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 --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     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 ------------------------------------------ tidyverse_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))
```

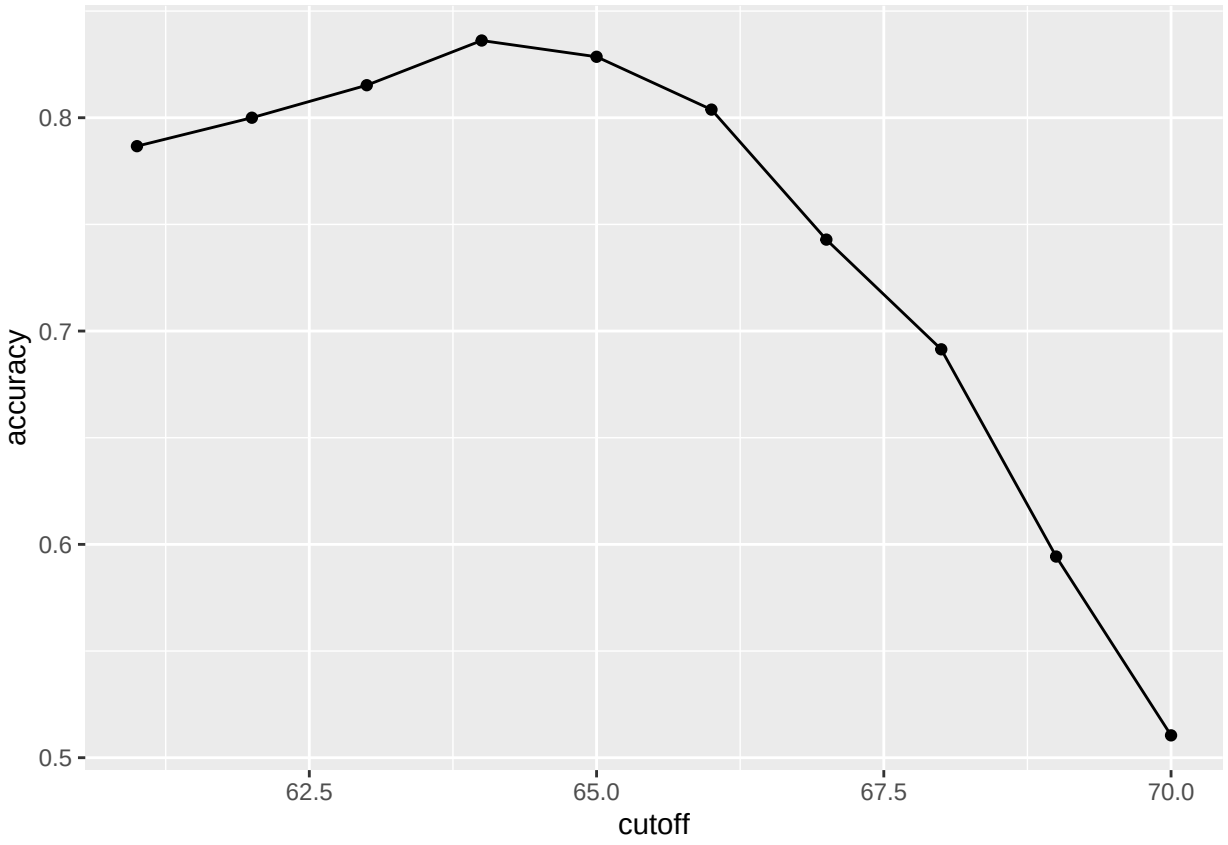
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # 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

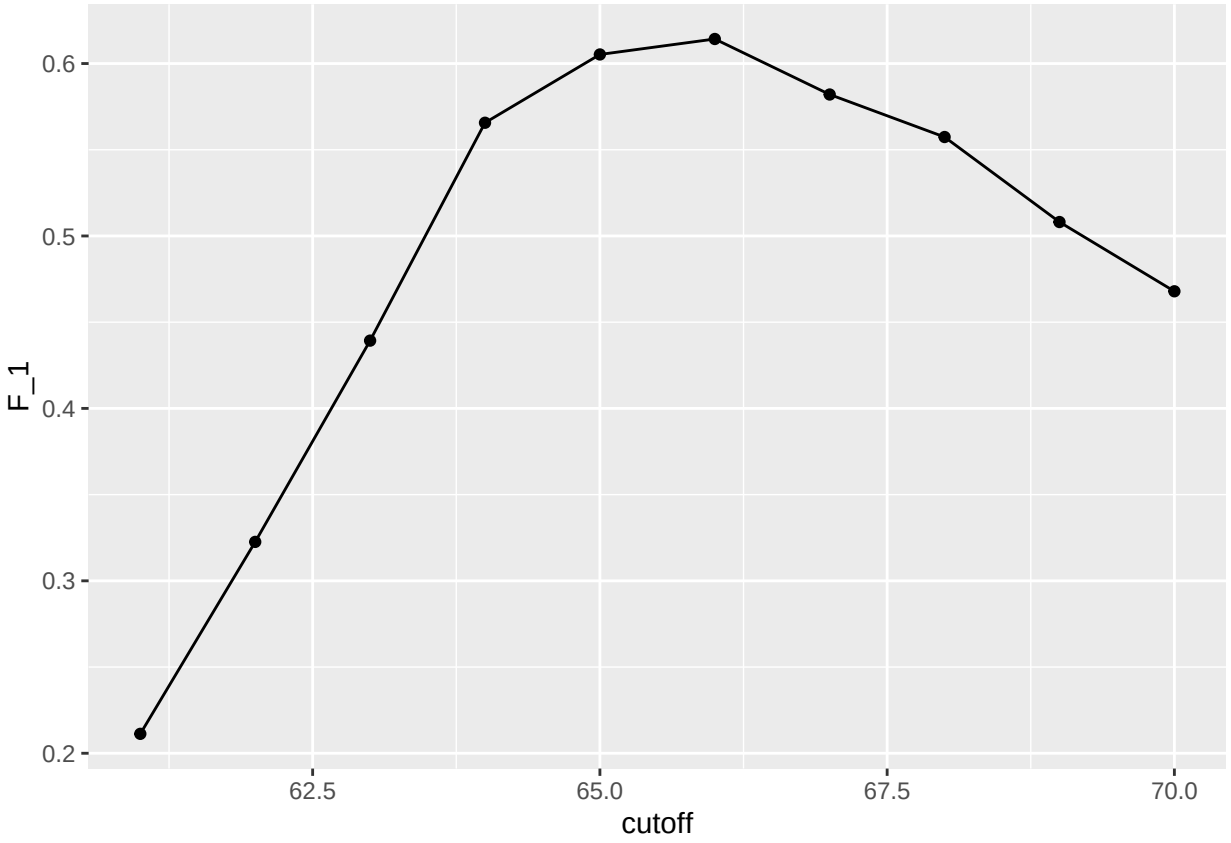
Key points

- 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.

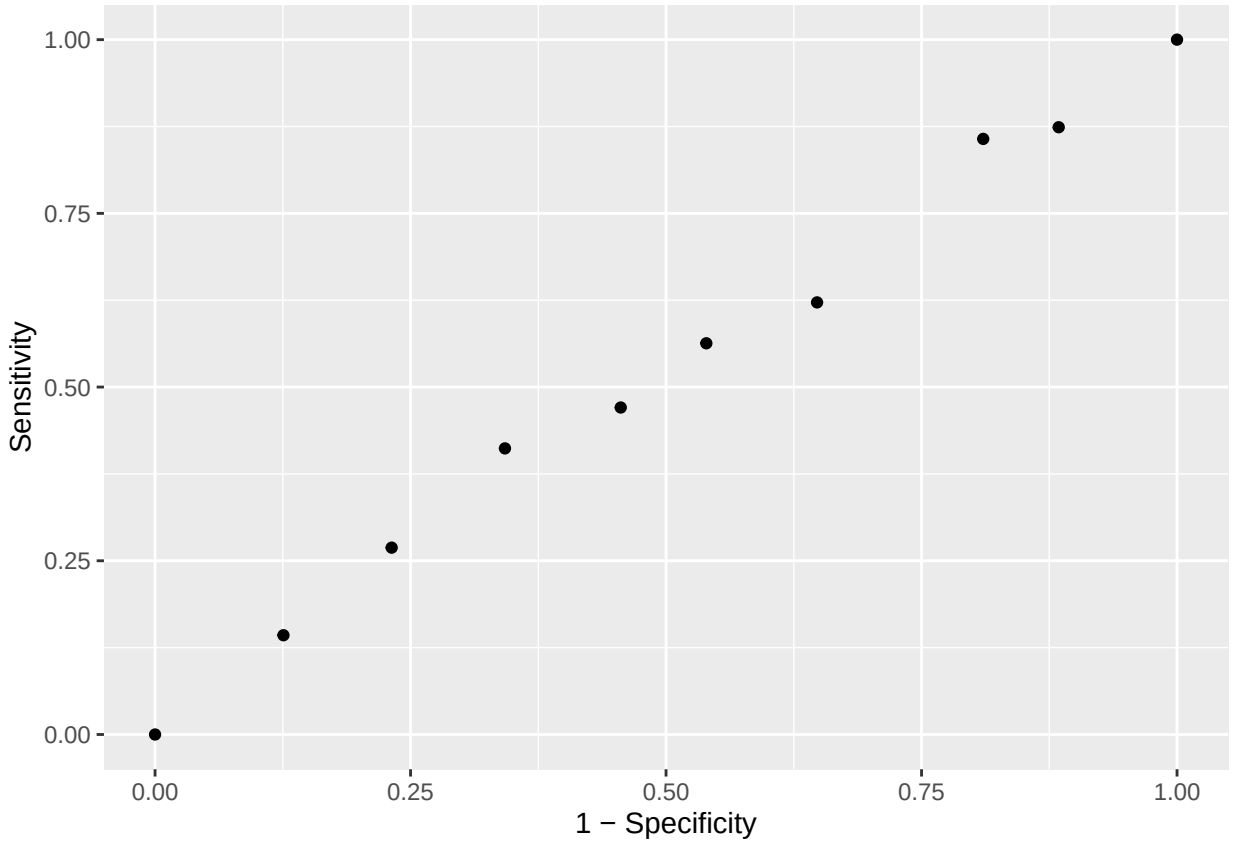
Code

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
```

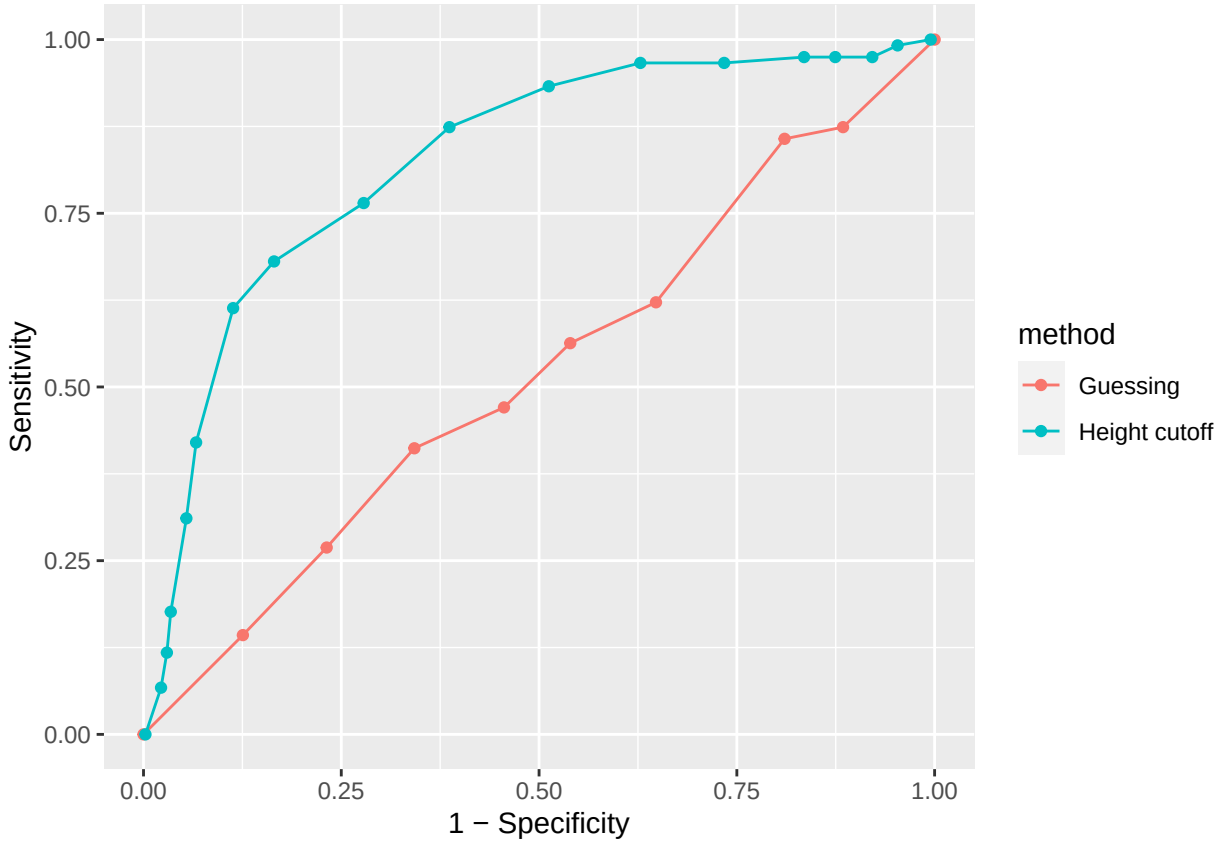
```
# ROC curve
probs <- seq(0, 1, length.out = 10)
guessing <- map_df(probs, function(p){
  y_hat <-
    sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guessing",
       FPR = 1 - specificity(y_hat, test_set$sex),
       TPR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

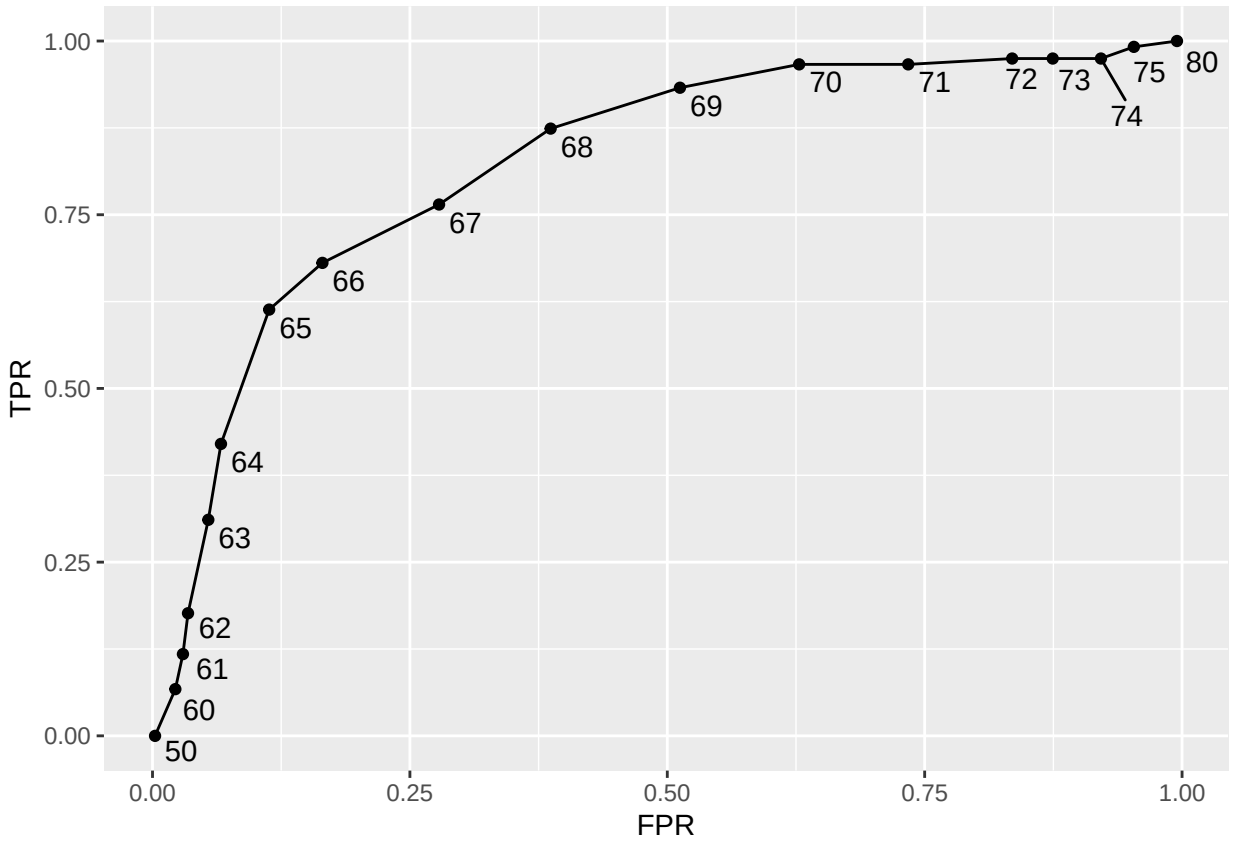
```
  geom_line() +
  geom_point() +
  xlab("1 - Specificity") +
  ylab("Sensitivity")
```




```
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)
```



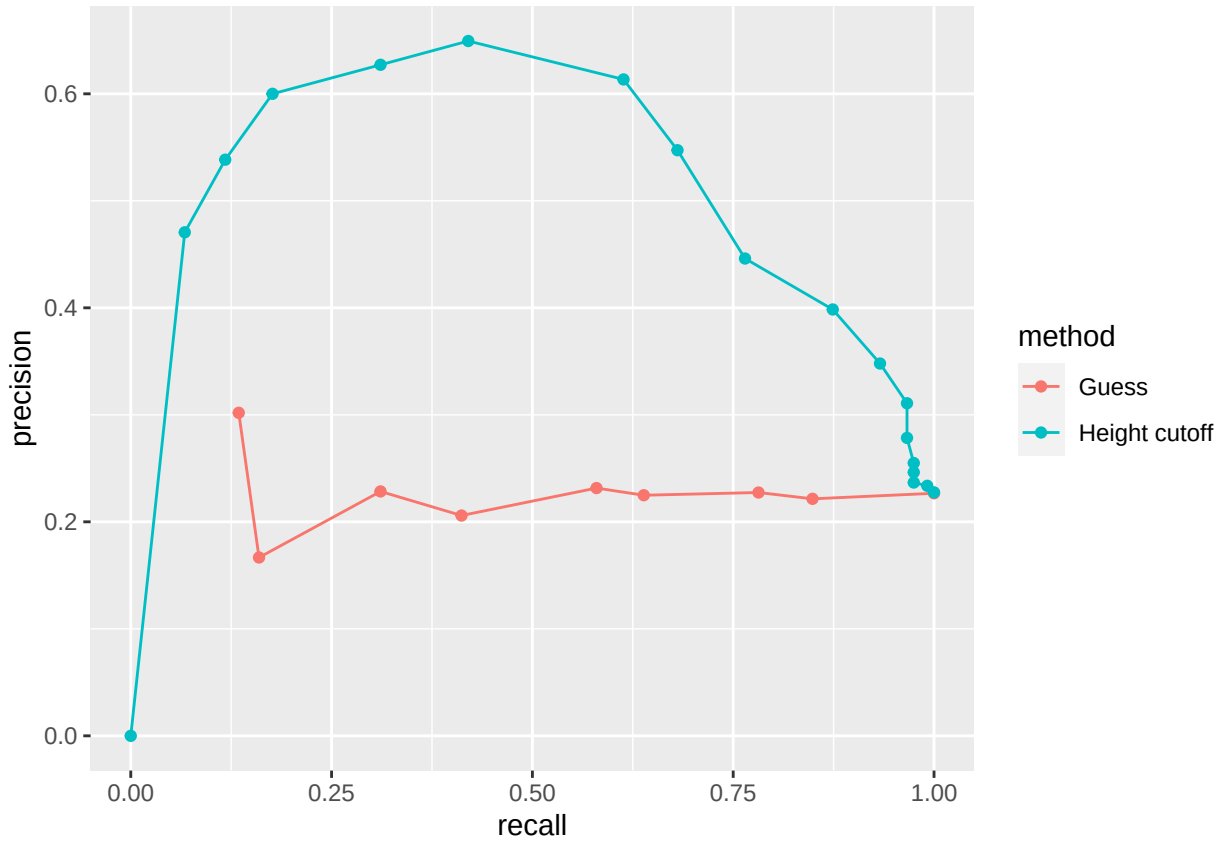

```
# plot precision against recall
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index),
                  replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guess",
    recall = sensitivity(y_hat, test_set$sex),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

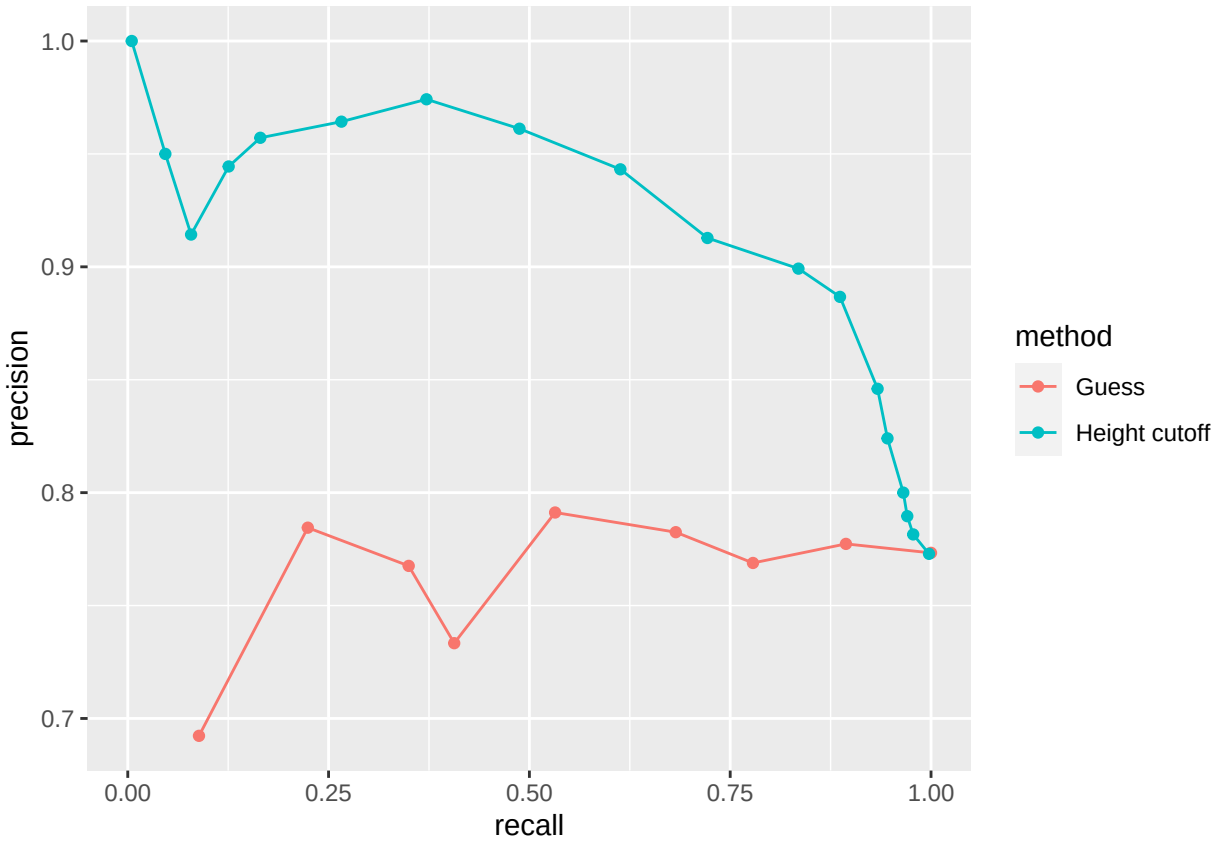



```
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE,
                  prob=c(p, 1-p)) %>%
    factor(levels = c("Male", "Female"))
  list(method = "Guess",
    recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
    precision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

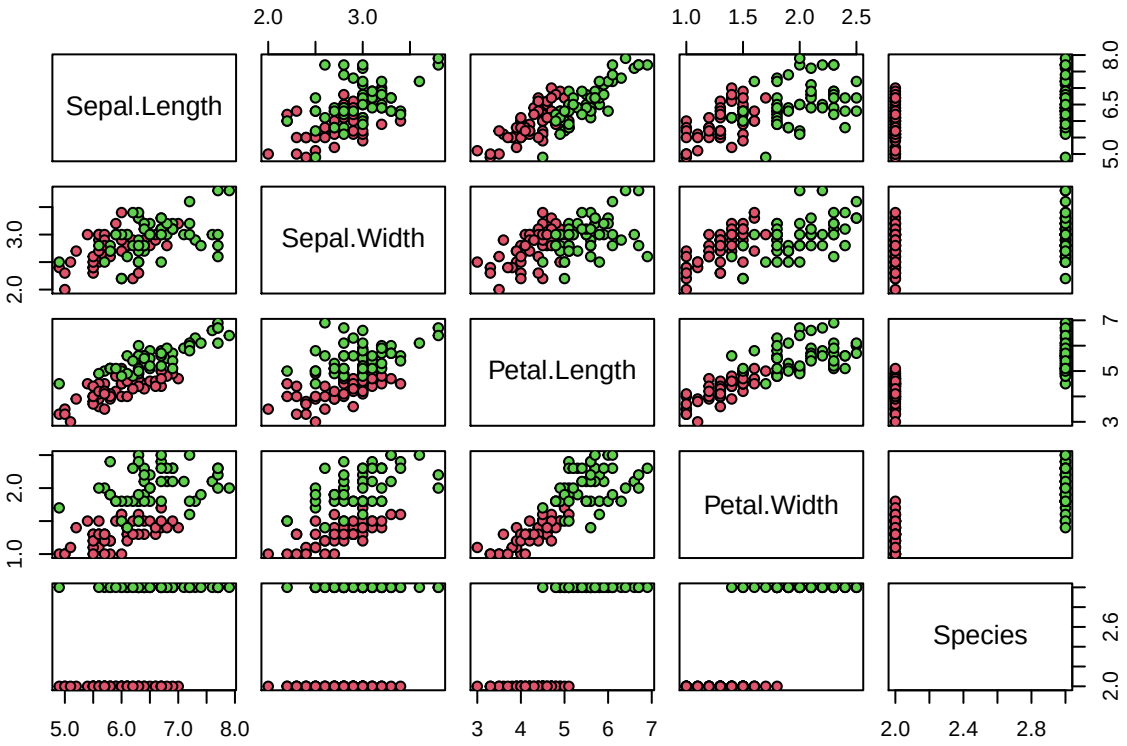
Notice that `Petal.Length` and `Petal.Width` in combination could potentially be more information than either feature alone.

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.

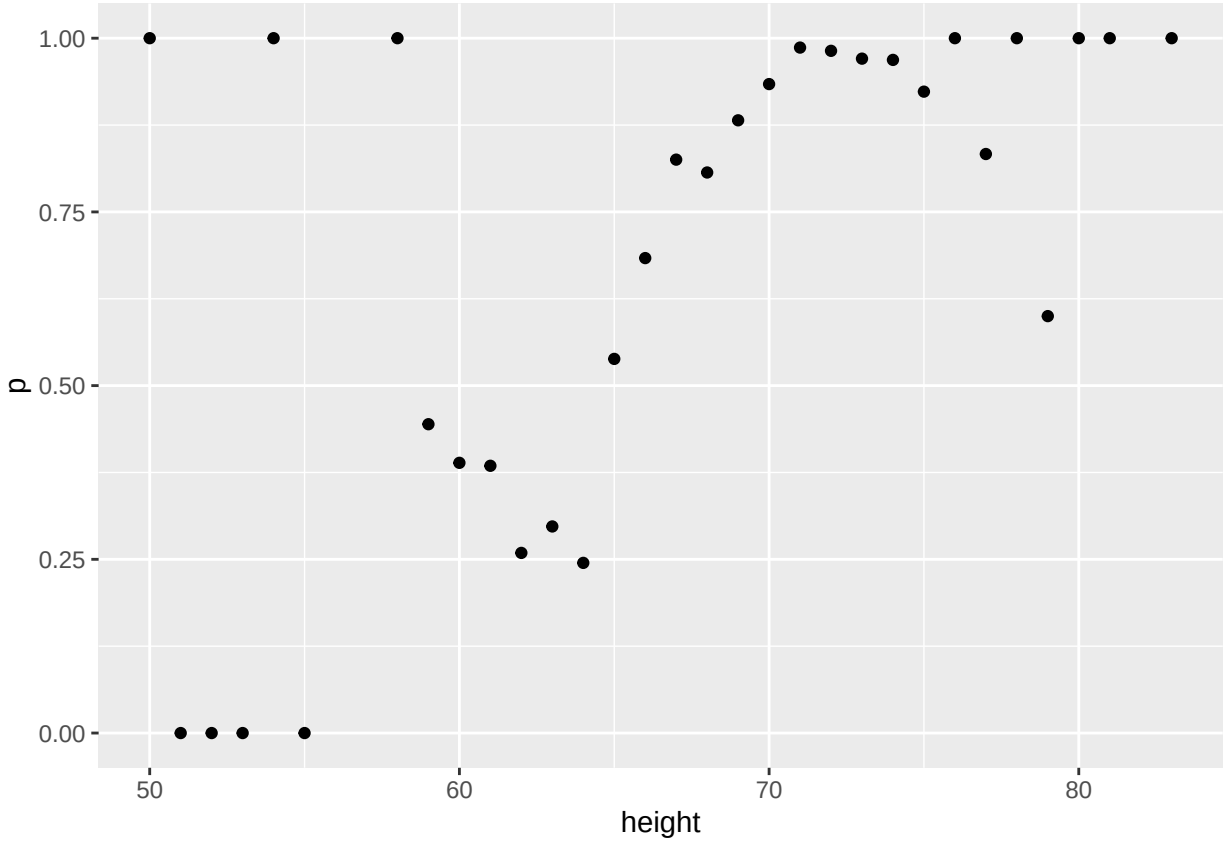
```
heights %>%
    mutate(height = round(height)) %>%
    summarize(p = mean(sex == "Male")) %>%
```

☒ 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.

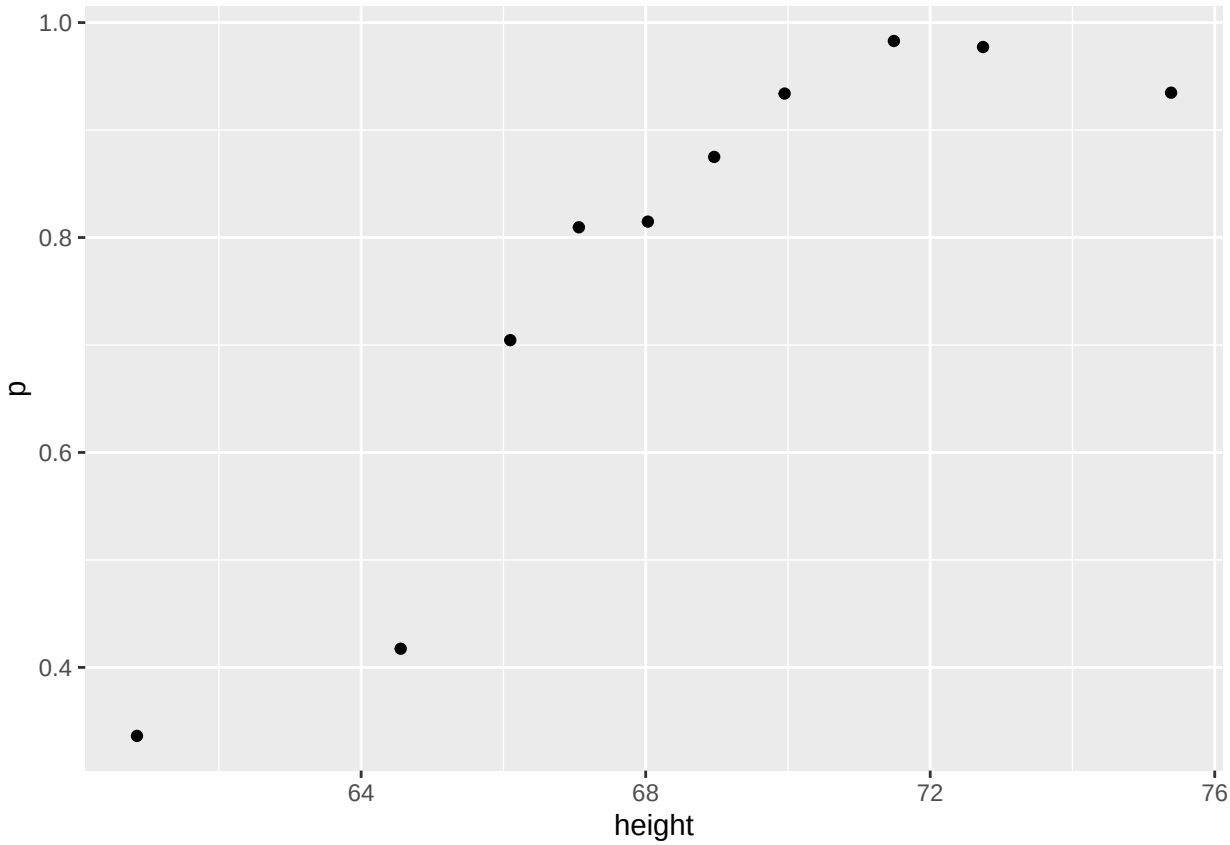
```
mutate(g = cut(female, quantile(height, ps), include.lowest = TRUE)) %>%
```

☐ 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
```

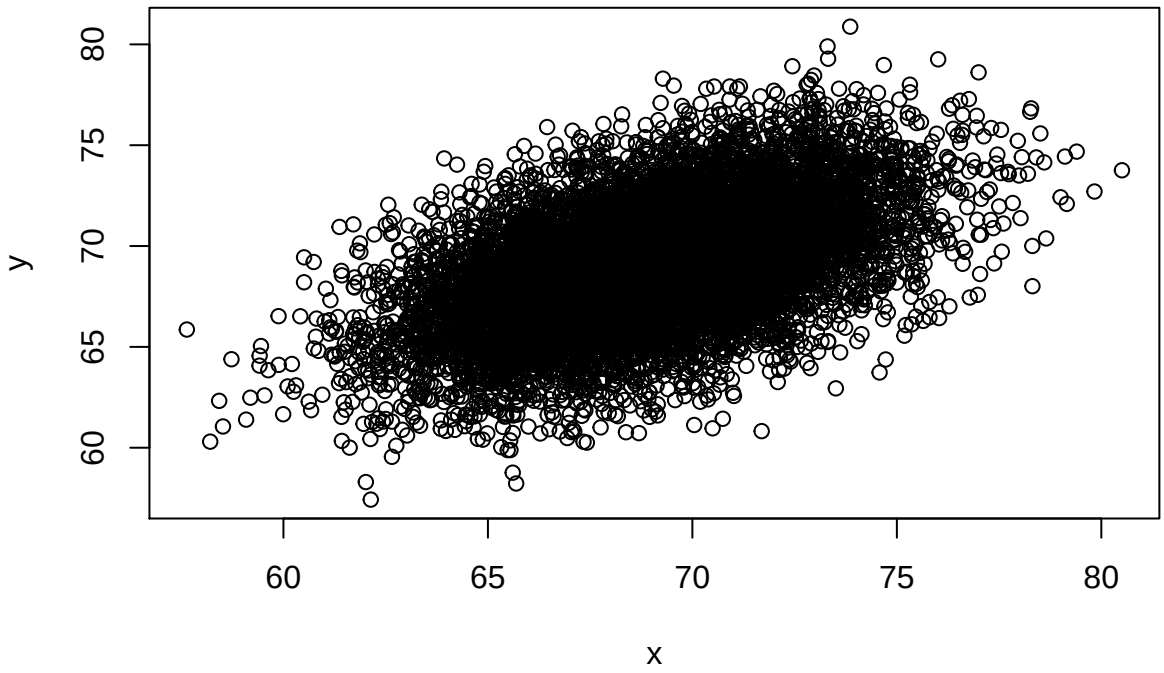
```
##
## Attaching package: 'MASS'
```

```
## 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.

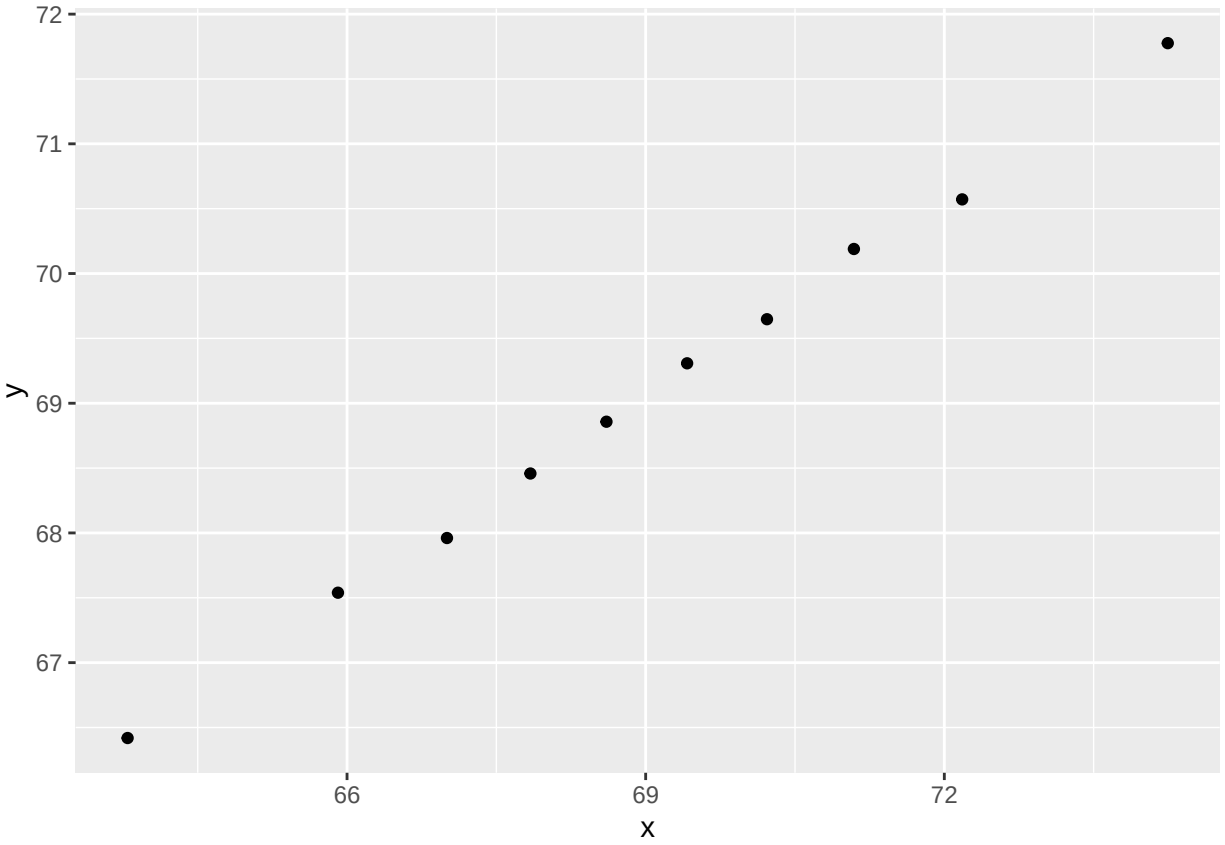
```
mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%
summarize(y = mean(y), x = mean(x)) %>%
```

☐ 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
```

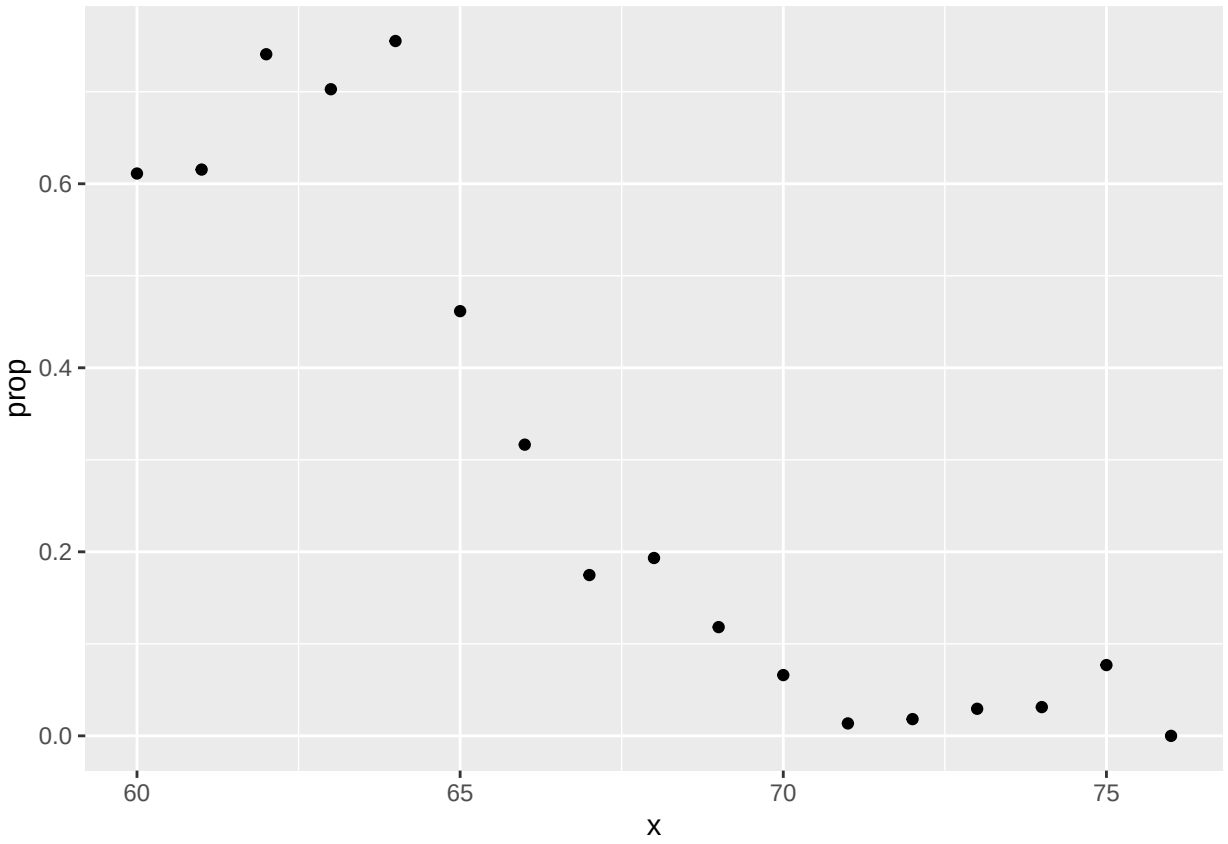
```r
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- heights %>% slice(-test_index)
test_set <- heights %>% slice(test_index)

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

There is a link to the relevant section of the textbook: Logistic regression

Key points

- **Logistic regression** is an extension of linear regression that assures that the estimate of conditional probability $Pr(Y = 1|X = x)$ is between 0 and 1. This approach makes use of the logistic transformation:

$g(p) = log\frac{p}{1-p}$

- With logistic regression, we model the conditional probability directly with:

$g\{Pr(Y = 1|X = x)\} = \beta_0 + \beta_1 x$

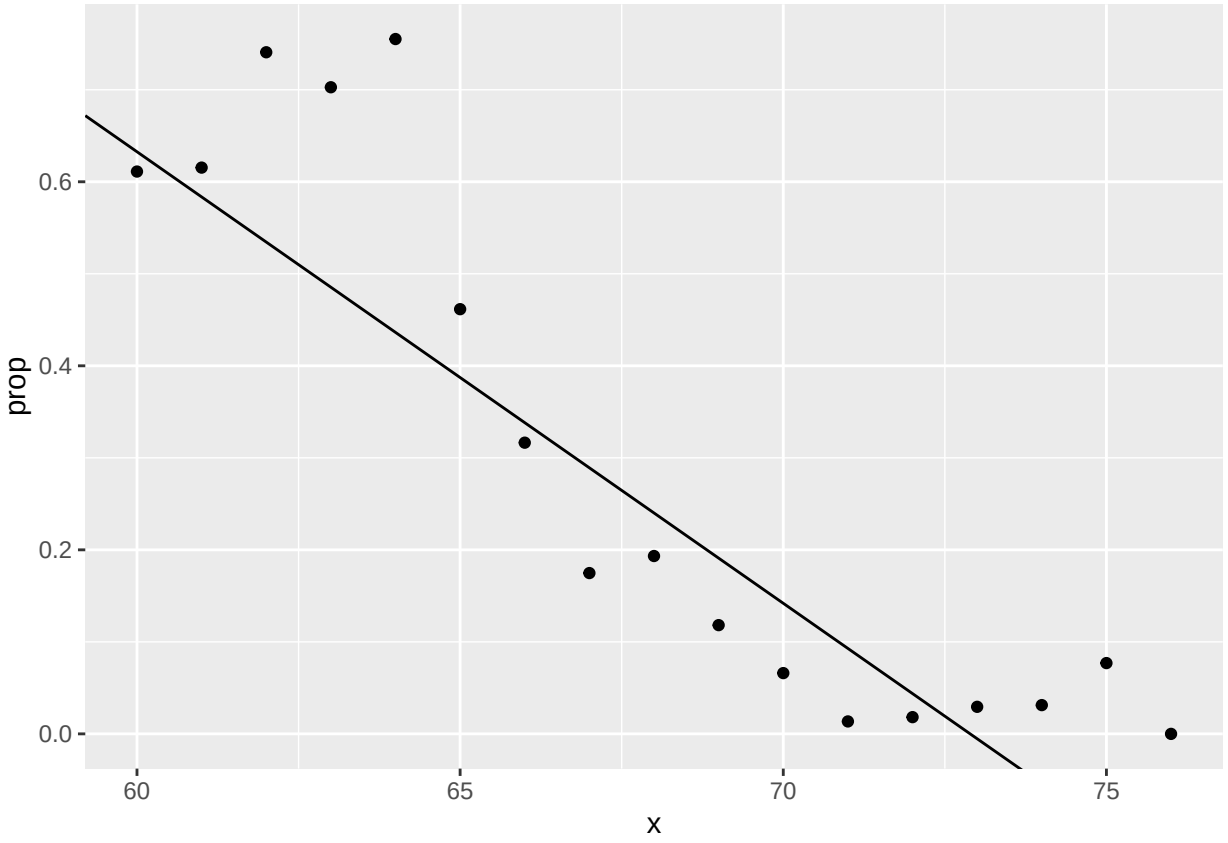
- Note that with this model, we can no longer use least squares. Instead we compute the **maximum likelihood estimate (MLE)**.

- In R, we can fit the logistic regression model with the function `glm()` (generalized linear models). If we want to compute the conditional probabilities, we want `type="response"` since the default is to return the logistic transformed values.

Code

```
heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(prop = mean(sex == "Female")) %>%
  ggplot(aes(x, prop)) +
  geom_point() +
  geom_abline(intercept = lm_fit$coef[1], slope = lm_fit$coef[2])
```

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




```
range(p_hat)
```

```
## [1] -0.397868  1.123309
```

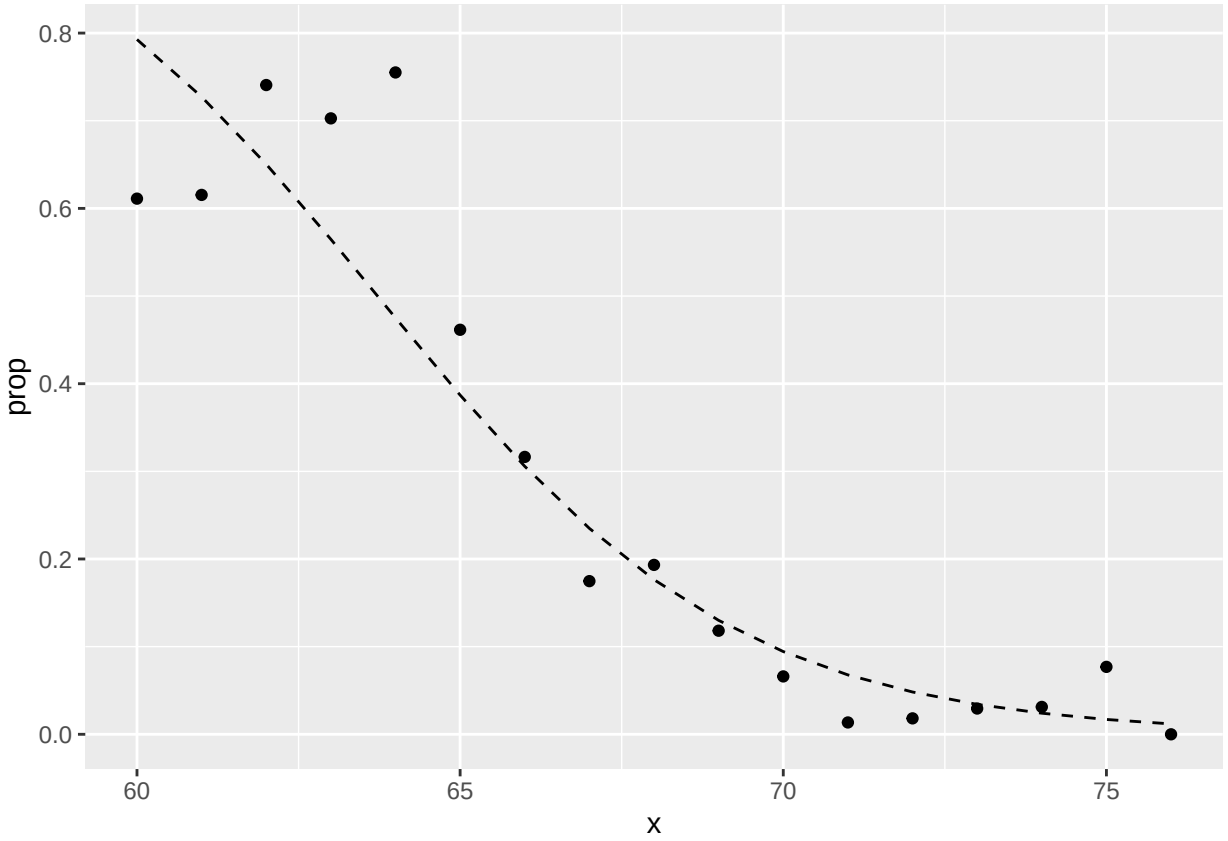
```
# fit logistic regression model
glm_fit <- train_set %>%
  mutate(y = as.numeric(sex == "Female")) %>%
  glm(y ~ height, data=., family = "binomial")
```

```
p_hat_logit <- predict(glm_fit, newdata = test_set, type = "response")

tmp <- heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(prop = mean(sex == "Female"))
```

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

```
logistic_curve <- data.frame(x = seq(min(tmp$x), max(tmp$x))) %>%
  mutate(p_hat = plogis(glm_fit$coef[1] + glm_fit$coef[2]*x))
tmp %>%
  ggplot(aes(x, prop)) +
  geom_point() +
  geom_line(data = logistic_curve, mapping = aes(x, p_hat), lty = 2)
```




```
y_hat_logit <- ifelse(p_hat_logit > 0.5, "Female", "Male") %>% factor
confusionMatrix(y_hat_logit, test_set$sex)$overall[["Accuracy"]]
```

```
## [1] 0.7984791
```

Case Study: 2 or 7

There is a link to the relevant section of the textbook: Case study: 2 or 7

Key points

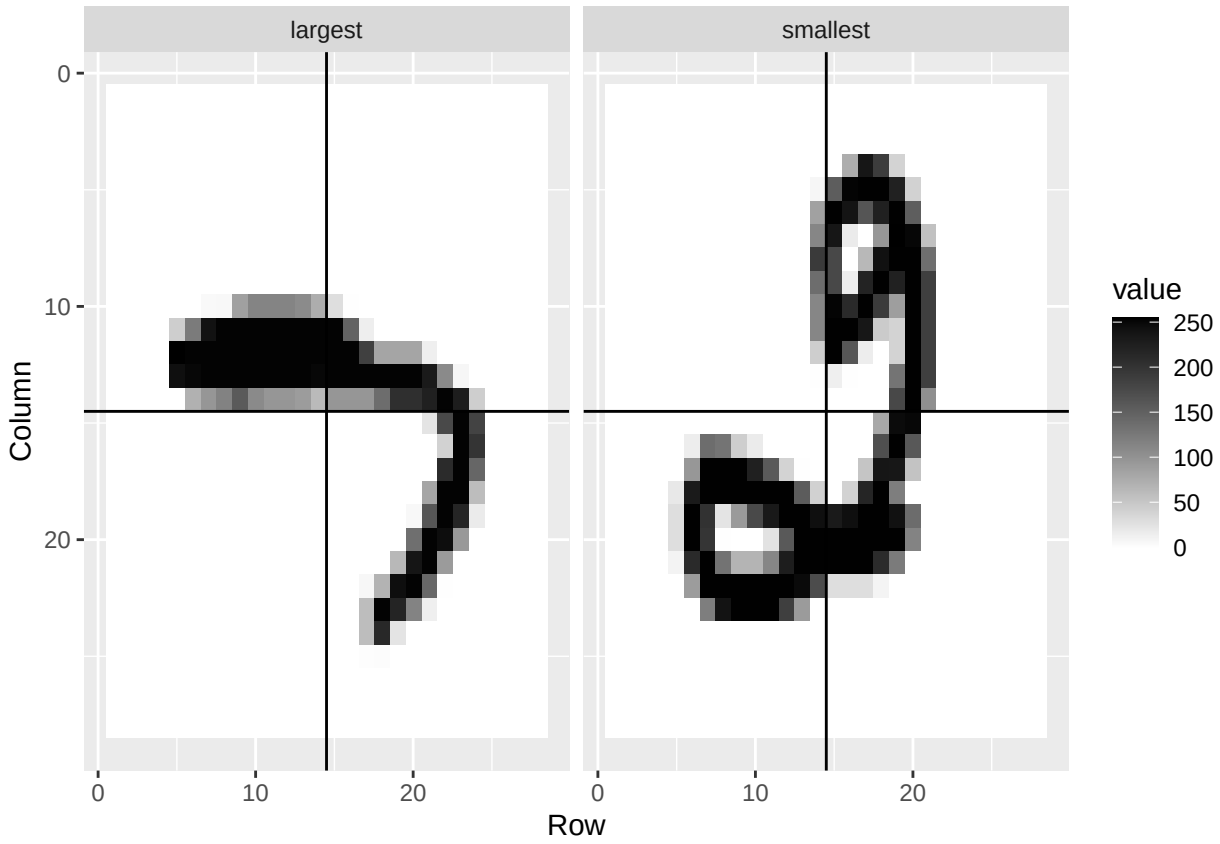
- In this case study we apply logistic regression to classify whether a digit is two or seven. We are interested in estimating a conditional probability that depends on two variables:

$$g\{p(x_1, x_2\} = g\{Pr(Y = 1|X_1 = x_1, X_2 = x_2)\} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

- Through this case, we know that logistic regression forces our estimates to be a **plane** and our boundary to be a **line**. This implies that a logistic regression approach has no chance of capturing the **non-linear** nature of the true $p(x_1, x_2)$. Therefore, we need other more flexible methods that permit other shapes.

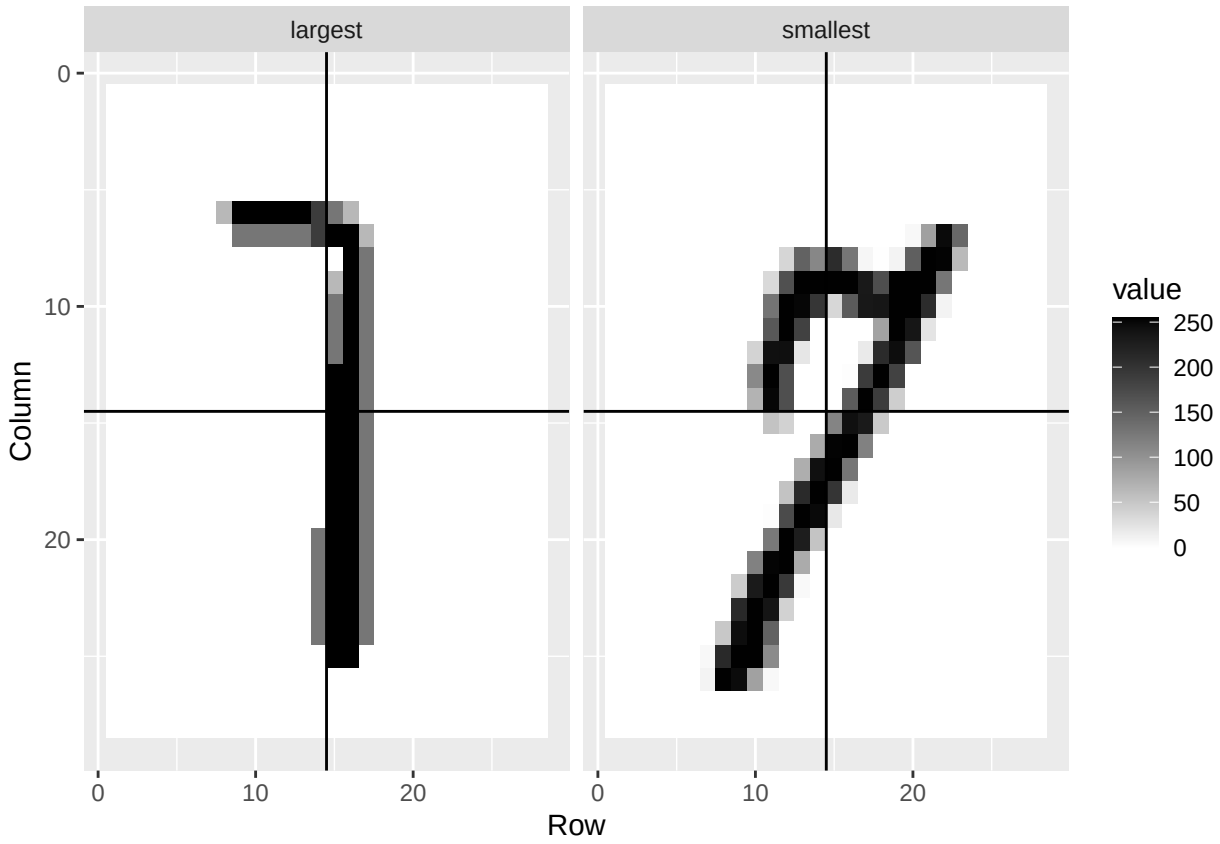
Code

```
mnist <- read_mnist()
is <- mnist_27$index_train[c(which.min(mnist_27$train$x_1), which.max(mnist_27$train$x_1))]
titles <- c("smallest","largest")
tmp <- lapply(1:2, function(i){
    expand.grid(Row=1:28, Column=1:28) %>%
        mutate(label=titles[i],
               value = mnist$train$images[is[i],])
})
tmp <- Reduce(rbind, tmp)
tmp %>% ggplot(aes(Row, Column, fill=value)) +
    geom_raster() +
    scale_y_reverse() +
    scale_fill_gradient(low="white", high="black") +
    facet_grid(.~label) +
    geom_vline(xintercept = 14.5) +
    geom_hline(yintercept = 14.5)
```




```
data("mnist_27")
mnist_27$train %>% ggplot(aes(x_1, x_2, color = y)) + geom_point()
```

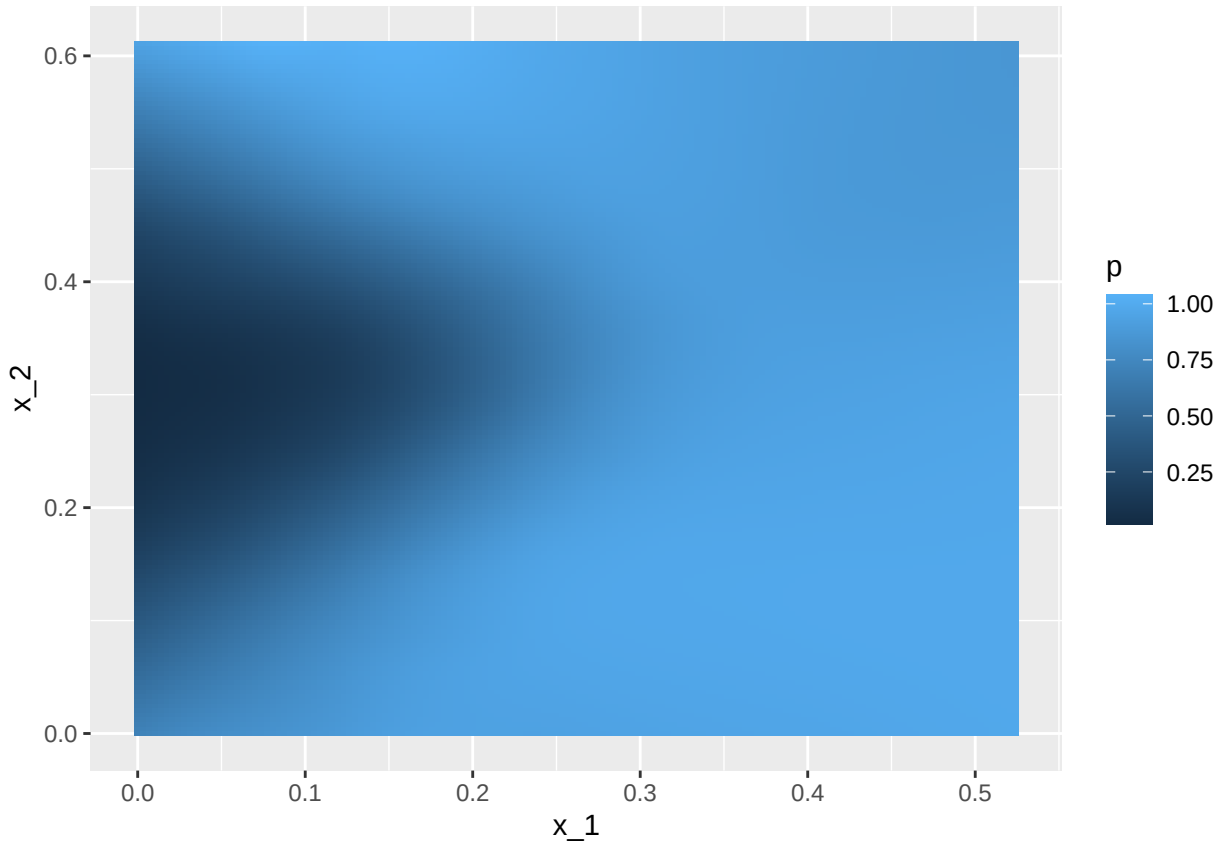

```
is <- mnist_27$index_train[c(which.min(mnist_27$train$x_2), which.max(mnist_27$train$x_2))]
titles <- c("smallest","largest")
tmp <- lapply(1:2, function(i){
    expand.grid(Row=1:28, Column=1:28) %>%
        mutate(label=titles[i],
               value = mnist$train$images[is[i],])
})
tmp <- Reduce(rbind, tmp)
tmp %>% ggplot(aes(Row, Column, fill=value)) +
    geom_raster() +
    scale_y_reverse() +
    scale_fill_gradient(low="white", high="black") +
    facet_grid(.~label) +
    geom_vline(xintercept = 14.5) +
    geom_hline(yintercept = 14.5)
```



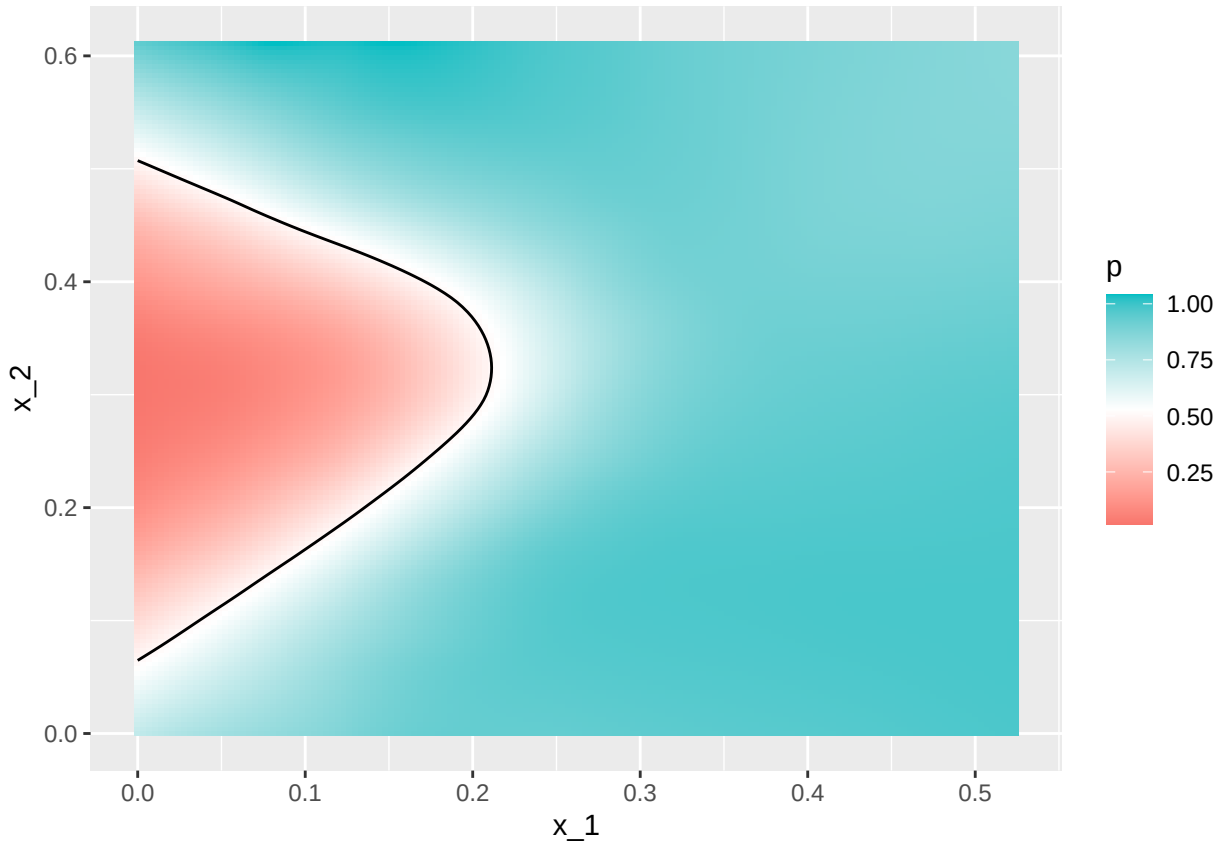

```
fit_glm <- glm(y ~ x_1 + x_2, data=mnist_27$train, family = "binomial")
p_hat_glm <- predict(fit_glm, mnist_27$test)
y_hat_glm <- factor(ifelse(p_hat_glm > 0.5, 7, 2))
confusionMatrix(data = y_hat_glm, reference = mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
##     0.76
```

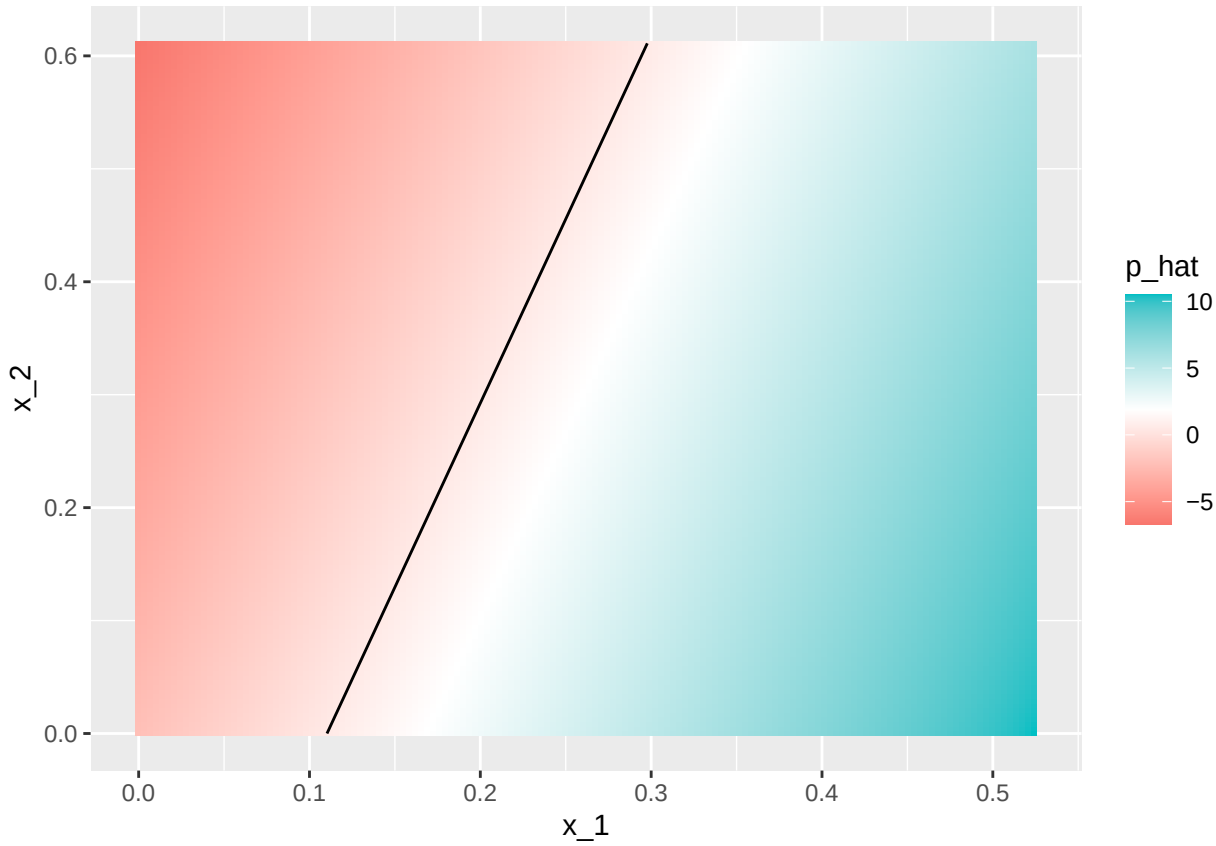
```
mnist_27$true_p %>% ggplot(aes(x_1, x_2, fill=p)) +
    geom_raster()
```



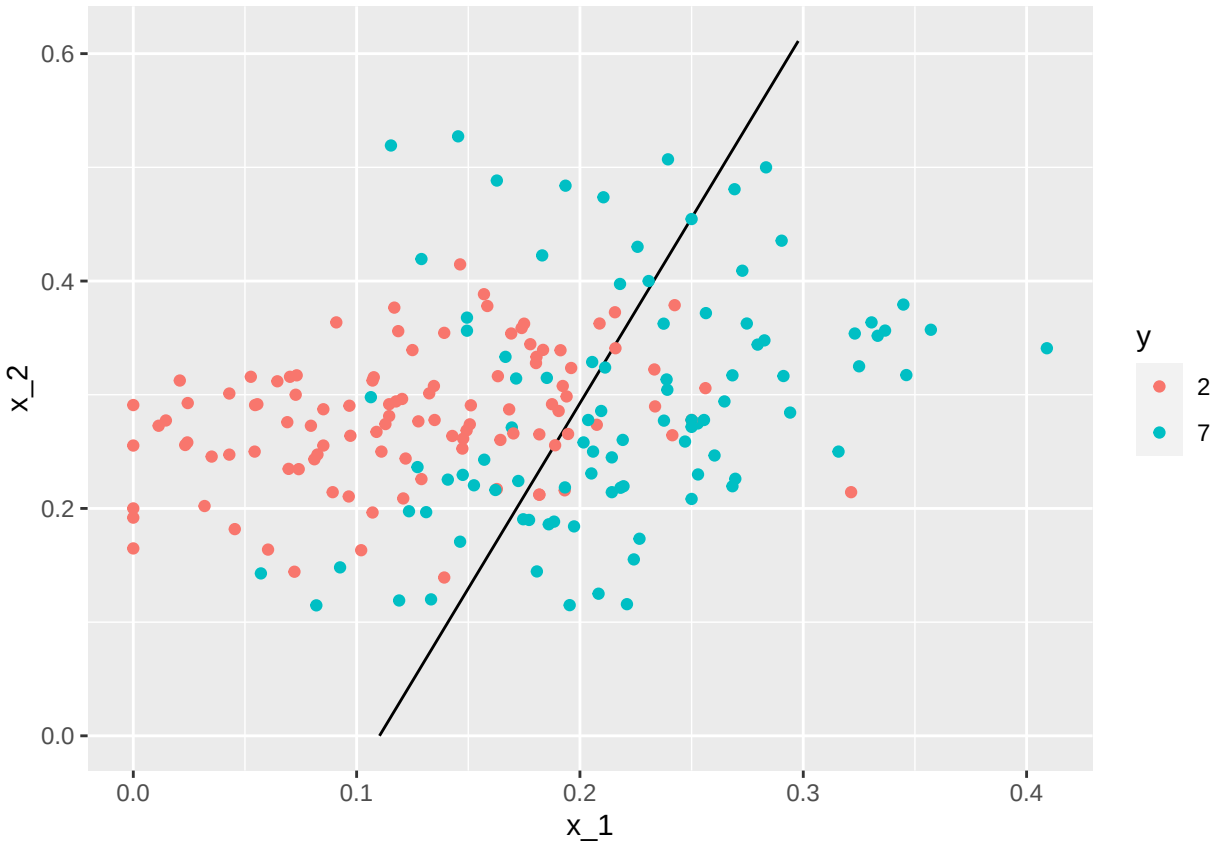

```
mnist_27$true_p %>% ggplot(aes(x_1, x_2, z=p, fill=p)) +
    geom_raster() +
    scale_fill_gradientn(colors=c("#F8766D","white","#00BFC4")) +
    stat_contour(breaks=c(0.5), color="black")
```




```
p_hat <- predict(fit_glm, newdata = mnist_27$true_p)
mnist_27$true_p %>%
    mutate(p_hat = p_hat) %>%
    ggplot(aes(x_1, x_2,  z=p_hat, fill=p_hat)) +
    geom_raster() +
    scale_fill_gradientn(colors=c("#F8766D","white","#00BFC4")) +
    stat_contour(breaks=c(0.5),color="black")
```




```
p_hat <- predict(fit_glm, newdata = mnist_27$true_p)
mnist_27$true_p %>%
    mutate(p_hat = p_hat) %>%
    ggplot() +
    stat_contour(aes(x_1, x_2, z=p_hat), breaks=c(0.5), color="black") +
    geom_point(mapping = aes(x_1, x_2, color=y), data = mnist_27$test)
```



Comprehension Check - Logistic Regression

1. Define a dataset using the following code:

```
# 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
```

```
make_data <- function(n = 1000, p = 0.5,
                mu_0 = 0, mu_1 = 2,
                sigma_0 = 1,  sigma_1 = 1){

y <- rbinom(n, 1, p)
f_0 <- rnorm(n, mu_0, sigma_0)
f_1 <- rnorm(n, mu_1, sigma_1)
x <- ifelse(y == 1, f_1, f_0)

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

list(train = data.frame(x = x, y = as.factor(y)) %>% slice(-test_index),
    test = data.frame(x = x, y = as.factor(y)) %>% slice(test_index))
}
dat <- make_data()
```

Note that we have defined a variable `x` that is predictive of a binary outcome `y`:

`dat$train %>% ggplot(aes(x, color = y)) + geom_density()`.

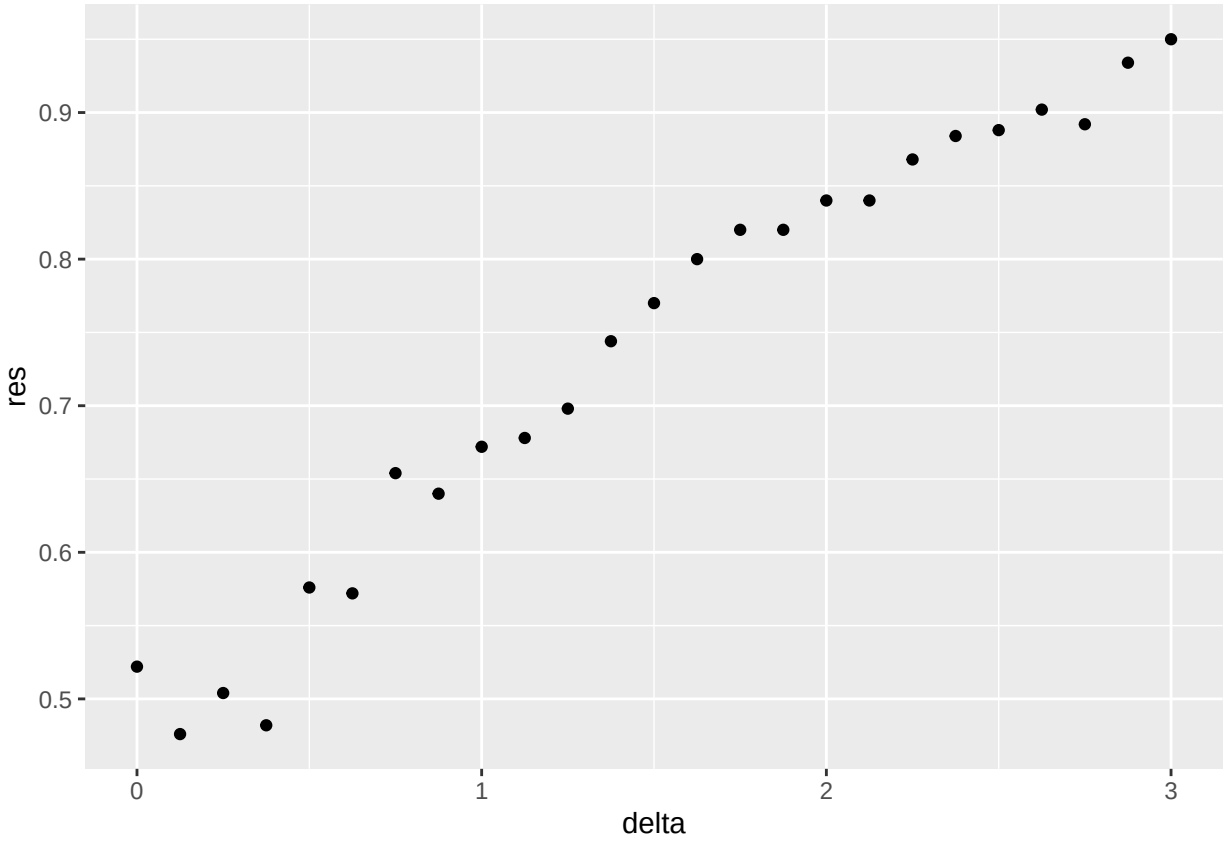
Set the seed to 1, then use the `make_data()` function defined above to generate 25 different datasets with `mu_1 <- seq(0, 3, len=25)`. Perform logistic regression on each of the 25 different datasets (predict 1 if $p > 0.5$) and plot accuracy (`res` in the figures) vs mu_1 (`delta` in the figures).

Which is the correct plot?

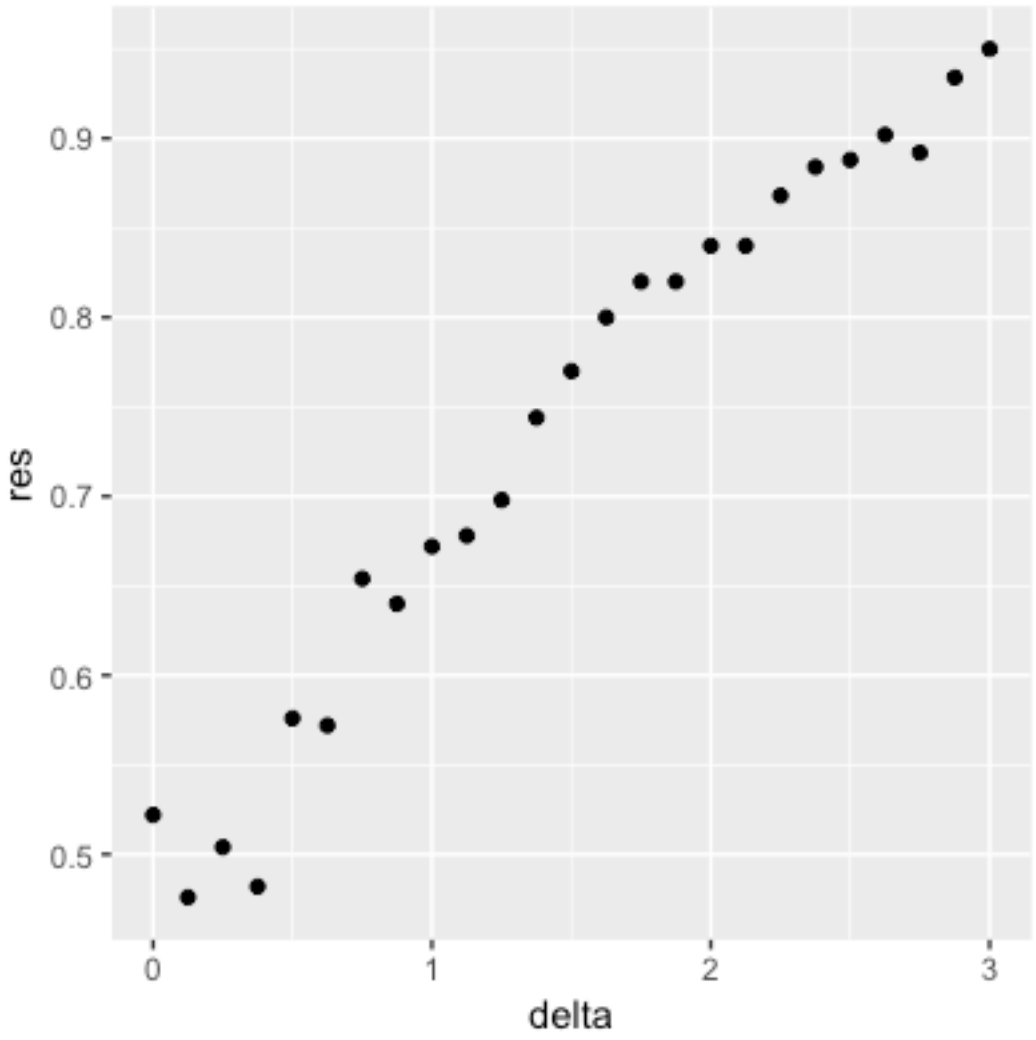
```
set.seed(1) #if you are using R 3.5 or earlier
set.seed(1, sample.kind="Rounding") #if you are using R 3.6 or later
```

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

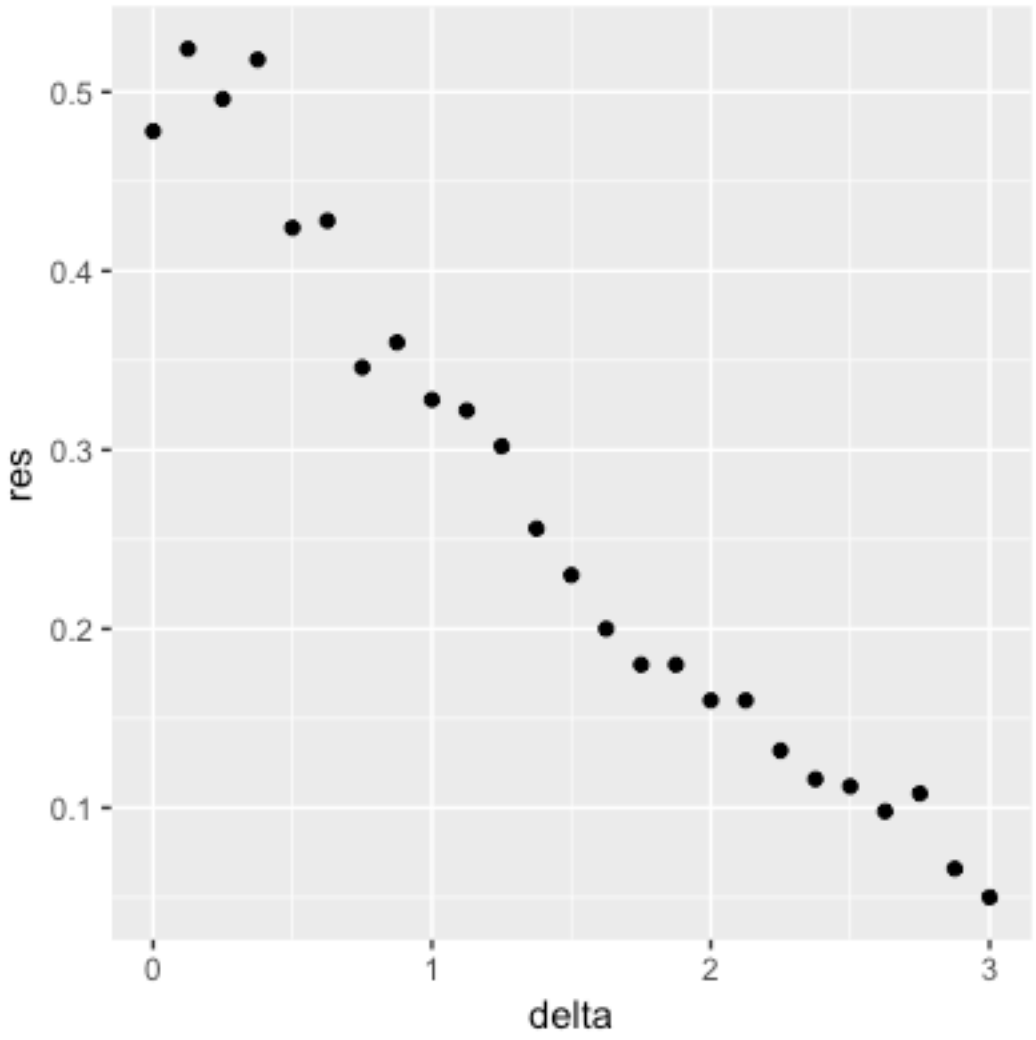
```
delta <- seq(0, 3, len = 25)
res <- sapply(delta, function(d){
    dat <- make_data(mu_1 = d)
    fit_glm <- dat$train %>% glm(y ~ x, family = "binomial", data = .)
    y_hat_glm <- ifelse(predict(fit_glm, dat$test) > 0.5, 1, 0) %>% factor(levels = c(0, 1))
    mean(y_hat_glm == dat$test$y)
})
qplot(delta, res)
```



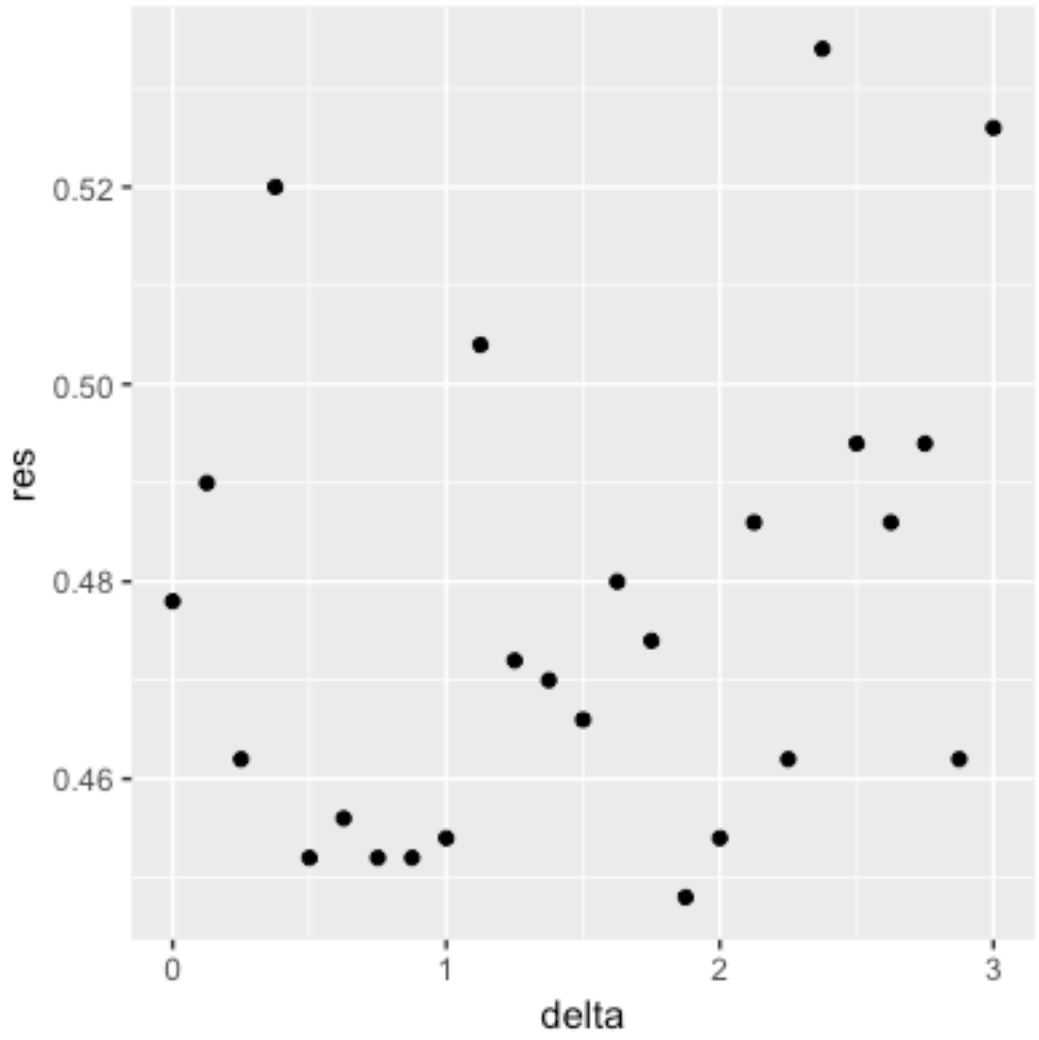
☒ A.



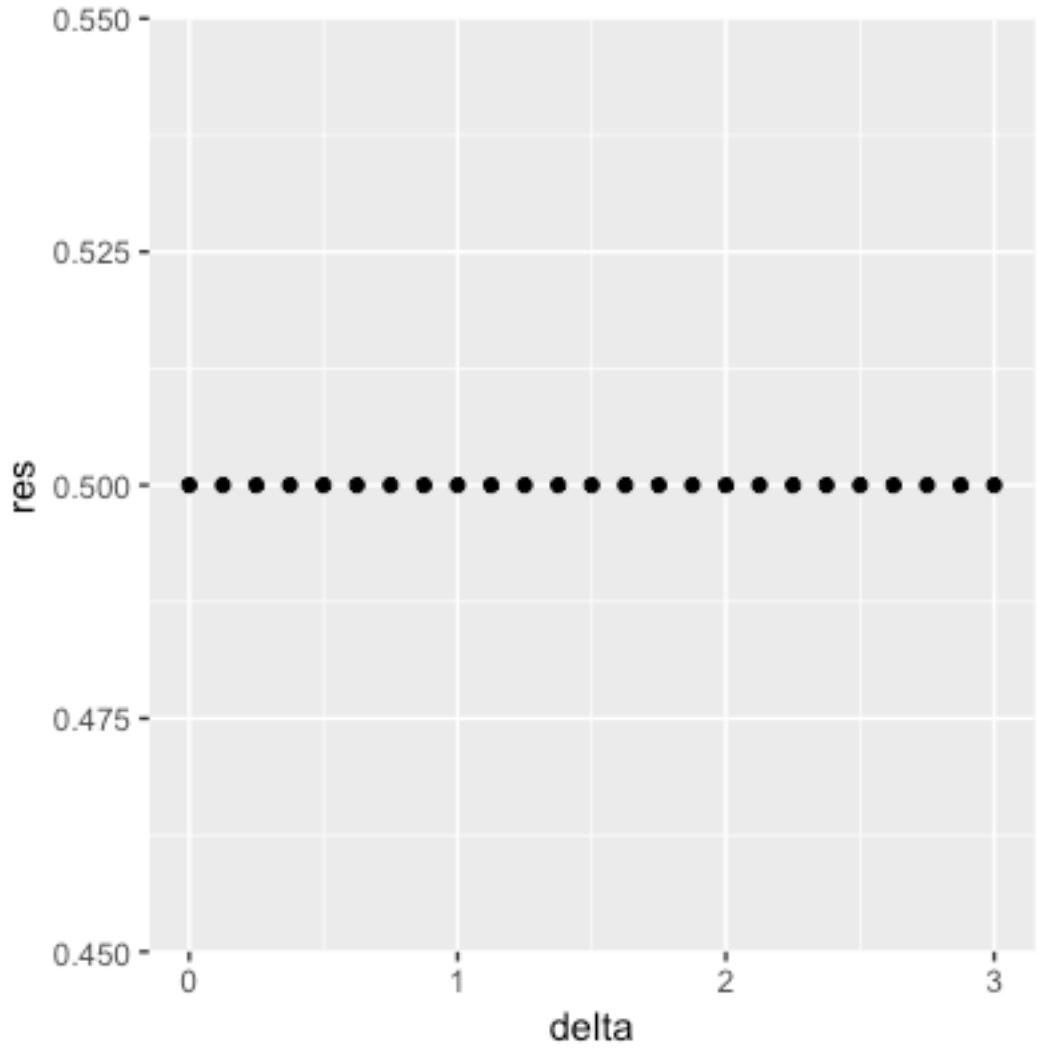
☐ B.



□ C.



□ D.



Introduction to Smoothing

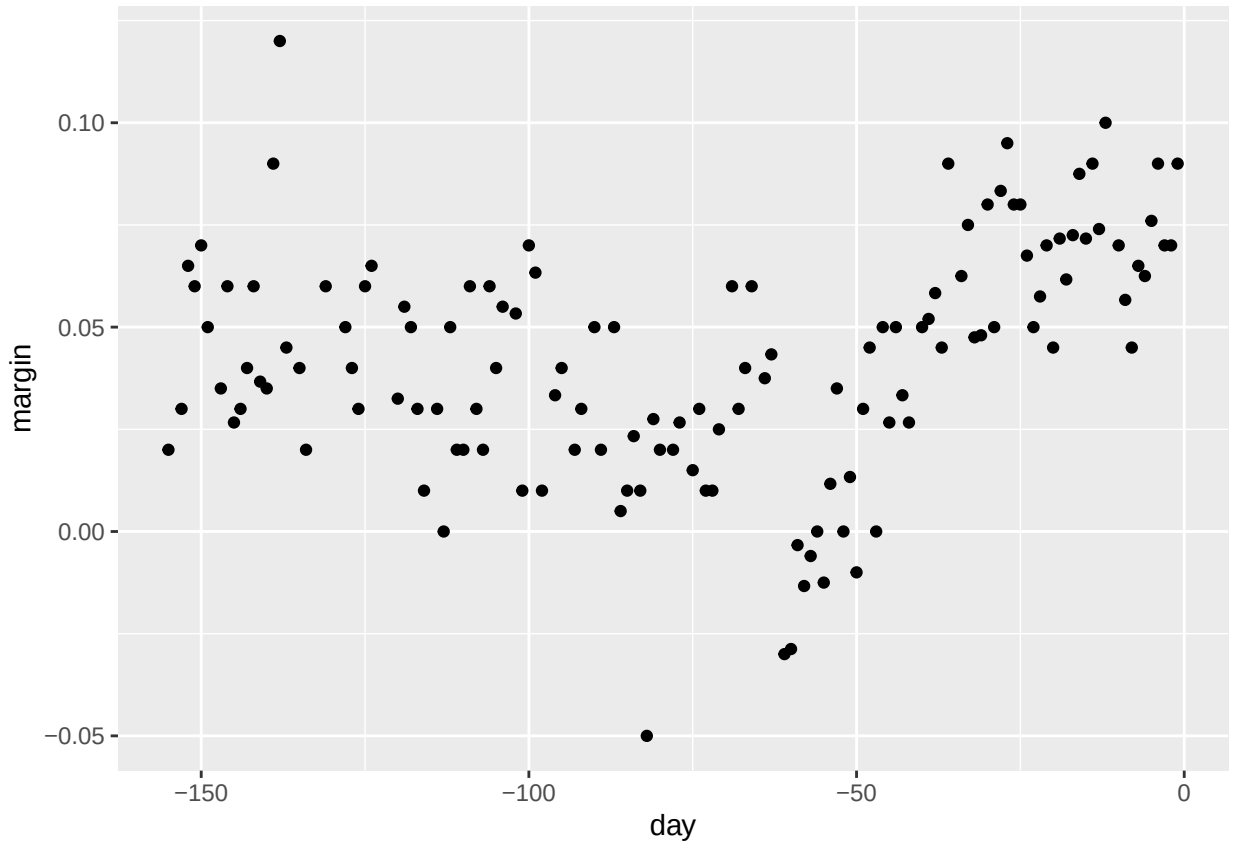
There is a link to the relevant section of the textbook: Smoothing

Key points

- **Smoothing** is a very powerful technique used all across data analysis. It is designed to detect trends in the presence of noisy data in cases in which the shape of the trend is unknown.
- The concepts behind smoothing techniques are extremely useful in machine learning because **conditional expectations/probabilities** can be thought of as **trends** of unknown shapes that we need to estimate in the presence of uncertainty.

Code

```
data("polls_2008")
qplot(day, margin, data = polls_2008)
```



Bin Smoothing and Kernels

There is a link to the relevant sections of the textbook: Bin smoothing and Kernels

Key points

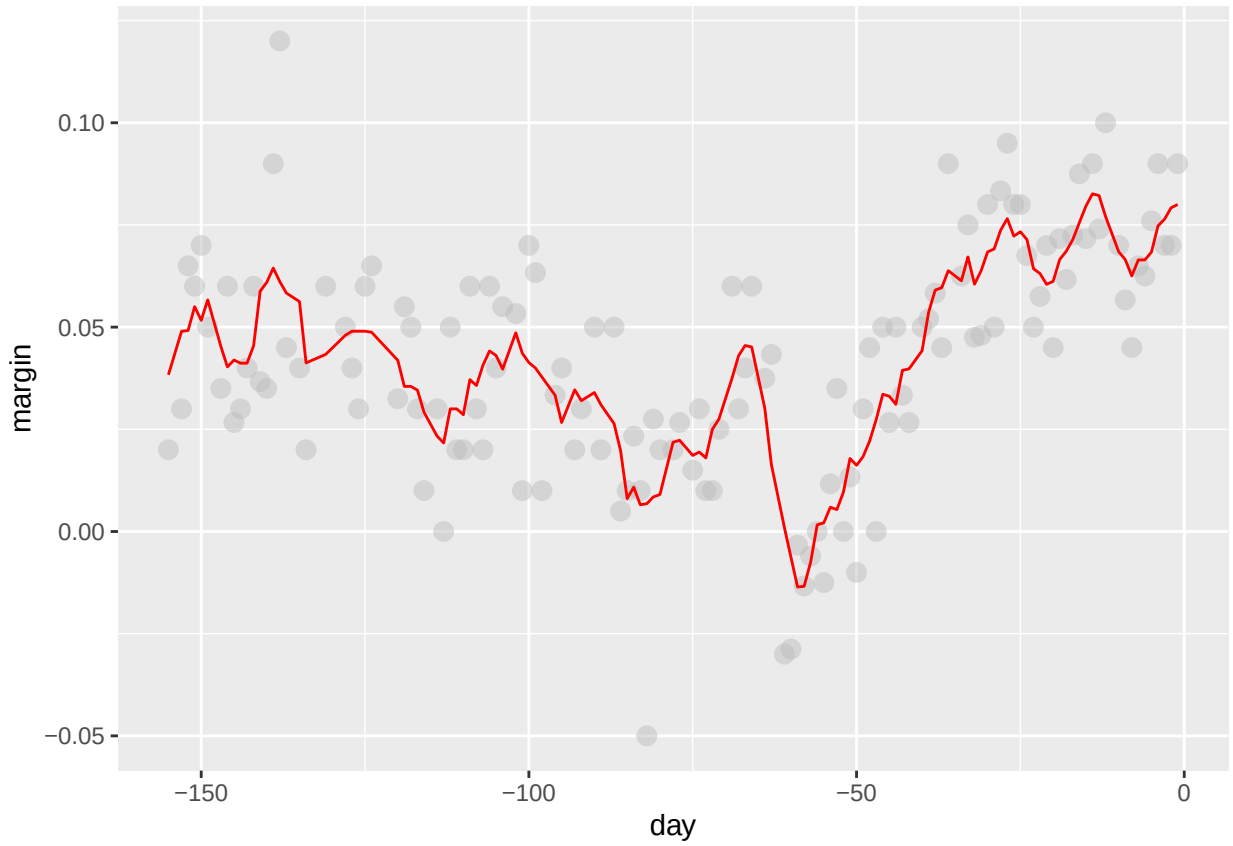
- The general idea of smoothing is to group data points into strata in which the value of $f(x)$ can be assumed to be constant. We can make this assumption because we think $f(x)$ changes slowly and, as a result, $f(x)$ is almost constant in small windows of time.
- This assumption implies that a good estimate for $f(x)$ is the average of the Y_i values in the window. The estimate is:

$\hat{f}(x_0) = \frac{1}{N_0} \sum_{i \in A_0} Y_i$

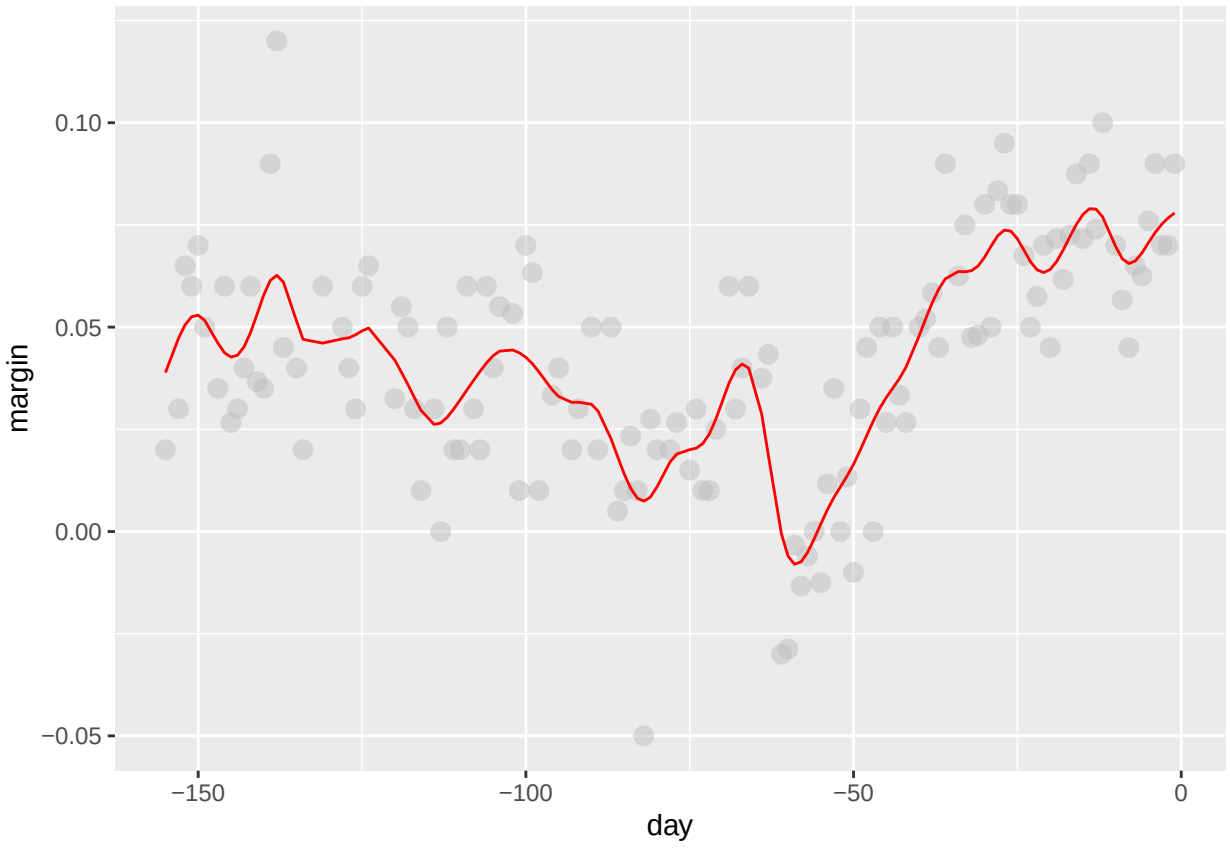
- In smoothing, we call the size of the interval $|x - x_0|$ satisfying the particular condition the window size, bandwidth or span.

Code

```
# bin smoothers
span <- 7
fit <- with(polls_2008,ksmooth(day, margin, x.points = day, kernel="box", bandwidth =span))
polls_2008 %>% mutate(smooth = fit$y) %>%
    ggplot(aes(day, margin)) +
    geom_point(size = 3, alpha = .5, color = "grey") +
    geom_line(aes(day, smooth), color="red")
```




```
# kernel
span <- 7
fit <- with(polls_2008, ksmooth(day, margin,  x.points = day, kernel="normal", bandwidth = span))
polls_2008 %>% mutate(smooth = fit$y) %>%
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = .5, color = "grey") +
  geom_line(aes(day, smooth), color="red")
```



Local Weighted Regression (loess)

There is a link to the relevant section of the textbook: Local weighted regression

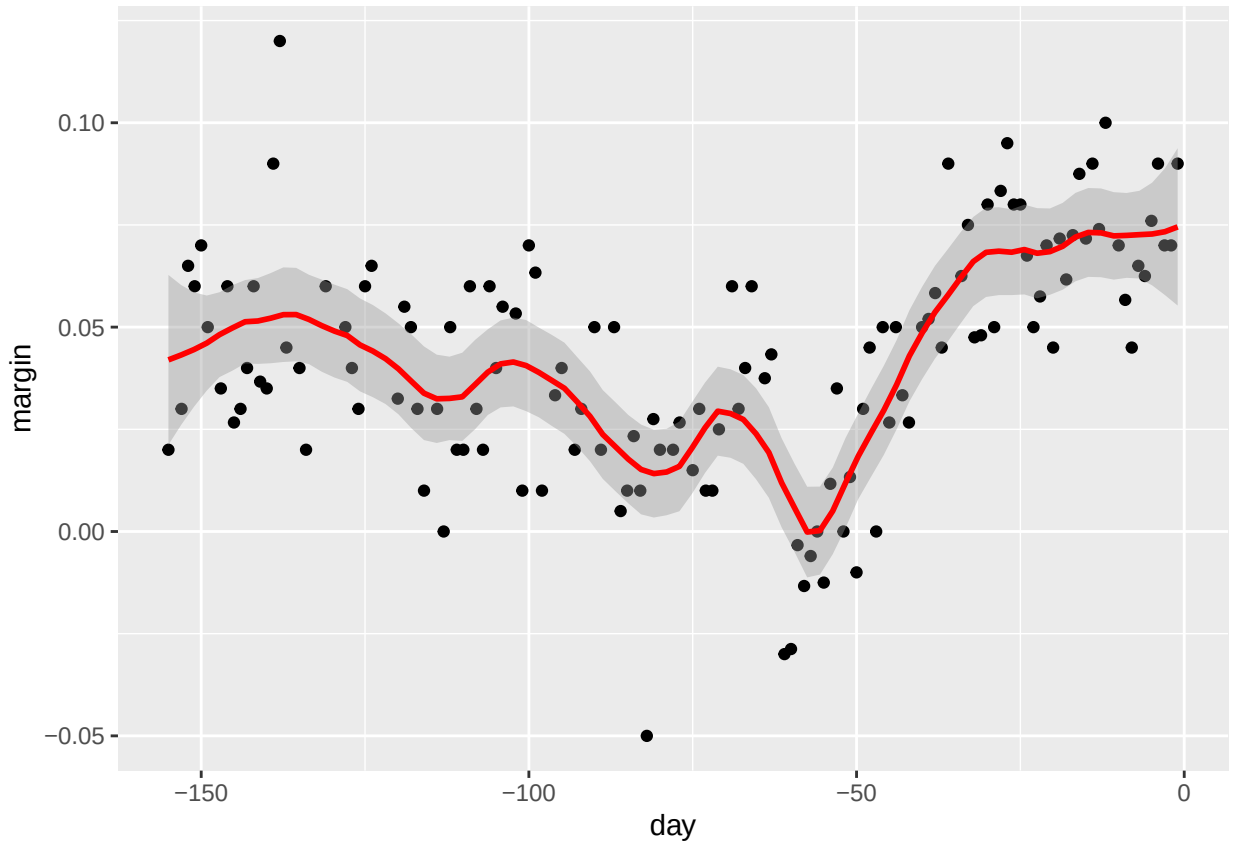
Key points

- A limitation of the bin smoothing approach is that we need small windows for the approximately constant assumptions to hold which may lead to imprecise estimates of $f(x)$. **Local weighted regression (loess)** permits us to consider larger window sizes.
- One important difference between loess and bin smoother is that we assume the smooth function is locally **linear** in a window instead of constant.
- The result of loess is a smoother fit than bin smoothing because we use larger sample sizes to estimate our local parameters.

Code

```
polls_2008 %>% ggplot(aes(day, margin)) +
  geom_point() +
  geom_smooth(color="red", span = 0.15, method = "loess", method.args = list(degree=1))
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Comprehension Check - Smoothing

1. In the Wrangling course of this series, PH125.6x, we used the following code to obtain mortality counts for Puerto Rico for 2015-2018:

```
if(!require(purrr)) install.packages("purrr")
if(!require(pdftools)) install.packages("pdftools")
```

```
## Loading required package: pdftools
```

```
## Using poppler version 0.73.0
```

```
library(tidyverse)
library(lubridate)
library(purrr)
library(pdftools)

fn <- system.file("extdata", "RD-Mortality-Report_2015-18-180531.pdf", package="dslabs")
dat <- map_df(str_split(pdf_text(fn), "\n"), function(s){
    s <- str_trim(s)
    header_index <- str_which(s, "2015")[1]
    tmp <- str_split(s[header_index], "\\s+", simplify = TRUE)
    month <- tmp[1]
    header <- tmp[-1]
```

```
    tail_index  <- str_which(s, "Total")
    n <- str_count(s, "\\d+")
    out <- c(1:header_index, which(n==1), which(n>=28), tail_index:length(s))
    s[-out] %>%
        str_remove_all("[^\\d\\s]") %>%
        str_trim() %>%
        str_split_fixed("\\s+", n = 6) %>%
        .[,1:5] %>%
        as_data_frame() %>%
        setNames(c("day", header)) %>%
        mutate(month = month,
            day = as.numeric(day)) %>%
        gather(year, deaths, -c(day, month)) %>%
        mutate(deaths = as.numeric(deaths))
}) %>%
    mutate(month = recode(month, "JAN" = 1, "FEB" = 2, "MAR" = 3, "APR" = 4, "MAY" = 5, "JUN" = 6,
                          "JUL" = 7, "AGO" = 8, "SEP" = 9, "OCT" = 10, "NOV" = 11, "DEC" = 12)) %>%
    mutate(date = make_date(year, month, day)) %>%
        dplyr::filter(date <= "2018-05-01")
```

```
## Warning: `as_data_frame()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

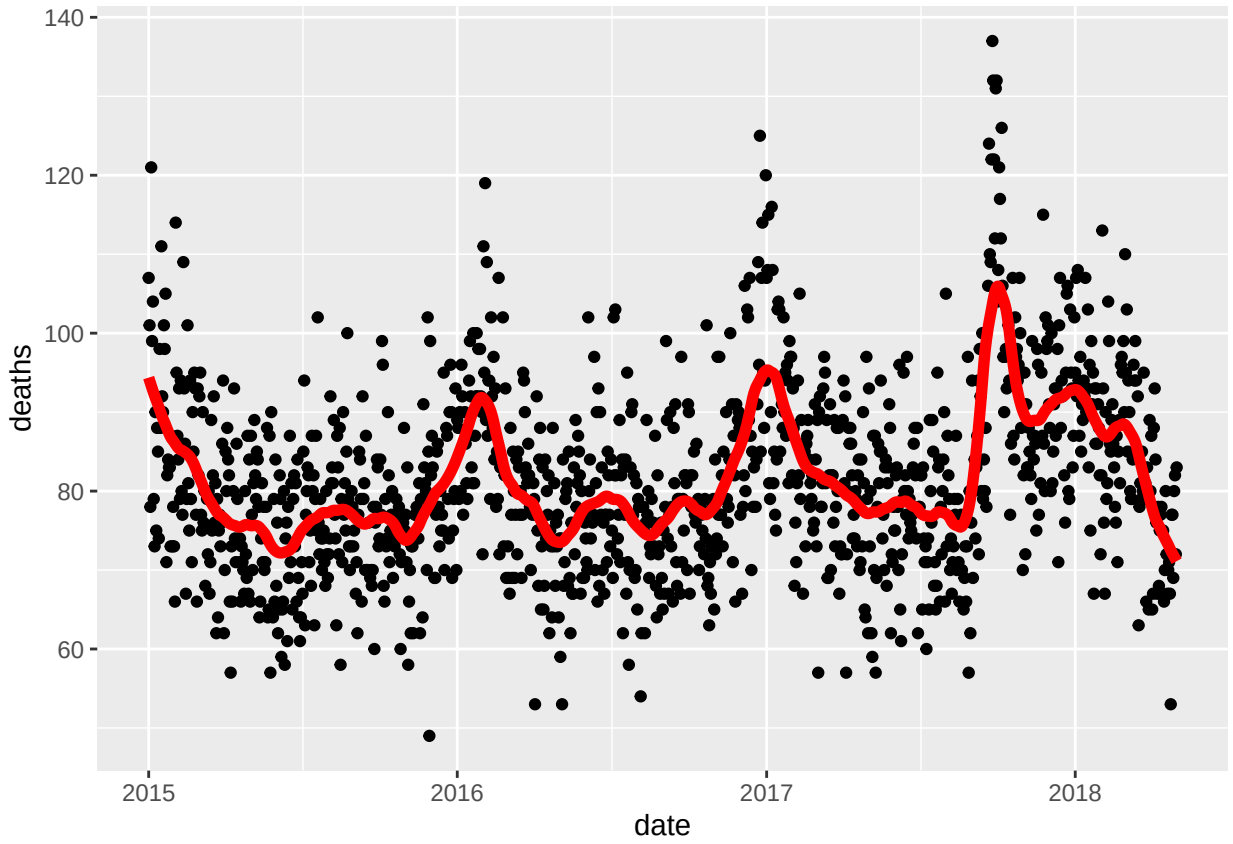
```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Use the `loess()` function to obtain a smooth estimate of the expected number of deaths as a function of date. Plot this resulting smooth function. Make the span about two months long.

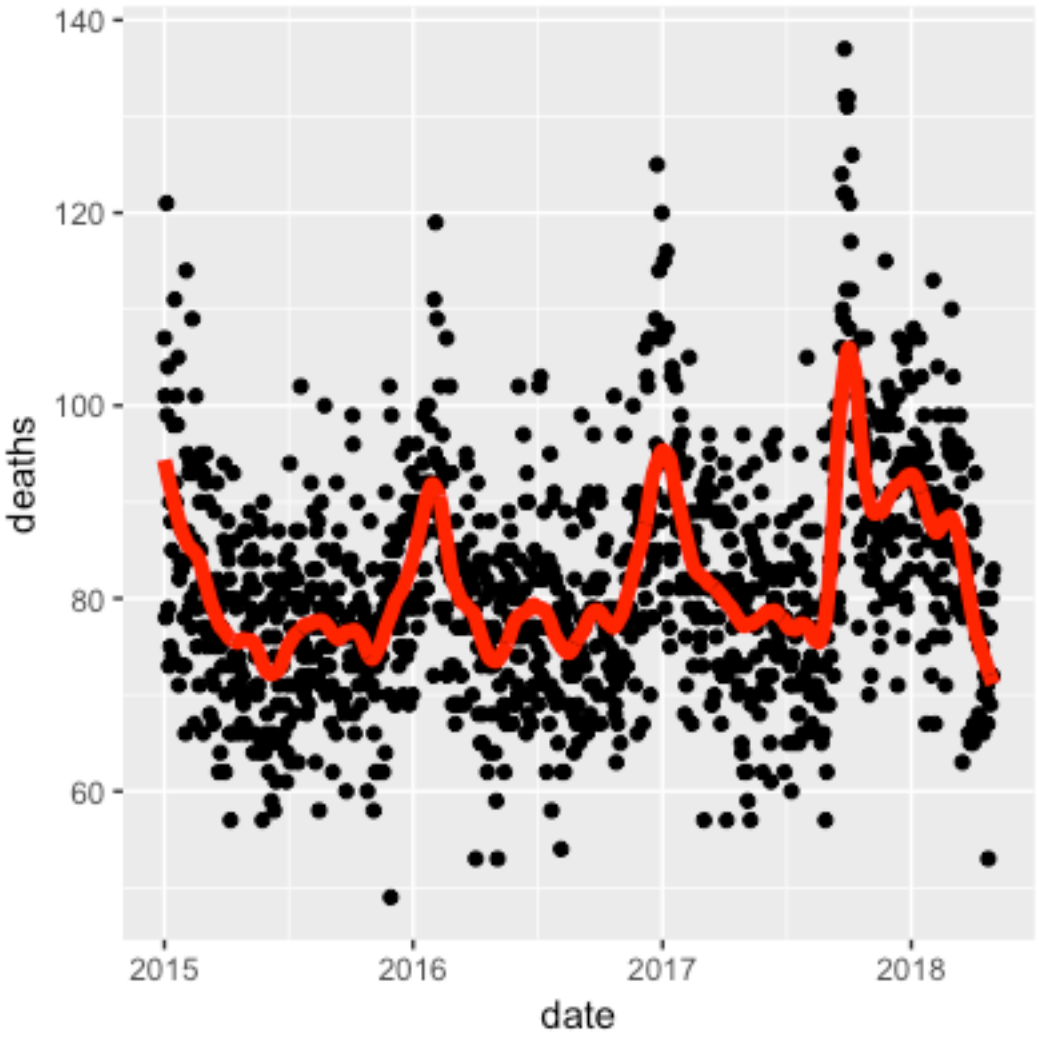
Which of the following plots is correct?

```
span <- 60 / as.numeric(diff(range(dat$date)))
fit <- dat %>% mutate(x = as.numeric(date)) %>% loess(deaths ~ x, data = ., span = span, degree = 1)
dat %>% mutate(smooth = predict(fit, as.numeric(date))) %>%
    ggplot() +
    geom_point(aes(date, deaths)) +
    geom_line(aes(date, smooth), lwd = 2, col = "red")
```

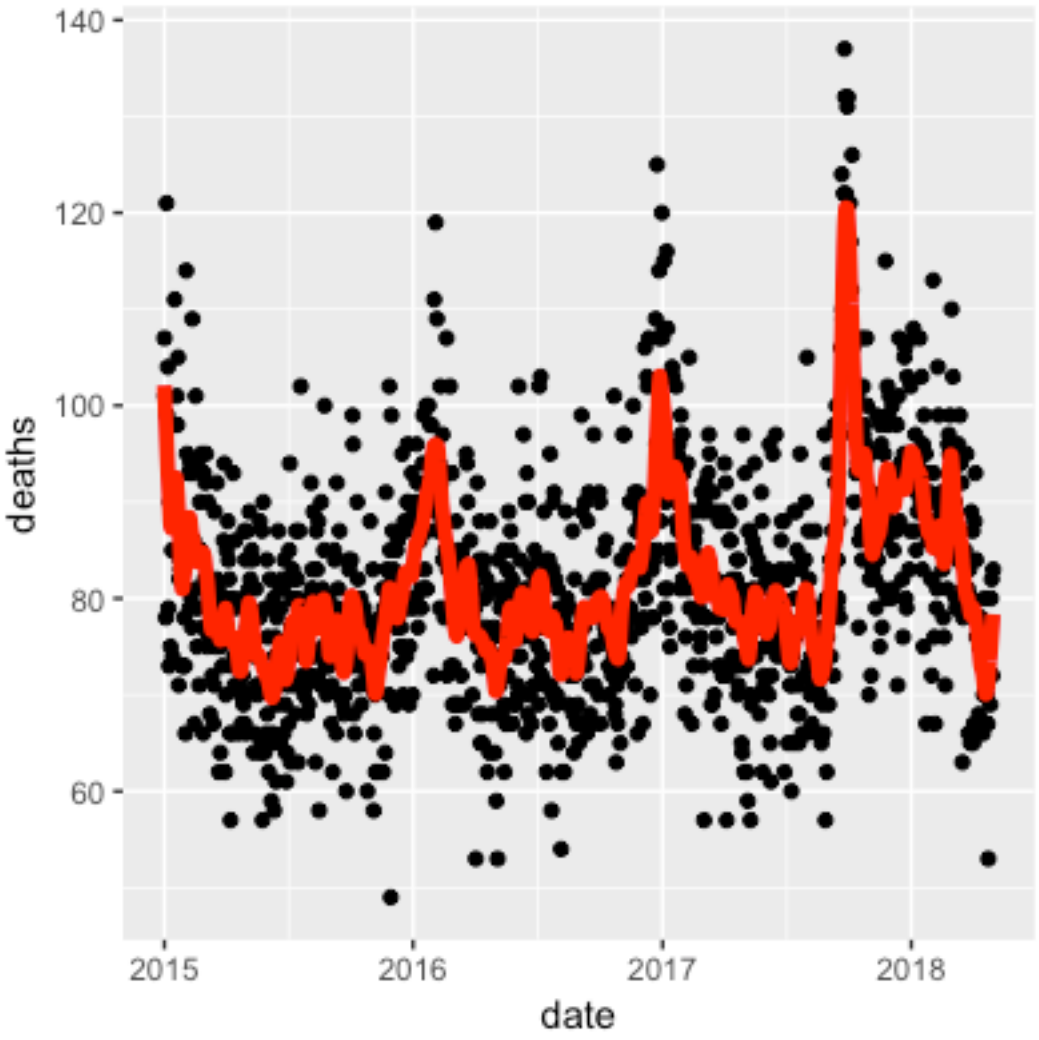
```
## Warning: Removed 1 rows containing missing values (geom_point).
```



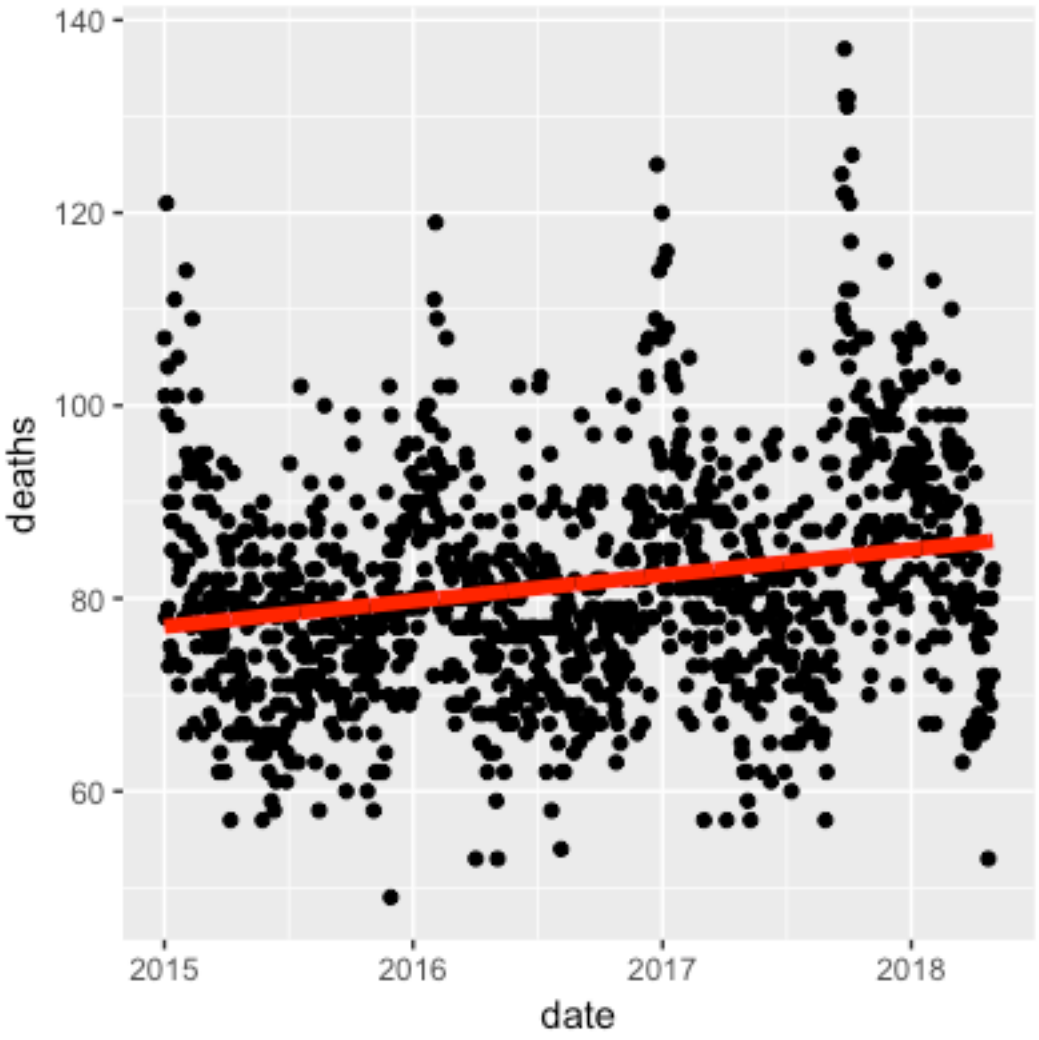
☒ A.



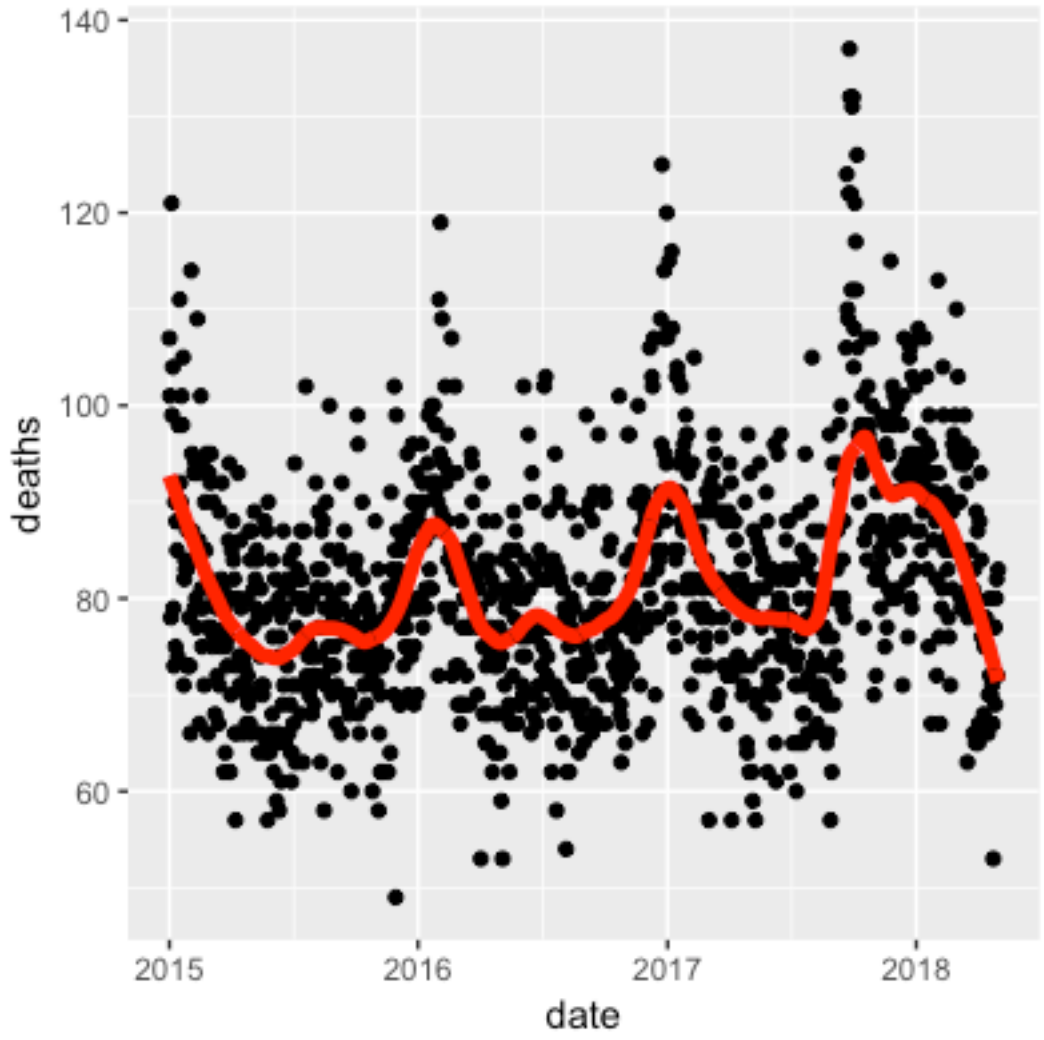
☐ B.



□ C.



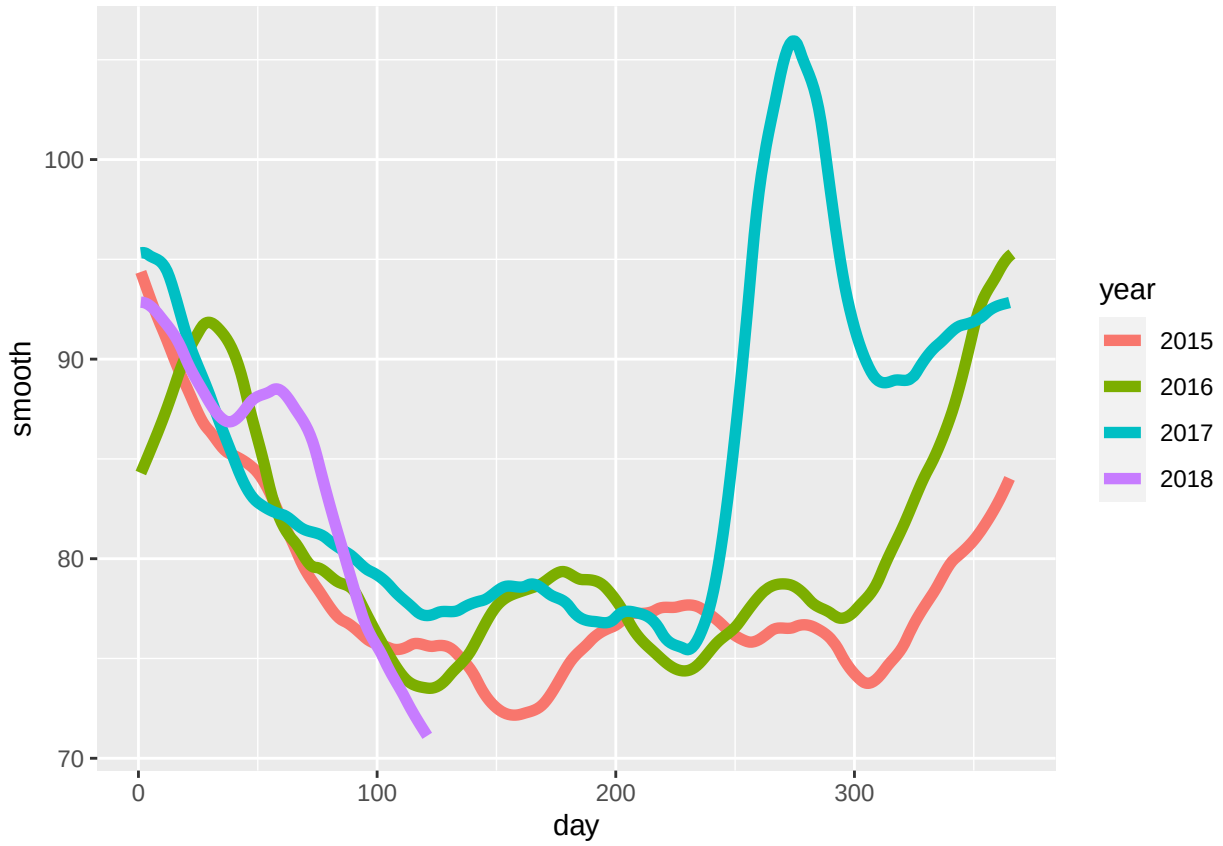
□ D.



2. Work with the same data as in Q1 to plot smooth estimates against day of the year, all on the same plot, but with different colors for each year.

Which code produces the desired plot?

```
dat %>%
    mutate(smooth = predict(fit, as.numeric(date)), day = yday(date), year = as.character(year(date))) %
    ggplot(aes(day, smooth, col = year)) +
    geom_line(lwd = 2)
```



☐ A.

```
dat %>%
    mutate(smooth = predict(fit), day = yday(date), year = as.character(year(date))) %>%
    ggplot(aes(day, smooth, col = year)) +
    geom_line(lwd = 2)
```

☐ B.

```
dat %>%
    mutate(smooth = predict(fit, as.numeric(date)), day = mday(date), year = as.character(year(date))) %
    ggplot(aes(day, smooth, col = year)) +
    geom_line(lwd = 2)
```

☐ C.

```
 dat %>%
    mutate(smooth = predict(fit, as.numeric(date)), day = yday(date), year = as.character(year(date))) %
    ggplot(aes(day, smooth)) +
    geom_line(lwd = 2)
```

☒ D.

```
dat %>%
    mutate(smooth = predict(fit, as.numeric(date)), day = yday(date), year = as.character(year(date))) %
    ggplot(aes(day, smooth, col = year)) +
    geom_line(lwd = 2)
```

3. Suppose we want to predict 2s and 7s in the `mnist_27` dataset with just the second covariate. Can we do this? On first inspection it appears the data does not have much predictive power.

In fact, if we fit a regular logistic regression the coefficient for `x_2` is not significant!

This can be seen using this code:

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

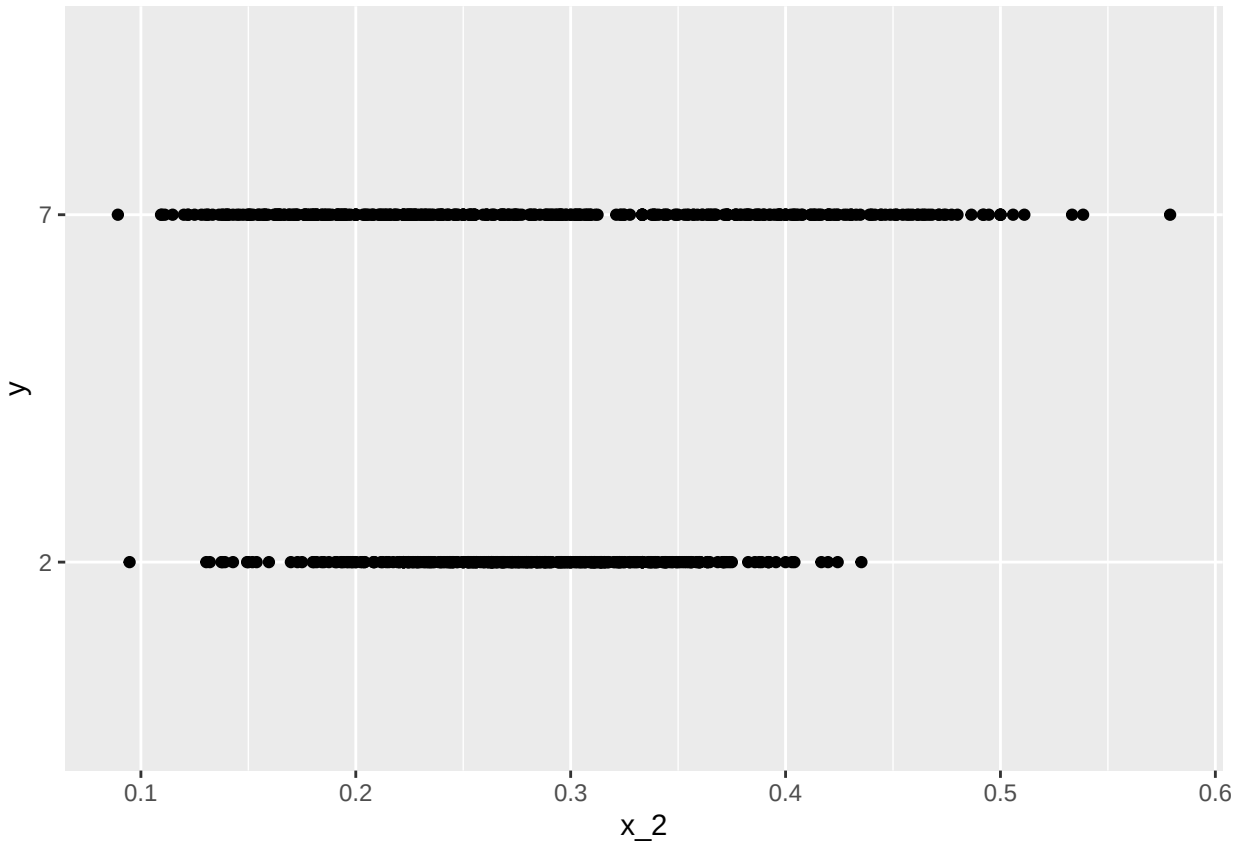
```
## Loading required package: broom
```

```
library(broom)
mnist_27$train %>% glm(y ~ x_2, family = "binomial", data = .) %>% tidy()
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)  -0.0907     0.247    -0.368   0.713
## 2 x_2           0.685      0.827     0.829   0.407
```

Plotting a scatterplot here is not useful since `y` is binary:

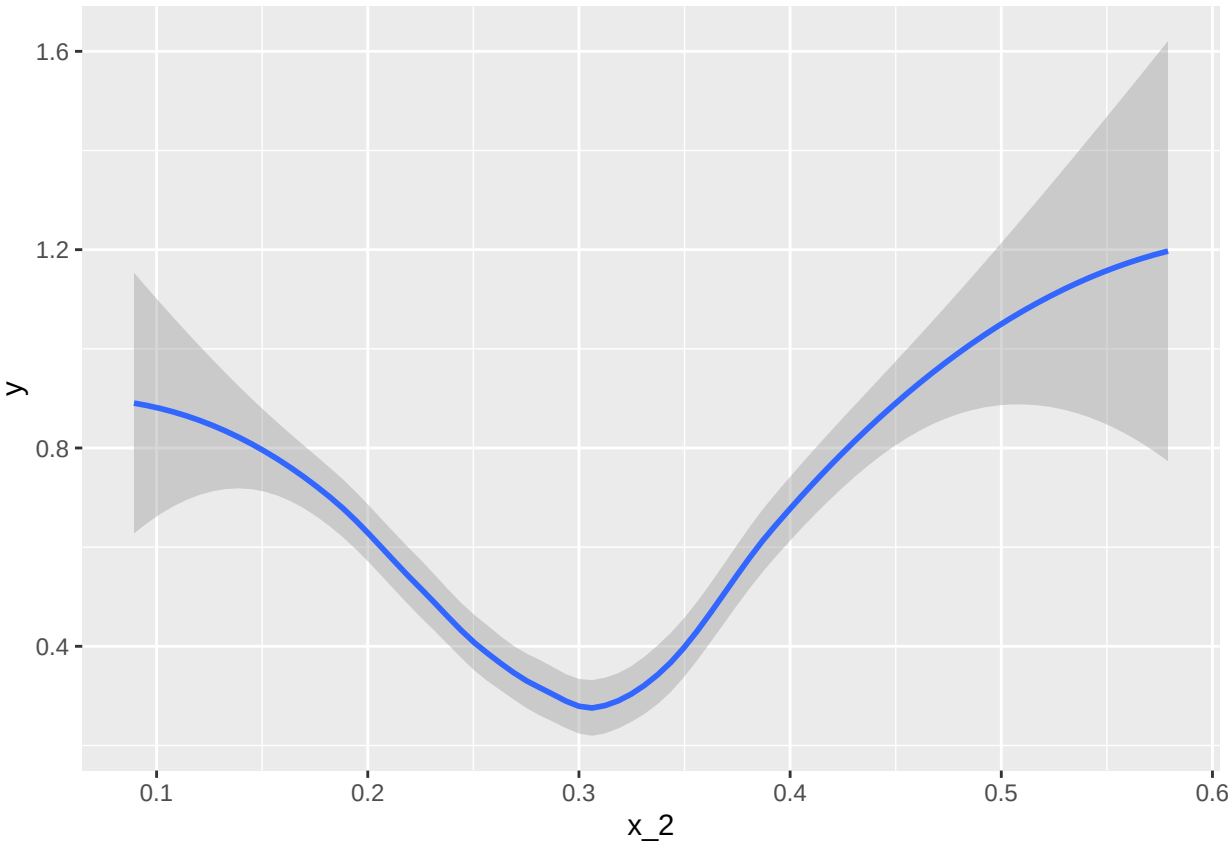
```
qplot(x_2, y, data = mnist_27$train)
```



Fit a loess line to the data above and plot the results. What do you observe?

```
mnist_27$train %>%
    mutate(y = ifelse(y=="7", 1, 0)) %>%
    ggplot(aes(x_2, y)) +
    geom_smooth(method = "loess")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



☐ A. There is no predictive power and the conditional probability is linear.
☐ B. There is no predictive power and the conditional probability is non-linear.
☐ C. There is predictive power and the conditional probability is linear.
☒ D. There is predictive power and the conditional probability is non-linear.

Matrices

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

Key points

- The main reason for using matrices is that certain mathematical operations needed to develop efficient code can be performed using techniques from a branch of mathematics called **linear algebra**.
- **Linear algebra** and **matrix notation** are key elements of the language used in academic papers describing machine learning techniques.

Code

```
if(!exists("mnist")) mnist <- read_mnist()

class(mnist$train$images)
```

```
## [1] "matrix" "array"
```

```
x <- mnist$train$images[1:1000,]
y <- mnist$train$labels[1:1000]
```

Matrix Notation

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

Key points

- In matrix algebra, we have three main types of objects: **scalars**, **vectors**, and **matrices**.
  - **Scalar:** $\alpha = 1$
  - **Vector:** $X_1 = \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{N,1} \end{pmatrix}$
  - **Matrix:** $X = [X_1 X_2] = \begin{pmatrix} x_{1,1} & x_{1,2} \\ \vdots & \vdots \\ x_{N,1} & x_{N,2} \end{pmatrix}$
- In R, we can extract the dimension of a matrix with the function `dim()`. We can convert a vector into a matrix using the function `as.matrix()`.

Code

```
length(x[,1])
```

```
## [1] 1000
```

```
x_1 <- 1:5
x_2 <- 6:10
cbind(x_1, x_2)
```

```
##      x_1 x_2
## [1,]   1   6
## [2,]   2   7
## [3,]   3   8
## [4,]   4   9
## [5,]   5  10
```

```
dim(x)
```

```
## [1] 1000  784
```

```
dim(x_1)
```

```
## NULL
```

```
dim(as.matrix(x_1))
```

```
## [1] 5 1
```

```
dim(x)
```

```
## [1] 1000  784
```

Converting a Vector to a Matrix

There is a link to the relevant section of the textbook: Converting a vector to a matrix

Key points

- In R, we can **convert a vector into a matrix** with the `matrix()` function. The matrix is filled in by column, but we can fill by row by using the `byrow` argument. The function `t()` can be used to directly transpose a matrix.
- Note that the matrix function **recycles values in the vector** without warning if the product of columns and rows does not match the length of the vector.

Code

```
my_vector <- 1:15

# fill the matrix by column
mat <- matrix(my_vector, 5, 3)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
# fill by row
mat_t <- matrix(my_vector, 3, 5, byrow = TRUE)
mat_t
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
```

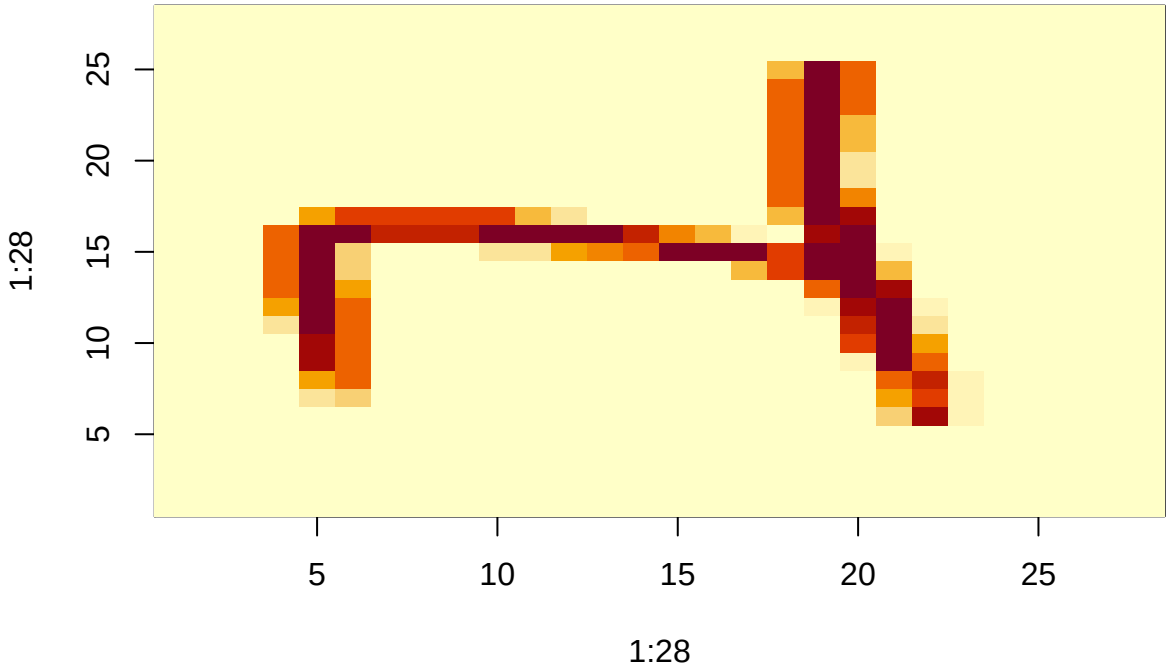
```
identical(t(mat), mat_t)
```

```
## [1] TRUE
```

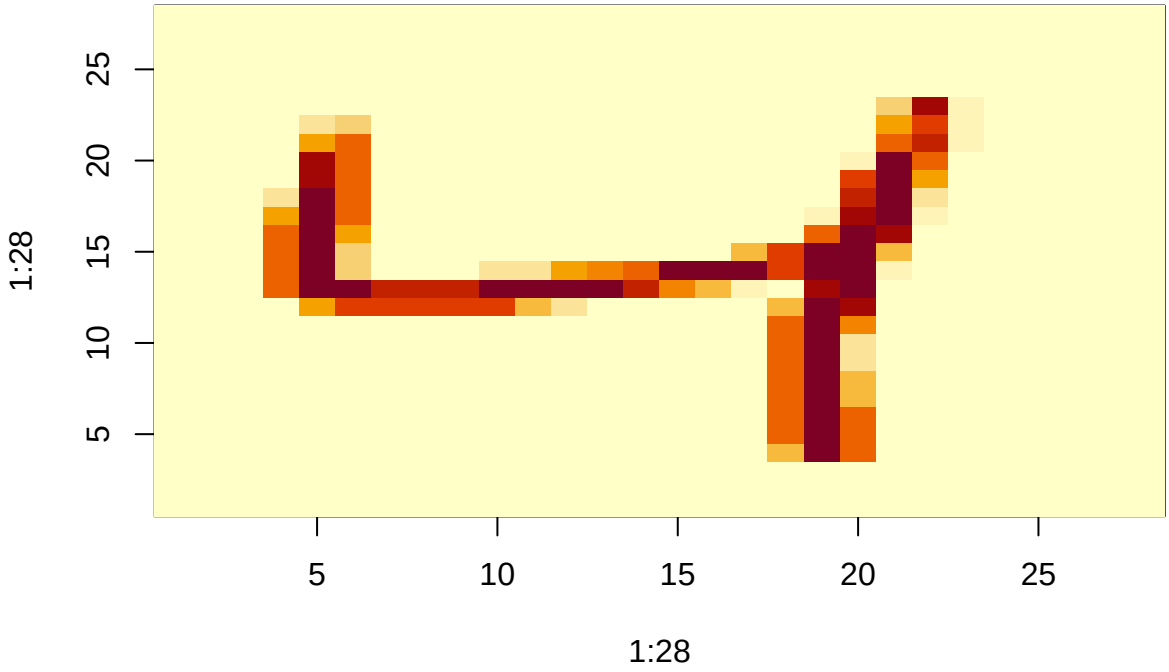
```
matrix(my_vector, 5, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11    1    6
## [2,]    2    7   12    2    7
## [3,]    3    8   13    3    8
## [4,]    4    9   14    4    9
## [5,]    5   10   15    5   10
```

```
grid <- matrix(x[3,], 28, 28)
image(1:28, 1:28, grid)
```




```
# flip the image back
image(1:28, 1:28, grid[, 28:1])
```



Row and Column Summaries and Apply

There is a link to the relevant section of the textbook: Row and column summaries

Key points

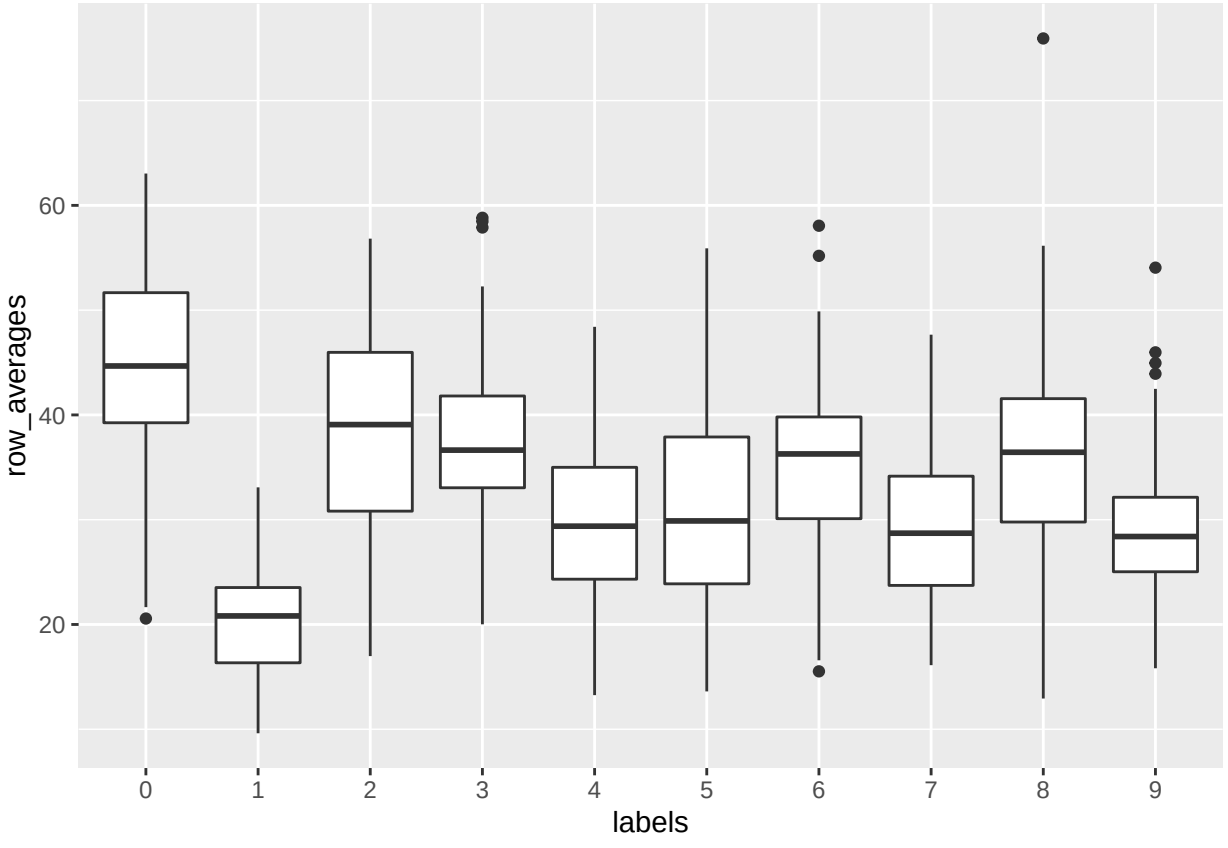
- The function `rowSums()` computes the sum of each row.
- The function `rowMeans()` computes the average of each row.
- We can compute the column sums and averages using the functions `colSums()` and `colMeans()`.
- The **matrixStats** package adds functions that performs operations on each row or column very efficiently, including the functions `rowSds()` and `colSds()`.
- The `apply()` function lets you apply any function to a matrix. The first argument is the **matrix**, the second is the **dimension** (1 for rows, 2 for columns), and the third is the **function**.

Code

```
sums <- rowSums(x)
avg <- rowMeans(x)

data_frame(labels = as.factor(y), row_averages = avg) %>%
    qplot(labels, row_averages, data = ., geom = "boxplot")
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```




```
avgs <- apply(x, 1, mean)
sds <- apply(x, 2, sd)
```

Filtering Columns Based on Summaries

There is a link to the relevant section of the textbook: Filtering columns based on summaries

Key points

- The operations used to extract columns: `x[,c(351,352)]`.
- The operations used to extract rows: `x[c(2,3),]`.
- We can also use logical indexes to determine which columns or rows to keep: `new_x <- x[ ,colSds(x) > 60]`.
- **Important note:** if you select only one column or only one row, the result is no longer a matrix but a **vector**. We can **preserve the matrix class** by using the argument `drop=FALSE`.

Code

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

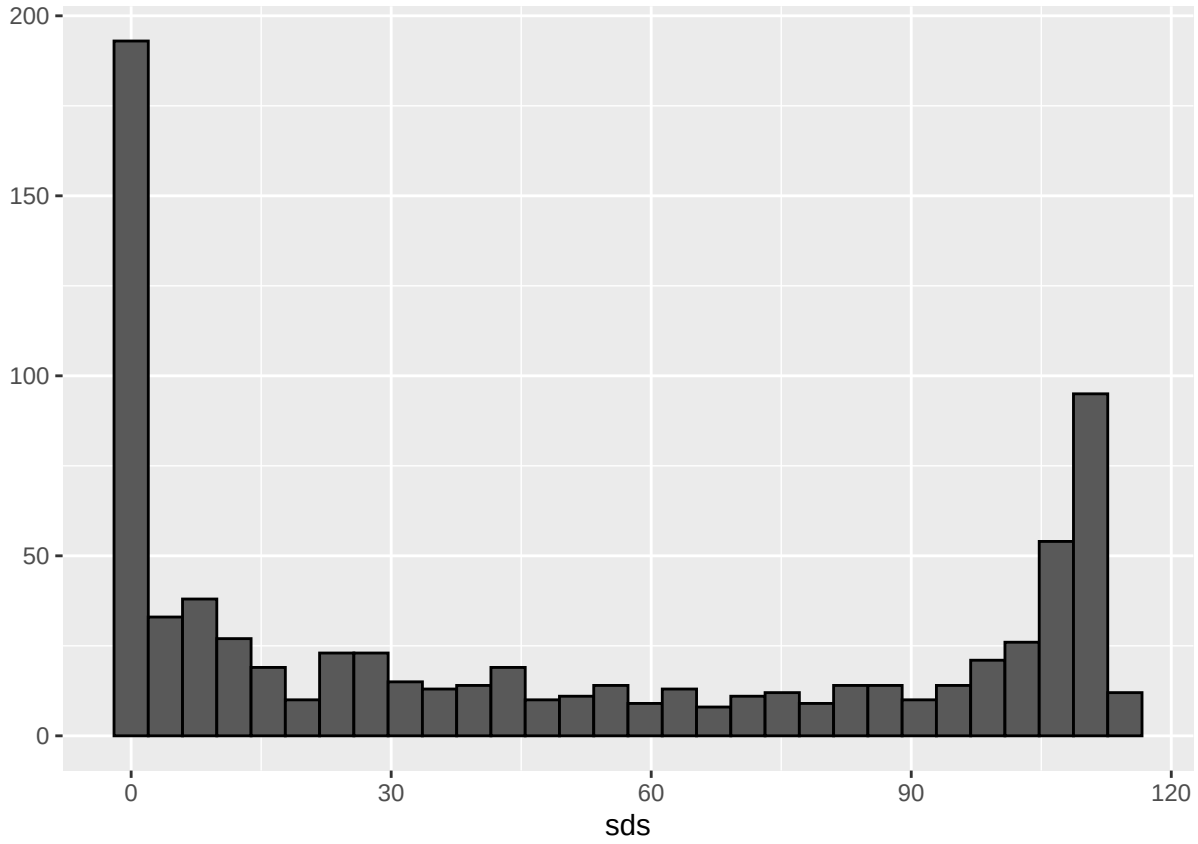
```
## Loading required package: matrixStats
```

```
##
## Attaching package: 'matrixStats'
```

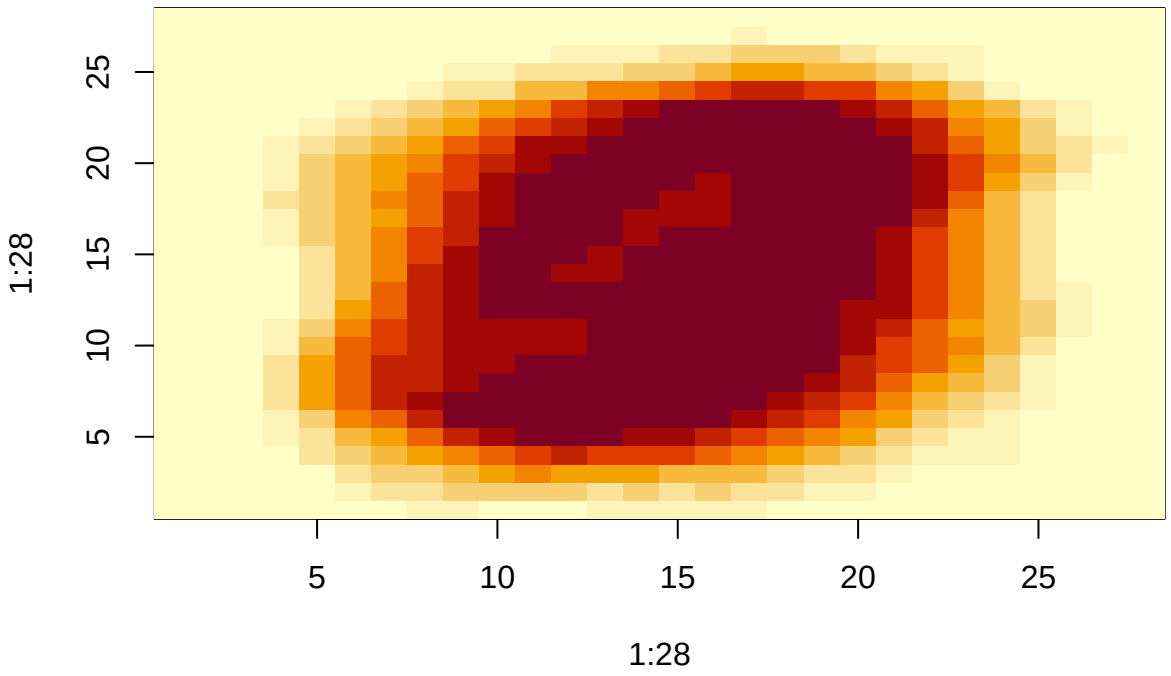
```
## The following object is masked from 'package:dplyr':
##
##     count
```

```
library(matrixStats)

sds <- colSds(x)
qplot(sds, bins = "30", color = I("black"))
```




```
image(1:28, 1:28, matrix(sds, 28, 28)[, 28:1])
```




```
#extract columns and rows
x[ ,c(351,352)]
```

```
##         [,1] [,2]
##    [1,]   70    0
##    [2,]    0    0
##    [3,]    0    0
##    [4,]  205  253
##    [5,]    8   78
##    [6,]    0    0
##    [7,]  253  253
##    [8,]   91  212
##    [9,]  254  143
##   [10,]    0    0
##   [11,]  254  254
##   [12,]   78   79
##   [13,]  254  248
##   [14,]    0  114
##   [15,]  254  109
##   [16,]    0    0
##   [17,]    0    0
##   [18,]   80  223
##   [19,]    0    0
##   [20,]    8   43
##   [21,]  109  109
##   [22,]   96  204
##   [23,]    0    0
##   [24,]  142  255
##   [25,]   32  254
##   [26,]  250  253
##   [27,]    0    0
##   [28,]  253  253
##   [29,]    0    0
##   [30,]    2    0
##   [31,]  253  253
##   [32,]  253  253
##   [33,]    0    0
##   [34,]  228  216
##   [35,]  225    0
##   [36,]  141   86
##   [37,]  107    0
##   [38,]    0    0
##   [39,]    0   15
##   [40,]    0    0
##   [41,]  253  253
##   [42,]  232  233
##   [43,]    0  182
##   [44,]   71  173
##   [45,]  253  203
##   [46,]   44  199
##   [47,]    0  154
##   [48,]    0    0
##   [49,]  169  254
##   [50,]  252  176
##   [51,]  254  254
##   [52,]    0    0
##   [53,]    0    0
```

```
##   [54,]   24  242
##   [55,]   71  122
##   [56,]    0  186
##   [57,]    0    0
##   [58,]    0    0
##   [59,]  111  189
##   [60,]  229  254
##   [61,]    0    0
##   [62,]    0  227
##   [63,]    0    0
##   [64,]  253  251
##   [65,]    0    0
##   [66,]  216  151
##   [67,]  128  128
##   [68,]  254  254
##   [69,]    0    0
##   [70,]   29    0
##   [71,]  253  122
##   [72,]   69    0
##   [73,]  254  204
##   [74,]   17  179
##   [75,]  253  252
##   [76,]  182   15
##   [77,]  254  254
##   [78,]  251  253
##   [79,]  173  253
##   [80,]   10    0
##   [81,]  252  253
##   [82,]    0    0
##   [83,]    0    0
##   [84,]    0  128
##   [85,]    0    0
##   [86,]  253  253
##   [87,]  253  253
##   [88,]   21   52
##   [89,]    0    0
##   [90,]    0    0
##   [91,]    0    0
##   [92,]   53   53
##   [93,]    0    0
##   [94,]   70  236
##   [95,]   38    0
##   [96,]    0    0
##   [97,]    0   26
##   [98,]   38   38
##   [99,]  253  240
##  [100,]   69  253
##  [101,]    0    0
##  [102,]   66    0
##  [103,]  254   95
##  [104,]    0    0
##  [105,]  251    0
##  [106,]  253  253
##  [107,]    0    0
```

```
##  [108,]  191  255
##  [109,]    0    0
##  [110,]  163    8
##  [111,]   78  253
##  [112,]   55  139
##  [113,]  252  253
##  [114,]  252  252
##  [115,]    0    0
##  [116,]    0    0
##  [117,]    0   15
##  [118,]  253  253
##  [119,]    0    0
##  [120,]   14    0
##  [121,]    0    0
##  [122,]    0    0
##  [123,]    0  150
##  [124,]    0    0
##  [125,]  253  233
##  [126,]  254  178
##  [127,]    0    0
##  [128,]   61    1
##  [129,]  253  253
##  [130,]  192  252
##  [131,]  254  247
##  [132,]    0    5
##  [133,]  253  253
##  [134,]  141  240
##  [135,]  253  251
##  [136,]  252  252
##  [137,]  254  179
##  [138,]  255  255
##  [139,]  244  253
##  [140,]    0    0
##  [141,]    0    0
##  [142,]  131   44
##  [143,]    0    0
##  [144,]  162  255
##  [145,]   72  142
##  [146,]    0    0
##  [147,]    0   34
##  [148,]    0    0
##  [149,]    0    0
##  [150,]  252  252
##  [151,]  221  254
##  [152,]    0    0
##  [153,]  232  254
##  [154,]    5   89
##  [155,]  253  213
##  [156,]    0   36
##  [157,]    0    0
##  [158,]  179  242
##  [159,]   50   50
##  [160,]    0   90
##  [161,]  254  254
```

```
##  [162,]  229  254
##  [163,]    0    0
##  [164,]   76  243
##  [165,]    0    0
##  [166,]   63  167
##  [167,]    0    0
##  [168,]    0    0
##  [169,]  253  252
##  [170,]  105    4
##  [171,]   37  168
##  [172,]   69  168
##  [173,]  255  152
##  [174,]  170    0
##  [175,]  252  253
##  [176,]  185    8
##  [177,]  254  253
##  [178,]  251  253
##  [179,]    0    0
##  [180,]   59  106
##  [181,]    0  178
##  [182,]    0    0
##  [183,]  176  253
##  [184,]    0   64
##  [185,]  253  226
##  [186,]    0    0
##  [187,]    0    0
##  [188,]  254  254
##  [189,]    0    0
##  [190,]  252  252
##  [191,]  167  254
##  [192,]    0    0
##  [193,]    0    0
##  [194,]   32   32
##  [195,]    0    0
##  [196,]  148  149
##  [197,]    0    0
##  [198,]  250  225
##  [199,]  104  252
##  [200,]    0   11
##  [201,]  253  169
##  [202,]  157  252
##  [203,]  100  247
##  [204,]  162  216
##  [205,]    0    0
##  [206,]  253  251
##  [207,]    0    0
##  [208,]    0    0
##  [209,]  253  253
##  [210,]    0    0
##  [211,]    0    0
##  [212,]  253  254
##  [213,]  199  253
##  [214,]    0   20
##  [215,]    0    0
```

```
##  [216,]  253  253
##  [217,]    0    0
##  [218,]    0    0
##  [219,]  106  239
##  [220,]  181   84
##  [221,]    0    0
##  [222,]    0   31
##  [223,]  152  244
##  [224,]    0    0
##  [225,]    0   61
##  [226,]  253  227
##  [227,]    0  136
##  [228,]    0    0
##  [229,]    0    0
##  [230,]    0    0
##  [231,]    0    0
##  [232,]  253  251
##  [233,]    0    0
##  [234,]    0    0
##  [235,]    0    2
##  [236,]  253  253
##  [237,]    0    0
##  [238,]    0    0
##  [239,]    0    0
##  [240,]   98   88
##  [241,]  253  252
##  [242,]    0    0
##  [243,]  254  254
##  [244,]    0    0
##  [245,]    0  169
##  [246,]  255  255
##  [247,]    0    0
##  [248,]    0    2
##  [249,]  254  252
##  [250,]    0    0
##  [251,]    0    1
##  [252,]  253  253
##  [253,]  253  252
##  [254,]    0    0
##  [255,]  254  254
##  [256,]  253  253
##  [257,]  253  171
##  [258,]    0    0
##  [259,]    0    0
##  [260,]  254  231
##  [261,]    0    0
##  [262,]    0    0
##  [263,]    0    0
##  [264,]    0    0
##  [265,]    0    0
##  [266,]  236   62
##  [267,]   77    0
##  [268,]    0   90
##  [269,]    0   93
```

```
##  [270,]  253  253
##  [271,]  251   57
##  [272,]    0    0
##  [273,]  125  168
##  [274,]  127  127
##  [275,]  232    8
##  [276,]    0    0
##  [277,]  191  254
##  [278,]    0    0
##  [279,]  245  254
##  [280,]    0  128
##  [281,]    0   51
##  [282,]  253  255
##  [283,]    0    0
##  [284,]    0    0
##  [285,]  253  253
##  [286,]    0    0
##  [287,]  253  253
##  [288,]  254  251
##  [289,]    0    0
##  [290,]    0    0
##  [291,]  252  253
##  [292,]  253  253
##  [293,]    2   45
##  [294,]    0    0
##  [295,]    0    0
##  [296,]  133  160
##  [297,]    0    0
##  [298,]    0    0
##  [299,]  253  253
##  [300,]    0  155
##  [301,]   42  235
##  [302,]    0    0
##  [303,]    0    0
##  [304,]    0    0
##  [305,]   29   29
##  [306,]    0    0
##  [307,]  100  176
##  [308,]    0    0
##  [309,]    0    0
##  [310,]  232  253
##  [311,]  235  254
##  [312,]    0    0
##  [313,]  183  102
##  [314,]    0   35
##  [315,]    0    0
##  [316,]  243  253
##  [317,]  255  255
##  [318,]    0    0
##  [319,]  241  224
##  [320,]    0    5
##  [321,]    0    0
##  [322,]  230  253
##  [323,]    0    0
```

```
##  [324,]    0    0
##  [325,]    0    0
##  [326,]    0    0
##  [327,]    0    0
##  [328,]  253  253
##  [329,]   45    0
##  [330,]    0    0
##  [331,]   70   70
##  [332,]    0    0
##  [333,]    0    0
##  [334,]  184  184
##  [335,]    0  183
##  [336,]  211   86
##  [337,]    0    0
##  [338,]    0    0
##  [339,]    0    0
##  [340,]    0    0
##  [341,]    0   64
##  [342,]  253  255
##  [343,]  132  152
##  [344,]  252  241
##  [345,]    0    0
##  [346,]  158  254
##  [347,]    8  134
##  [348,]    0    0
##  [349,]  205  254
##  [350,]    0    0
##  [351,]    0    3
##  [352,]  180  253
##  [353,]  253  207
##  [354,]    0    0
##  [355,]    0  102
##  [356,]  254  254
##  [357,]  253  253
##  [358,]  211  253
##  [359,]  254   95
##  [360,]    0    0
##  [361,]  253  253
##  [362,]  160  252
##  [363,]    0    0
##  [364,]    0   96
##  [365,]    0    0
##  [366,]    0    0
##  [367,]  253  217
##  [368,]    0    0
##  [369,]  254  254
##  [370,]    0    0
##  [371,]  253  253
##  [372,]    0    0
##  [373,]    0   43
##  [374,]    0    0
##  [375,]  121  252
##  [376,]    0    0
##  [377,]    0    0
```

```
##  [378,]    0    0
##  [379,]    0    0
##  [380,]    0    3
##  [381,]    0    0
##  [382,]    0    0
##  [383,]  254   84
##  [384,]    0    0
##  [385,]    0   56
##  [386,]    0   52
##  [387,]  252  240
##  [388,]    0    0
##  [389,]    0    0
##  [390,]    0    0
##  [391,]   38  233
##  [392,]  197  173
##  [393,]   53  232
##  [394,]   64   64
##  [395,]  181    0
##  [396,]    0    0
##  [397,]    0    0
##  [398,]  207  252
##  [399,]  253  158
##  [400,]   27    0
##  [401,]    0    0
##  [402,]    0    0
##  [403,]    0    0
##  [404,]  105    0
##  [405,]  253  253
##  [406,]   93  239
##  [407,]  253   58
##  [408,]   42   27
##  [409,]  254  195
##  [410,]    0    0
##  [411,]  229  253
##  [412,]    0    0
##  [413,]    0  100
##  [414,]    0    0
##  [415,]    0   70
##  [416,]    0    0
##  [417,]  253  251
##  [418,]   58    0
##  [419,]    7  221
##  [420,]    0   45
##  [421,]  252  253
##  [422,]    0    0
##  [423,]    0   77
##  [424,]    0    0
##  [425,]  253  253
##  [426,]   23   29
##  [427,]  252  252
##  [428,]    0    0
##  [429,]  135  246
##  [430,]    0    0
##  [431,]    0    0
```

```
##  [432,]    0    0
##  [433,]    0    0
##  [434,]  253  253
##  [435,]    0    0
##  [436,]    0    0
##  [437,]    0    0
##  [438,]   40    8
##  [439,]    0   34
##  [440,]  254  254
##  [441,]    0    0
##  [442,]    0   47
##  [443,]    0    0
##  [444,]   99  253
##  [445,]  222  246
##  [446,]  252  209
##  [447,]    0    0
##  [448,]  172  253
##  [449,]   12  161
##  [450,]    0    0
##  [451,]  251  180
##  [452,]    0    0
##  [453,]  254  253
##  [454,]    0    0
##  [455,]  254  223
##  [456,]  237  252
##  [457,]  252  252
##  [458,]    0    0
##  [459,]    0    0
##  [460,]   49  159
##  [461,]    0    0
##  [462,]    0    0
##  [463,]    0    0
##  [464,]    0    0
##  [465,]    0    0
##  [466,]    0    0
##  [467,]   98  254
##  [468,]    0    0
##  [469,]    0    0
##  [470,]    0    0
##  [471,]    0    0
##  [472,]   51   51
##  [473,]  154  250
##  [474,]    0    0
##  [475,]    0    0
##  [476,]  211  253
##  [477,]    0    0
##  [478,]    0    0
##  [479,]  114  253
##  [480,]  254  253
##  [481,]    0    0
##  [482,]    0    0
##  [483,]    0    0
##  [484,]    0    0
##  [485,]  253  132
```

```
##  [486,]    0    0
##  [487,]   67    0
##  [488,]    0    9
##  [489,]  254  255
##  [490,]    0    0
##  [491,]  253  250
##  [492,]    0  255
##  [493,]  252  250
##  [494,]    0    0
##  [495,]    0    0
##  [496,]  253  253
##  [497,]  202  203
##  [498,]    0    0
##  [499,]    0    0
##  [500,]  130   76
##  [501,]    0    0
##  [502,]    0    0
##  [503,]    0    0
##  [504,]  115   34
##  [505,]  105    0
##  [506,]    0    0
##  [507,]    0    0
##  [508,]  143  253
##  [509,]  254  254
##  [510,]  160  253
##  [511,]  253  224
##  [512,]   12  118
##  [513,]    0    0
##  [514,]    0    0
##  [515,]  148  237
##  [516,]    0    0
##  [517,]    0    0
##  [518,]   24    0
##  [519,]    0    7
##  [520,]    0    0
##  [521,]    0    0
##  [522,]  128   25
##  [523,]    0    0
##  [524,]    0    0
##  [525,]    0    0
##  [526,]    0    0
##  [527,]    0    0
##  [528,]   12    0
##  [529,]  221   62
##  [530,]    0   51
##  [531,]    0    0
##  [532,]    0    0
##  [533,]  253  253
##  [534,]   18  246
##  [535,]  204  252
##  [536,]  128  253
##  [537,]    0    0
##  [538,]  156  127
##  [539,]  254  254
```

```
##  [540,]    0   42
##  [541,]  114    0
##  [542,]    0    0
##  [543,]  151    0
##  [544,]    0    0
##  [545,]  189  112
##  [546,]    0  164
##  [547,]  252  253
##  [548,]    0   15
##  [549,]    0    0
##  [550,]   82  202
##  [551,]    0    8
##  [552,]    0    0
##  [553,]  215  254
##  [554,]  206  252
##  [555,]  251  253
##  [556,]    0    0
##  [557,]  253  253
##  [558,]  253  253
##  [559,]  115    0
##  [560,]  110  231
##  [561,]    0  136
##  [562,]  254  254
##  [563,]    0    0
##  [564,]    0   23
##  [565,]    0    0
##  [566,]  113  206
##  [567,]    0   71
##  [568,]    0    0
##  [569,]    0    0
##  [570,]    0   22
##  [571,]    0    0
##  [572,]   25  119
##  [573,]  255  255
##  [574,]  246  253
##  [575,]  253  128
##  [576,]   21   22
##  [577,]  194  113
##  [578,]    0    0
##  [579,]    0    0
##  [580,]    0    0
##  [581,]   43  225
##  [582,]  253  253
##  [583,]    0    0
##  [584,]  112  166
##  [585,]    0    0
##  [586,]    0    0
##  [587,]    0    0
##  [588,]  253  253
##  [589,]   70  254
##  [590,]    0    0
##  [591,]    0  157
##  [592,]    0    0
##  [593,]    0    6
```

```
##  [594,]  179  253
##  [595,]  221  253
##  [596,]    0   32
##  [597,]    0    0
##  [598,]  252   82
##  [599,]    0    0
##  [600,]    0    0
##  [601,]  111  245
##  [602,]    0    0
##  [603,]  253   65
##  [604,]   64    0
##  [605,]   47  254
##  [606,]    0   14
##  [607,]   10  168
##  [608,]    7  160
##  [609,]    0    0
##  [610,]  252  252
##  [611,]    0    0
##  [612,]   23  172
##  [613,]    0    0
##  [614,]  253  247
##  [615,]    0    0
##  [616,]    0    0
##  [617,]    0    0
##  [618,]    0    0
##  [619,]  253    0
##  [620,]    0    0
##  [621,]  252  253
##  [622,]    0    0
##  [623,]  253  255
##  [624,]   50    7
##  [625,]    0    0
##  [626,]    0    0
##  [627,]    0    0
##  [628,]    0    0
##  [629,]  182  253
##  [630,]  206  253
##  [631,]   68   41
##  [632,]    0    0
##  [633,]   47    5
##  [634,]   18    0
##  [635,]    0   80
##  [636,]    0    0
##  [637,]    0    0
##  [638,]  193  254
##  [639,]  254  177
##  [640,]    0    0
##  [641,]   84   19
##  [642,]  236  253
##  [643,]    0    0
##  [644,]  253  253
##  [645,]  254  254
##  [646,]  253  253
##  [647,]  164  253
```

```
##  [648,]    0    0
##  [649,]  229  254
##  [650,]    5    0
##  [651,]   88  211
##  [652,]    0    0
##  [653,]  252  229
##  [654,]    0    0
##  [655,]    0    9
##  [656,]    0    0
##  [657,]    5    0
##  [658,]    0    0
##  [659,]    0    0
##  [660,]    8  128
##  [661,]   25    0
##  [662,]    0   29
##  [663,]   19    0
##  [664,]    0    0
##  [665,]    0   10
##  [666,]  235  239
##  [667,]    0    0
##  [668,]  255  128
##  [669,]    0    0
##  [670,]    0    0
##  [671,]   14   51
##  [672,]  253  253
##  [673,]    0    0
##  [674,]    0    0
##  [675,]  244   89
##  [676,]  253  253
##  [677,]  254  230
##  [678,]   20    0
##  [679,]  253  253
##  [680,]  239  249
##  [681,]    0    0
##  [682,]    0    0
##  [683,]    0    0
##  [684,]    0    0
##  [685,]    0    0
##  [686,]  254  254
##  [687,]    0    0
##  [688,]    0    0
##  [689,]   13  221
##  [690,]    0    0
##  [691,]    0    0
##  [692,]  206  253
##  [693,]  131  178
##  [694,]   57  144
##  [695,]   73  253
##  [696,]  252  252
##  [697,]    0   47
##  [698,]    0    0
##  [699,]  253  253
##  [700,]  237  165
##  [701,]    0    0
```

```
##  [702,]    0    0
##  [703,]    0    0
##  [704,]    0    0
##  [705,]   17   65
##  [706,]  253  253
##  [707,]   49  189
##  [708,]   51   92
##  [709,]  133  254
##  [710,]    0    0
##  [711,]  253   72
##  [712,]  252  252
##  [713,]  180    0
##  [714,]    0   55
##  [715,]  113  254
##  [716,]  254  253
##  [717,]  249  127
##  [718,]    0    0
##  [719,]  253  254
##  [720,]  251  253
##  [721,]  253  246
##  [722,]    0    0
##  [723,]    8    0
##  [724,]    0    0
##  [725,]    0    0
##  [726,]  252  252
##  [727,]  254  218
##  [728,]    0    0
##  [729,]    0   51
##  [730,]    0    0
##  [731,]    0    0
##  [732,]  253  253
##  [733,]  209  253
##  [734,]    0    0
##  [735,]  122  198
##  [736,]    0    0
##  [737,]  255   29
##  [738,]   32    0
##  [739,]  254   59
##  [740,]    0    5
##  [741,]  254  139
##  [742,]    0    0
##  [743,]    0    0
##  [744,]    7    0
##  [745,]  226  226
##  [746,]   73    0
##  [747,]    0  219
##  [748,]  176  253
##  [749,]  194   71
##  [750,]    9    0
##  [751,]    0   29
##  [752,]  253  254
##  [753,]  252  252
##  [754,]    0    0
##  [755,]    0    0
```

```
##  [756,]    0    0
##  [757,]  208  208
##  [758,]  246  230
##  [759,]  251  252
##  [760,]    0    0
##  [761,]  243   40
##  [762,]  177    8
##  [763,]    0    0
##  [764,]    0    0
##  [765,]    0   57
##  [766,]  253  253
##  [767,]  203  204
##  [768,]  254  200
##  [769,]  208  199
##  [770,]  252  253
##  [771,]    0    0
##  [772,]  110  110
##  [773,]   15  178
##  [774,]    0    0
##  [775,]    0    0
##  [776,]   60  100
##  [777,]    0    0
##  [778,]  241  101
##  [779,]    0    0
##  [780,]  253  252
##  [781,]  253  252
##  [782,]    7    0
##  [783,]    0    0
##  [784,]  253  253
##  [785,]  224  252
##  [786,]    0    0
##  [787,]    0    0
##  [788,]    0    0
##  [789,]    0    0
##  [790,]  254  254
##  [791,]    0    0
##  [792,]  218  253
##  [793,]  242   78
##  [794,]    0    0
##  [795,]    7    0
##  [796,]    0   54
##  [797,]   24    0
##  [798,]    0   10
##  [799,]    0    0
##  [800,]  253  254
##  [801,]    0  103
##  [802,]  132  253
##  [803,]    0   78
##  [804,]    0    6
##  [805,]    0    0
##  [806,]  254  254
##  [807,]    0   15
##  [808,]  144  254
##  [809,]  252  154
```

```
##  [810,]  253  252
##  [811,]  116  137
##  [812,]  253  253
##  [813,]    0   54
##  [814,]    0  131
##  [815,]  141  210
##  [816,]  203  223
##  [817,]    0    0
##  [818,]  254  254
##  [819,]    0    0
##  [820,]    0    0
##  [821,]    0    0
##  [822,]  253  253
##  [823,]    2   41
##  [824,]   13  126
##  [825,]    0  135
##  [826,]    0    0
##  [827,]    0    0
##  [828,]    0    0
##  [829,]    0    0
##  [830,]    5    0
##  [831,]  252  253
##  [832,]  137  184
##  [833,]  255  253
##  [834,]  253  252
##  [835,]    0    0
##  [836,]  253  252
##  [837,]   82  223
##  [838,]  254  254
##  [839,]  252  253
##  [840,]    0    0
##  [841,]  253  204
##  [842,]    0    0
##  [843,]  253  253
##  [844,]  254  253
##  [845,]    0    0
##  [846,]  249  253
##  [847,]    0    0
##  [848,]    0    0
##  [849,]    0    0
##  [850,]   64    0
##  [851,]    0    0
##  [852,]    0    0
##  [853,]   59    0
##  [854,]    0    0
##  [855,]    0    0
##  [856,]    0    0
##  [857,]  254  253
##  [858,]  252  252
##  [859,]    0    0
##  [860,]    0    0
##  [861,]    0    0
##  [862,]  253  134
##  [863,]    0  190
```

```
##  [864,]   77  254
##  [865,]  159  254
##  [866,]  242  253
##  [867,]    0    0
##  [868,]  253  253
##  [869,]    0    0
##  [870,]    8    0
##  [871,]  253  253
##  [872,]  240  254
##  [873,]    0    0
##  [874,]    0    0
##  [875,]  253  253
##  [876,]  253  253
##  [877,]   44  249
##  [878,]    0    0
##  [879,]  243  174
##  [880,]   97   97
##  [881,]    0    0
##  [882,]    6   86
##  [883,]    0    0
##  [884,]    0    0
##  [885,]   82  253
##  [886,]  197  253
##  [887,]  114    0
##  [888,]    1   25
##  [889,]    0    0
##  [890,]    0    0
##  [891,]  252  253
##  [892,]  240  253
##  [893,]  181   20
##  [894,]    0    0
##  [895,]  203  254
##  [896,]  254  253
##  [897,]    0    0
##  [898,]    0    0
##  [899,]    0    0
##  [900,]   24    0
##  [901,]    6  191
##  [902,]    0    0
##  [903,]    0    0
##  [904,]    0    0
##  [905,]    0    0
##  [906,]  104  254
##  [907,]    0  152
##  [908,]    0    8
##  [909,]   67  160
##  [910,]  253  253
##  [911,]    0    0
##  [912,]    0    0
##  [913,]    0    0
##  [914,]   37  167
##  [915,]    0    0
##  [916,]   35    0
##  [917,]    7  108
```

```
##  [918,]    0    0
##  [919,]   71  241
##  [920,]  254  254
##  [921,]  253  253
##  [922,]    0    0
##  [923,]    1    0
##  [924,]    0   64
##  [925,]  198  198
##  [926,]    0  170
##  [927,]    0    0
##  [928,]    0    0
##  [929,]    0    0
##  [930,]    0    0
##  [931,]    0    0
##  [932,]    0    0
##  [933,]  123  254
##  [934,]  251  225
##  [935,]    0    0
##  [936,]   14   69
##  [937,]   89  253
##  [938,]    0    0
##  [939,]  190  252
##  [940,]   94    0
##  [941,]    0    0
##  [942,]  150  254
##  [943,]  163  238
##  [944,]    7    0
##  [945,]  168  169
##  [946,]    0    0
##  [947,]   75  231
##  [948,]    1    0
##  [949,]  128  254
##  [950,]    0    0
##  [951,]  116  253
##  [952,]  241  254
##  [953,]    0    0
##  [954,]  254  254
##  [955,]    0    0
##  [956,]    0    0
##  [957,]   74   53
##  [958,]    8    0
##  [959,]  253  253
##  [960,]  253  253
##  [961,]    0    0
##  [962,]  234  254
##  [963,]    0    0
##  [964,]   98  253
##  [965,]  222   25
##  [966,]    0    0
##  [967,]  241  189
##  [968,]    0    0
##  [969,]    0   46
##  [970,]    0    0
##  [971,]    6    6
```

```
##  [972,]    0    0
##  [973,]    0    0
##  [974,]   23    0
##  [975,]  231  254
##  [976,]  254  254
##  [977,]    0   32
##  [978,]   15    0
##  [979,]  155    0
##  [980,]    6    0
##  [981,]  135  243
##  [982,]    0    0
##  [983,]  253  201
##  [984,]  198  254
##  [985,]    0    0
##  [986,]   22    0
##  [987,]    3  171
##  [988,]    0    0
##  [989,]    0    0
##  [990,]    0    0
##  [991,]    0    0
##  [992,]  221  151
##  [993,]  254  172
##  [994,]  156  253
##  [995,]    0    0
##  [996,]  254  254
##  [997,]    0    0
##  [998,]    0    0
##  [999,]  103   64
## [1000,]  139    0
```

```
x[c(2,3),]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
```

```
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,99] [,100] [,101] [,102] [,103] [,104] [,105] [,106] [,107] [,108]
## [1,]     0      0      0      0      0      0      0      0      0      0
## [2,]     0      0      0      0      0      0      0      0      0      0
##      [,109] [,110] [,111] [,112] [,113] [,114] [,115] [,116] [,117] [,118]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,119] [,120] [,121] [,122] [,123] [,124] [,125] [,126] [,127] [,128]
## [1,]      0      0      0      0      0      0      0      0      0     51
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137] [,138]
## [1,]    159    253    159     50      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,139] [,140] [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,149] [,150] [,151] [,152] [,153] [,154] [,155] [,156] [,157] [,158]
## [1,]      0      0      0      0      0      0     48    238    252    252
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,159] [,160] [,161] [,162] [,163] [,164] [,165] [,166] [,167] [,168]
## [1,]    252    237      0      0      0      0      0      0      0      0
## [2,]      0      0     67    232     39      0      0      0      0      0
##      [,169] [,170] [,171] [,172] [,173] [,174] [,175] [,176] [,177] [,178]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0     62     81      0      0      0      0
##      [,179] [,180] [,181] [,182] [,183] [,184] [,185] [,186] [,187] [,188]
## [1,]      0      0      0     54    227    253    252    239    233    252
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,189] [,190] [,191] [,192] [,193] [,194] [,195] [,196] [,197] [,198]
## [1,]     57      6      0      0      0      0      0      0      0      0
## [2,]    120    180     39      0      0      0      0      0      0      0
##      [,199] [,200] [,201] [,202] [,203] [,204] [,205] [,206] [,207] [,208]
## [1,]      0      0      0      0      0      0      0      0      0     10
## [2,]      0      0    126    163      0      0      0      0      0      0
##      [,209] [,210] [,211] [,212] [,213] [,214] [,215] [,216] [,217] [,218]
## [1,]     60    224    252    253    252    202     84    252    253    122
## [2,]      0      0      0      0      0      0      0      2    153    210
##      [,219] [,220] [,221] [,222] [,223] [,224] [,225] [,226] [,227] [,228]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]     40      0      0      0      0      0      0      0      0      0
##      [,229] [,230] [,231] [,232] [,233] [,234] [,235] [,236] [,237] [,238]
## [1,]      0      0      0      0      0      0      0    163    252    252
## [2,]    220    163      0      0      0      0      0      0      0      0
##      [,239] [,240] [,241] [,242] [,243] [,244] [,245] [,246] [,247] [,248]
## [1,]    252    253    252    252     96    189    253    167      0      0
## [2,]      0      0      0      0      0     27    254    162      0      0
##      [,249] [,250] [,251] [,252] [,253] [,254] [,255] [,256] [,257] [,258]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0    222    163
##      [,259] [,260] [,261] [,262] [,263] [,264] [,265] [,266] [,267] [,268]
## [1,]      0      0      0      0     51    238    253    253    190    114
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,269] [,270] [,271] [,272] [,273] [,274] [,275] [,276] [,277] [,278]
```

```
## [1,]    253    228     47     79    255    168      0      0      0      0
## [2,]      0      0      0    183    254    125      0      0      0      0
##      [,279] [,280] [,281] [,282] [,283] [,284] [,285] [,286] [,287] [,288]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0     46    245    163      0      0
##      [,289] [,290] [,291] [,292] [,293] [,294] [,295] [,296] [,297] [,298]
## [1,]      0     48    238    252    252    179     12     75    121     21
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,299] [,300] [,301] [,302] [,303] [,304] [,305] [,306] [,307] [,308]
## [1,]      0      0    253    243     50      0      0      0      0      0
## [2,]      0    198    254     56      0      0      0      0      0      0
##      [,309] [,310] [,311] [,312] [,313] [,314] [,315] [,316] [,317] [,318]
## [1,]      0      0      0      0      0      0      0      0     38    165
## [2,]      0      0      0    120    254    163      0      0      0      0
##      [,319] [,320] [,321] [,322] [,323] [,324] [,325] [,326] [,327] [,328]
## [1,]    253    233    208     84      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0     23    231
##      [,329] [,330] [,331] [,332] [,333] [,334] [,335] [,336] [,337] [,338]
## [1,]    253    252    165      0      0      0      0      0      0      0
## [2,]    254     29      0      0      0      0      0      0      0      0
##      [,339] [,340] [,341] [,342] [,343] [,344] [,345] [,346] [,347] [,348]
## [1,]      0      0      0      0      0      7    178    252    240     71
## [2,]      0    159    254    120      0      0      0      0      0      0
##      [,349] [,350] [,351] [,352] [,353] [,354] [,355] [,356] [,357] [,358]
## [1,]     19     28      0      0      0      0      0      0    253    252
## [2,]      0      0      0      0      0      0    163    254    216     16
##      [,359] [,360] [,361] [,362] [,363] [,364] [,365] [,366] [,367] [,368]
## [1,]    195      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0    159
##      [,369] [,370] [,371] [,372] [,373] [,374] [,375] [,376] [,377] [,378]
## [1,]      0      0      0     57    252    252     63      0      0      0
## [2,]    254     67      0      0      0      0      0      0      0      0
##      [,379] [,380] [,381] [,382] [,383] [,384] [,385] [,386] [,387] [,388]
## [1,]      0      0      0      0      0      0    253    252    195      0
## [2,]      0     14     86    178    248    254     91      0      0      0
##      [,389] [,390] [,391] [,392] [,393] [,394] [,395] [,396] [,397] [,398]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0    159    254     85
##      [,399] [,400] [,401] [,402] [,403] [,404] [,405] [,406] [,407] [,408]
## [1,]      0    198    253    190      0      0      0      0      0      0
## [2,]      0      0      0     47     49    116    144    150    241    243
##      [,409] [,410] [,411] [,412] [,413] [,414] [,415] [,416] [,417] [,418]
## [1,]      0      0      0      0    255    253    196      0      0      0
## [2,]    234    179    241    252     40      0      0      0      0      0
##      [,419] [,420] [,421] [,422] [,423] [,424] [,425] [,426] [,427] [,428]
## [1,]      0      0      0      0      0      0      0      0     76    246
## [2,]      0      0      0      0      0    150    253    237    207    207
##      [,429] [,430] [,431] [,432] [,433] [,434] [,435] [,436] [,437] [,438]
## [1,]    252    112      0      0      0      0      0      0      0      0
## [2,]    207    253    254    250    240    198    143     91     28      5
##      [,439] [,440] [,441] [,442] [,443] [,444] [,445] [,446] [,447] [,448]
## [1,]      0      0    253    252    148      0      0      0      0      0
## [2,]    233    250      0      0      0      0      0      0      0      0
##      [,449] [,450] [,451] [,452] [,453] [,454] [,455] [,456] [,457] [,458]
```

```
## [1,]      0      0      0      0      0      0     85    252    230     25
## [2,]      0      0      0      0    119    177    177    177    177    177
##      [,459] [,460] [,461] [,462] [,463] [,464] [,465] [,466] [,467] [,468]
## [1,]      0      0      0      0      0      0      0      0      7    135
## [2,]     98     56      0      0      0      0      0    102    254    220
##      [,469] [,470] [,471] [,472] [,473] [,474] [,475] [,476] [,477] [,478]
## [1,]    253    186     12      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,479] [,480] [,481] [,482] [,483] [,484] [,485] [,486] [,487] [,488]
## [1,]      0      0      0      0     85    252    223      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,489] [,490] [,491] [,492] [,493] [,494] [,495] [,496] [,497] [,498]
## [1,]      0      0      0      0      0      7    131    252    225     71
## [2,]      0      0      0      0      0    169    254    137      0      0
##      [,499] [,500] [,501] [,502] [,503] [,504] [,505] [,506] [,507] [,508]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,509] [,510] [,511] [,512] [,513] [,514] [,515] [,516] [,517] [,518]
## [1,]      0      0     85    252    145      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,519] [,520] [,521] [,522] [,523] [,524] [,525] [,526] [,527] [,528]
## [1,]      0      0     48    165    252    173      0      0      0      0
## [2,]      0      0      0    169    254     57      0      0      0      0
##      [,529] [,530] [,531] [,532] [,533] [,534] [,535] [,536] [,537] [,538]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,539] [,540] [,541] [,542] [,543] [,544] [,545] [,546] [,547] [,548]
## [1,]     86    253    225      0      0      0      0      0      0    114
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,549] [,550] [,551] [,552] [,553] [,554] [,555] [,556] [,557] [,558]
## [1,]    238    253    162      0      0      0      0      0      0      0
## [2,]      0    169    254     57      0      0      0      0      0      0
##      [,559] [,560] [,561] [,562] [,563] [,564] [,565] [,566] [,567] [,568]
## [1,]      0      0      0      0      0      0      0      0     85    252
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,569] [,570] [,571] [,572] [,573] [,574] [,575] [,576] [,577] [,578]
## [1,]    249    146     48     29     85    178    225    253    223    167
## [2,]      0      0      0      0      0      0      0      0      0    169
##      [,579] [,580] [,581] [,582] [,583] [,584] [,585] [,586] [,587] [,588]
## [1,]     56      0      0      0      0      0      0      0      0      0
## [2,]    255     94      0      0      0      0      0      0      0      0
##      [,589] [,590] [,591] [,592] [,593] [,594] [,595] [,596] [,597] [,598]
## [1,]      0      0      0      0      0      0     85    252    252    252
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,599] [,600] [,601] [,602] [,603] [,604] [,605] [,606] [,607] [,608]
## [1,]    229    215    252    252    252    196    130      0      0      0
## [2,]      0      0      0      0      0      0      0    169    254     96
##      [,609] [,610] [,611] [,612] [,613] [,614] [,615] [,616] [,617] [,618]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,619] [,620] [,621] [,622] [,623] [,624] [,625] [,626] [,627] [,628]
## [1,]      0      0      0      0     28    199    252    252    253    252
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,629] [,630] [,631] [,632] [,633] [,634] [,635] [,636] [,637] [,638]
```

```
## [1,]    252    233    145      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0    169    254    153      0      0
##      [,639] [,640] [,641] [,642] [,643] [,644] [,645] [,646] [,647] [,648]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,649] [,650] [,651] [,652] [,653] [,654] [,655] [,656] [,657] [,658]
## [1,]      0      0      0     25    128    252    253    252    141     37
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,659] [,660] [,661] [,662] [,663] [,664] [,665] [,666] [,667] [,668]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0    169    255    153      0      0      0      0
##      [,669] [,670] [,671] [,672] [,673] [,674] [,675] [,676] [,677] [,678]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,679] [,680] [,681] [,682] [,683] [,684] [,685] [,686] [,687] [,688]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,689] [,690] [,691] [,692] [,693] [,694] [,695] [,696] [,697] [,698]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0     96    254    153      0      0      0      0      0      0
##      [,699] [,700] [,701] [,702] [,703] [,704] [,705] [,706] [,707] [,708]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,709] [,710] [,711] [,712] [,713] [,714] [,715] [,716] [,717] [,718]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,719] [,720] [,721] [,722] [,723] [,724] [,725] [,726] [,727] [,728]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,729] [,730] [,731] [,732] [,733] [,734] [,735] [,736] [,737] [,738]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,739] [,740] [,741] [,742] [,743] [,744] [,745] [,746] [,747] [,748]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,749] [,750] [,751] [,752] [,753] [,754] [,755] [,756] [,757] [,758]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,759] [,760] [,761] [,762] [,763] [,764] [,765] [,766] [,767] [,768]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,769] [,770] [,771] [,772] [,773] [,774] [,775] [,776] [,777] [,778]
## [1,]      0      0      0      0      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0      0      0      0      0
##      [,779] [,780] [,781] [,782] [,783] [,784]
## [1,]      0      0      0      0      0      0
## [2,]      0      0      0      0      0      0
```

```
new_x <- x[ ,colSds(x) > 60]
dim(new_x)
```

```
## [1] 1000  314
```

```
class(x[,1])
```

```
## [1] "integer"
```

```
dim(x[1,])
```

```
## NULL
```

```
#preserve the matrix class
class(x[ , 1, drop=FALSE])
```

```
## [1] "matrix" "array"
```

```
dim(x[, 1, drop=FALSE])
```

```
## [1] 1000    1
```

Indexing with Matrices and Binarizing the Data

There is a link to the relevant sections of the textbook: Indexing with matrices and Binarizing the data

Key points

- We can use logical operations with matrices:

```
mat <- matrix(1:15, 5, 3)
mat[mat > 6 & mat < 12] <- 0
```

- We can also binarize the data using just matrix operations:

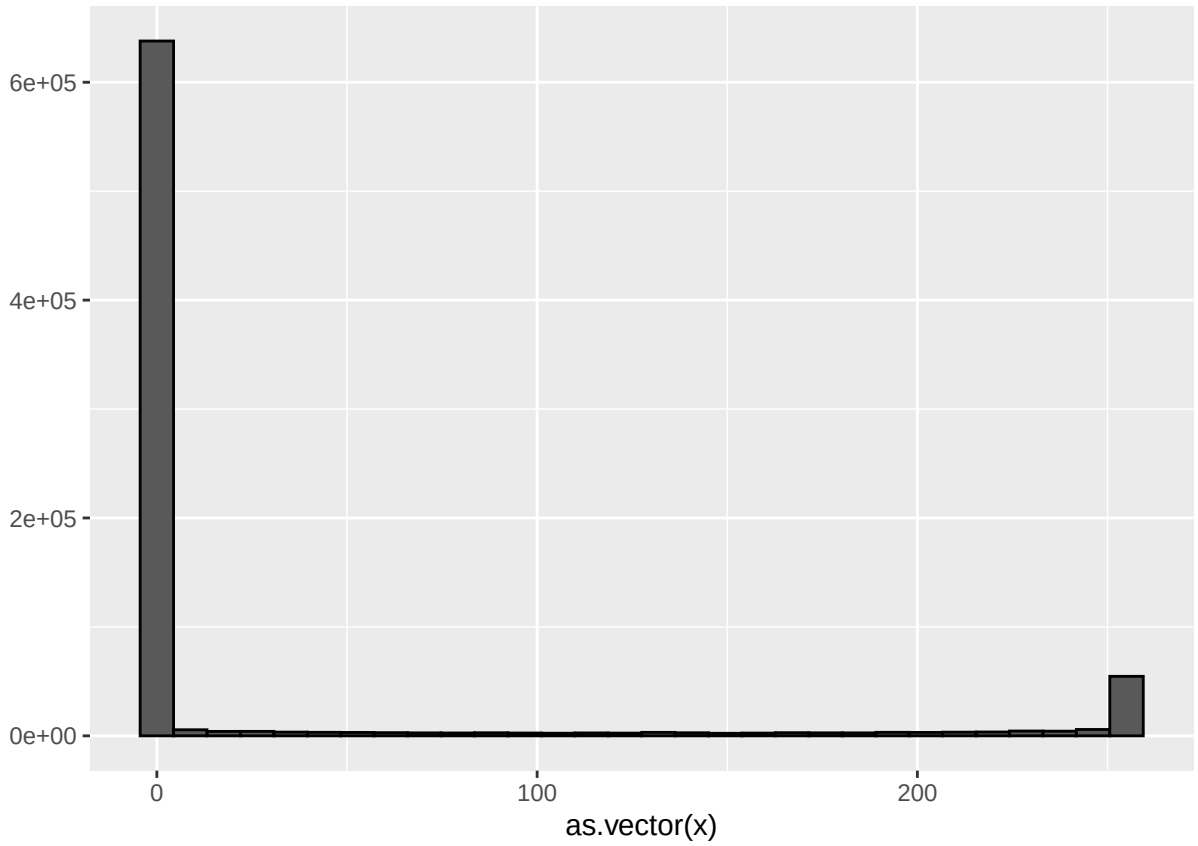
```
bin_x <- x
bin_x[bin_x < 255/2] <- 0
bin_x[bin_x > 255/2] <- 1
```

Code

```
#index with matrices
mat <- matrix(1:15, 5, 3)
as.vector(mat)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
qplot(as.vector(x), bins = 30, color = I("black"))
```




```
new_x <- x
new_x[new_x < 50] <- 0

mat <- matrix(1:15, 5, 3)
mat[mat < 3] <- 0
mat
```

```
##      [,1] [,2] [,3]
## [1,]    0    6   11
## [2,]    0    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
mat <- matrix(1:15, 5, 3)
mat[mat > 6 & mat < 12] <- 0
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    6    0
## [2,]    2    0   12
## [3,]    3    0   13
## [4,]    4    0   14
## [5,]    5    0   15
```

```r
#binarize the data
bin_x <- x
bin_x[bin_x < 255/2] <- 0
bin_x[bin_x > 255/2] <- 1
bin_X <- (x > 255/2)*1
```

Vectorization for Matrices and Matrix Algebra Operations

There is a link to the relevant sections of the textbook: Vectorization for matrices and Matrix algebra operations

Key points

- We can scale each row of a matrix using this line of code:

```r
(x - rowMeans(x)) / rowSds(x)
```

- To scale each column of a matrix, we use this code:

```r
t(t(X) - colMeans(X))
```

- We can also use a function called `sweep()` that works similarly to `apply()`. It takes each entry of a vector and subtracts it from the corresponding row or column:

```r
X_mean_0 <- sweep(x, 2, colMeans(x))
```

- Matrix multiplication: `t(x) %*% x`
- The cross product: `crossprod(x)`
- The inverse of a function: `solve(crossprod(x))`
- The QR decomposition: `qr(x)`

Code

```r
#scale each row of a matrix
(x - rowMeans(x)) / rowSds(x)

#scale each column
t(t(x) - colMeans(x))
```

```r
#take each entry of a vector and subtracts it from the corresponding row or column
x_mean_0 <- sweep(x, 2, colMeans(x))

#divide by the standard deviation
x_mean_0 <- sweep(x, 2, colMeans(x))
x_standardized <- sweep(x_mean_0, 2, colSds(x), FUN = "/")
```

Comprehension Check - Working with Matrices

1. Which line of code correctly creates a 100 by 10 matrix of randomly generated normal numbers and assigns it to `x`?

☐ A. `x <- matrix(rnorm(1000), 100, 100)`

☒ B. `x <- matrix(rnorm(100*10), 100, 10)`

☐ C. `x <- matrix(rnorm(100*10), 10, 10)`

☐ D. `x <- matrix(rnorm(100*10), 10, 100)`

2. Write the line of code that would give you the specified information about the matrix `x` that you generated in q1. Do not include any spaces in your line of code.

Dimension of x: `dim(x)`

Number of rows of x: `nrow(x)` or `dim(x)[1]` or `length(x[,1])`

Number of columns of x: `ncol(x)` or `dim(x)[2]` or `length(x[1,])`

3. Which of the following lines of code would add the scalar 1 to row 1, the scalar 2 to row 2, and so on, for the matrix `x`? Select ALL that apply.

☒ A. `x <- x + seq(nrow(x))`

☐ B. `x <- 1:nrow(x)`

☐ C. `x <- sweep(x, 2, 1:nrow(x),"+")`

☒ D. `x <- sweep(x, 1, 1:nrow(x),"+")`

4. Which of the following lines of code would add the scalar 1 to column 1, the scalar 2 to column 2, and so on, for the matrix `x`? Select ALL that apply.

☐ A. `x <- 1:ncol(x)`

☐ B. `x <- 1:col(x)`

☒ C. `x <- sweep(x, 2, 1:ncol(x), FUN = "+")`

☐ D. `x <- -x`

5. Which code correctly computes the average of each row of x?

☐ A. `mean(x)`

☐ B. `rowMedians(x)`

☐ C. `sapply(x,mean)`

☐ D. `rowSums(x)`

☒ E. `rowMeans(x)`

Which code correctly computes the average of each column of x?

☐ A. `mean(x)`

☐ B. `sapply(x,mean)`

☒ C. `colMeans(x)`

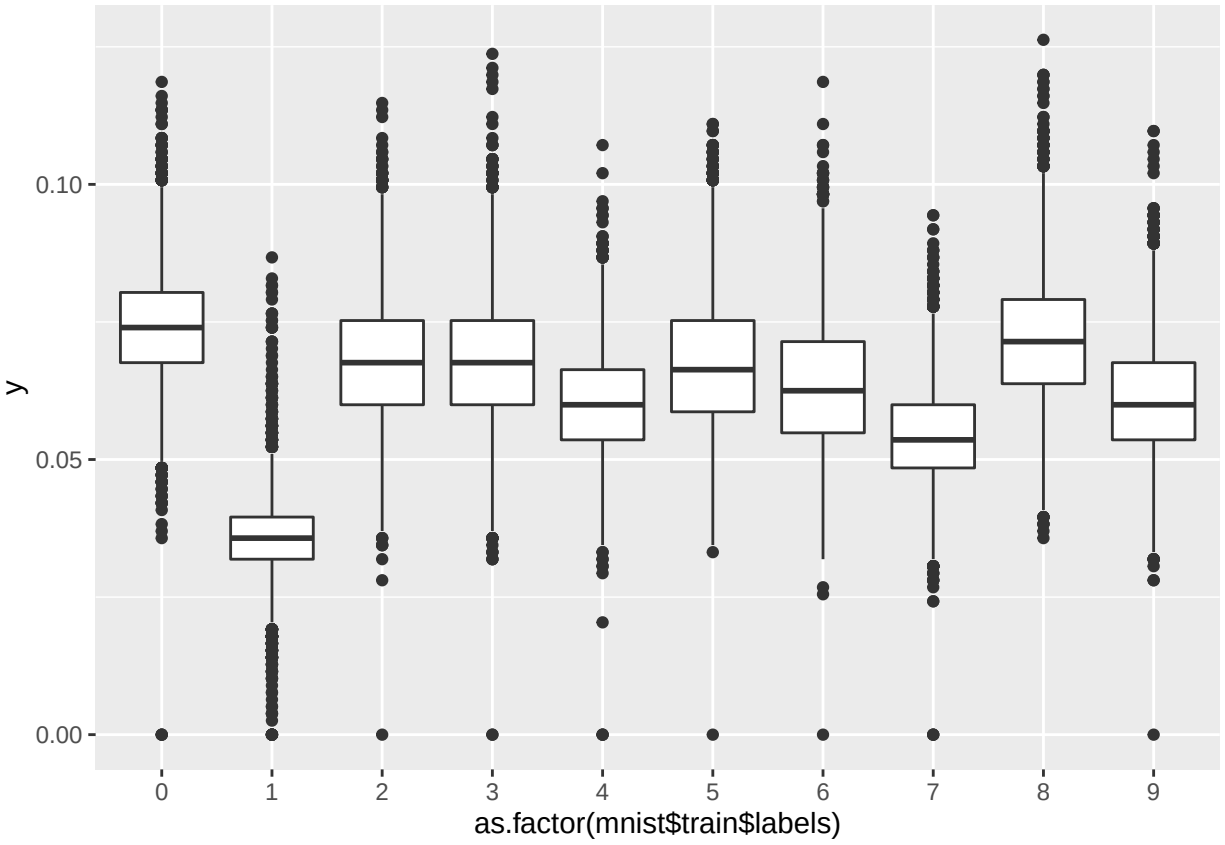
☐ D. `colMedians(x)`

☐ C. `colSums(x)`

6. For each observation in the mnist training data, compute the proportion of pixels that are in the **grey area**, defined as values between 50 and 205 (but not including 50 and 205). (To visualize this, you can make a boxplot by digit class.)

What proportion of the 60000*784 pixels in the mnist training data are in the grey area overall, defined as values between 50 and 205? Report your answer to at least 3 significant digits.

```
mnist <- read_mnist()
y <- rowMeans(mnist$train$images>50 & mnist$train$images<205)
qplot(as.factor(mnist$train$labels), y, geom = "boxplot")
```




```
mean(y) # proportion of pixels
```

```
## [1] 0.06183703
```

Section 4 - Distance, Knn, Cross Validation, and Generative Models

In the **Distance, kNN, Cross Validation, and Generative Models** section, you will learn about different types of discriminative and generative approaches for machine learning algorithms.

After completing this section, you will be able to:

- Use the **k-nearest neighbors (kNN)** algorithm.
- Understand the problems of **overtraining** and **oversmoothing**.
- Use **cross-validation** to reduce the **true error** and the **apparent error**.
- Use **generative models** such as **naive Bayes, quadratic discriminant analysis (qda),** and **linear discriminant analysis (lda)** for machine learning.

This section has three parts: **nearest neighbors, cross-validation,** and **generative models**.

Distance

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

Key points

- Most clustering and machine learning techniques rely on being able to define distance between observations, using features or predictors.
- With high dimensional data, a quick way to compute all the distances at once is to use the function `dist()`, which computes the distance between each row and produces an object of class `dist()`:

```
d <- dist(x)
```

- We can also compute distances between predictors. If N is the number of observations, the distance between two predictors, say 1 and 2, is:

$$\text{dist}(1,2) = \sqrt{\sum_{i=1}^{N}(x_{i,1} - x_{i,2})^2}$$

- To compute the distance between all pairs of the 784 predictors, we can transpose the matrix first and then use dist():

```
d <- dist(t(x))
```

Code

```
if(!exists("mnist")) mnist <- read_mnist()
set.seed(0) # if using R 3.5 or earlier
set.seed(0, sample.kind = "Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(0, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
ind <- which(mnist$train$labels %in% c(2,7)) %>% sample(500)

#the predictors are in x and the labels in y
x <- mnist$train$images[ind,]
y <- mnist$train$labels[ind]

y[1:3]
```

```
## [1] 7 7 2
```

```
x_1 <- x[1,]
x_2 <- x[2,]
x_3 <- x[3,]

#distance between two numbers
sqrt(sum((x_1 - x_2)^2))
```

```
## [1] 2079.753
```

```
sqrt(sum((x_1 - x_3)^2))
```

```
## [1] 2252.129
```

```
sqrt(sum((x_2 - x_3)^2))
```

```
## [1] 2642.906
```

```
#compute distance using matrix algebra
sqrt(crossprod(x_1 - x_2))
```

```
##          [,1]
## [1,] 2079.753
```

```
sqrt(crossprod(x_1 - x_3))
```

```
##          [,1]
## [1,] 2252.129
```

```
sqrt(crossprod(x_2 - x_3))
```

```
##          [,1]
## [1,] 2642.906
```

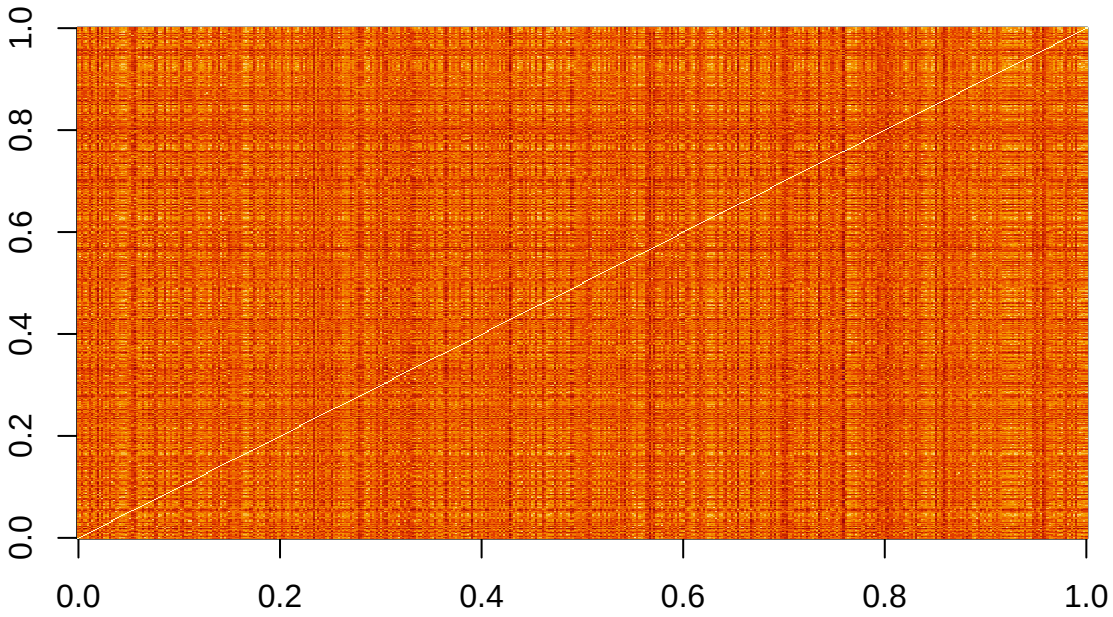
```
#compute distance between each row
d <- dist(x)
class(d)
```

```
## [1] "dist"
```

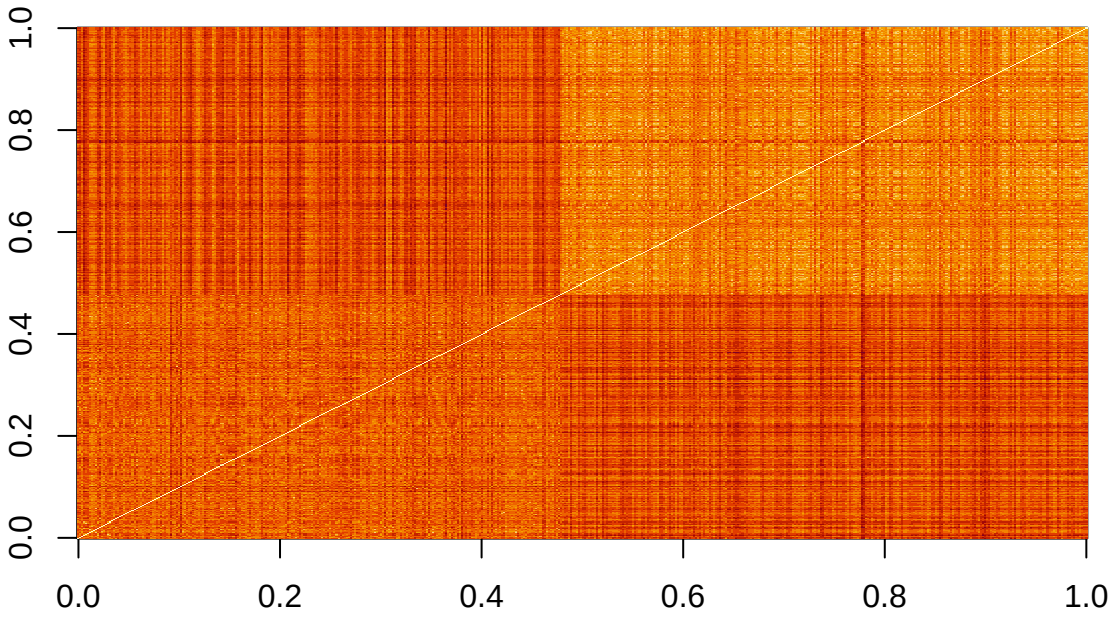
```r
as.matrix(d)[1:3,1:3]
```

```
##          1        2        3
## 1    0.000 2079.753 2252.129
## 2 2079.753    0.000 2642.906
## 3 2252.129 2642.906    0.000
```

```r
#visualize these distances
image(as.matrix(d))
```




```r
#order the distance by labels
image(as.matrix(d)[order(y), order(y)])
```

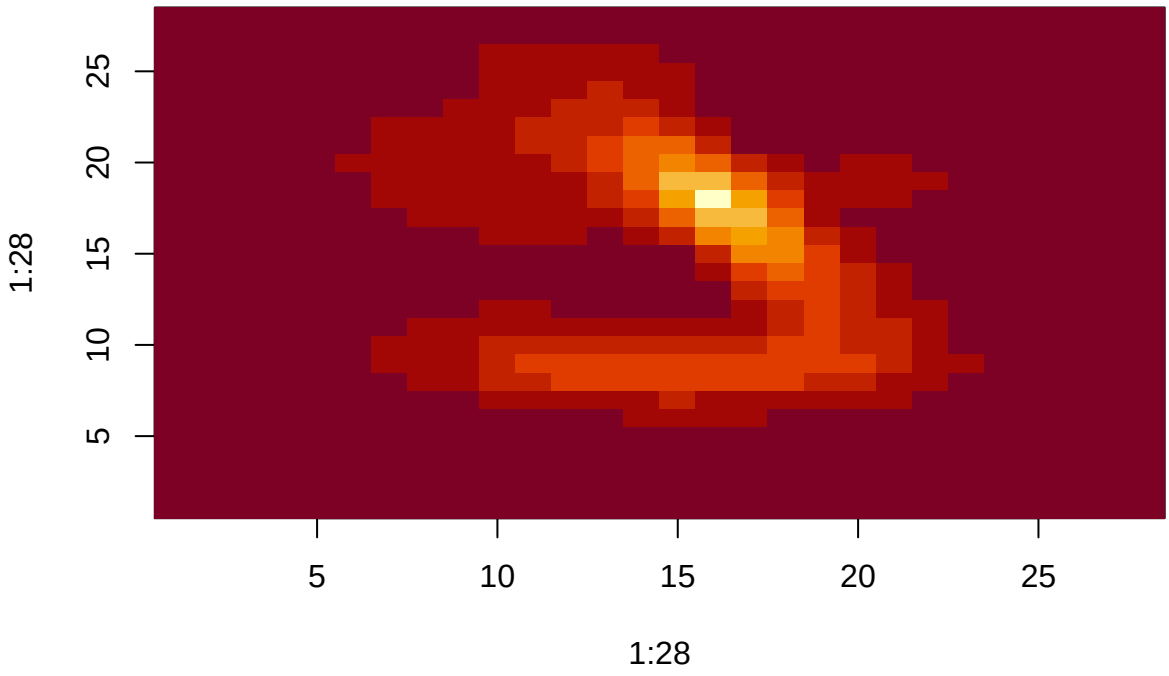


```
#compute distance between predictors
d <- dist(t(x))
dim(as.matrix(d))
```

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

```
d_492 <- as.matrix(d)[492,]

image(1:28, 1:28, matrix(d_492, 28, 28))
```



Comprehension Check - Distance

1. Load the following dataset:

```
data(tissue_gene_expression)
```

This dataset includes a matrix `x`:

```
dim(tissue_gene_expression$x)
```

```
## [1] 189 500
```

This matrix has the gene expression levels of 500 genes from 189 biological samples representing seven different tissues. The tissue type is stored in `y`:

```
table(tissue_gene_expression$y)
```

```
##
##  cerebellum       colon endometrium hippocampus      kidney       liver
##          38          34          15          31          39          26
##    placenta
##           6
```

Which of the following lines of code computes the Euclidean distance between each observation and stores it in the object `d`?

```
d <- dist(tissue_gene_expression$x)
```

- ☐ A. `d <- dist(tissue_gene_expression$x, distance='maximum')`
- ☐ B. `d <- dist(tissue_gene_expression)`
- ☒ C. `d <- dist(tissue_gene_expression$x)`
- ☐ D. `d <- cor(tissue_gene_expression$x)`

2. Using the dataset from Q1, compare the distances between observations 1 and 2 (both cerebellum), observations 39 and 40 (both colon), and observations 73 and 74 (both endometrium).

Distance-wise, are samples from tissues of the same type closer to each other than tissues of different type?

```
ind <- c(1, 2, 39, 40, 73, 74)
as.matrix(d)[ind,ind]
```

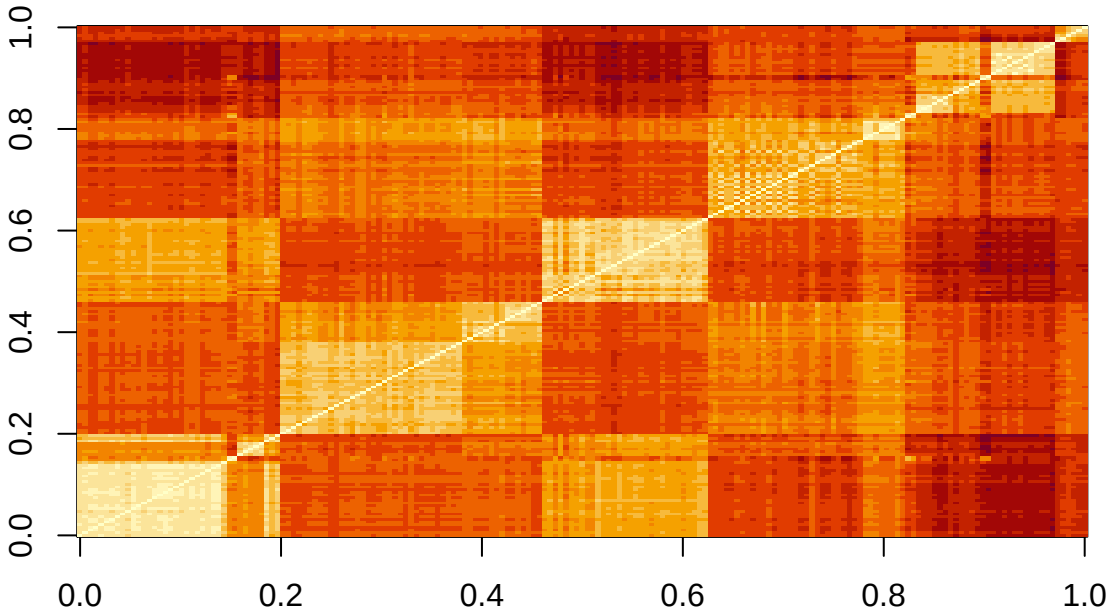
```
##               cerebellum_1 cerebellum_2   colon_1   colon_2 endometrium_1
## cerebellum_1      0.000000     7.005922 22.694801 22.699755      21.12763
## cerebellum_2      7.005922     0.000000 22.384821 22.069557      20.87910
## colon_1          22.694801    22.384821  0.000000  8.191935      14.99672
## colon_2          22.699755    22.069557  8.191935  0.000000      14.80355
## endometrium_1    21.127629    20.879099 14.996715 14.803545       0.00000
## endometrium_2    21.780792    20.674802 18.089213 17.004456      14.29405
##               endometrium_2
## cerebellum_1       21.78079
## cerebellum_2       20.67480
## colon_1            18.08921
## colon_2            17.00446
## endometrium_1      14.29405
## endometrium_2       0.00000
```

- ☐ A. No, the samples from the same tissue type are not necessarily closer.
- ☐ B. The two colon samples are close to each other, but the samples from the other two tissues are not.
- ☐ C. The two cerebellum samples are close to each other, but the samples from the other two tissues are not.
- ☒ D. Yes, the samples from the same tissue type are closer to each other.

3. Make a plot of all the distances using the image() function to see if the pattern you observed in Q2 is general.

Which code would correctly make the desired plot?

```
image(as.matrix(d))
```



☐ A. `image(d)`

☒ B. `image(as.matrix(d))`

☐ C. `d`

☐ D. `image()`

Knn

There is a link to the relevant section of the textbook: k-nearest neighbors

Key points

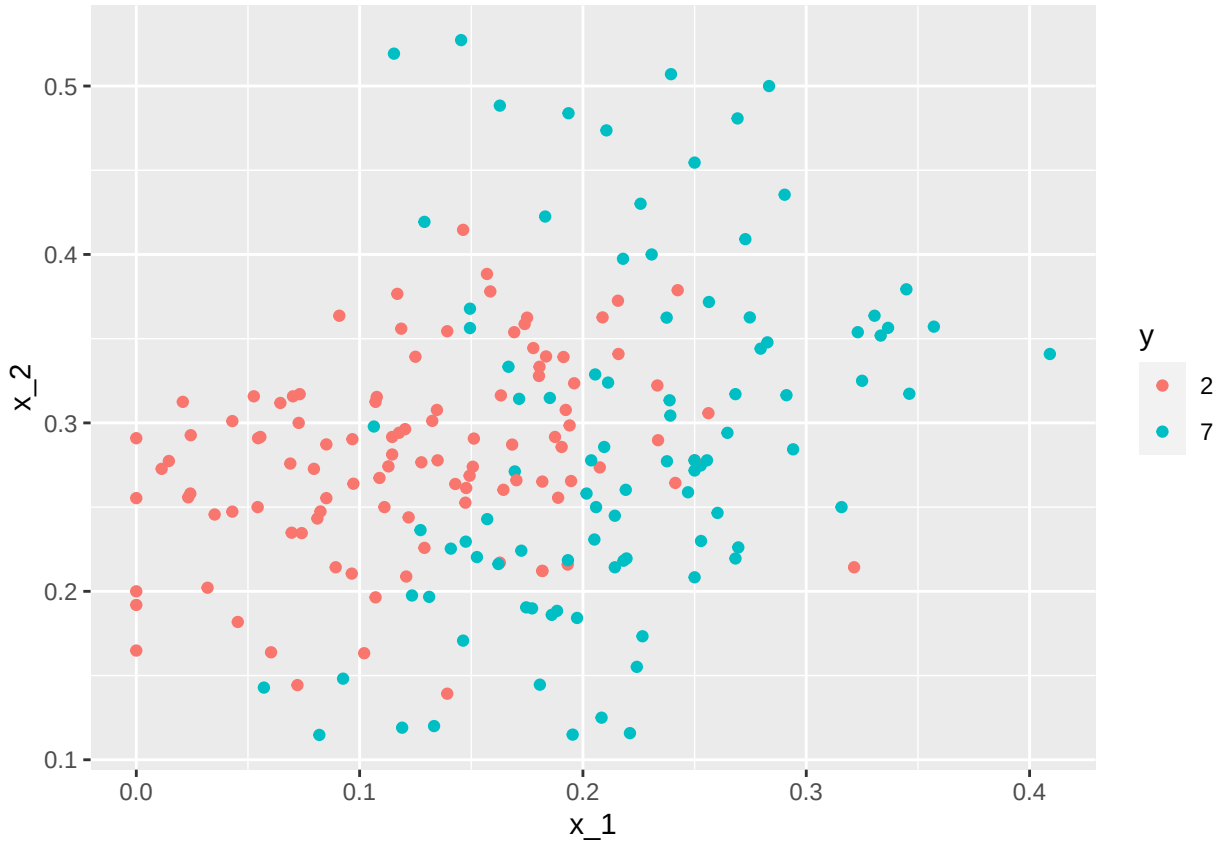
- **K-nearest neighbors (kNN)** estimates the conditional probabilities in a similar way to bin smoothing. However, kNN is easier to adapt to multiple dimensions.
- Using kNN, for any point (x_1, x_2) for which we want an estimate of $p(x_1, x_2)$, we look for the **k nearest points** to (x_1, x_2) and take an average of the 0s and 1s associated with these points. We refer to the set of points used to compute the average as the **neighborhood**. Larger values of k result in smoother estimates, while smaller values of k result in more flexible and more wiggly estimates.
- To implement the algorithm, we can use the `knn3()` function from the **caret** package. There are two ways to call this function:

1. We need to specify a formula and a data frame. The formula looks like this: outcome $\sim$ predictor$_1$ + predictor$_2$ + predictor$_3$. The `predict()` function for `knn3` produces a probability for each class.
2. We can also call the function with the first argument being the matrix predictors and the second a vector of outcomes, like this:

```
x <- as.matrix(mnist_27$train[,2:3])
y <- mnist_27$train$y
knn_fit <- knn3(x,y)
```

Code

```
data("mnist_27")
mnist_27$test %>% ggplot(aes(x_1, x_2, color = y)) + geom_point()
```




```
#logistic regression
library(caret)
fit_glm <- glm(y~x_1+x_2, data=mnist_27$train, family="binomial")
p_hat_logistic <- predict(fit_glm, mnist_27$test)
y_hat_logistic <- factor(ifelse(p_hat_logistic > 0.5, 7, 2))
confusionMatrix(data = y_hat_logistic, reference = mnist_27$test$y)$overall[1]
```

```
## Accuracy
##     0.76
```

```
#fit knn model
knn_fit <- knn3(y ~ ., data = mnist_27$train)

x <- as.matrix(mnist_27$train[,2:3])
y <- mnist_27$train$y
knn_fit <- knn3(x, y)

knn_fit <- knn3(y ~ ., data = mnist_27$train, k=5)

y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
##    0.815
```

Over-training and Over-smoothing

There is a link to the relevant sections of the textbook: Over-training and Over-smoothing

Key points

- **Over-training** is the reason that we have higher accuracy in the train set compared to the test set. Over-training is at its worst when we set $k = 1$. With $k = 1$, the estimate for each (x_1, x_2) in the training set is obtained with just the y corresponding to that point.
- When we try a larger k, the k might be so large that it does not permit enough flexibility. We call this **over-smoothing**.
- Note that if we use the test set to pick this k, we should not expect the accompanying accuracy estimate to extrapolate to the real world. This is because even here we broke a golden rule of machine learning: **we selected the k using the test set. Cross validation** also provides an estimate that takes this into account.

Code

```
y_hat_knn <- predict(knn_fit, mnist_27$train, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$train$y)$overall["Accuracy"]
```

```
## Accuracy
##   0.8825
```

```
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
##    0.815
```

```
#fit knn with k=1
knn_fit_1 <- knn3(y ~ ., data = mnist_27$train, k = 1)
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$train, type = "class")
confusionMatrix(data=y_hat_knn_1, reference=mnist_27$train$y)$overall[["Accuracy"]]
```

```
## [1] 0.995
```

```
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$test, type = "class")
confusionMatrix(data=y_hat_knn_1, reference=mnist_27$test$y)$overall[["Accuracy"]]
```

```
## [1] 0.74
```

```
#fit knn with k=401
knn_fit_401 <- knn3(y ~ ., data = mnist_27$train, k = 401)
y_hat_knn_401 <- predict(knn_fit_401, mnist_27$test, type = "class")
confusionMatrix(data=y_hat_knn_401, reference=mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
##     0.79
```

```
#pick the k in knn
ks <- seq(3, 251, 2)
library(purrr)
accuracy <- map_df(ks, function(k){
    fit <- knn3(y ~ ., data = mnist_27$train, k = k)
    y_hat <- predict(fit, mnist_27$train, type = "class")
    cm_train <- confusionMatrix(data = y_hat, reference = mnist_27$train$y)
    train_error <- cm_train$overall["Accuracy"]
    y_hat <- predict(fit, mnist_27$test, type = "class")
    cm_test <- confusionMatrix(data = y_hat, reference = mnist_27$test$y)
    test_error <- cm_test$overall["Accuracy"]

tibble(train = train_error, test = test_error)
})


#pick the k that maximizes accuracy using the estimates built on the test data
ks[which.max(accuracy$test)]
```

```
## [1] 41
```

```
max(accuracy$test)
```

```
## [1] 0.86
```

Comprehension Check - Nearest Neighbors

1. Previously, we used logistic regression to predict sex based on height. Now we are going to use knn to do the same. Set the seed to 1, then use the **caret** package to partition the **dslabs** `heights` data into a training and test set of equal size. Use the `sapply()` function to perform knn with `k` values of `seq(1, 101, 3)` and calculate F1 scores with the `F_meas()` function using the default value of the relevant argument.

What is the max value of `F_1`?

At what value of `k` does the max occur?

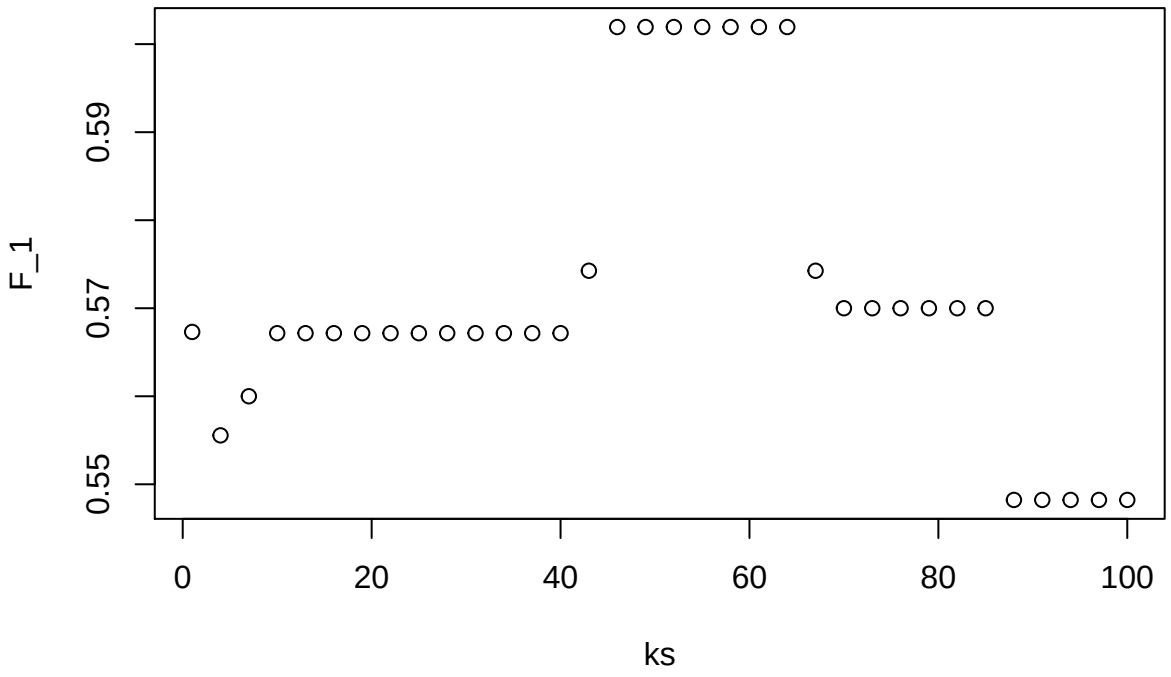
```
data("heights")

# 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(heights$sex, times = 1, p = 0.5, list = FALSE)
test_set <- heights[test_index, ]
train_set <- heights[-test_index, ]

ks <- seq(1, 101, 3)
F_1 <- sapply(ks, function(k){
    fit <- knn3(sex ~ height, data = train_set, k = k)
    y_hat <- predict(fit, test_set, type = "class") %>%
        factor(levels = levels(train_set$sex))
    F_meas(data = y_hat, reference = test_set$sex)
})
plot(ks, F_1)
```




```
max(F_1)
```

```
## [1] 0.6019417
```

```
ks[which.max(F_1)]
```

```
## [1] 46
```

2. Next we will use the same gene expression example used in the Comprehension Check: Distance exercises. You can load it like this:

```
library(dslabs)
library(caret)
data("tissue_gene_expression")
```

First, set the seed to 1 and split the data into training and test sets with `p = 0.5`. Then, report the accuracy you obtain from predicting tissue type using KNN with `k = seq(1, 11, 2)` using `sapply()` or `map_df()`. Note: use the `createDataPartition()` function outside of `sapply()` or `map_df()`.

```
# 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
```

```
y <- tissue_gene_expression$y
x <- tissue_gene_expression$x
test_index <- createDataPartition(y, list = FALSE)
sapply(seq(1, 11, 2), function(k){
    fit <- knn3(x[-test_index,], y[-test_index], k = k)
    y_hat <- predict(fit, newdata = data.frame(x=x[test_index,]),
                type = "class")
mean(y_hat == y[test_index])
})
```

```
## [1] 0.9895833 0.9687500 0.9479167 0.9166667 0.9166667 0.9062500
```

K-fold cross validation

There is a link to the relevant section of the textbook: K-fold cross validation

Key points

- For k**-fold cross validation**, we divide the dataset into a training set and a test set. We train our algorithm exclusively on the training set and use the test set only for evaluation purposes.
- For each set of algorithm parameters being considered, we want an **estimate of the MSE and then we will choose the parameters with the smallest MSE**. In k-fold cross validation, we randomly split the observations into k non-overlapping sets, and repeat the calculation for MSE for each of these sets. Then, we compute the average MSE and obtain an estimate of our loss. Finally, we can select the optimal parameter that minimized the MSE.
- In terms of how to select k for cross validation, **larger values of** k **are preferable but they will also take much more** computational time. For this reason, the choices of $k = 5$ and $k = 10$ are common.

Comprehension Check - Cross-validation

1. Generate a set of random predictors and outcomes using the following code:

```
# set.seed(1996) #if you are using R 3.5 or earlier
set.seed(1996, sample.kind="Rounding") #if you are using R 3.6 or later
```

```
## Warning in set.seed(1996, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
n <- 1000
p <- 10000
x <- matrix(rnorm(n*p), n, p)
colnames(x) <- paste("x", 1:ncol(x), sep = "_")
y <- rbinom(n, 1, 0.5) %>% factor()

x_subset <- x[ ,sample(p, 100)]
```

Because `x` and `y` are completely independent, you should not be able to predict `y` using `x` with accuracy greater than 0.5. Confirm this by running cross-validation using logistic regression to fit the model. Because we have so many predictors, we selected a random sample `x_subset`. Use the subset when training the model.

Which code correctly performs this cross-validation?

```
fit <- train(x_subset, y, method = "glm")
fit$results
```

```
##   parameter  Accuracy      Kappa AccuracySD    KappaSD
## 1      none 0.5078406 0.01318925 0.02336971 0.04626366
```

☐ A.

```
fit <- train(x_subset, y)
fit$results
```

☒ B.

```
fit <- train(x_subset, y, method = "glm")
fit$results
```

☐ C.

```
fit <- train(y, x_subset, method = "glm")
fit$results
```

☐ D.

```
fit <- test(x_subset, y, method = "glm")
fit$results
```

2. Now, instead of using a random selection of predictors, we are going to search for those that are most predictive of the outcome. We can do this by comparing the values for the $y = 1$ group to those in the $y = 0$ group, for each predictor, using a t-test. You can do perform this step like this:

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

```
## Loading required package: BiocManager
```

```
## Bioconductor version 3.11 (BiocManager 1.30.10), ?BiocManager::install for help
```

```
BiocManager::install("genefilter")
```

```
## Bioconductor version 3.11 (BiocManager 1.30.10), R 4.0.2 (2020-06-22)
```

```
## Installing package(s) 'genefilter'
```

```
##
## The downloaded binary packages are in
##  /var/folders/6m/nz2p76pn679b692c99t644bm0000gn/T//RtmpgmksPi/downloaded_packages
```

```
library(genefilter)
```

```
##
## Attaching package: 'genefilter'
```

```
## The following objects are masked from 'package:matrixStats':
##
##     rowSds, rowVars
```

```
## The following object is masked from 'package:MASS':
##
##     area
```

```
## The following object is masked from 'package:readr':
##
##     spec
```

```
tt <- colttests(x, y)
```

Which of the following lines of code correctly creates a vector of the p-values called pvals?

```
pvals <- tt$p.value
```

- ☐ A. `pvals <- tt$dm`
- ☐ B. `pvals <- tt$statistic`
- ☐ C. `pvals <- tt`
- ☒ D. `pvals <- tt$p.value`

3. Create an index `ind` with the column numbers of the predictors that were "statistically significantly" associated with `y`. Use a p-value cutoff of 0.01 to define "statistically significantly."

How many predictors survive this cutoff?

```
ind <- which(pvals <= 0.01)
length(ind)
```

```
## [1] 108
```

4. Now re-run the cross-validation after redefinining `x_subset` to be the subset of `x` defined by the columns showing "statistically significant" association with `y`.

What is the accuracy now?

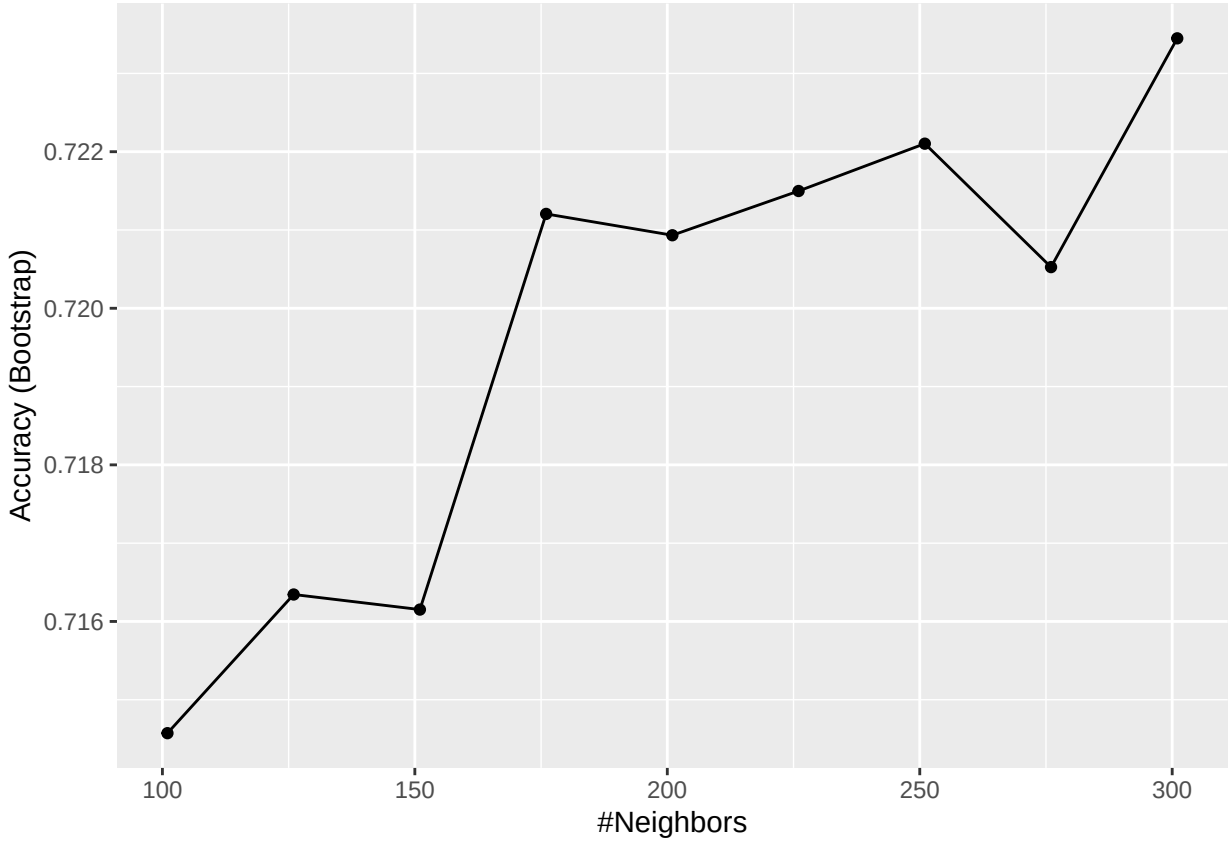
```
x_subset <- x[,ind]
fit <- train(x_subset, y, method = "glm")
fit$results
```

```
##   parameter  Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.7571395 0.5134142 0.01922097 0.03805696
```

5. Re-run the cross-validation again, but this time using kNN. Try out the following grid `k = seq(101, 301, 25)` of tuning parameters. Make a plot of the resulting accuracies.

Which code is correct?

```
fit <- train(x_subset, y, method = "knn", tuneGrid = data.frame(k = seq(101, 301, 25)))
ggplot(fit)
```



☒ A.

```
fit <- train(x_subset, y, method = "knn", tuneGrid = data.frame(k = seq(101, 301, 25)))
ggplot(fit)
```

☐ B.

```
fit <- train(x_subset, y, method = "knn")
ggplot(fit)
```

☐ C.

```
fit <- train(x_subset, y, method = "knn", tuneGrid = data.frame(k = seq(103, 301, 25)))
ggplot(fit)
```

☐ D.

```
fit <- train(x_subset, y, method = "knn", tuneGrid = data.frame(k = seq(101, 301, 5)))
ggplot(fit)
```

6. In the previous exercises, we see that despite the fact that `x` and `y` are completely independent, we were able to predict `y` with accuracy higher than 70%. We must be doing something wrong then.

What is it?

☐ A. The function `train()` estimates accuracy on the same data it uses to train the algorithm.
☐ B. We are overfitting the model by including 100 predictors.
☒ C. We used the entire dataset to select the columns used in the model.
☐ D. The high accuracy is just due to random variability.