Homework 12

Ethan Chang - ehc586

Please submit as a knitted HTML file on Canvas before the due date!

For all questions, include the R commands/functions that you used to find your answer. Answers without supporting code will not receive credit.

> **Review of how to submit this assignment** All homework assignments will be completed using R Markdown. These `.Rmd` files consist of text/syntax (formatted using Markdown) alongside embedded R code. When you have completed the assignment (by adding R code inside codeblocks and supporting text outside of the codeblocks), create your document as follows:

> - Click the arrow next to the "Knit" button (above)
> - Choose "Knit to HTML" and wait; fix any errors if applicable
> - Go to Files pane and put checkmark next to the correct HTML file
> - Click on the blue gear icon ("More") and click Export
> - Download the file and then upload to Canvas

For this homework, we will perform principal component analysis (PCA) on the famous *MNIST* dataset. The MNIST dataset contains handwritten digits on a 28-by-28 grid (784 pixels total) with the intensity of the pixels between 0 and 255.

First, we load the data and requisite packages. This might take some time because you need to download the data; feel free to save the data once downloaded so that you don't need to keep downloading it.

```
library(tidyverse)
library(randomForest)

## Loading the data
mnist_url <- str_c("https://github.com/cerndb/dist-keras/blob/master/examples/",
                   "data/mnist.csv?raw=true")
mnist_raw <- read_csv(mnist_url, col_names = TRUE)
```

NOTE: This file won't knit properly until you have completed the whole assignment. Just run things inside RStudio until you finish the assignment and then knit at the end.

Question 1 (1pt)

We will only be interested in classifying the digits 0 and 1. **Filter the dataset down to only the labels 0 and 1 and store this dataset as `mnist_01`.**

```
mnist_01 <- mnist_raw %>% filter(label <= 1)
```

Question 2 (1pt)

The following code takes a vector of length 784 and converts it to an image using the `geom_contour_filled` function:

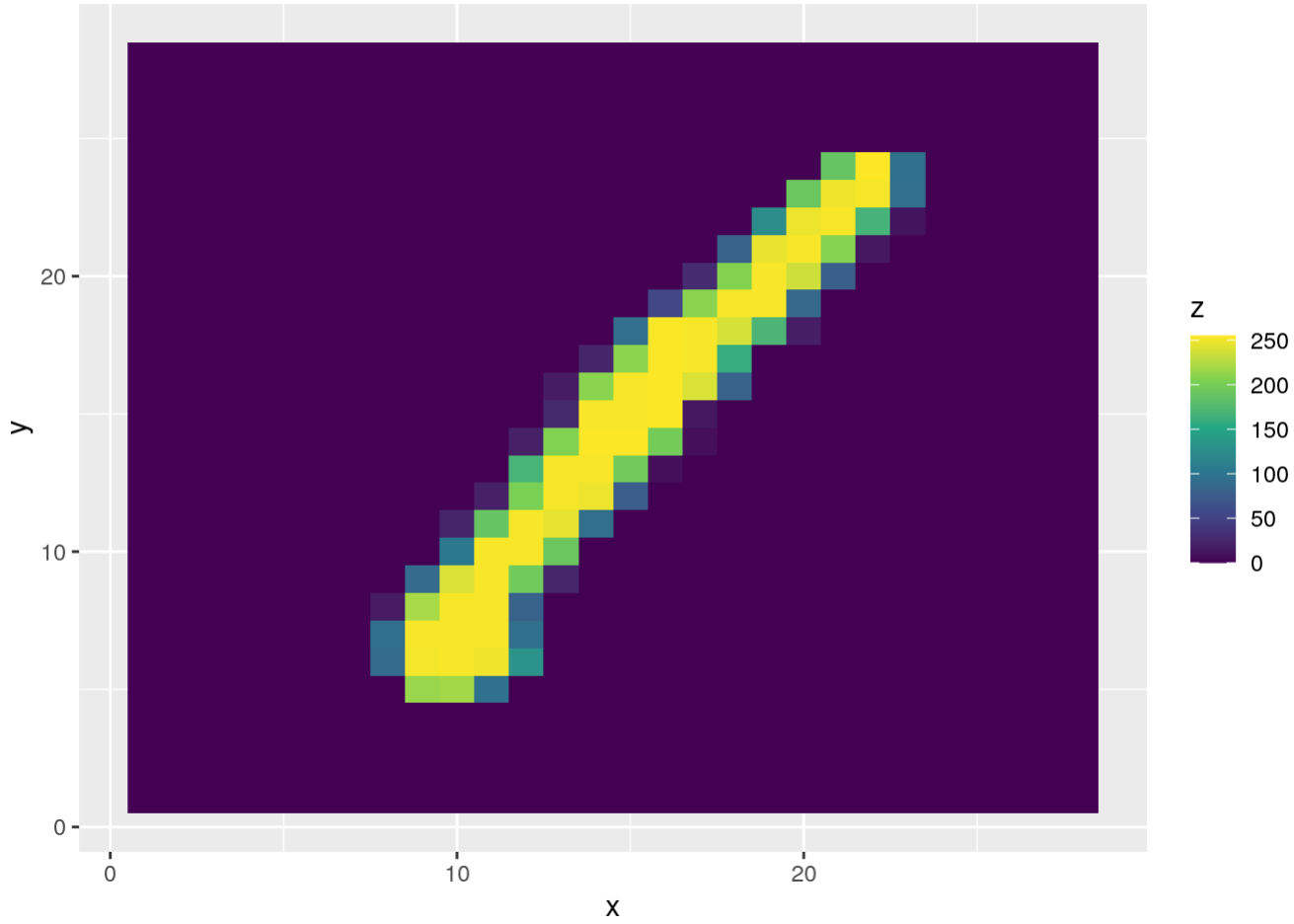
```
draw_image <- function(digit) {
  x <- rep(1:28, times = 28)
  y <- rep(28:1, each = 28)
  df <- data.frame(x = x, y = y, z = digit)
  ggplot(df, aes(x = x, y = y, fill = z)) +
    geom_raster() +
    scale_fill_viridis_c()
}
```

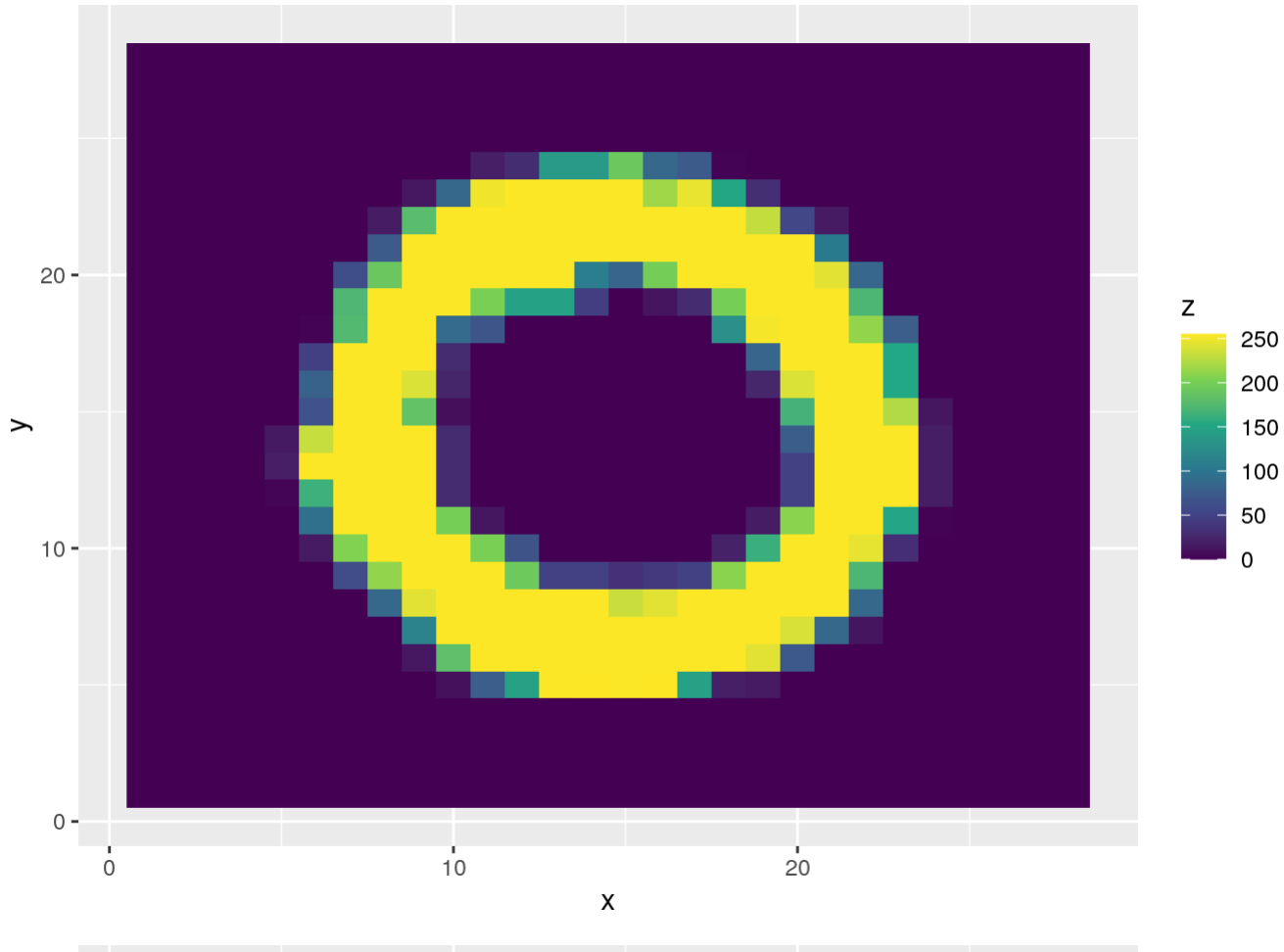
The following code will grab pixel intensities and labels:

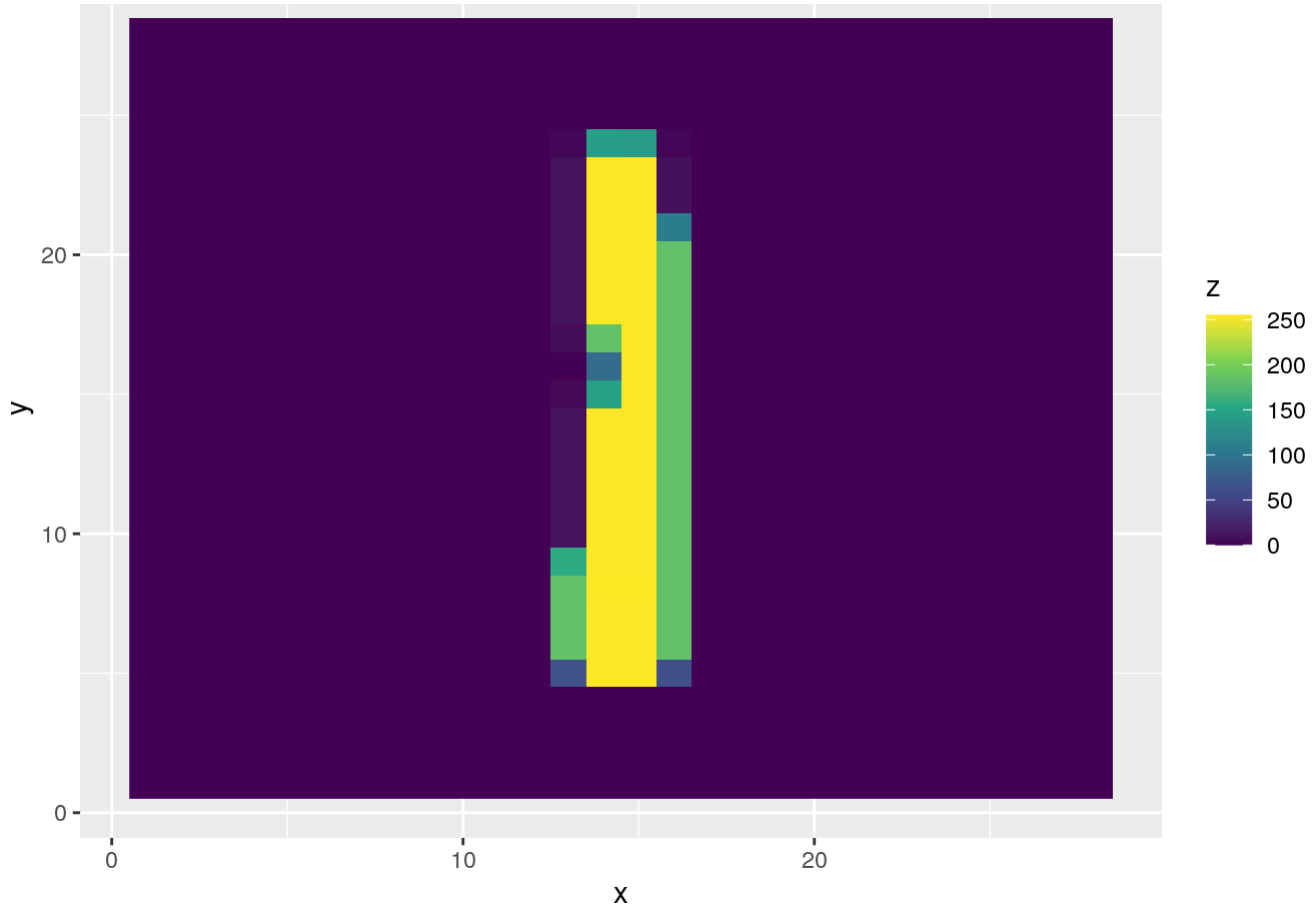
```
intensities <- mnist_01 %>% dplyr::select(-label) %>% as.matrix()
labels <- mnist_01 %>% pull(label)
```

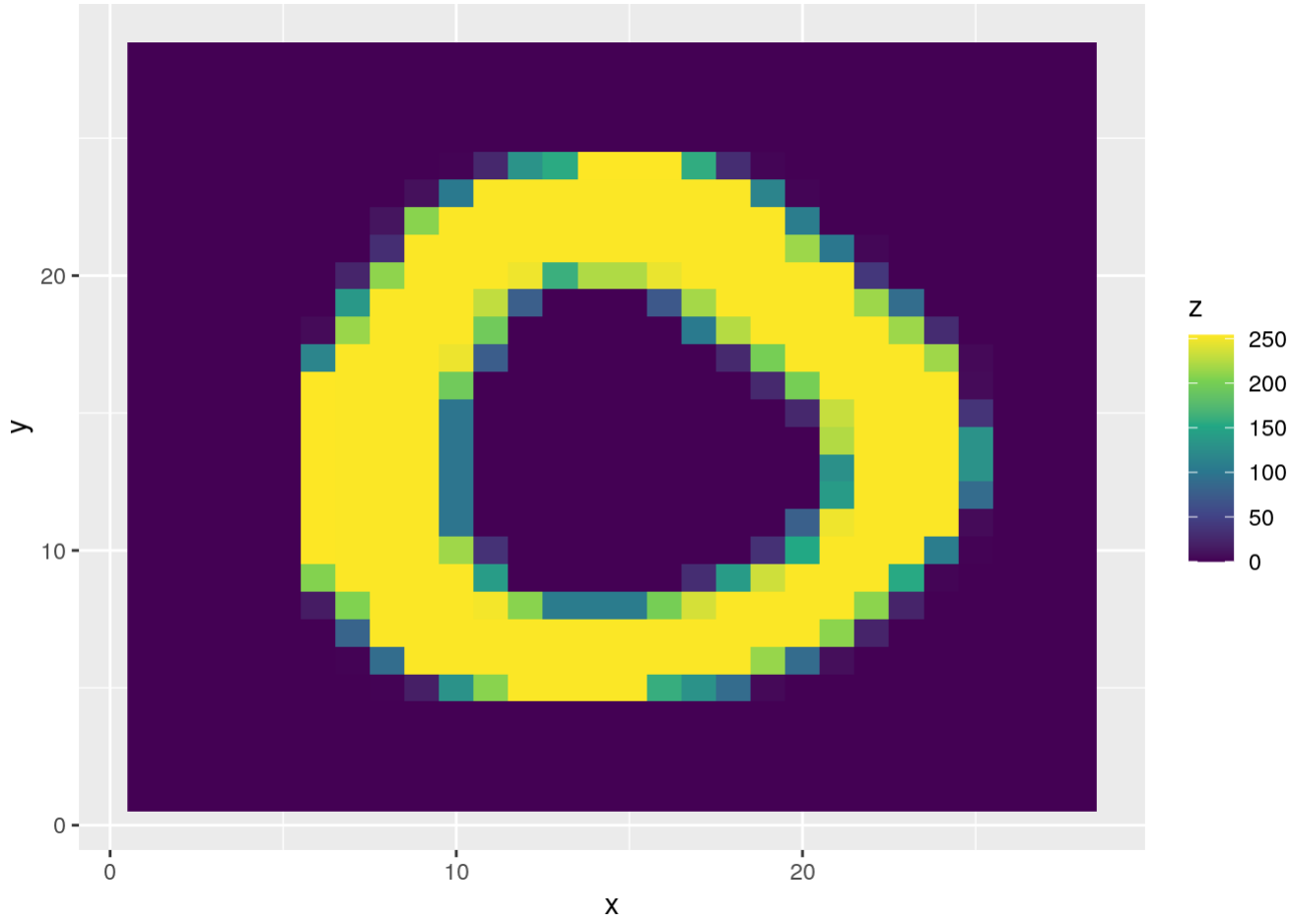
Use the `draw_image()` function to make images for the first 5 rows of the data using the pixel intensities. Can you tell the difference between the 0s and 1s?

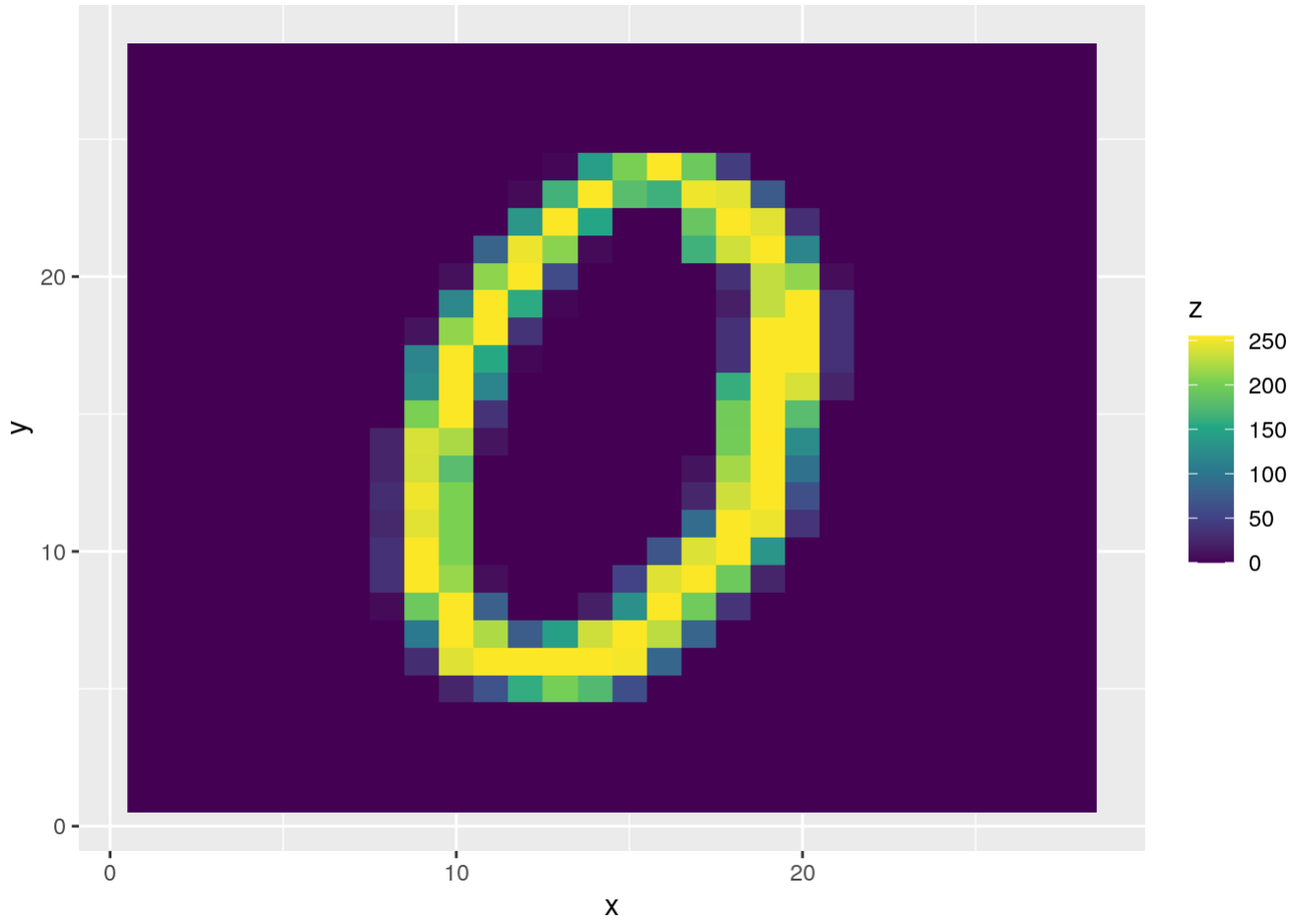
```
## Your code here
for (row in 1:5){
  intensities[row,] %>% draw_image() %>% print()
}
```


20
10
0
y
0
10
20
x
z
250
200
150
100
50
0


20
10
0
y
0
10
20
x
z
250
200
150
100
50
0


20
10
0
y
0
10
20
x
z
250
200
150
100
50
0


20
10
0
y
0
10
20
x
z
250
200
150
100
50
0


20
10
0
y
0
10
20
x
z
250
200
150
100
50
0

Answer: Yes I can tell the difference between the 0s and 1s. It goes 1 0 1 0 0.

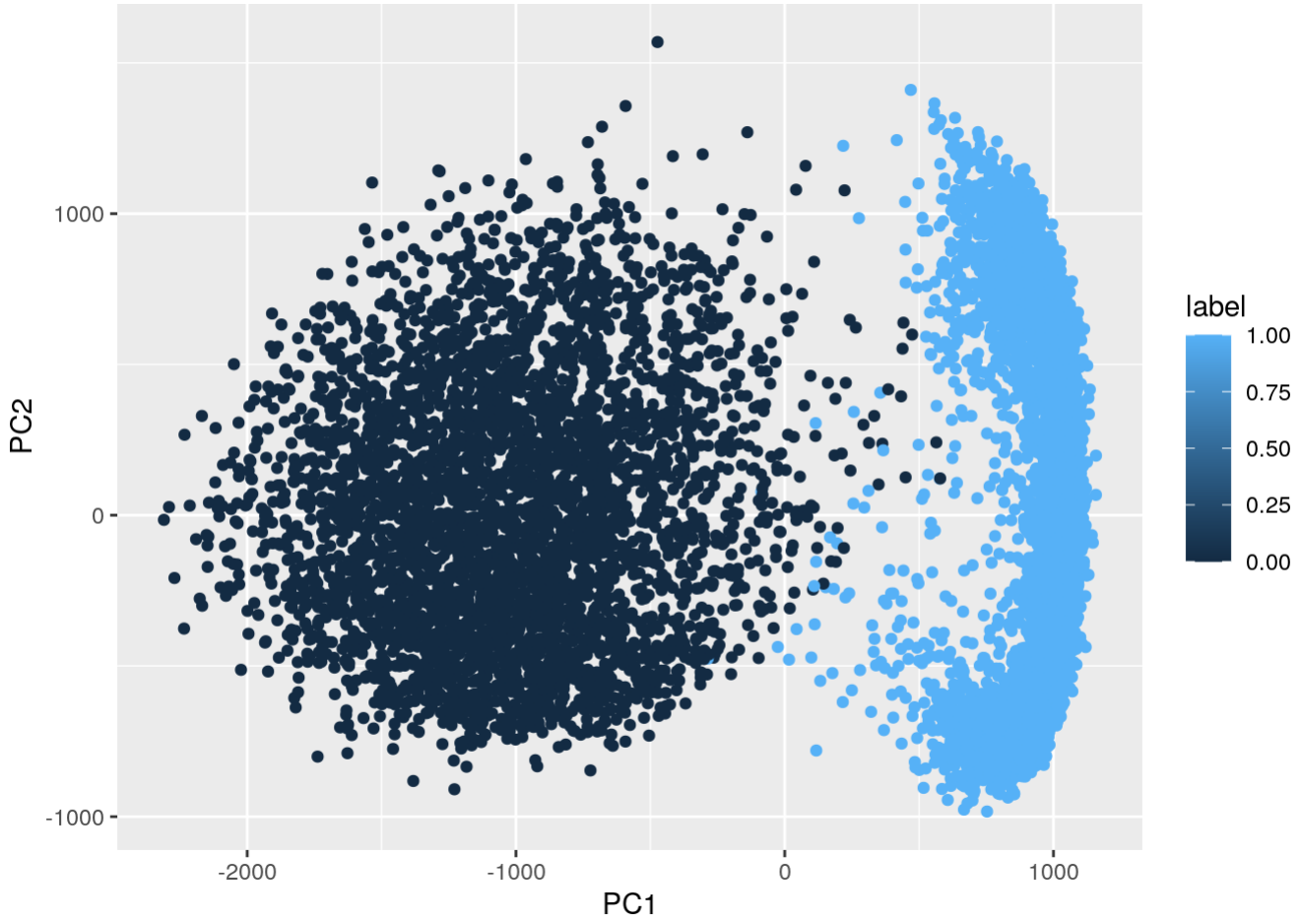
Question 3 (2pts)

Let's now compute the principal components of `intensities` using `prcomop` .

```
pca_mnist <- intensities %>% prcomp()
pcs <- data.frame(pca_mnist$x) %>% mutate(label = labels)
```

Then plot the first principal component against the second principal component and color the points according to `label` . **Do either of the two principal components do a good job of distinguishing the 0s and 1s? Explain.**

```
## Your code here
ggplot(pcs, aes(x=PC1, y=PC2, color=label)) + geom_point()
```

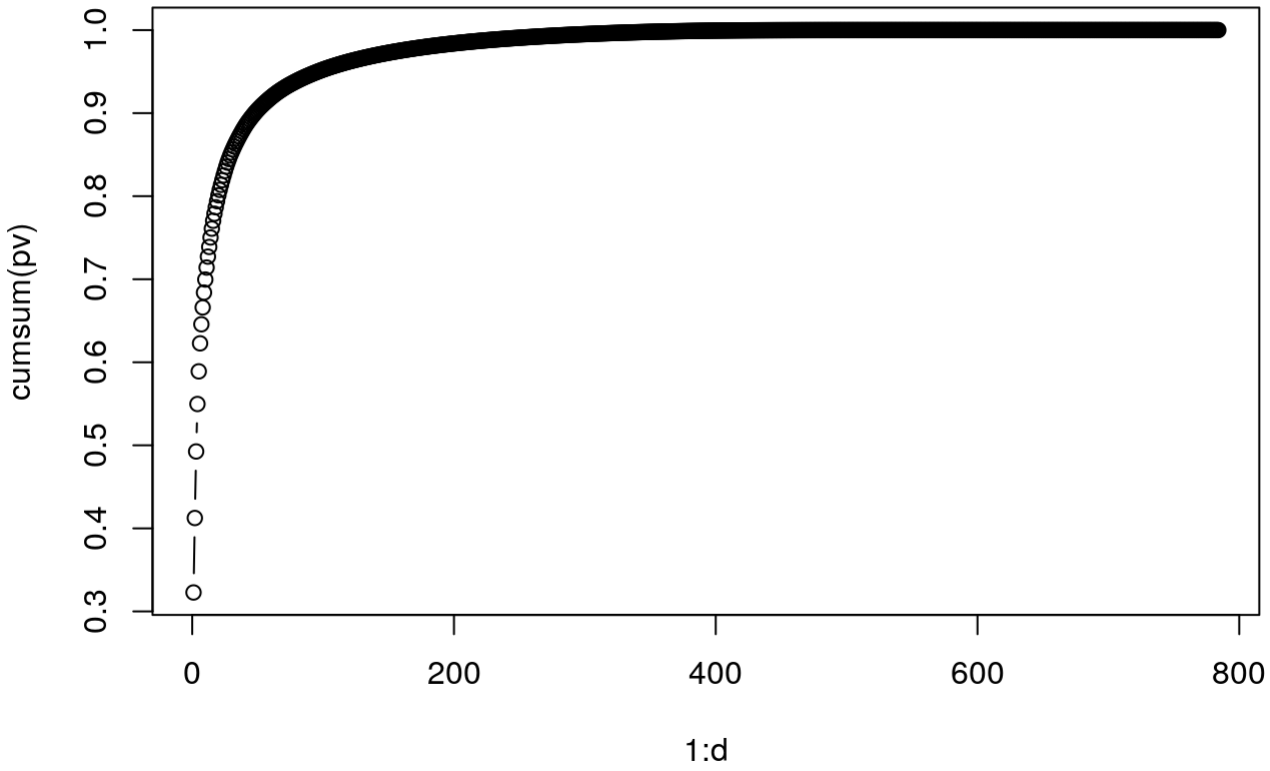


Answer: Based on the above plot, it can be seen that PC1 does a good job of distinguishing the 0s and 1s. In general, it can be seen that a positive PC1 is associated with the 1s label while a negative PC1 is associated with the 0s label.

Question 4 (1pt)

Plot the proportiion of variance explained by the first `d` principal components against `d` for `d = 1, ..., 784` . How many principal components are needed to account for 90% of the variability in the data?

```
## Your code here
pv <- pca_mnist$sdev^2 / sum(pca_mnist$sdev^2)
d <- nrow(pca_mnist$rotation)
plot(1:d, cumsum(pv), type="b")
```




```
for (i in 1:length(pv)){
  if (cumsum(pv)[i] > 0.9){
    break
  }
}
cat(i, "principal components for 90% of variability in data")
```

```
## 48 principal components for 90% of variability in data
```

Answer: From the above plot, it can be seen that 48 principal components are needed to account for 90% of the variability in the data.

Question 5 (2pts)

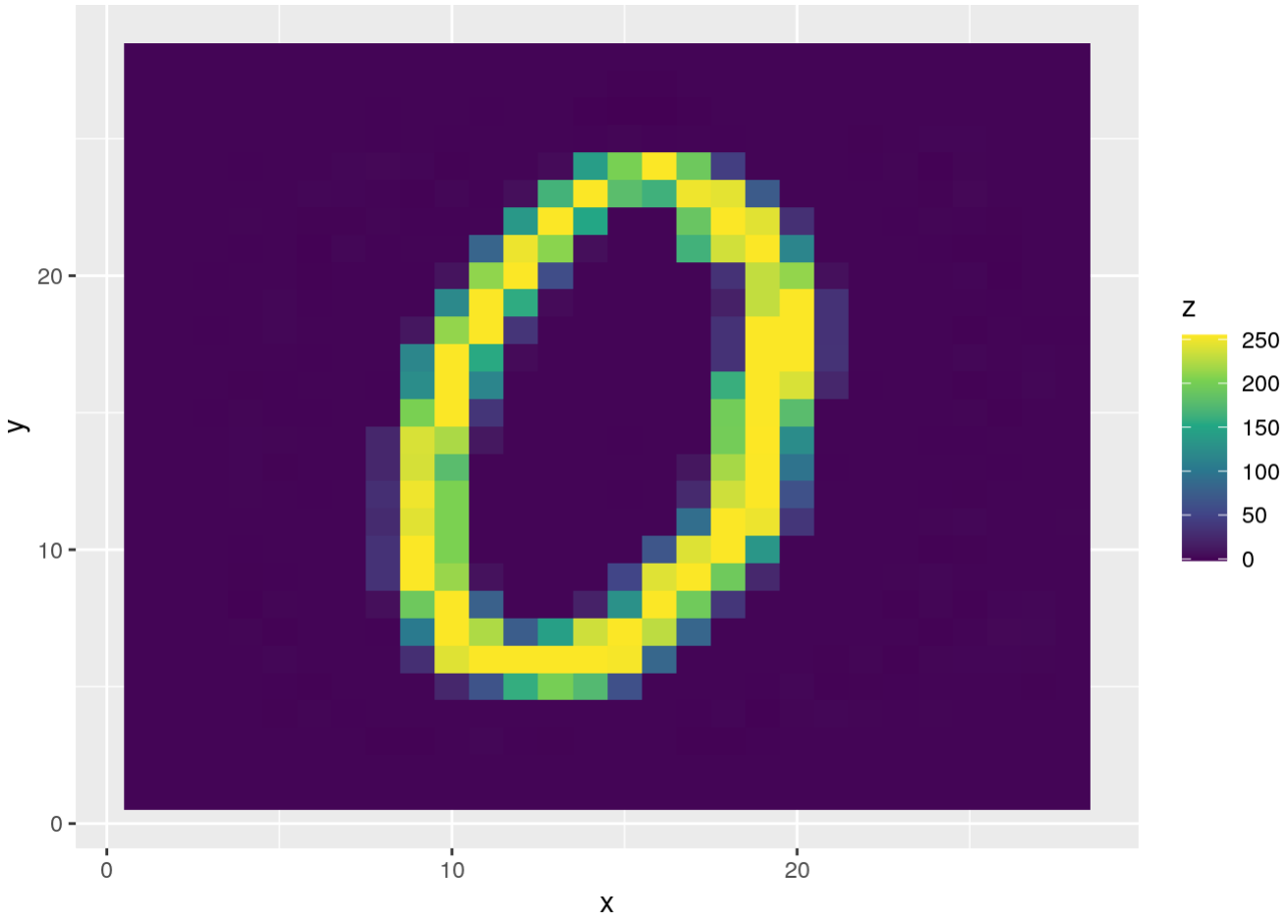
One interesting application of PCA is to perform *image compression* - if most of the data can be reduced to (say) 50 principal components then we no longer need to store the whole 28-by-28 image. The following code, it turns out, will do the compression for us:

```
mean_intensities <- colMeans(intensities)
compress_image <- function(row, num_components) {
  G <- pca_mnist$rotation[,1:num_components]
  compressed <- G %*% t(G) %*% (intensities[row,] - mean_intensities) +
    mean_intensities
  return(compressed)
}
```

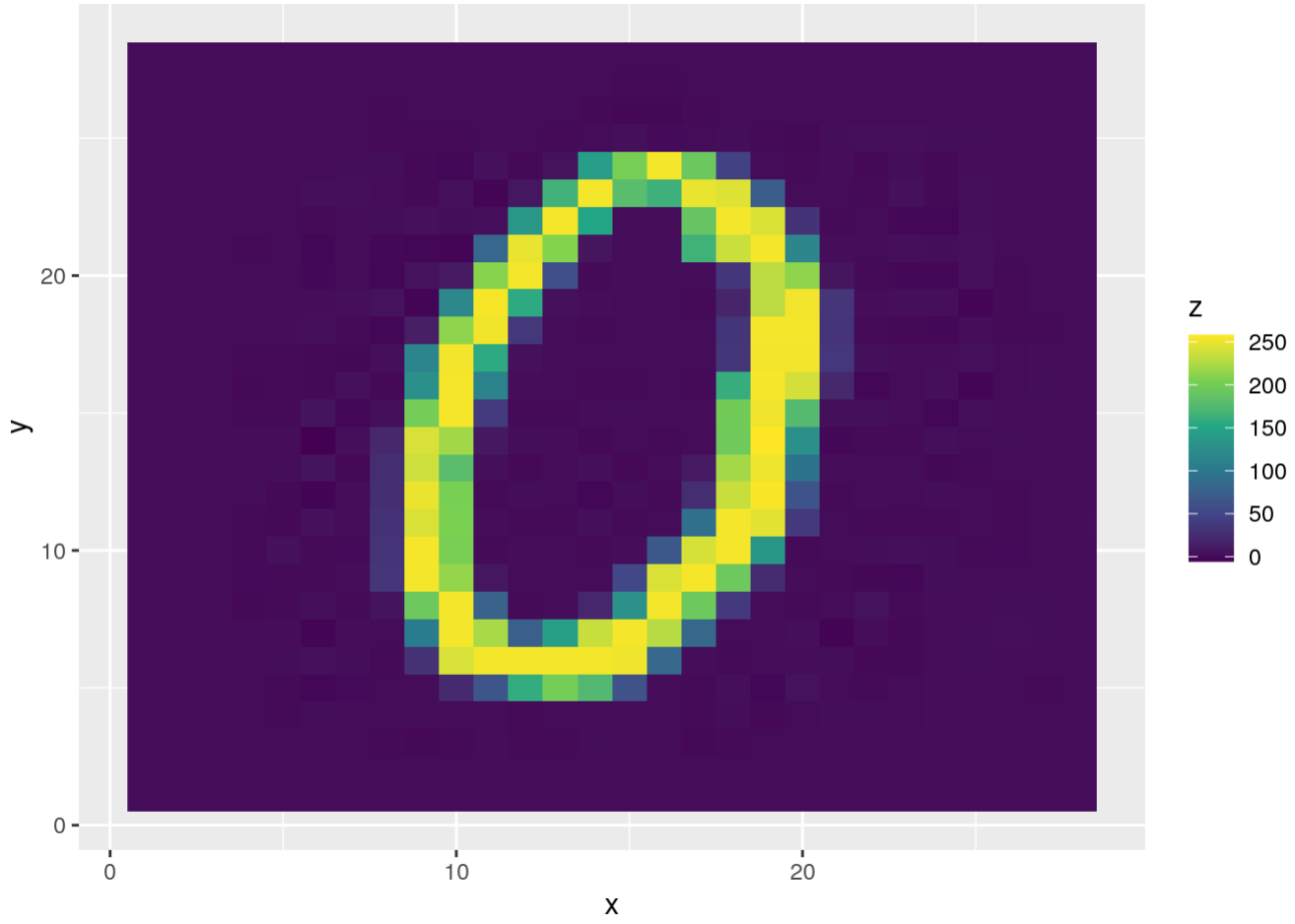
The argument `row` tells us which image we want to compress while `num_components` tells us how many PCs we shoould use to describe the image (for example, if `num_components = 1` then we are reducing the image all the way down to a single number, whereas if `num_components = 28^2` then we are not reducing the data at all).

Play with `compress_image`, feeding the results into `draw_image` to visualize; for concreteness, print 5 of the images you make. How many principal components do you need to do a good job at compressing? No hard answer here, just give your personal opinion about how many you think yuo need to do a good job.

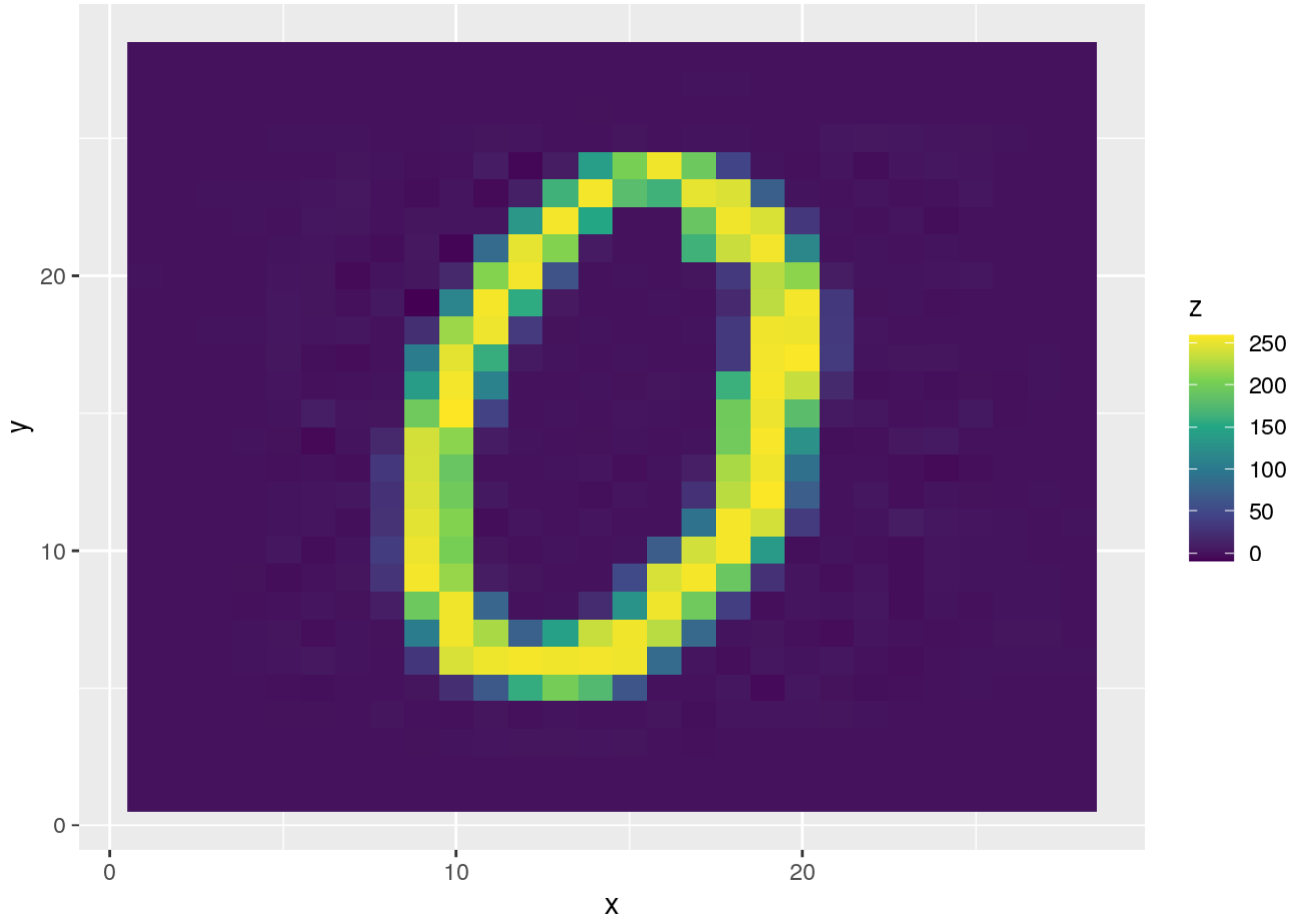
```
## Your code here
compress_image(5,20^2) %>% draw_image()
```



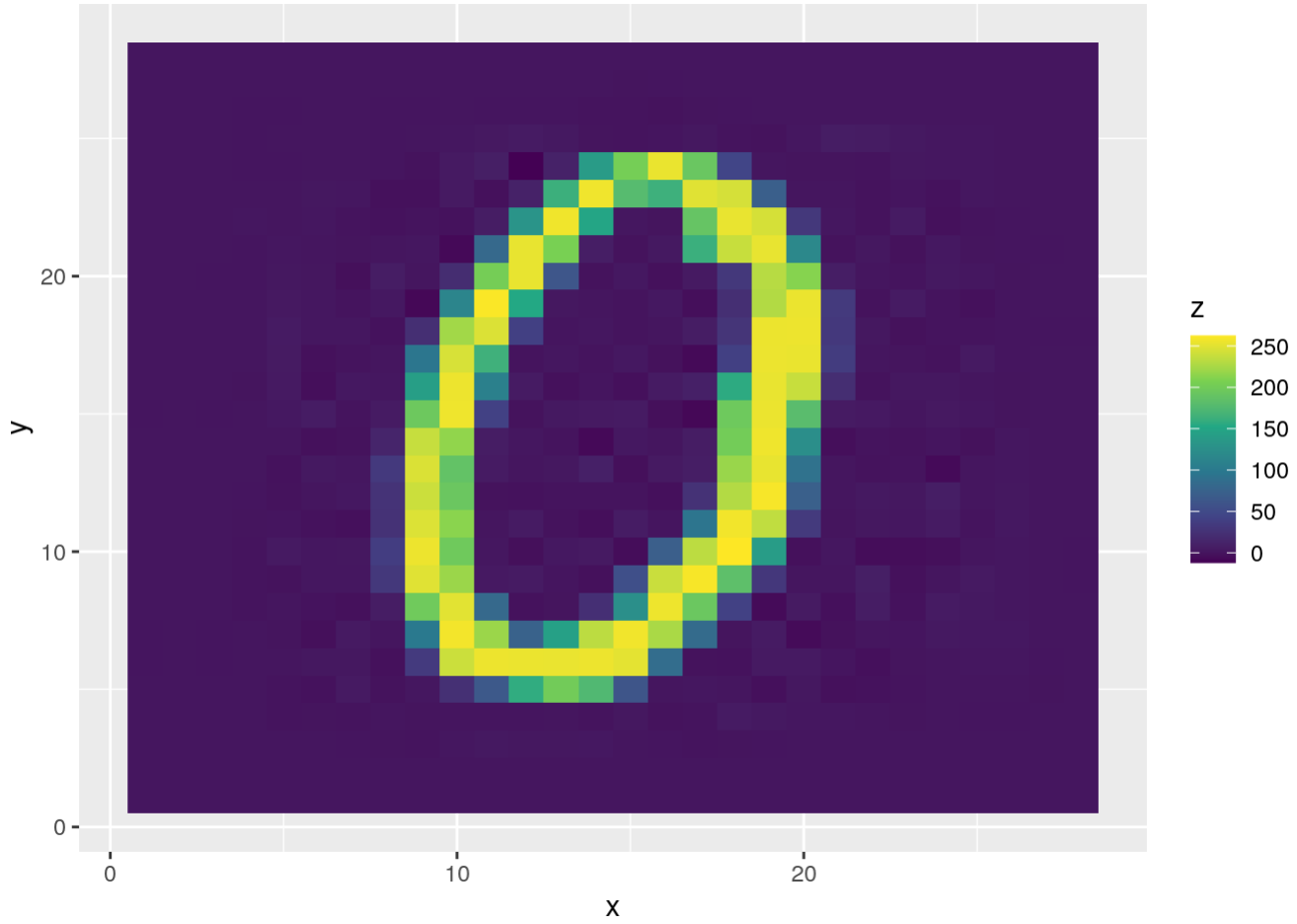

```
compress_image(5,19^2) %>% draw_image()
```



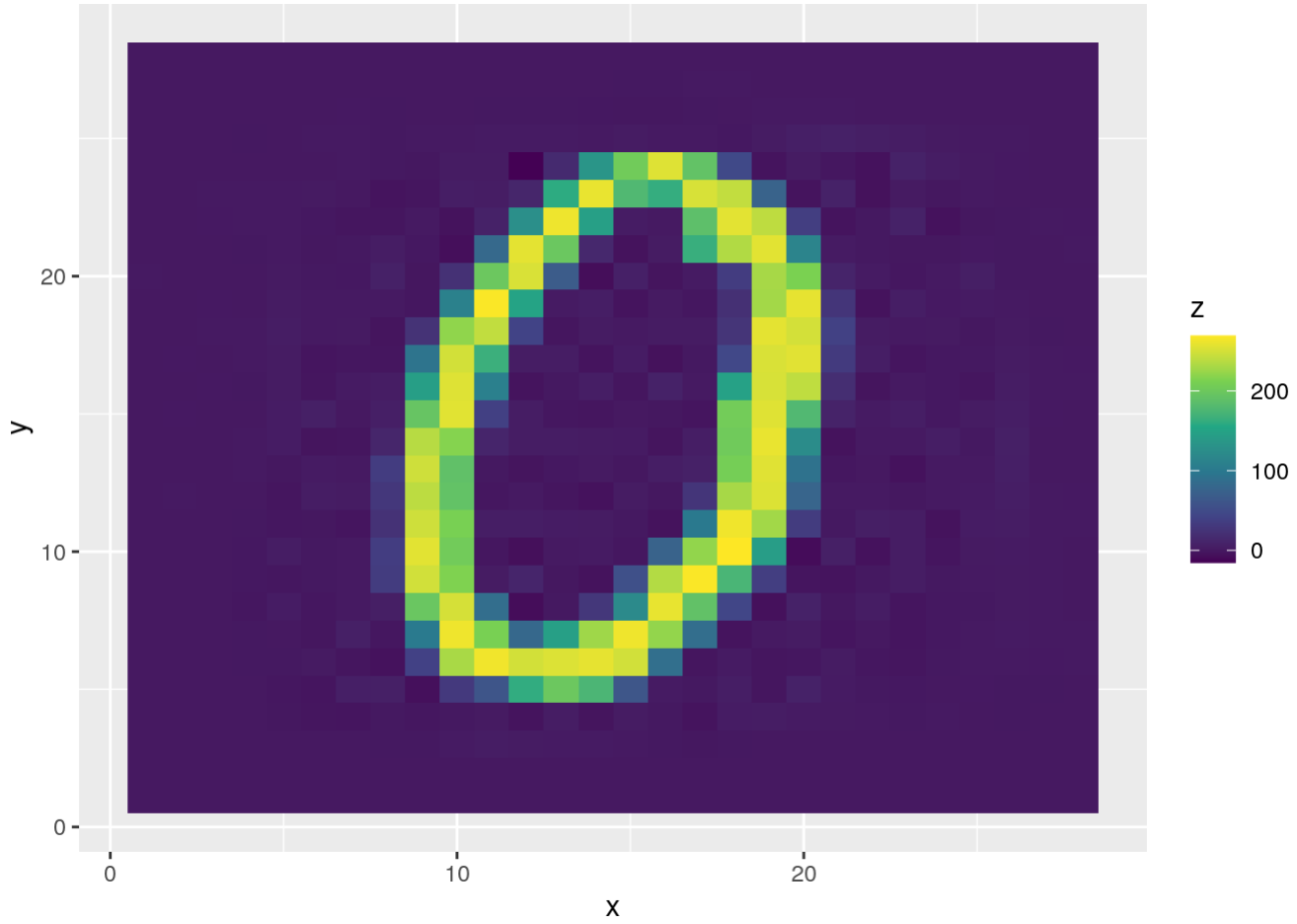

```
compress_image(5,18^2) %>% draw_image()
```




```
compress_image(5,17^2) %>% draw_image()
```




```
compress_image(5,16^2) %>% draw_image()
```



Answer: In my opinion, I would say that we need 20^2 principal components to do a good job of compressing the data. If we decrease it any more, we would start to introduce “noise” to the images and they become less clear. 20^2 is the smallest number that retains the same quality as the original 28^2 image.

Question 6 (2pts)

The neat thing about what we see in the picture in Question 3 is that the principal components were able to group the 0s and 1s together *without even knowing the labels!* this suggests that we could use the principal components to predict whether a digit is 0 or 1.

The following code divides the data into training and testing sets for you to use:

```
set.seed(1239812)

idx_train <- sample(1:nrow(mnist_01), size = floor(nrow(mnist_01) * .7))
mnist_pca_train <- pcs[idx_train,]
mnist_pca_test  <- pcs[-idx_train,]
```

So, for example, to evaluate the model for the first two PCs, we could do the following:

```
glm_2 <- glm(label ~ .,
  family = binomial,
  data = mnist_pca_train %>% dplyr::select(label, PC1:PC2)
)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
predictions_2 <- ifelse(
  predict(glm_2, mnist_pca_test, type = 'response') > 0.5, 1, 0)

mean(predictions_2 == mnist_pca_test$label)
```

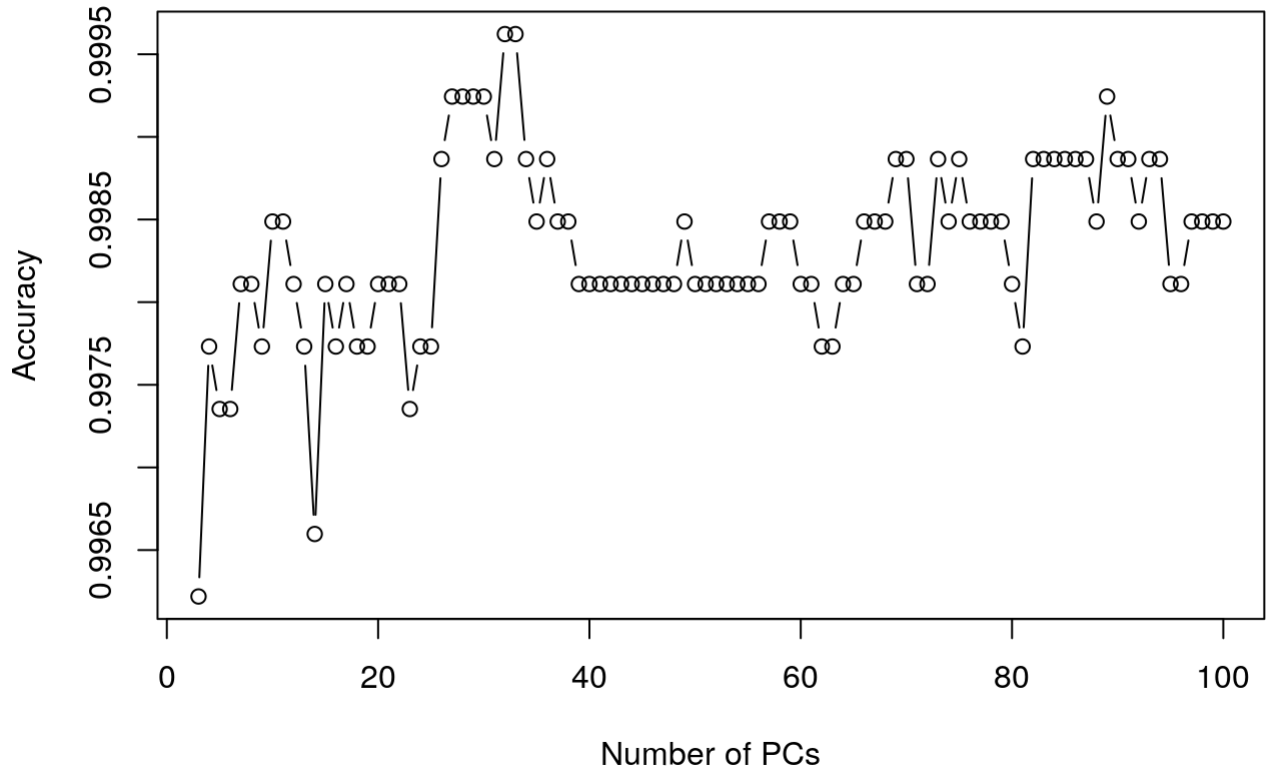
```
## [1] 0.9965974
```

Repeat this analysis, varying the number of PCs from 3 to 100, and record the test-set accuracy of each number of PCs. Which number of PCs results in the best accuracy on the test set? Plot the accuracy against the number of PCs as well.

```
## Your code here
mnist_pca_colnames <- mnist_pca_train %>% colnames()
test_accuracy <- c()

for (i in 3:100){
  glm_2 <- glm(label ~ .,
    family = binomial,
    data = mnist_pca_train %>% dplyr::select(label, PC1:mnist_pca_colnames[i])
  )

  predictions_2 <- ifelse(
    predict(glm_2, mnist_pca_test, type = 'response') > 0.5, 1, 0)

  test_accuracy[i-2] <- mean(predictions_2 == mnist_pca_test$label)
}

which(test_accuracy %in% max(test_accuracy)) + 2
```

```
## [1] 32 33
```

```
plot(3:100, test_accuracy, xlab="Number of PCs", ylab="Accuracy", type="b")
```



Answer: 32 and 33 PCs both result in the best accuracy on the test set.

Question 7 (1pt)

Instead of using logistic regression, use a random forest with 50 PCs. How does the result compare to logistic regression? Use the same train/test split.

```
## Your code here
rf <- randomForest(label ~ ., mnist_pca_train %>% dplyr::select(label, PC1:PC50) %>% mutate(labe
l = as.factor(label)))

predictions_rf <- predict(rf, mnist_pca_test)

mean(predictions_rf == mnist_pca_test$label)
```

```
## [1] 0.9988658
```

```
test_accuracy[50-2]
```

```
## [1] 0.9981096
```

Answer: When using random forest with 50 PCs, the accuracy was 0.9988658, whereas the accuracy when using logistic regression was 0.9981096. Overall, the accuracy for both methods with 50 PCs were extraordinarily close, with the accuracy for random forest being slightly higher by 0.0007562.