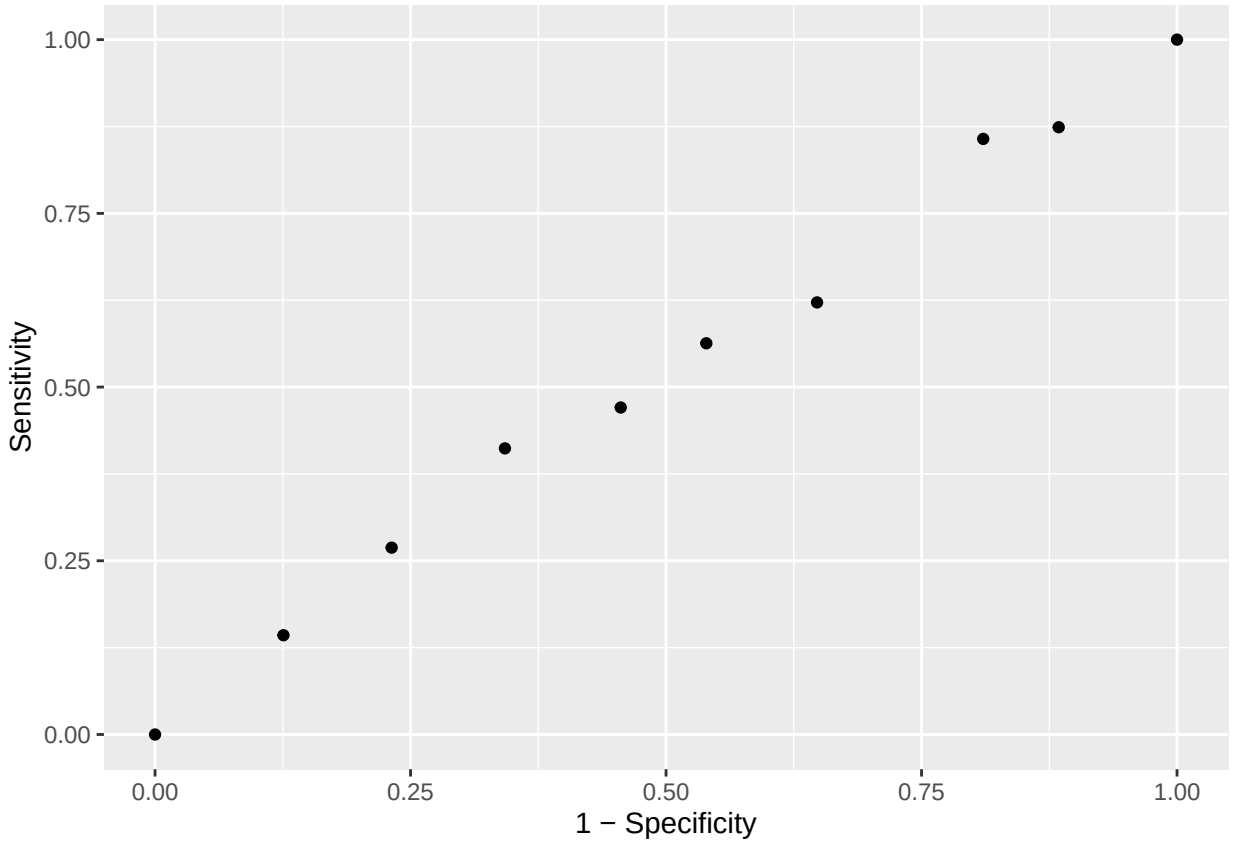
Code

Note: your results and plots may be slightly different.

```
p <- 0.9
n <- length(test_index)
y_hat <- sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
  factor(levels = levels(test_set$sex))
mean(y_hat == test_set$sex)
```

```
## [1] 0.7180952
```

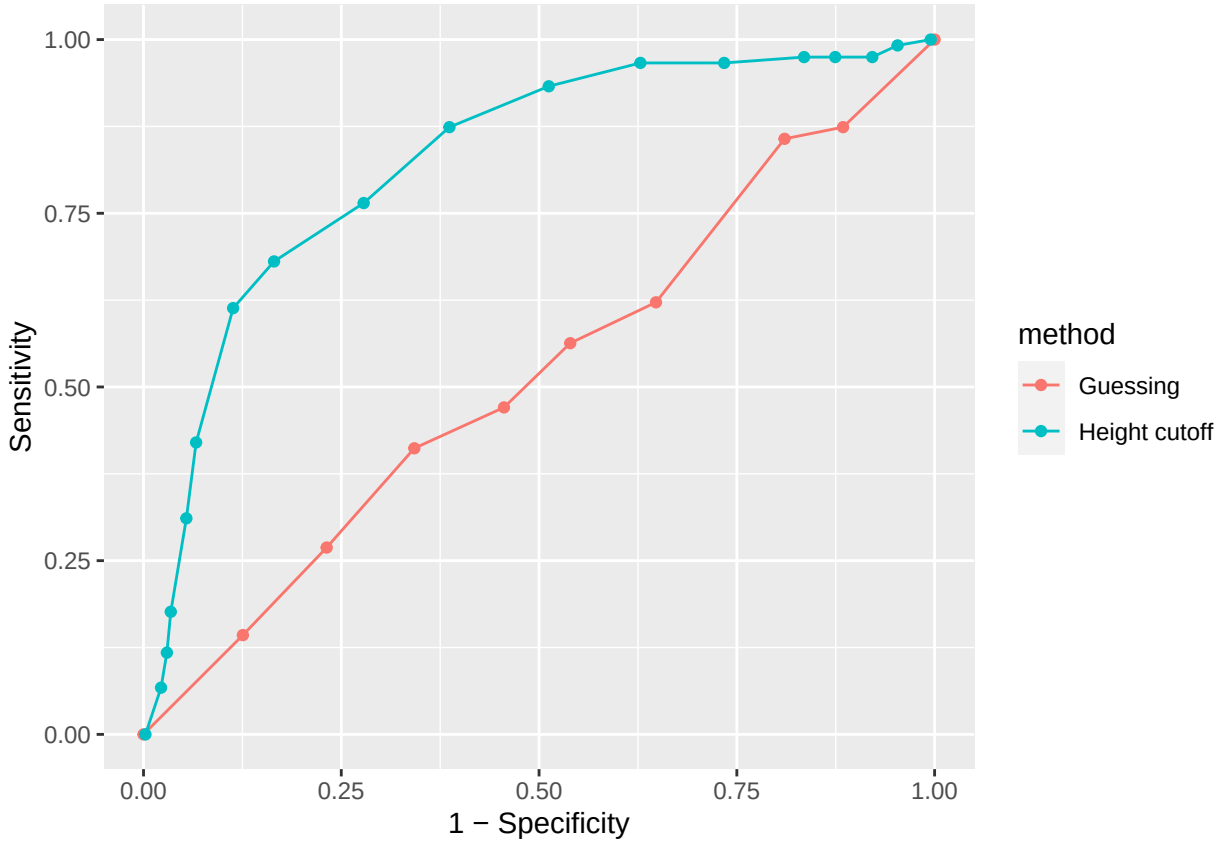
```
# ROC curve
probs <- seq(0, 1, length.out = 10)
guessing <- map_df(probs, function(p){
  y_hat <-
    sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guessing",
       FPR = 1 - specificity(y_hat, test_set$sex),
       TPR = sensitivity(y_hat, test_set$sex))
})
guessing %>% qplot(FPR, TPR, data =., xlab = "1 - Specificity", ylab = "Sensitivity")
```




```
cutoffs <- c(50, seq(60, 75), 80)
height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
   list(method = "Height cutoff",
        FPR = 1-specificity(y_hat, test_set$sex),
        TPR = sensitivity(y_hat, test_set$sex))
})

# plot both curves together
bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(FPR, TPR, color = method)) +
```

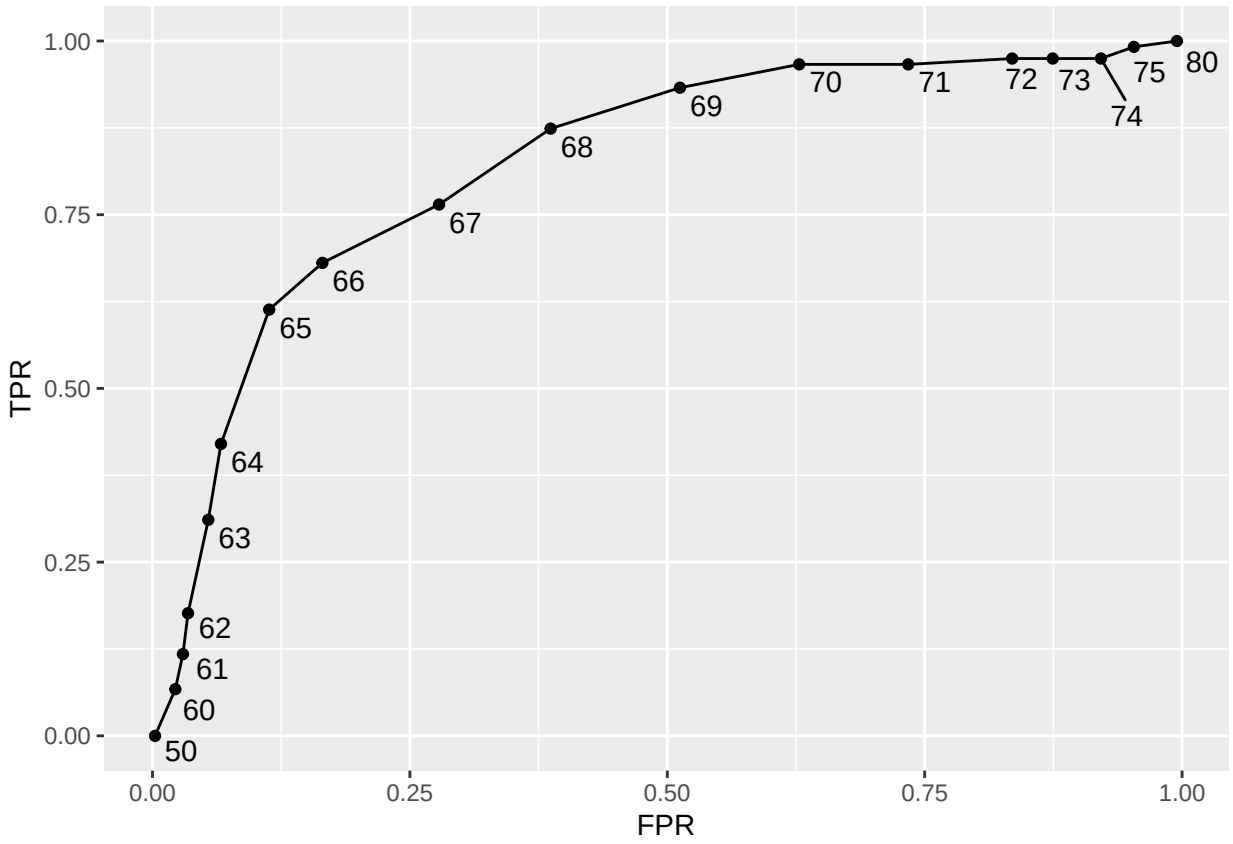
```
  geom_line() +
  geom_point() +
  xlab("1 - Specificity") +
  ylab("Sensitivity")
```




```
if(!require(ggrepel)) install.packages("ggrepel")
```

```
## Loading required package: ggrepel
```

```
library(ggrepel)
map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
   list(method = "Height cutoff",
        cutoff = x,
        FPR = 1-specificity(y_hat, test_set$sex),
        TPR = sensitivity(y_hat, test_set$sex))
}) %>%
  ggplot(aes(FPR, TPR, label = cutoff)) +
  geom_line() +
  geom_point() +
  geom_text_repel(nudge_x = 0.01, nudge_y = -0.01)
```



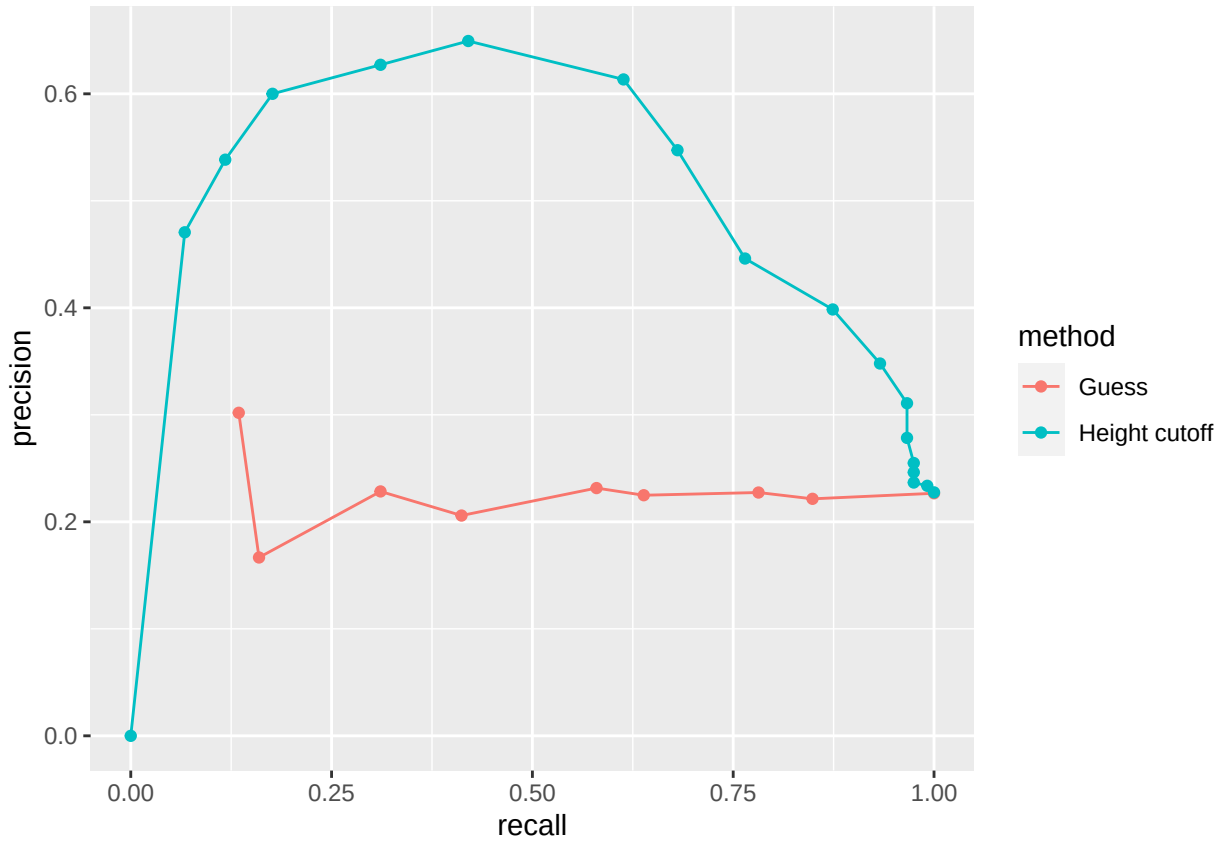

```
# plot precision against recall
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index),
                  replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guess",
    recall = sensitivity(y_hat, test_set$sex),
    precision = precision(y_hat, test_set$sex))
})

height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       recall = sensitivity(y_hat, test_set$sex),
    precision = precision(y_hat, test_set$sex))
})

bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(recall, precision, color = method)) +
  geom_line() +
  geom_point()
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

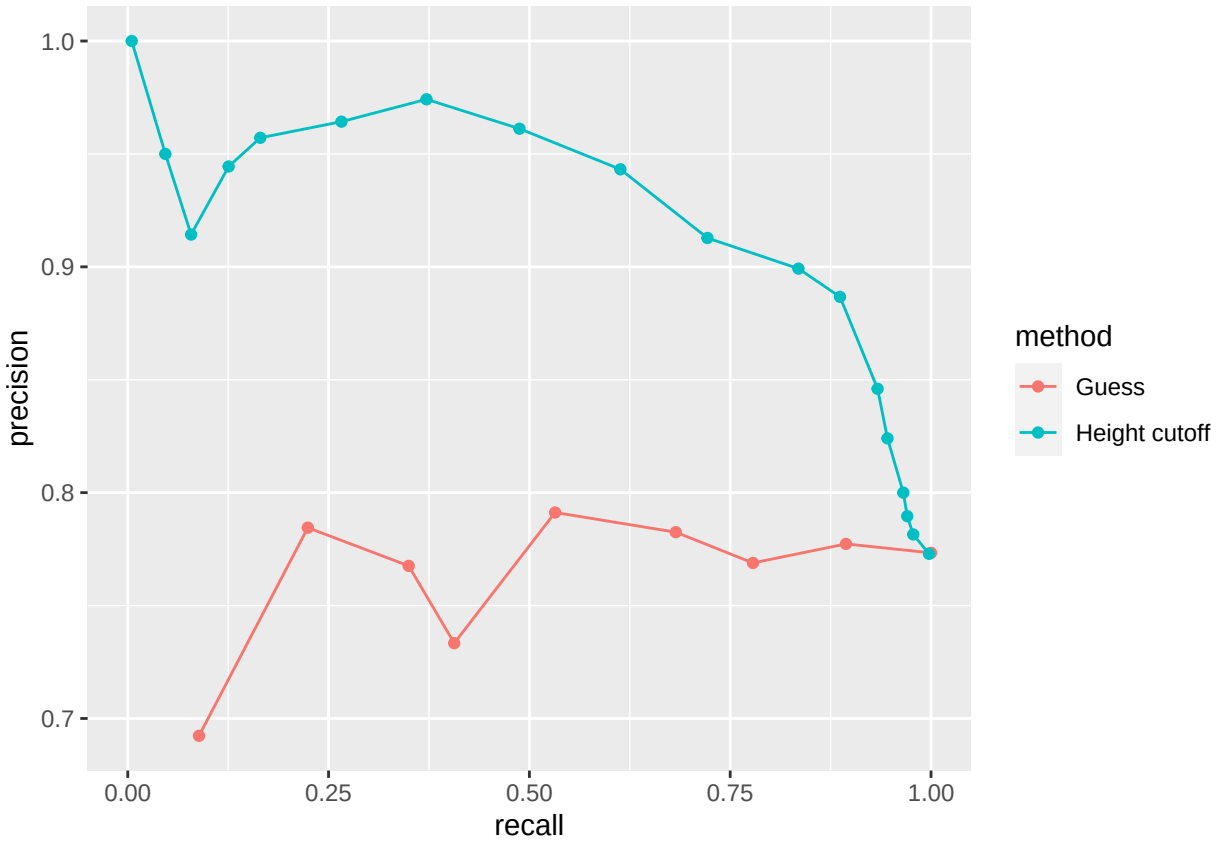



```
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE,
                  prob=c(p, 1-p)) %>%
    factor(levels = c("Male", "Female"))
  list(method = "Guess",
    recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
    precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))
})

height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Male", "Female"))
  list(method = "Height cutoff",
       recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
    precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))
})
bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(recall, precision, color = method)) +
  geom_line() +
  geom_point()
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).

## Warning: Removed 1 rows containing missing values (geom_point).
```



Comprehension Check - Practice with Machine Learning, Part 1

The following questions all ask you to work with the dataset described below.

The `reported_heights` and `heights` datasets were collected from three classes taught in the Departments of Computer Science and Biostatistics, as well as remotely through the Extension School. The Biostatistics class was taught in 2016 along with an online version offered by the Extension School. On 2016-01-25 at 8:15 AM, during one of the lectures, the instructors asked student to fill in the sex and height questionnaire that populated the `reported_heights` dataset. The online students filled out the survey during the next few days, after the lecture was posted online. We can use this insight to define a variable which we will call `type`, to denote the type of student, `inclass` or `online`.

The code below sets up the dataset for you to analyze in the following exercises:

```
if(!require(dplyr)) install.packages("dplyr")
if(!require(lubridate)) install.packages("lubridate")
```

```
## Loading required package: lubridate
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(dplyr)
library(lubridate)
data(reported_heights)

dat <- mutate(reported_heights, date_time = ymd_hms(time_stamp)) %>%
  filter(date_time >= make_date(2016, 01, 25) & date_time < make_date(2016, 02, 1)) %>%
  mutate(type = ifelse(day(date_time) == 25 & hour(date_time) == 8 & between(minute(date_time), 15, 30)
  select(sex, type)

y <- factor(dat$sex, c("Female", "Male"))
x <- dat$type
```

1. The `type` column of `dat` indicates whether students took classes in person ("inclass") or online ("online"). What proportion of the inclass group is female? What proportion of the online group is female?

Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
dat %>% group_by(type) %>% summarize(prop_female = mean(sex == "Female"))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   type    prop_female
##   <chr>         <dbl>
## 1 inclass       0.667
## 2 online        0.378
```

2. In the course videos, height cutoffs were used to predict sex. Instead of height, use the `type` variable to predict sex. Assume that for each class type the students are either all male or all female, based on the most prevalent sex in each class type you calculated in Q1. Report the accuracy of your prediction of sex based on type. You do not need to split the data into training and test sets.

Enter your accuracy as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
y_hat <- ifelse(x == "online", "Male", "Female") %>%
      factor(levels = levels(y))
mean(y_hat==y)
```

```
## [1] 0.6333333
```

3. Write a line of code using the `table()` function to show the confusion matrix between `y_hat` and `y`. Use the **exact** format `function(a, b)` for your answer and do not name the columns and rows. Your answer should have exactly one space.

```
table(y_hat, y)
```

```
##         y
## y_hat    Female Male
##   Female     26   13
##   Male       42   69
```

4. What is the sensitivity of this prediction? You can use the `sensitivity()` function from the **caret** package. Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
sensitivity(y_hat, y)
```

```
## [1] 0.3823529
```

5. What is the specificity of this prediction? You can use the `specificity()` function from the **caret** package. Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
specificity(y_hat, y)
```

```
## [1] 0.8414634
```

6. What is the prevalence (% of females) in the `dat` dataset defined above? Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
mean(y == "Female")
```

```
## [1] 0.4533333
```

Comprehension Check - Practice with Machine Learning, Part 2

We will practice building a machine learning algorithm using a new dataset, `iris`, that provides multiple predictors for us to use to train. To start, we will remove the `setosa` species and we will focus on the `versicolor` and `virginica` iris species using the following code:

```
data(iris)
iris <- iris[-which(iris$Species=='setosa'),]
y <- iris$Species
```

The following questions all involve work with this dataset.

7. First let us create an even split of the data into `train` and `test` partitions using `createDataPartition()` from the **caret** package. The code with a missing line is given below:

```
# set.seed(2) # if using R 3.5 or earlier
set.seed(2, sample.kind="Rounding") # if using R 3.6 or later
# line of code
test <- iris[test_index,]
train <- iris[-test_index,]
```

Which code should be used in place of # line of code above?

- ☐ A. test_index <- createDataPartition(y,times=1,p=0.5)
- ☐ B. test_index <- sample(2,length(y),replace=FALSE)
- ☒ C. test_index <- createDataPartition(y,times=1,p=0.5,list=FALSE)
- ☐ D. test_index <- rep(1,length(y))

```
# set.seed(2) # if using R 3.5 or earlier
set.seed(2, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y,times=1,p=0.5,list=FALSE)
```

```
## Warning in createDataPartition(y, times = 1, p = 0.5, list = FALSE): Some
## classes have no records ( setosa ) and these will be ignored
```

```
test <- iris[test_index,]
train <- iris[-test_index,]
```

8. Next we will figure out the singular feature in the dataset that yields the greatest overall accuracy when predicting species. You can use the code from the introduction and from Q7 to start your analysis.

Using only the `train` iris dataset, for each feature, perform a simple search to find the cutoff that produces the highest accuracy, predicting virginica if greater than the cutoff and versicolor otherwise. Use the `seq` function over the range of each feature by intervals of 0.1 for this search.

Which feature produces the highest accuracy?

```
foo <- function(x){
    rangedValues <- seq(range(x)[1],range(x)[2],by=0.1)
    sapply(rangedValues,function(i){
        y_hat <- ifelse(x>i,'virginica','versicolor')
        mean(y_hat==train$Species)
    })
}
predictions <- apply(train[,-5],2,foo)
sapply(predictions,max)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         0.70         0.62         0.96         0.94
```

- ☐ A. Sepal.Length
- ☐ B. Sepal.Width
- ☒ C. Petal.Length
- ☐ D. Petal.Width

9. For the feature selected in Q8, use the smart cutoff value from the training data to calculate overall accuracy in the test data. What is the overall accuracy?

```
predictions <- foo(train[,3])
rangedValues <- seq(range(train[,3])[1],range(train[,3])[2],by=0.1)
cutoffs <-rangedValues[which(predictions==max(predictions))]

y_hat <- ifelse(test[,3]>cutoffs[1],'virginica','versicolor')
mean(y_hat==test$Species)
```

```
## [1] 0.9
```

10. Notice that we had an overall accuracy greater than 96% in the training data, but the overall accuracy was lower in the test data. This can happen often if we overtrain. In fact, it could be the case that a single feature is not the best choice. For example, a combination of features might be optimal. Using a single feature and optimizing the cutoff as we did on our training data can lead to overfitting.

Given that we know the test data, we can treat it like we did our training data to see if the same feature with a different cutoff will optimize our predictions.

Which feature best optimizes our overall accuracy?

```
foo <- function(x){
    rangedValues <- seq(range(x)[1],range(x)[2],by=0.1)
    sapply(rangedValues,function(i){
        y_hat <- ifelse(x>i,'virginica','versicolor')
        mean(y_hat==test$Species)
    })
}
predictions <- apply(test[,-5],2,foo)
sapply(predictions,max)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         0.78         0.64         0.90         0.94
```

- ☐ A. Sepal.Length
- ☐ B. Sepal.Width
- ☐ C. Petal.Length
- ☒ D. Petal.Width

11. Now we will perform some exploratory data analysis on the data.

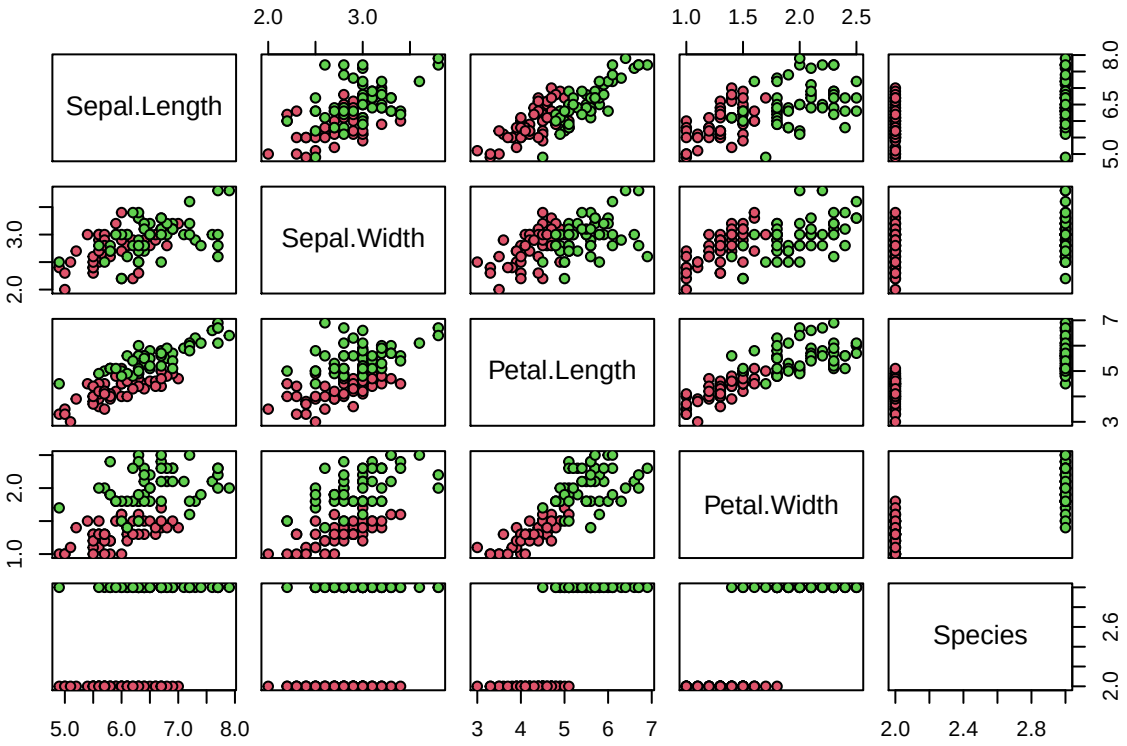
Notice that `Petal.Length` and `Petal.Width` in combination could potentially be more information than either feature alone.

Optimize the the cutoffs for `Petal.Length` and `Petal.Width` separately in the train dataset by using the `seq` function with increments of 0.1. Then, report the overall accuracy when applied to the test dataset by creating a rule that predicts virginica if `Petal.Length` is greater than the length cutoff OR `Petal.Width` is greater than the width cutoff, and versicolor otherwise.

What is the overall accuracy for the test data now?

```
data(iris)
iris <- iris[-which(iris$Species=='setosa'),]
y <- iris$Species

plot(iris,pch=21,bg=iris$Species)
```




```
# set.seed(2) # if using R 3.5 or earlier
set.seed(2, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y,times=1,p=0.5,list=FALSE)
```

```
## Warning in createDataPartition(y, times = 1, p = 0.5, list = FALSE): Some
## classes have no records ( setosa ) and these will be ignored
```

```
test <- iris[test_index,]
train <- iris[-test_index,]

petalLengthRange <- seq(range(train$Petal.Length)[1],range(train$Petal.Length)[2],by=0.1)
petalWidthRange <- seq(range(train$Petal.Width)[1],range(train$Petal.Width)[2],by=0.1)

length_predictions <- sapply(petalLengthRange,function(i){
        y_hat <- ifelse(train$Petal.Length>i,'virginica','versicolor')
        mean(y_hat==train$Species)
    })
length_cutoff <- petalLengthRange[which.max(length_predictions)] # 4.7

width_predictions <- sapply(petalWidthRange,function(i){
        y_hat <- ifelse(train$Petal.Width>i,'virginica','versicolor')
        mean(y_hat==train$Species)
    })
width_cutoff <- petalWidthRange[which.max(width_predictions)] # 1.5
```

```
y_hat <- ifelse(test$Petal.Length>length_cutoff | test$Petal.Width>width_cutoff,'virginica','versicolor
mean(y_hat==test$Species)
```

```
## [1] 0.88
```

Conditional probabilities

There is a link to the relevant section of the textbook: Conditional probabilities

Key points

- Conditional probabilities for each class:

$p_k(x) = Pr(Y = k|X = x), for\, k = 1, ..., K$

- In machine learning, this is referred to as **Bayes' Rule**. This is a theoretical rule because in practice we don't know $p(x)$. Having a good estimate of the $p(x)$ will suffice for us to build optimal prediction models, since we can control the balance between specificity and sensitivity however we wish. In fact, estimating these conditional probabilities can be thought of as the main challenge of machine learning.

Conditional expectations and loss function

There is a link to the relevant sections of the textbook: Conditional expectations and Loss functions

Key points

- Due to the connection between **conditional probabilities** and **conditional expectations**:

$p_k(x) = Pr(Y = k|X = x)$, for $k = 1, ..., K$

we often only use the expectation to denote both the conditional probability and conditional expectation.

- For continuous outcomes, we define a loss function to evaluate the model. The most commonly used one is **MSE (Mean Squared Error)**. The reason why we care about the conditional expectation in machine learning is that the expected value minimizes the MSE:

$\hat{Y} = E(Y|X = x)$ minimizes $E\{(\hat{Y} - Y)^2|X = x\}$

Due to this property, a succinct description of the main task of machine learning is that we use data to estimate for any set of features. **The main way in which competing machine learning algorithms differ is in their approach to estimating this expectation**.

Comprehension Check - Conditional Probabilities, Part 1

1. In a previous module, we covered Bayes' theorem and the Bayesian paradigm. Conditional probabilities are a fundamental part of this previous covered rule.

$P(A|B) = P(B|A)\frac{P(A)}{P(B)}$

We first review a simple example to go over conditional probabilities.

Assume a patient comes into the doctor's office to test whether they have a particular disease.

- The test is positive 85% of the time when tested on a patient with the disease (high sensitivity): $P(\text{test}+|\text{disease}) = 0.85$
- The test is negative 90% of the time when tested on a healthy patient (high specificity): $P(\text{test}-|\text{heathy}) = 0.90$
- The disease is prevalent in about 2% of the community: $P(\text{disease}) = 0.02$

Using Bayes' theorem, calculate the probability that you have the disease if the test is positive.

$P(\text{disease}|\text{test}+) = P(\text{test}+|\text{disease}) \times \frac{P(\text{disease})}{P(\text{test}+)} = \frac{P(\text{test}+|\text{disease})P(\text{disease})}{P(\text{test}+|\text{disease})P(\text{disease})+P(\text{test}+|\text{healthy})P(\text{healthy})]} = \frac{0.85 \times 0.02}{0.85 \times 0.02 + 0.1 \times 0.98} = 0.1478261$

The following 4 questions (Q2-Q5) all relate to implementing this calculation using R.

We have a hypothetical population of 1 million individuals with the following conditional probabilities as described below:

- The test is positive 85% of the time when tested on a patient with the disease (high sensitivity): $P(\text{test}+|\text{disease}) = 0.85$
- The test is negative 90% of the time when tested on a healthy patient (high specificity): $P(\text{test}-|\text{heathy}) = 0.90$
- The disease is prevalent in about 2% of the community: $P(\text{disease}) = 0.02$

Here is some sample code to get you started:

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
disease <- sample(c(0,1), size=1e6, replace=TRUE, prob=c(0.98,0.02))
test <- rep(NA, 1e6)
test[disease==0] <- sample(c(0,1), size=sum(disease==0), replace=TRUE, prob=c(0.90,0.10))
test[disease==1] <- sample(c(0,1), size=sum(disease==1), replace=TRUE, prob=c(0.15, 0.85))
```

2. What is the probability that a test is positive?

```
mean(test)
```

```
## [1] 0.114509
```

3. What is the probability that an individual has the disease if the test is negative?

```
mean(disease[test==0])
```

```
## [1] 0.003461356
```

4. What is the probability that you have the disease if the test is positive? Remember: calculate the conditional probability the disease is positive assuming a positive test.

```
mean(disease[test==1]==1)
```

```
## [1] 0.1471762
```

5. Compare the prevalence of disease in people who test positive to the overall prevalence of disease.

If a patient's test is positive, by how many times does that increase their risk of having the disease? First calculate the probability of having the disease given a positive test, then divide by the probability of having the disease.

```
mean(disease[test==1]==1)/mean(disease==1)
```

```
## [1] 7.389106
```

Comprehension Check - Conditional Probabilities, Part 2

6. We are now going to write code to compute conditional probabilities for being male in the heights dataset. Round the heights to the closest inch. Plot the estimated conditional probability $P(x) = \Pr(\text{Male}|\text{height} = x)$.

Part of the code is provided here:

```
data("heights")
# MISSING CODE
    qplot(height, p, data =.)
```

Which of the following blocks of code can be used to replace **# MISSING CODE** to make the correct plot?

☐ A.

```
heights %>%
    group_by(height) %>%
    summarize(p = mean(sex == "Male")) %>%
```

☐ B.

```
heights %>%
    mutate(height = round(height)) %>%
    group_by(height) %>%
    summarize(p = mean(sex == "Female")) %>%
```

☐ C.

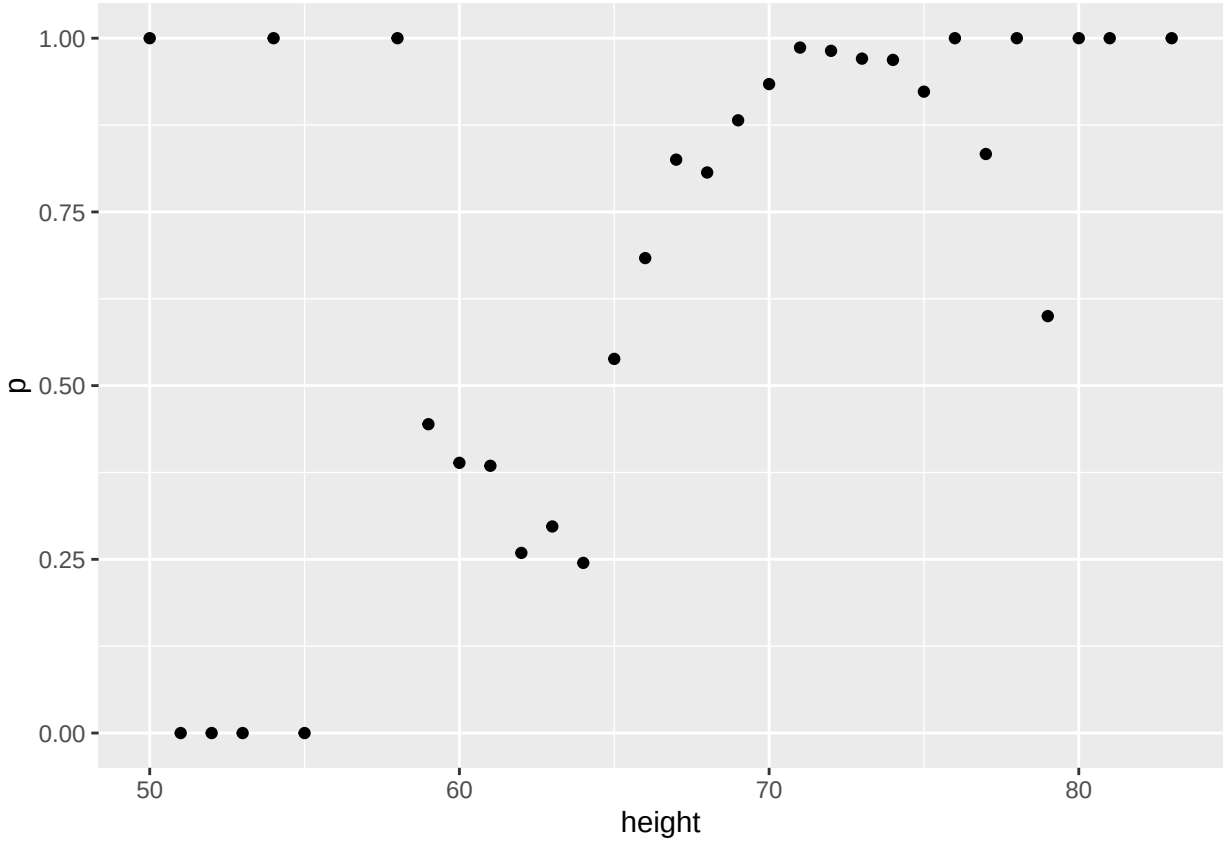
```
heights %>%
    mutate(height = round(height)) %>%
    summarize(p = mean(sex == "Male")) %>%
```

☒ D.

```
heights %>%
    mutate(height = round(height)) %>%
    group_by(height) %>%
    summarize(p = mean(sex == "Male")) %>%
```

```
data("heights")
heights %>%
    mutate(height = round(height)) %>%
    group_by(height) %>%
    summarize(p = mean(sex == "Male")) %>%
    qplot(height, p, data =.)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



7. In the plot we just made in Q6 we see high variability for low values of height. This is because we have few data points. This time use the quantile $0.1, 0.2, \ldots, 0.9$ and the `cut()` function to assure each group has the same number of points. Note that for any numeric vector `x`, you can create groups based on quantiles like this: `cut(x, quantile(x, seq(0, 1, 0.1)), include.lowest = TRUE)`.

Part of the code is provided here:

```
ps <- seq(0, 1, 0.1)
heights %>%
    # MISSING CODE
```

```
    group_by(g) %>%
    summarize(p = mean(sex == "Male"), height = mean(height)) %>%
    qplot(height, p, data =.)
```

Which of the following lines of code can be used to replace # **MISSING CODE** to make the correct plot?

☐ A.

```
mutate(g = cut(male, quantile(height, ps), include.lowest = TRUE)) %>%
```

☒ B.

```
mutate(g = cut(height, quantile(height, ps), include.lowest = TRUE)) %>%
```

☐ C.

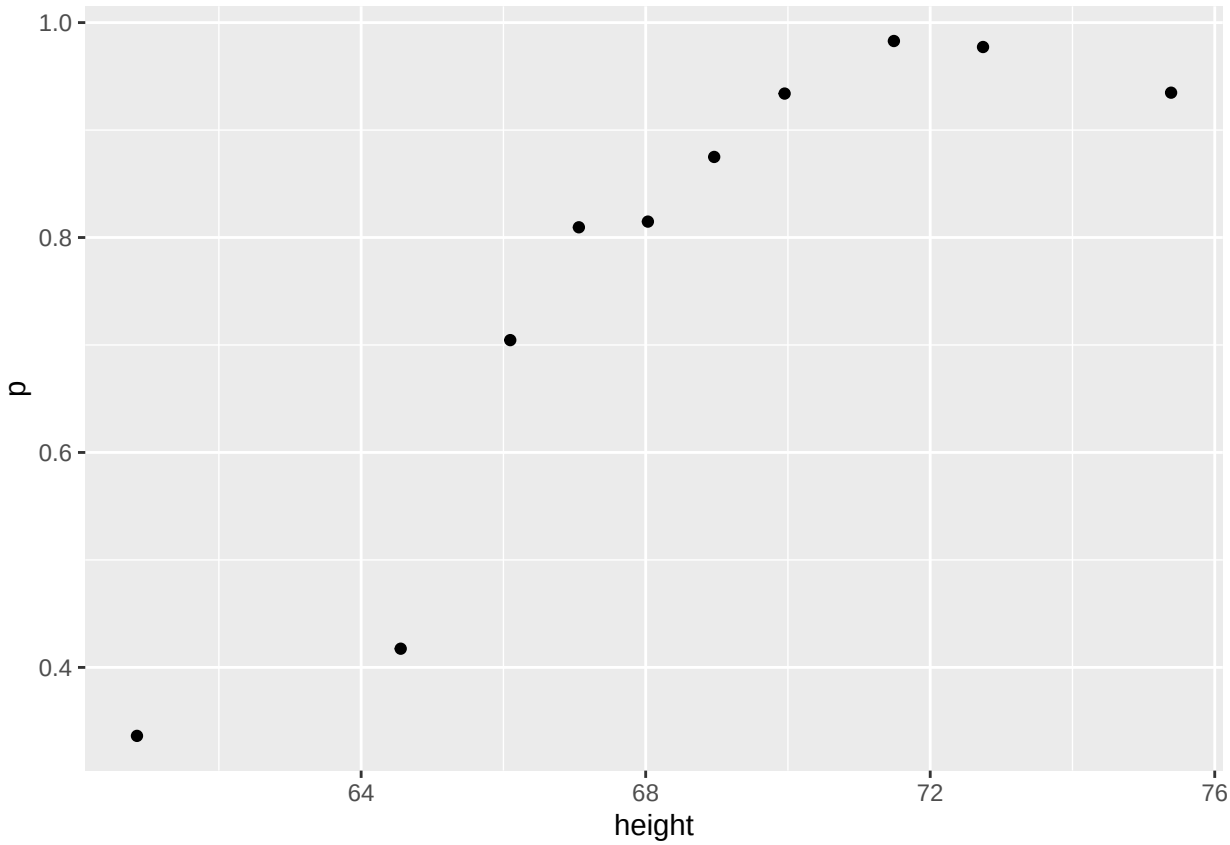
```
mutate(g = cut(female, quantile(height, ps), include.lowest = TRUE)) %>%
```

☐ D.

```
mutate(g = cut(height, quantile(height, ps))) %>%
```

```
ps <- seq(0, 1, 0.1)
heights %>%
    mutate(g = cut(height, quantile(height, ps), include.lowest = TRUE)) %>%
    group_by(g) %>%
    summarize(p = mean(sex == "Male"), height = mean(height)) %>%
    qplot(height, p, data =.)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



8. You can generate data from a bivariate normal distrubution using the **MASS** package using the following code:

```
if(!require(MASS)) install.packages("MASS")
```

```
## Loading required package: MASS
```

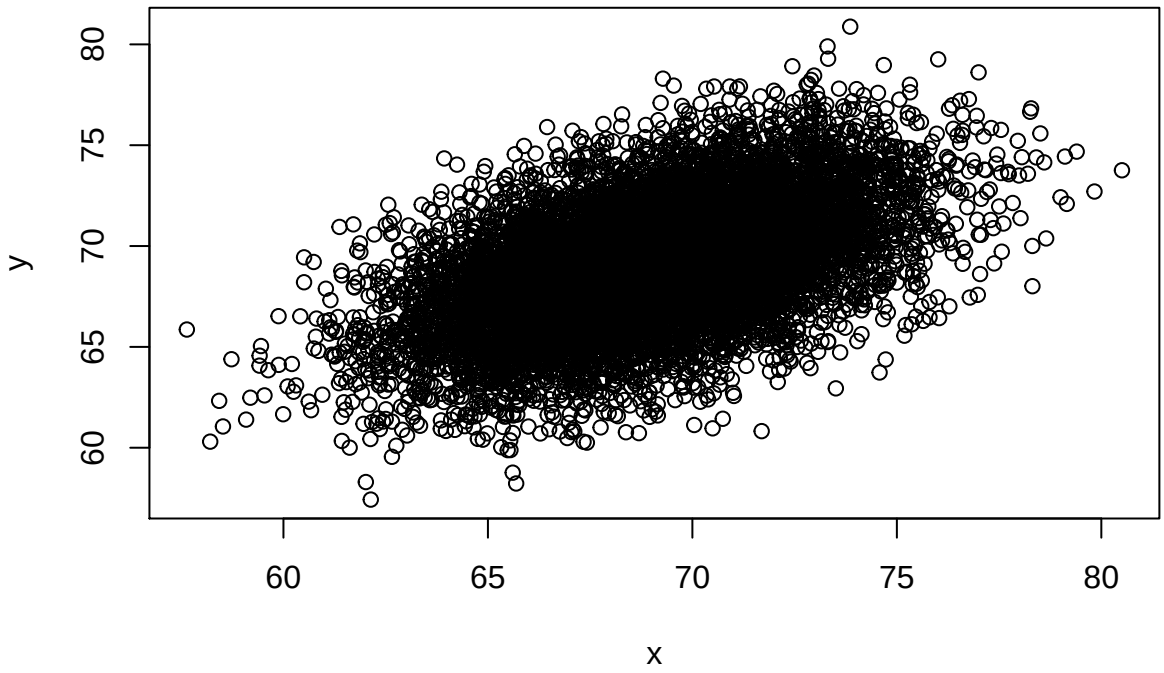
```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
Sigma <- 9*matrix(c(1,0.5,0.5,1), 2, 2)
dat <- MASS::mvrnorm(n = 10000, c(69, 69), Sigma) %>%
    data.frame() %>% setNames(c("x", "y"))
```

And you can make a quick plot using `plot(dat)`.

```
plot(dat)
```



Using an approach similar to that used in the previous exercise, let's estimate the conditional expectations and make a plot. Part of the code has again been provided for you:

```
ps <- seq(0, 1, 0.1)
dat %>%
    # MISSING CODE
    qplot(x, y, data =.)
```

Which of the following blocks of code can be used to replace # **MISSING CODE** to make the correct plot?

☒ A.

```
mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%
group_by(g) %>%
summarize(y = mean(y), x = mean(x)) %>%
```

☐ B.

```
mutate(g = cut(x, quantile(x, ps))) %>%
group_by(g) %>%
summarize(y = mean(y), x = mean(x)) %>%
```

☐ C.

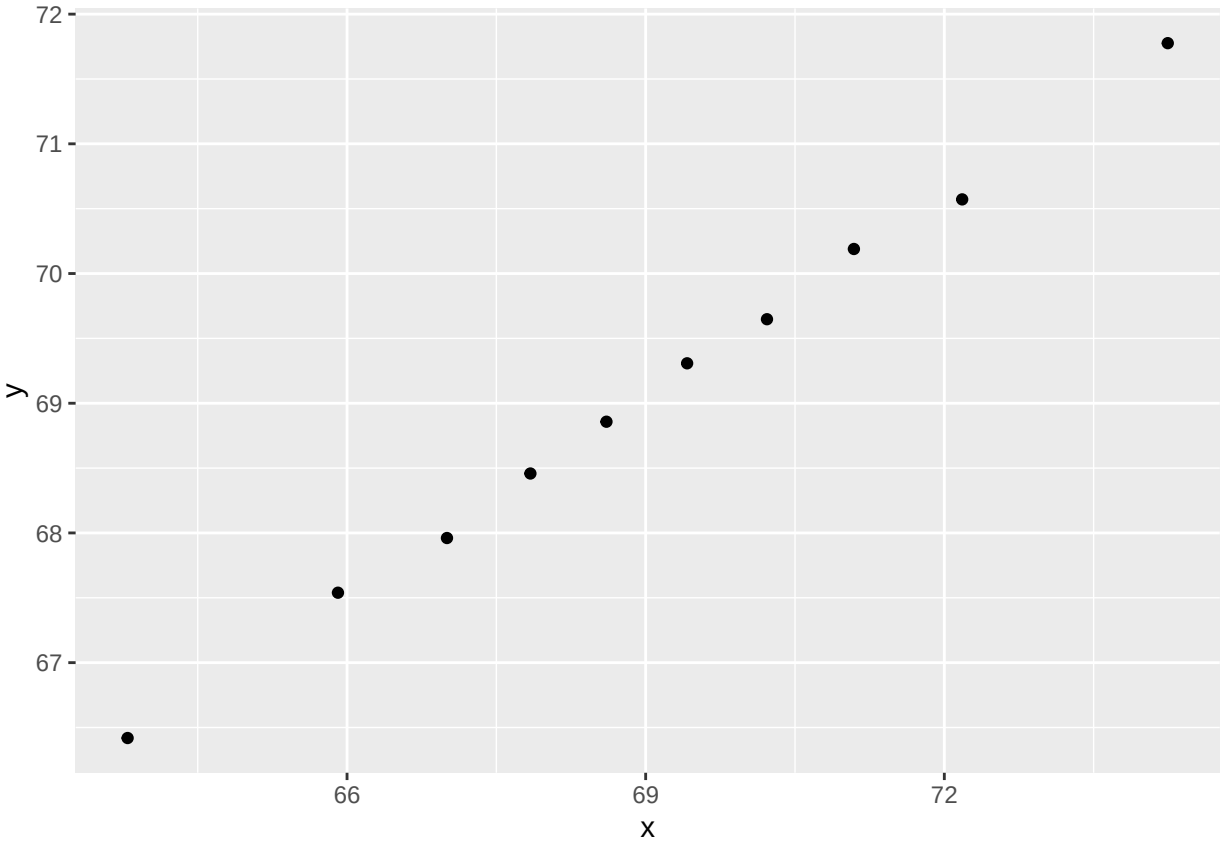
```
mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%
summarize(y = mean(y), x = mean(x)) %>%
```

☐ D.

```
mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%
group_by(g) %>%
summarize(y =(y), x =(x)) %>%
```

```
ps <- seq(0, 1, 0.1)
dat %>%
    mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%
    group_by(g) %>%
    summarize(y = mean(y), x = mean(x)) %>%
    qplot(x, y, data =.)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Section 3 - Linear Regression for Prediction, Smoothing, and Working with Matrices Overview

In the **Linear Regression for Prediction, Smoothing, and Working with Matrices Overview** section, you will learn why linear regression is a useful baseline approach but is often insufficiently flexible for more complex analyses, how to smooth noisy data, and how to use matrices for machine learning.

After completing this section, you will be able to:

- Use **linear regression for prediction** as a baseline approach.

- Use **logistic regression** for categorical data.
- Detect trends in noisy data using **smoothing** (also known as **curve fitting** or **low pass filtering**).
- Convert predictors to **matrices** and outcomes to **vectors** when all predictors are numeric (or can be converted to numerics in a meaningful way).
- Perform basic **matrix algebra** calculations.

This section has three parts: **linear regression for prediction**, **smoothing**, and **working with matrices**.

Linear Regression for Prediction

There is a link to the relevant section of the textbook: Linear regression for prediction

Key points

- Linear regression can be considered a machine learning algorithm. Although it can be too rigid to be useful, it works rather well for some challenges. It also serves as a baseline approach: if you can't beat it with a more complex approach, you probably want to stick to linear regression.

Code

Note: the seed was not set before `createDataPartition` so your results may be different.

```
if(!require(HistData)) install.packages("HistData")
```

```
## Loading required package: HistData
```

```
library(HistData)

galton_heights <- GaltonFamilies %>%
  filter(childNum == 1 & gender == "male") %>%
  dplyr::select(father, childHeight) %>%
  rename(son = childHeight)

y <- galton_heights$son
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)

train_set <- galton_heights %>% slice(-test_index)
test_set <- galton_heights %>% slice(test_index)

avg <- mean(train_set$son)
avg
```

```
## [1] 70.50114
```

```
mean((avg - test_set$son)^2)
```

```
## [1] 6.034931
```

```
# fit linear regression model
fit <- lm(son ~ father, data = train_set)
fit$coef
```

```
## (Intercept)      father
##  34.8934373   0.5170499
```

```
y_hat <- fit$coef[1] + fit$coef[2]*test_set$father
mean((y_hat - test_set$son)^2)
```

```
## [1] 4.632629
```

Predict Function

There is a link to the relevant section of the textbook: Predict function

Key points

- The `predict()` function takes a fitted object from functions such as `lm()` or `glm()` and a data frame with the new predictors for which to predict. We can use predict like this:

```
y_hat <- predict(fit, test_set)
```

- `predict()` is a generic function in R that calls other functions depending on what kind of object it receives. To learn about the specifics, you can read the help files using code like this:

```
?predict.lm    # or ?predict.glm
```

Code

```
y_hat <- predict(fit, test_set)
mean((y_hat - test_set$son)^2)
```

```
## [1] 4.632629
```

```
# read help files
?predict.lm
?predict.glm
```

Comprehension Check - Linear Regression

1. Create a data set using the following code:

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
n <- 100
Sigma <- 9*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%
      data.frame() %>% setNames(c("x", "y"))
```

We will build 100 linear models using the data above and calculate the mean and standard deviation of the combined models. First, set the seed to 1 again (make sure to use `sample.kind="Rounding"` if your R is version 3.6 or later). Then, within a `replicate()` loop, (1) partition the dataset into test and training sets with `p = 0.5` and using `dat$y` to generate your indices, (2) train a linear model predicting `y` from `x`, (3) generate predictions on the test set, and (4) calculate the RMSE of that model. Then, report the mean and standard deviation (SD) of the RMSEs from all 100 models.

Report all answers to at least 3 significant digits.

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
rmse <- replicate(100, {
    test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
    train_set <- dat %>% slice(-test_index)
    test_set <- dat %>% slice(test_index)
    fit <- lm(y ~ x, data = train_set)
    y_hat <- predict(fit, newdata = test_set)
    sqrt(mean((y_hat-test_set$y)^2))
})

mean(rmse)
```

```
## [1] 2.488661
```

```
sd(rmse)
```

```
## [1] 0.1243952
```

2. Now we will repeat the exercise above but using larger datasets. Write a function that takes a size `n`, then (1) builds a dataset using the code provided at the top of Q1 but with `n` observations instead of 100 and without the `set.seed(1)`, (2) runs the `replicate()` loop that you wrote to answer Q1, which builds 100 linear models and returns a vector of RMSEs, and (3) calculates the mean and standard deviation of the 100 RMSEs.

Set the seed to 1 (if using R 3.6 or later, use the argument `sample.kind="Rounding"`) and then use `sapply()` or `map()` to apply your new function to `n <- c(100, 500, 1000, 5000, 10000)`.

Hint: You only need to set the seed once before running your function; do not set a seed within your function. Also be sure to use `sapply()` or `map()` as you will get different answers running the simulations individually due to setting the seed.

```
# set.seed(1) # if R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
n <- c(100, 500, 1000, 5000, 10000)
res <- sapply(n, function(n){
    Sigma <- 9*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)
    dat <- MASS::mvrnorm(n, c(69, 69), Sigma) %>%
        data.frame() %>% setNames(c("x", "y"))
    rmse <- replicate(100, {
        test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
        train_set <- dat %>% slice(-test_index)
        test_set <- dat %>% slice(test_index)
        fit <- lm(y ~ x, data = train_set)
        y_hat <- predict(fit, newdata = test_set)
        sqrt(mean((y_hat-test_set$y)^2))
    })
    c(avg = mean(rmse), sd = sd(rmse))
})

res
```

```
##          [,1]       [,2]       [,3]       [,4]       [,5]
## avg 2.4977540 2.72095125 2.55554451 2.62482800 2.61844227
## sd  0.1180821 0.08002108 0.04560258 0.02309673 0.01689205
```

3. What happens to the RMSE as the size of the dataset becomes larger?

☒ A. On average, the RMSE does not change much as n gets larger, but the variability of the RMSE decreases.
☐ B. Because of the law of large numbers the RMSE decreases; more data means more precise estimates.
☐ C. n = 10000 is not sufficiently large. To see a decrease in the RMSE we would need to make it larger.
☐ D. The RMSE is not a random variable.

4. Now repeat the exercise from Q1, this time making the correlation between `x` and `y` larger, as in the following code:

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
n <- 100
Sigma <- 9*matrix(c(1.0, 0.95, 0.95, 1.0), 2, 2)
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%
    data.frame() %>% setNames(c("x", "y"))
```

Note what happens to RMSE - set the seed to 1 as before.

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
rmse <- replicate(100, {
    test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
    train_set <- dat %>% slice(-test_index)
    test_set <- dat %>% slice(test_index)
    fit <- lm(y ~ x, data = train_set)
    y_hat <- predict(fit, newdata = test_set)
    sqrt(mean((y_hat-test_set$y)^2))
})

mean(rmse)
```

```
## [1] 0.9099808
```

```
sd(rmse)
```

```
## [1] 0.06244347
```

5. Which of the following best explains why the RMSE in question 4 is so much lower than the RMSE in question 1?

☐ A. It is just luck. If we do it again, it will be larger.
☐ B. The central limit theorem tells us that the RMSE is normal.
☒ C. When we increase the correlation between x and y, x has more predictive power and thus provides a better estimate of y.
☐ D. These are both examples of regression so the RMSE has to be the same.

6. Create a data set using the following code.

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.25, 0.75, 0.25, 1.0), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%
    data.frame() %>% setNames(c("y", "x_1", "x_2"))
```

Note that `y` is correlated with both `x_1` and `x_2` but the two predictors are independent of each other, as seen by `cor(dat)`.

Set the seed to 1, then use the **caret** package to partition into test and training sets with `p = 0.5`. Compare the RMSE when using just `x_1`, just `x_2` and both `x_1` and `x_2`. Train a single linear model for each (not 100 like in the previous questions).

Which of the three models performs the best (has the lowest RMSE)?

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
train_set <- dat %>% slice(-test_index)
test_set <- dat %>% slice(test_index)

fit <- lm(y ~ x_1, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))
```

```
## [1] 0.600666
```

```
fit <- lm(y ~ x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))
```

```
## [1] 0.630699
```

```
fit <- lm(y ~ x_1 + x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))
```

```
## [1] 0.3070962
```

- ☐ A. x_1
- ☐ B. x_2
- ☒ C. x_1 and x_2

7. Report the lowest RMSE of the three models tested in Q6.

```
fit <- lm(y ~ x_1 + x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))
```

```
## [1] 0.3070962
```

8. Repeat the exercise from Q6 but now create an example in which `x_1` and `x_2` are highly correlated.

```
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.95, 0.75, 0.95, 1.0), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%
    data.frame() %>% setNames(c("y", "x_1", "x_2"))
```

Set the seed to 1, then use the **caret** package to partition into a test and training set of equal size. Compare the RMSE when using just `x_1`, just `x_2`, and both `x_1` and `x_2`.

Compare the results from Q6 and Q8. What can you conclude?

```r
# set.seed(1) # if using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
train_set <- dat %>% slice(-test_index)
test_set <- dat %>% slice(test_index)

fit <- lm(y ~ x_1, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))
```

```
## [1] 0.6592608
```

```r
fit <- lm(y ~ x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))
```

```
## [1] 0.640081
```

```r
fit <- lm(y ~ x_1 + x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))
```

```
## [1] 0.6597865
```

- ☐ A. Unless we include all predictors we have no predictive power.
- ☐ B. Adding extra predictors improves RMSE regardless of whether the added predictors are correlated with other predictors or not.
- ☐ C. Adding extra predictors results in over fitting.
- ☒ D. Adding extra predictors can improve RMSE substantially, but not when the added predictors are highly correlated with other predictors.

Regression for a Categorical Outcome

There is a link to the relevant section of the textbook: Regression for a categorical outcome

Key points

- The regression approach can be extended to categorical data. For example, we can try regression to estimate the conditional probability:

$p(x) = Pr(Y = 1|X = x) = \beta_0 + \beta_1 x$

- Once we have estimates β_0 and β_1, we can obtain an actual prediction $p(x)$. Then we can define a specific decision rule to form a prediction.

Code

```r
data("heights")
y <- heights$height

set.seed(2) #if you are using R 3.5 or earlier
set.seed(2, sample.kind = "Rounding") #if you are using R 3.6 or later
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

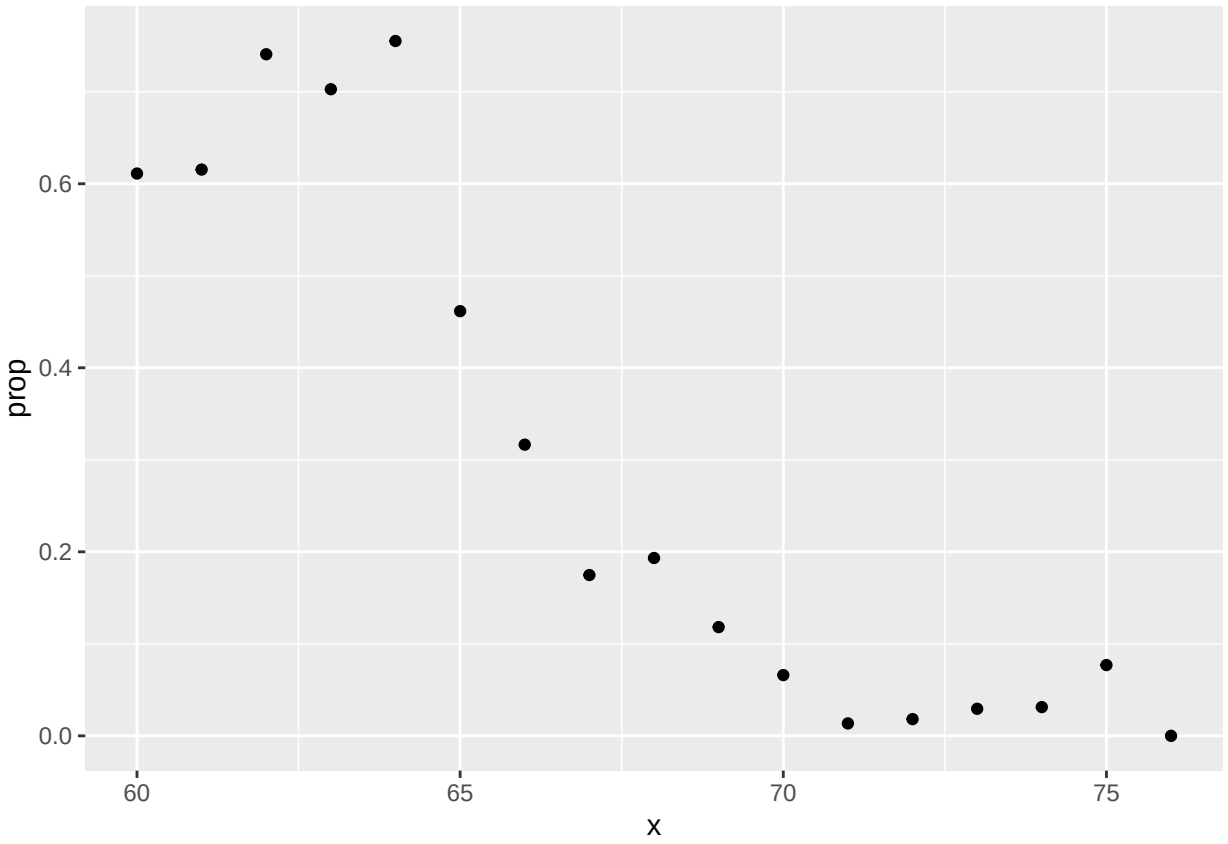
```r
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- heights %>% slice(-test_index)
test_set <- heights %>% slice(test_index)

train_set %>%
  filter(round(height)==66) %>%
  summarize(y_hat = mean(sex=="Female"))
```

```
##       y_hat
## 1 0.2424242
```

```r
heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(prop = mean(sex == "Female")) %>%
  ggplot(aes(x, prop)) +
  geom_point()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
lm_fit <- mutate(train_set, y = as.numeric(sex == "Female")) %>% lm(y ~ height, data = .)
p_hat <- predict(lm_fit, test_set)
y_hat <- ifelse(p_hat > 0.5, "Female", "Male") %>% factor()
confusionMatrix(y_hat, test_set$sex)$overall["Accuracy"]
```

```
##  Accuracy
## 0.7851711
```

Logistic Regression