Introduction to Regression Trees

Jaime Davila

4/20/2021

Goal: divide the data into chunks and then take the average of elements in a partition.

Introduction

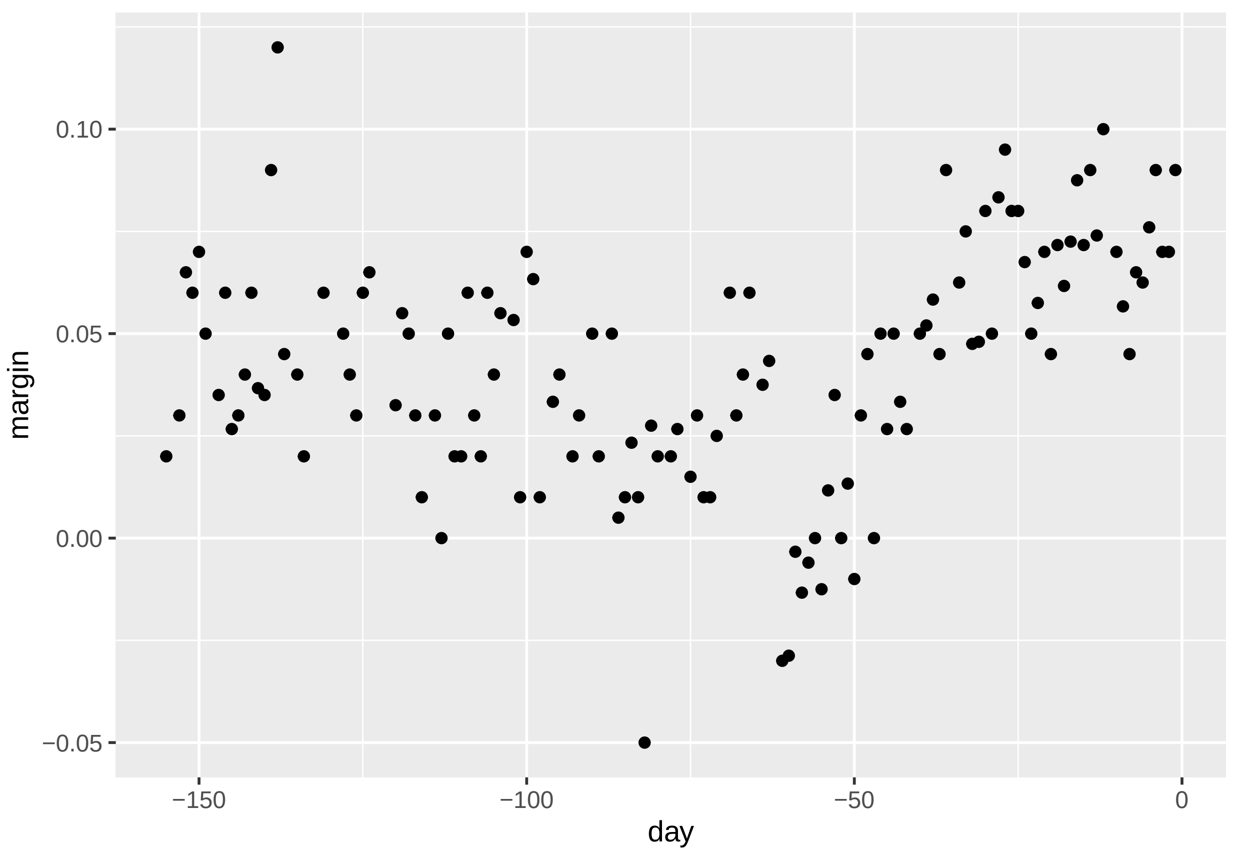
Today we will be using data from the presidential polls for the 2008 election (Obama vs McCain). Let's start by loading the dataset

```
library(dslabs)
data("polls_2008")
polls.2008.tbl <- tibble(polls_2008)
polls.2008.tbl
```

```
## # A tibble: 131 x 2
##      day margin
##    <dbl>  <dbl>
##  1  -155 0.0200
##  2  -153 0.0300
##  3  -152 0.065
##  4  -151 0.06
##  5  -150 0.07
##  6  -149 0.05
##  7  -147 0.035
##  8  -146 0.06
##  9  -145 0.0267
## 10  -144 0.0300
## # ... with 121 more rows
```

Notice that we only have two variables, the first one is `day` which measures the day until election day (day 0 is election night) and `margin` which is the average difference margin between Obama and McCain for that day. We can plot our data by doing

```
ggplot(polls.2008.tbl, aes(day, margin))+
  geom_point()
```



Using regression trees

We are interested in finding the **trend** of the margin using the day as our input variable. In particular we will be assuming that _____________ the trend for a period of days will be constant, so using a regression tree seems like the natural choice. (take average among a period of a few days = how to create chunks) So without further do, let's implement our usual steps using `tidymodels()`

- We define our testing/training dataset:

```
set.seed(123)
poll.split <- initial_split(polls.2008.tbl)
poll.train.tbl <- training(poll.split)
poll.test.tbl <- testing(poll.split)
```

- We define our regression tree model. Initially we will settle for `tree_depth` parameter of 2 and since the margin is a continous variable we will be using the `"regression"` mode.

```
poll.model <-
  decision_tree(tree_depth=2) %>%
  set_mode("regression") %>%
  set_engine("rpart")

poll.recipe <- recipe(margin ~ day, data=poll.train.tbl)

poll.wflow <- workflow() %>%
    add_recipe(poll.recipe) %>%
    add_model(poll.model)
```

- We train our model using our training data

```
poll.fit  <- fit(poll.wflow, poll.train.tbl)
```

- And we evaluate our model performance using our testing data

```
poll.final.tbl <- augment(poll.fit, poll.test.tbl)
rmse(poll.final.tbl, margin, .pred)
```

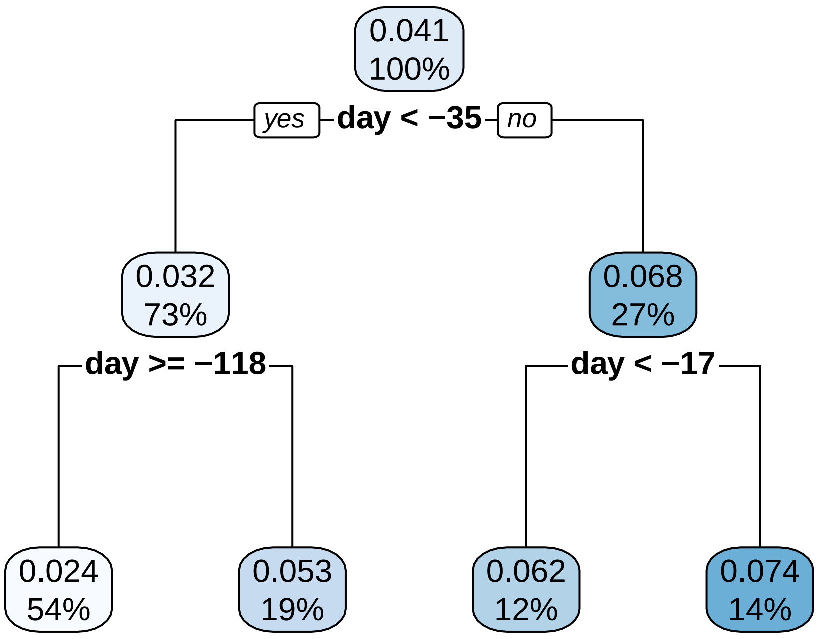
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0274
```

```
rsq(poll.final.tbl, margin, .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rsq     standard       0.220
```

We can visualize our regression tree as a tree

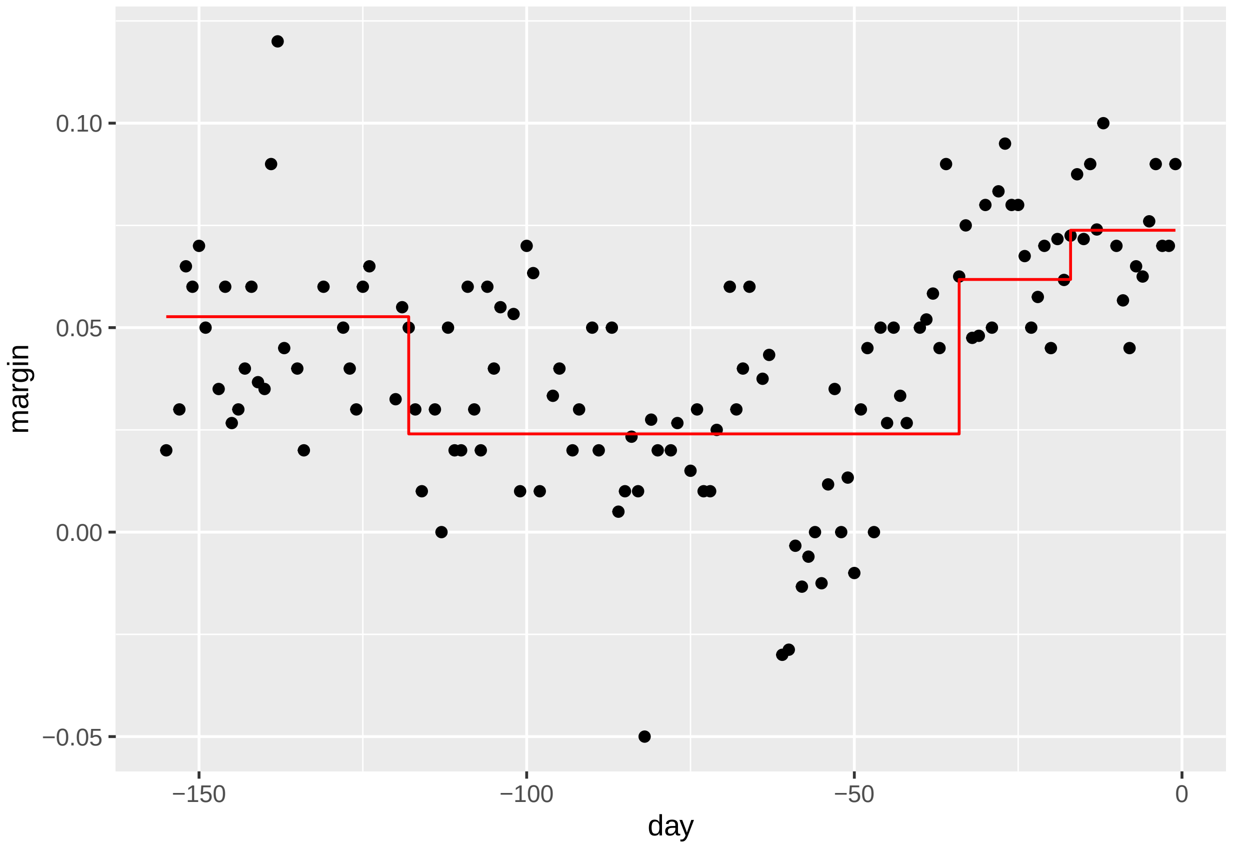
```
poll.fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```



need to review how to interpret this graph.

Or better yet we can see the trend obtained by the regression tree on our original dataset

```
augment(poll.fit, polls.2008.tbl) %>%
  ggplot()+
  geom_point(aes(day,margin))+
  geom_step(aes(day,.pred), col="red")
```



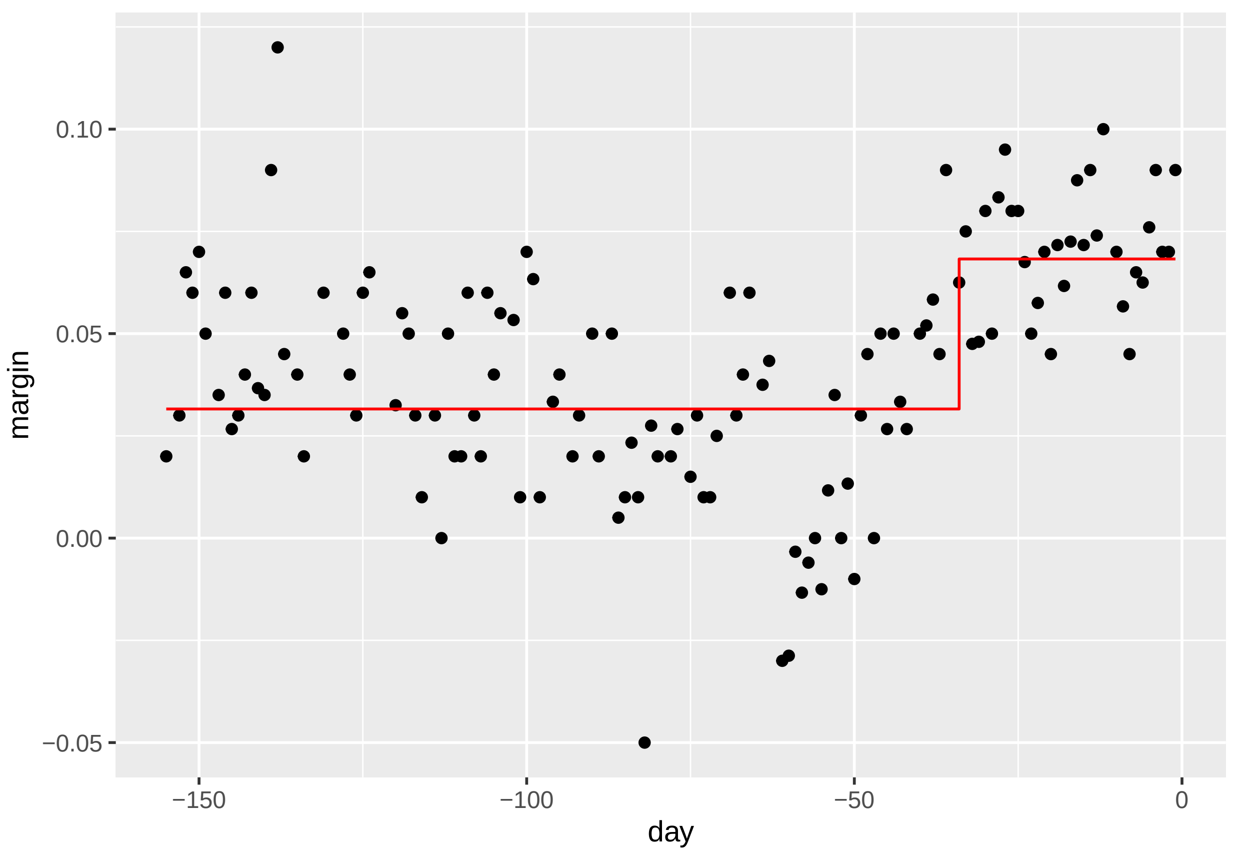
at what point should my tree stop? a large tree is not a good model (for every single observation). This would be overfitting.

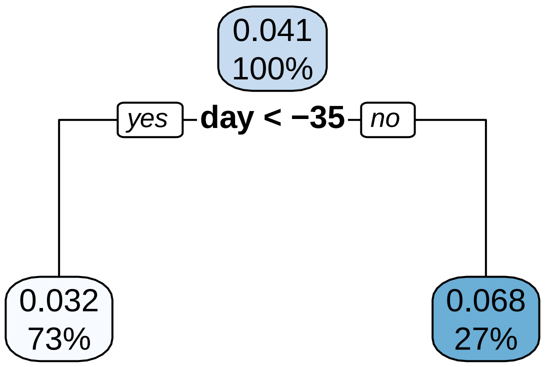
Understanding the parameters of regression trees

In the following exercises we will be exploring the process of the construction of the regression tree and how to optimize the selection of the parameters for our tree model.

1. Fill out the blanks of the function `calc_mse_tree` that receives two parameters, `tree_depth` and `cost_complexity`, creates a regression tree with such parameters and calculates the *mse* on the training data (yes that's correct, the *training* dataset). Test your function using `tree_depth`=1,2, while keeping `cost_complexity`=0.1

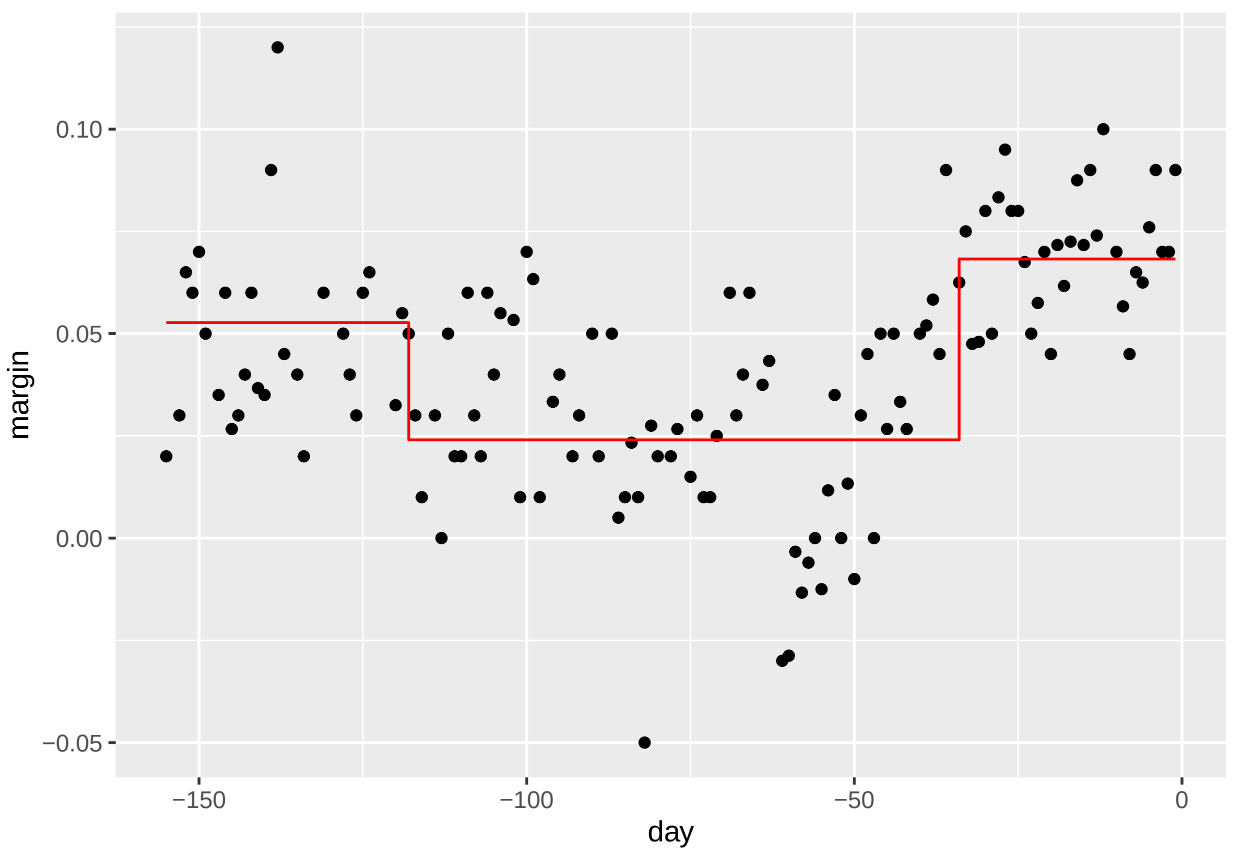
```
calc_mse <- function(tree_depth, cost_complexity) {
  poll.model <- decision_tree(tree_depth=tree_depth, cost_complexity = cost_complexity) %>%
  set_mode("regression") %>%
  set_engine("rpart")

  poll.recipe <- recipe(margin ~ day, data=poll.train.tbl)

  poll.wflow <- workflow() %>%
    add_recipe(poll.recipe) %>%
    add_model(poll.model)

   # Train your model
    poll.fit  <- fit(poll.wflow, poll.train.tbl)

   # Visualize your model
    print(augment(poll.fit, polls.2008.tbl) %>%
    ggplot()+
    geom_point(aes(day,margin))+
    geom_step(aes(day,.pred), col="red"))

    poll.fit %>%
      extract_fit_engine() %>%
      rpart.plot()

# Calculate and output the mse

    #poll.final.tbl <- augment(poll.fit, poll.test.tbl)
    #val <- rmse(poll.final.tbl, margin, .pred)#   mse is rmse^2
    #pull(val[3])^2

#prof:
  rmse <- augment(poll.fit, poll.train.tbl) %>%#why not the test? train is to see how the tree is growi
  rmse(margin, .pred) %>%
  pull(.estimate)
  rmse^2

}

calc_mse(1, 0.1)
```

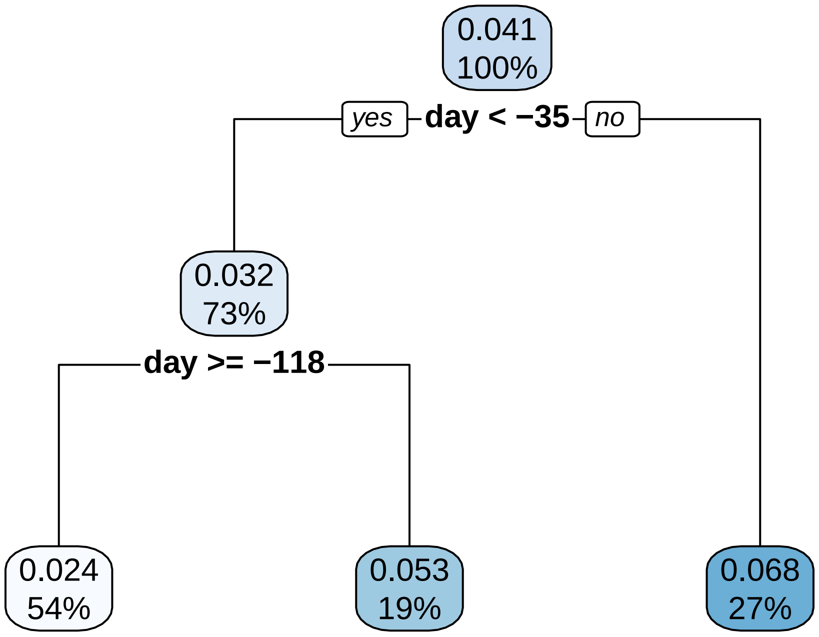

0.10
0.05
0.00
−0.05
margin
−150
−100
−50
0
day



```
## [1] 0.0005434706
```

```
calc_mse(2, 0.1)
```


0.10
0.05
0.00
−0.05
margin
−150
−100
−50
0
day

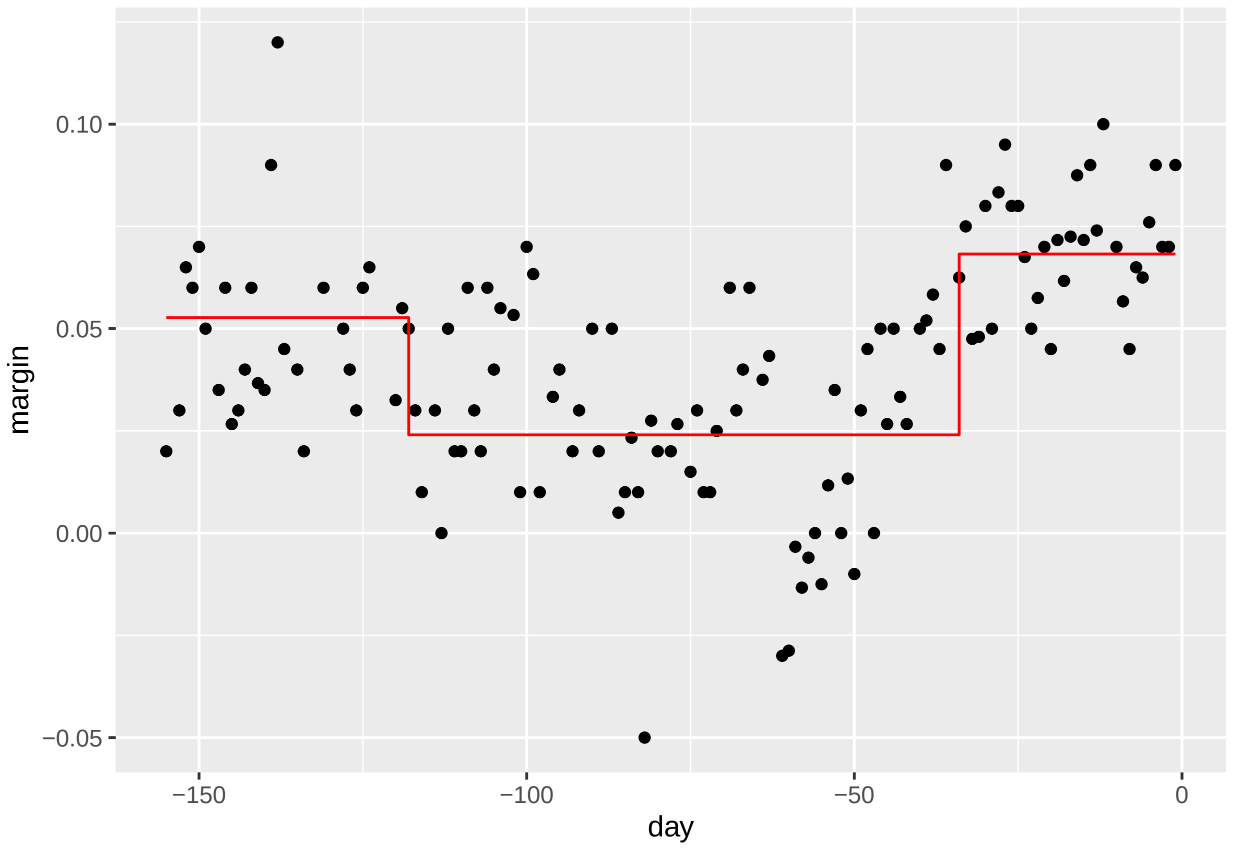


```
## [1] 0.0004262214
```

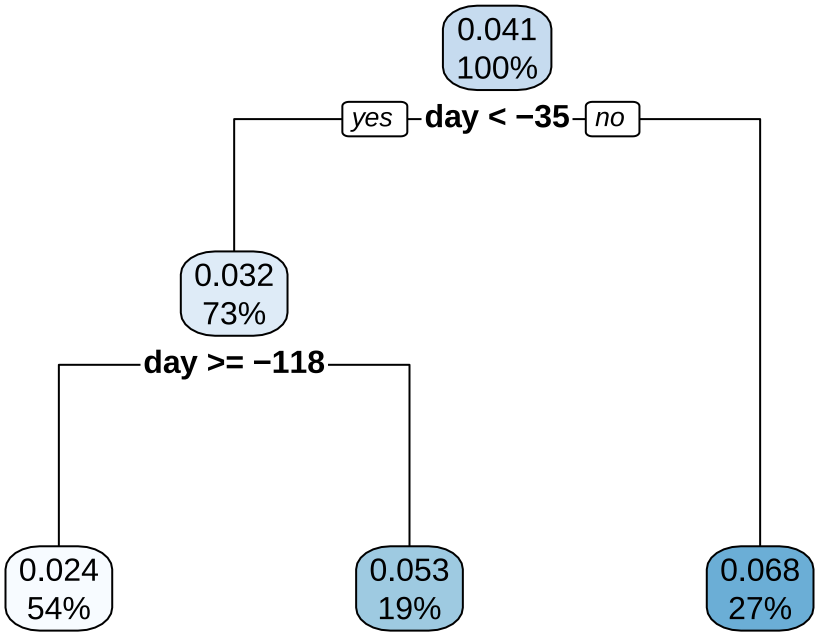
there is supposed to be a "relative improvement" of over 10%. "I keep growing my tree if the mse improves by this percentage".

2. In principle, every time that we add a level to our tree we can decrease our RSS. If we continue this approach indefinitely we could end up with a tree where every leaf is a single point which is a clear case of overfitting. The `complexity_parameter` (`cp`) controls the number of recursive splits your model takes. Roughly, it does this by measuring the difference in fit (measured by the MSE) by adding a new level and stopping if this value is less than the `cp` value. Armed with this knowledge explain why `calc_mse(3,0.1)` produces the same results as `calc_mse(2,0.1)`. Experiment changing the `cp` parameter so that you get a regression tree with three levels when you set the `tree_depth`=3. Change your parameters so that you get a regression tree with six levels.

```
calc_mse(3,0.1)
```

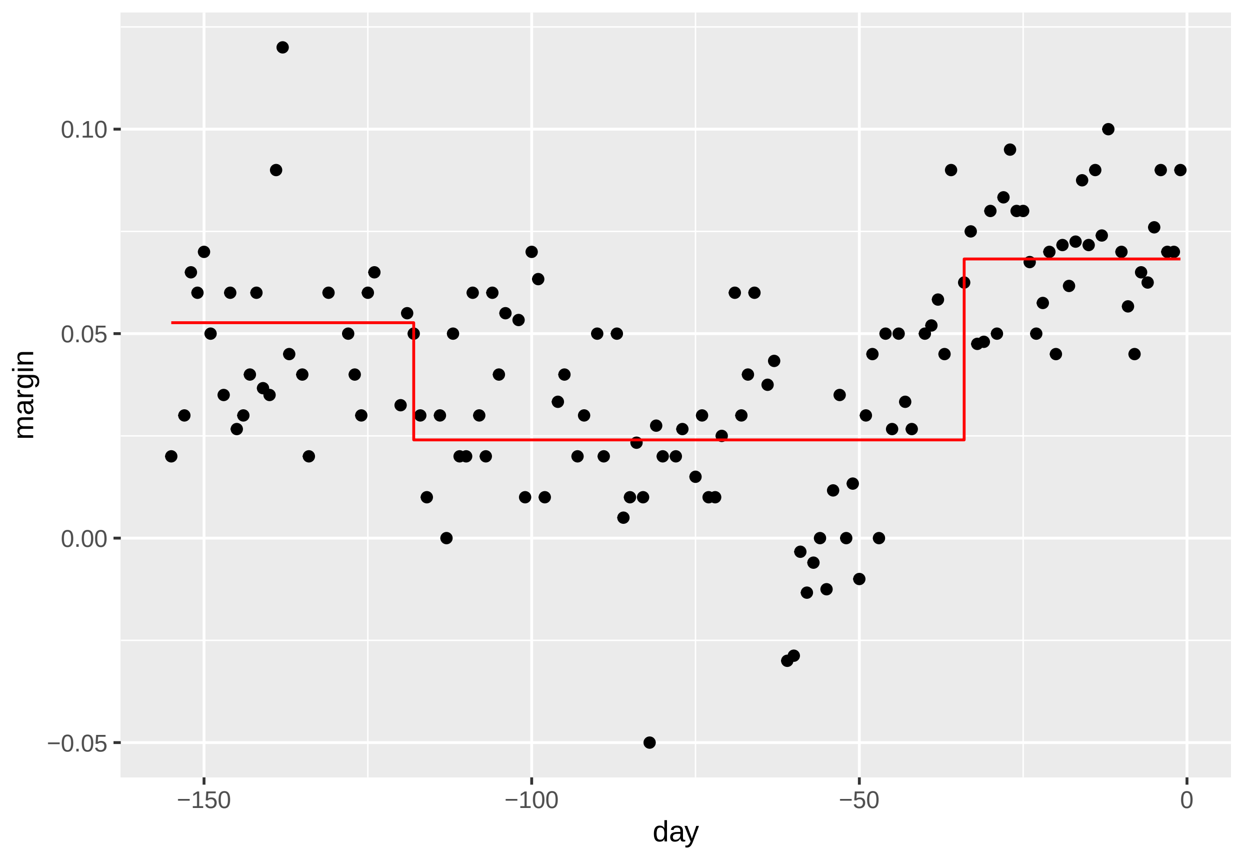


0.10
0.05
0.00
−0.05
margin
−150
−100
−50
0
day

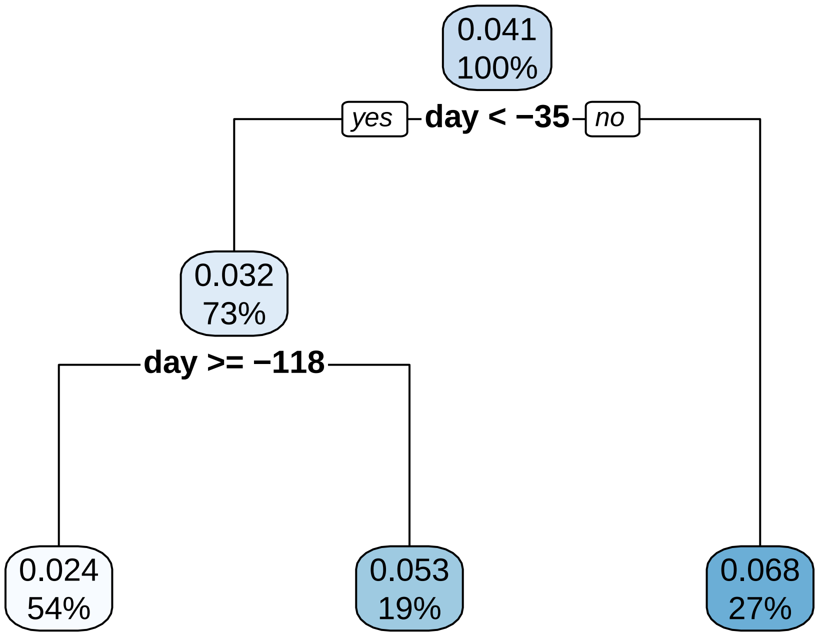


```
## [1] 0.0004262214
```

```
calc_mse(2,0.1)
```



0.10
0.05
0.00
−0.05
margin
−150
−100
−50
0
day

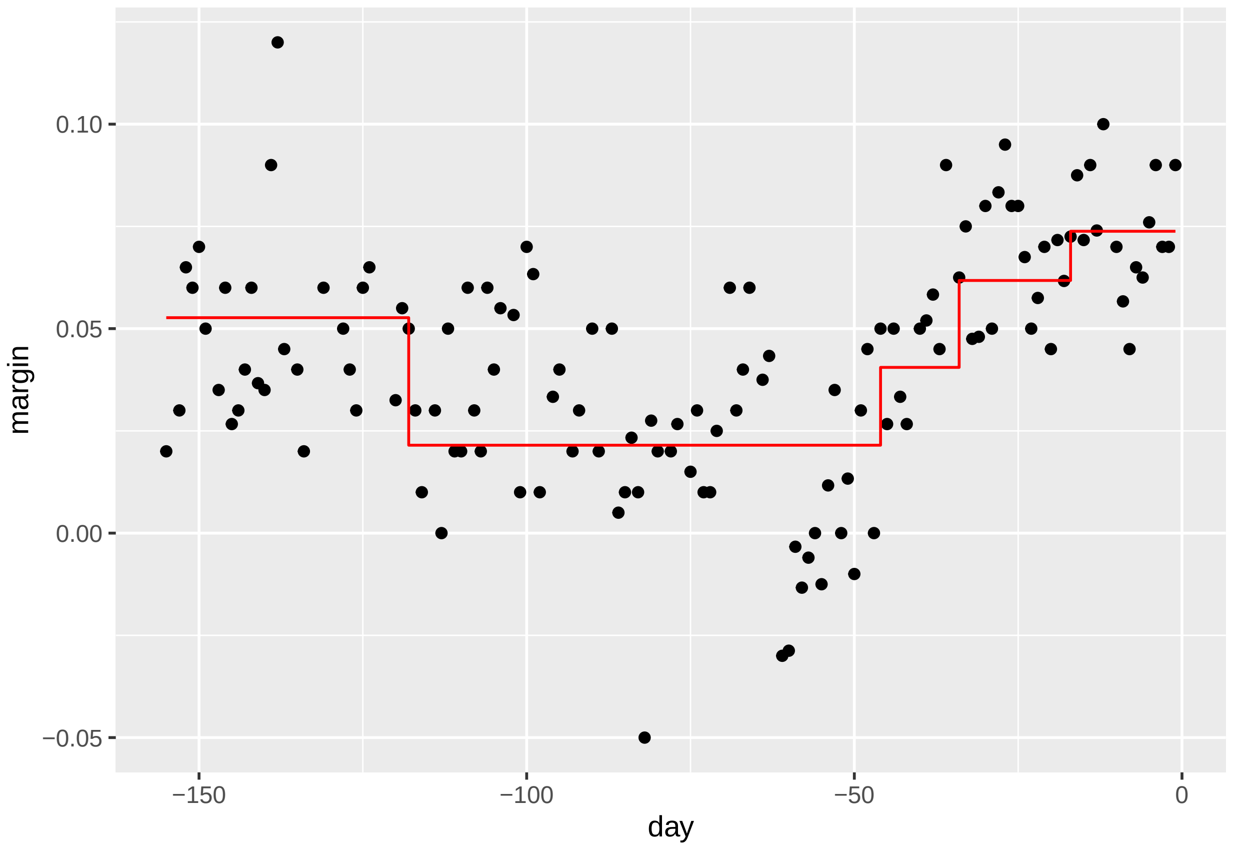


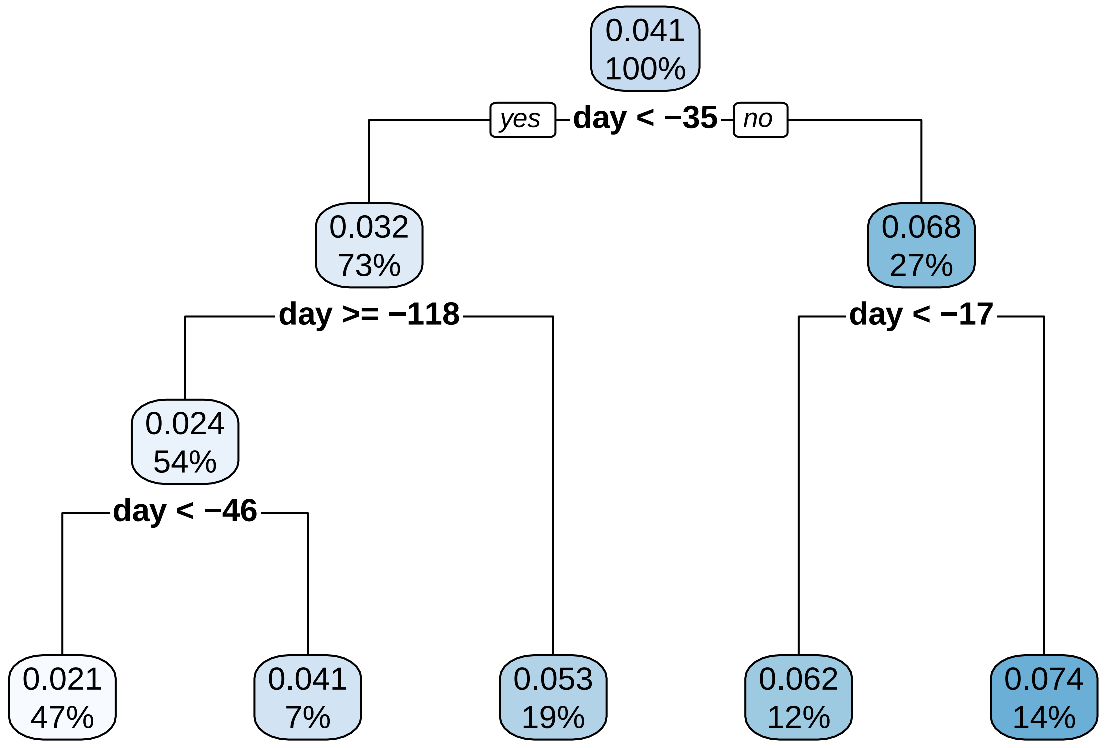
```
## [1] 0.0004262214
```

Why are these two values the same? The splits are both below 0.1. This occurs with a tree depth of 2.

Experiment changing the `cp` parameter so that you get a regression tree with three levels when you set the `tree_depth`=3:

```
calc_mse(3,0.01)
```

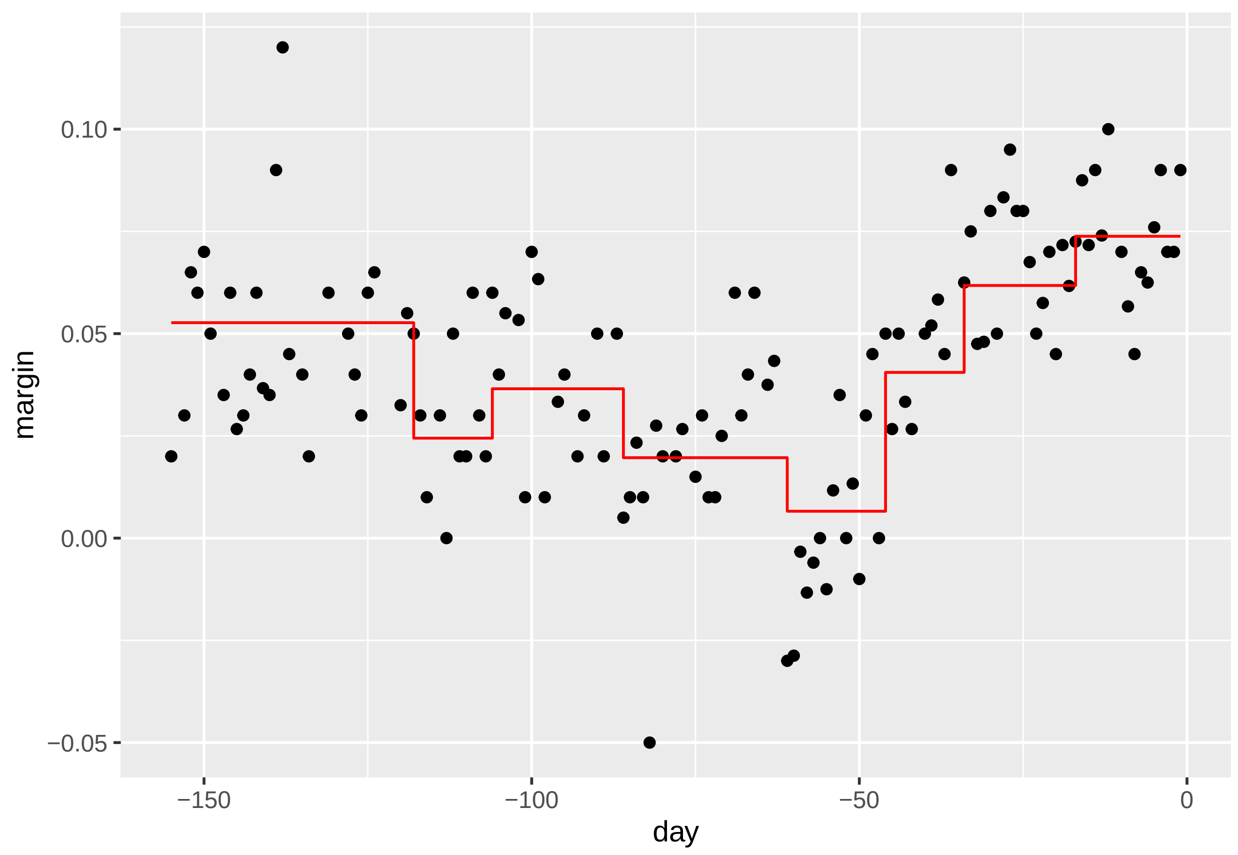

0.10
0.05
0.00
−0.05
margin
−150
−100
−50
0
day

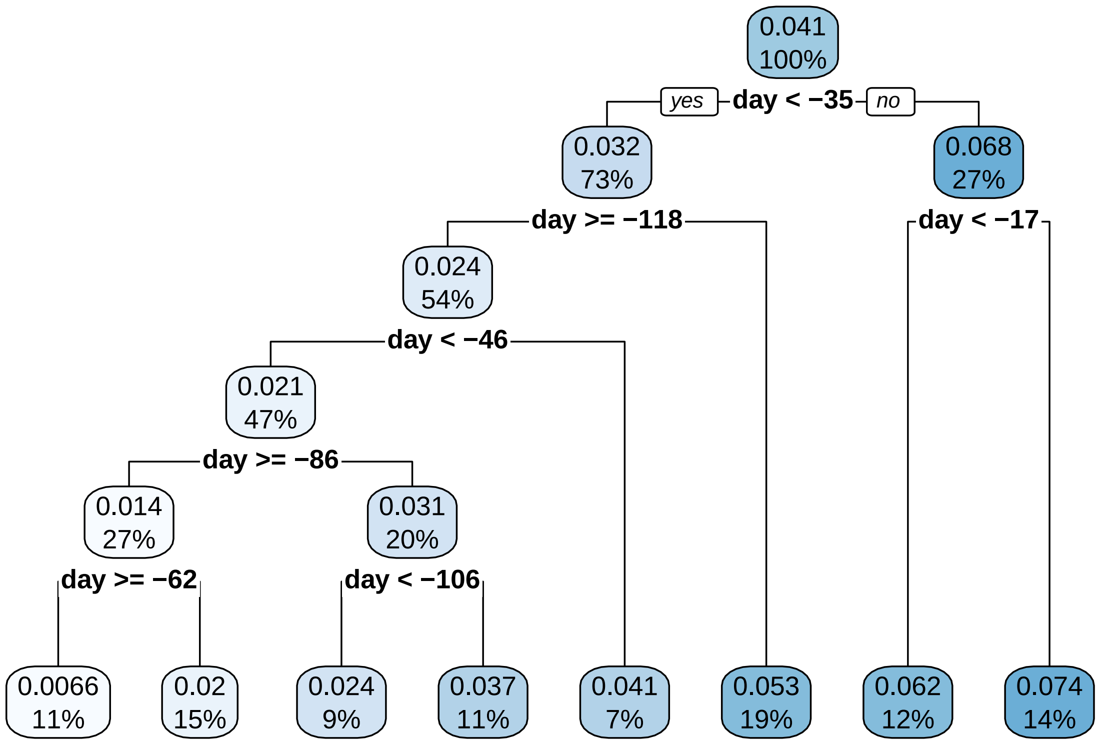


```
## [1] 0.0003942404
```

6 levels: can't figure out past 5.

```
calc_mse(6,0.001)
```


margin
0.10
0.05
0.00
−0.05
−150
−100
−50
0
day



```
## [1] 0.000342623
```

(select_by_one_std_err(poll_res, metric = “rmse”, desc(cost_complexity))) if you are minimizing rmse, you are also minimizing mse.

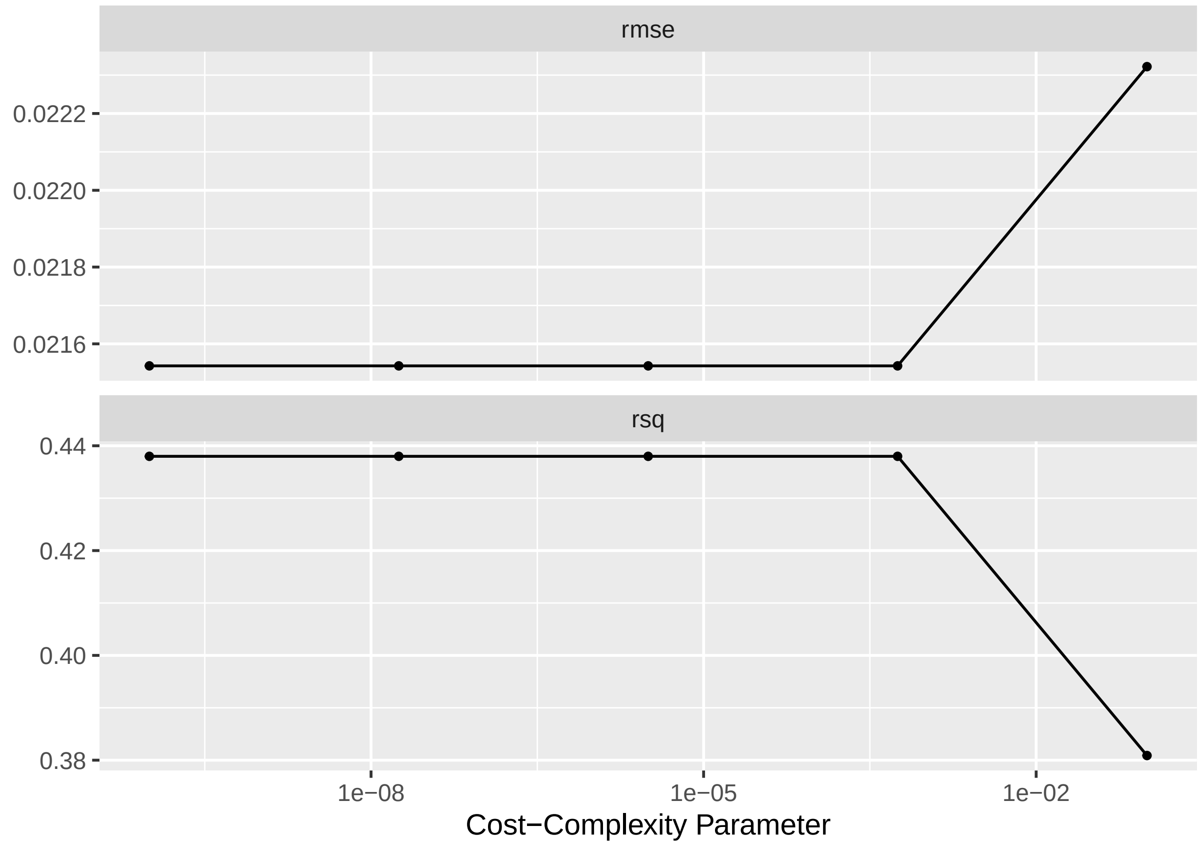
3. Using the following 10-fold cross-validation, find the optimal `cp` using the “one-standard-error” rule. Calculate the mse and plot the final model using your testing dataset

```
# Create the cross-validation dataset
set.seed(31416)
poll.folds <- vfold_cv(poll.train.tbl, v = 10)
```

The plot should look like a tequila glass sideways. do we include tree_depth = tune() somewhere? No. Huh.

```
poll.model <- decision_tree(cost_complexity = tune()) %>%#tree_depth=tune()
  set_mode("regression") %>%
  set_engine("rpart")

  #poll.recipe <- recipe(margin ~ day, data=poll.train.tbl)
  #reuse earlier recipe

  poll.wflow <- workflow() %>%
    add_recipe(poll.recipe) %>%
    add_model(poll.model)

  param_grid <- grid_regular(cost_complexity(), levels = 5)#levels = 10
```

```r
#calculate MSE
tune_res <- tune_grid(
  poll.wflow,
  resamples = poll.folds,
  grid = param_grid
  #, metrics = metric_set(accuracy)
)
autoplot(tune_res)
```



```r
show_best(tune_res, metric = "rmse")
```

```
## # A tibble: 5 x 7
##   cost_complexity .metric .estimator   mean     n std_err .config
##             <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <fct>
## 1    0.0000000001 rmse    standard   0.0215    10 0.00164 Preprocessor1_Model1
## 2    0.0000000178 rmse    standard   0.0215    10 0.00164 Preprocessor1_Model2
## 3    0.00000316   rmse    standard   0.0215    10 0.00164 Preprocessor1_Model3
## 4    0.000562     rmse    standard   0.0215    10 0.00164 Preprocessor1_Model4
## 5    0.1          rmse    standard   0.0223    10 0.00169 Preprocessor1_Model5
```

cost_complexity .metric .estimator mean n std_err .config

1 0.0000000001 rmse standard 0.0215 10 0.00164 Preprocessor1_Model01 2 0.000000001 rmse standard 0.0215 10 0.00164 Preprocessor1_Model02 3 0.00000001 rmse standard 0.0215 10 0.00164 Preprocessor1_Model03 4 0.0000001 rmse standard 0.0215 10 0.00164 Preprocessor1_Model04 5 0.000001 rmse standard 0.0215 10 0.00164 Preprocessor1_Model05

```
#prof:
#find the optimal `cp` using the "one-standard-error"
best.penalty <- select_by_one_std_err(tune_res, metric = "rmse", -cost_complexity)
poll.final.wf <- finalize_workflow(poll.wflow, best.penalty)
poll.final.fit <- fit(poll.final.wf, poll.train.tbl)
poll.final.rs <- last_fit(poll.final.wf, poll.split)
collect_metrics(poll.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <fct>
## 1 rmse    standard      0.0271 Preprocessor1_Model1
## 2 rsq     standard      0.235  Preprocessor1_Model1
```

```
#Calculate the mse: rmse^2
0.0271^2
```

```
## [1] 0.00073441
```

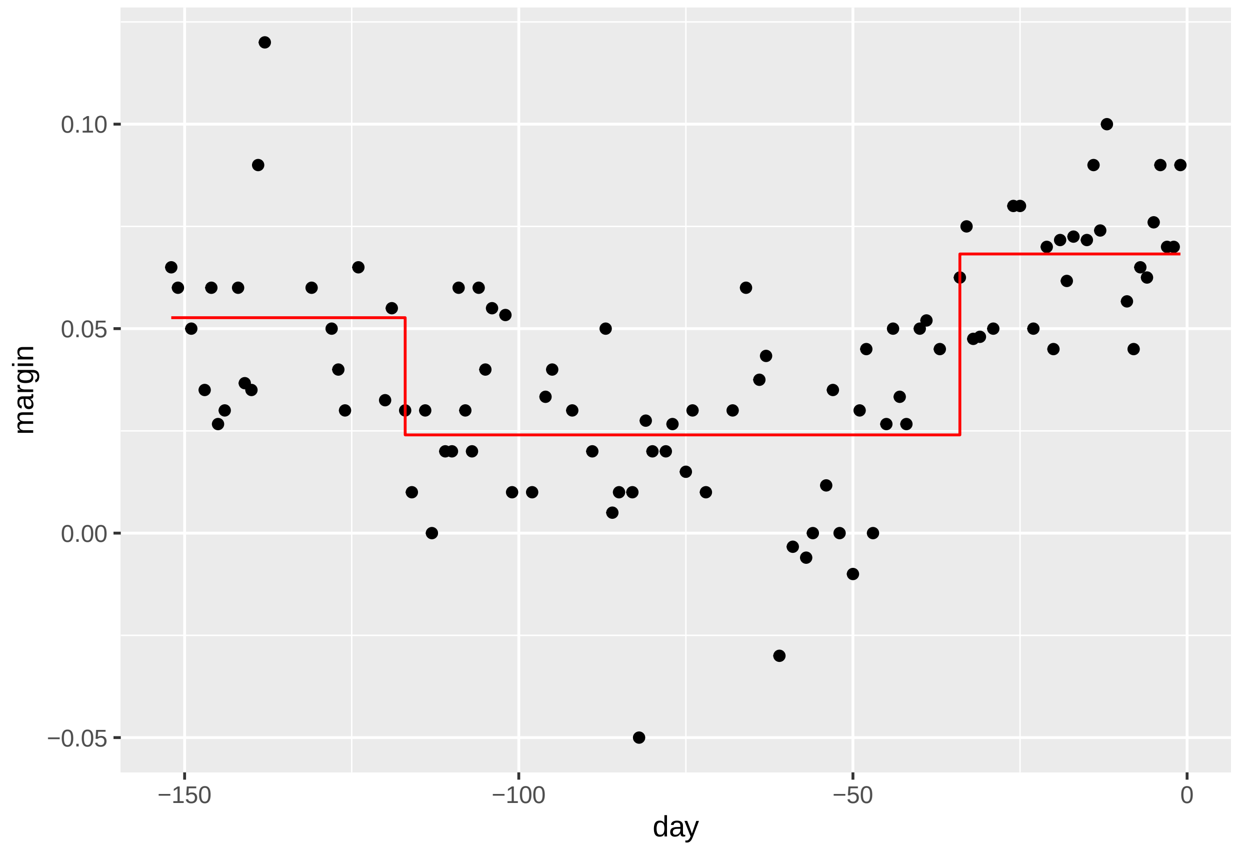
.metric .estimator .estimate .config

1 rmse standard 0.0271 Preprocessor1_Model1 2 rsq standard 0.235 Preprocessor1_Model1

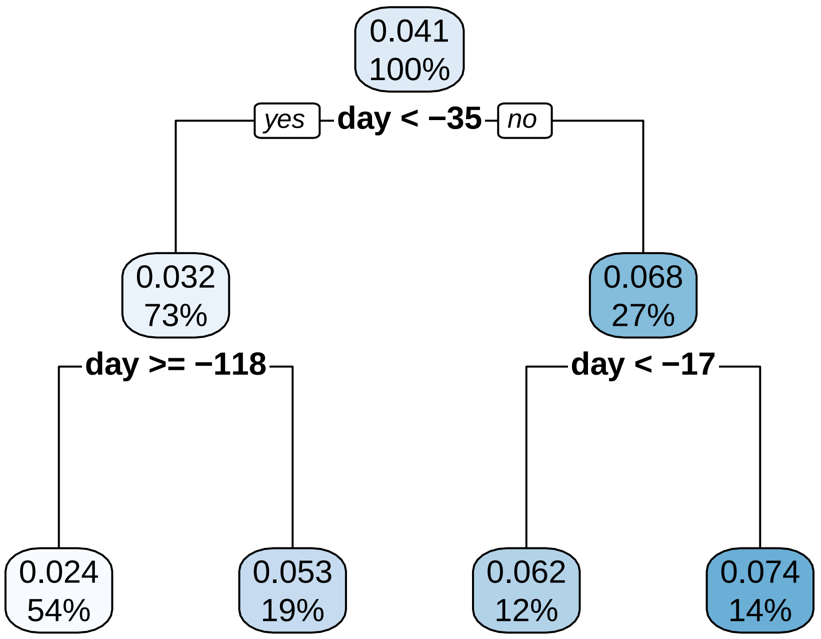
mse = 0.00073441

plot the final model using your testing dataset ****** is this correct? Why does the key show augment with the training dataset and not testing dataset?

```
#prof:
augment(poll.final.fit, poll.train.tbl) %>%
ggplot() +
geom_point(aes(day, margin))+
geom_step(aes(day, .pred), col = "red")
```



```
#and more.

poll.fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



Back to decision trees.

We would like to revisit one of our favorite problems, digit classification, this time using decision trees.

To do that first, let's create a subset of the MNIST dataset

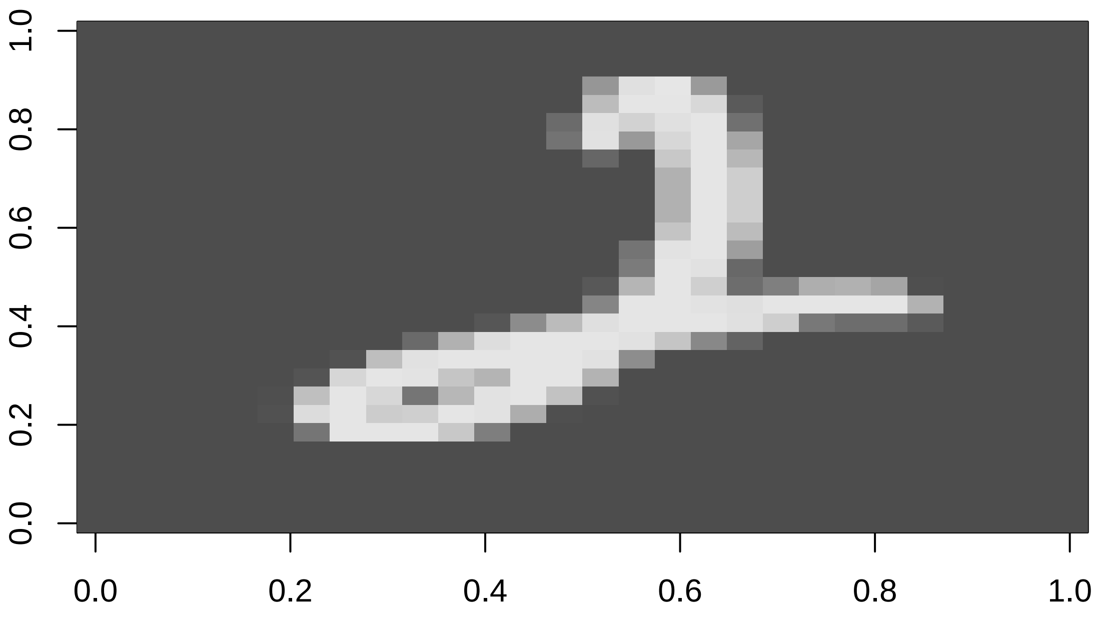
```
mnist <- read_mnist()
set.seed(2022)
index <- sample(nrow(mnist$train$images), 1000)#was 10000
train.tbl <- as_tibble (mnist$train$images[index,]) %>%
  mutate(digit = factor(mnist$train$labels[index]))

index <- sample(nrow(mnist$test$images), 100)#was 1000
test.tbl <- as_tibble (mnist$test$images[index,]) %>%
  mutate(digit = factor(mnist$test$labels[index]))
```

And let's subset this dataset to just `1`s and `2`s

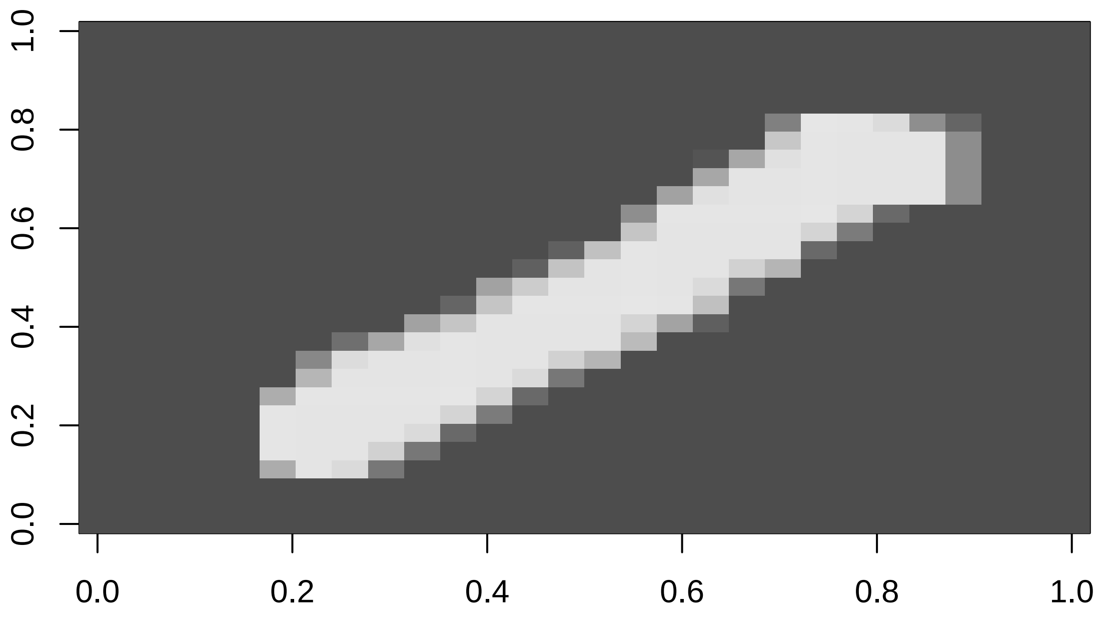
```
digits = c(1,2)

train.12.tbl = train.tbl %>%
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))

test.12.tbl = test.tbl %>%
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))
```

And let's keep some plotting functions in case we need them

```
plotImage <- function(dat,size=28){
  imag <- matrix(dat,nrow=size)[,28:1]
  image(imag,col=grey.colors(256), xlab = "", ylab="")
}

plot_row <- function(tbl) {
  ntbl <- tbl %>%
    select(-digit)
  plotImage(as.matrix(ntbl))
}
plot_row(train.12.tbl[100,])
```



```
plot_row(train.12.tbl["99",])
```



4. Using default parameters create a decision tree that would distinguish between 1s and 2s. Visualize the decision tree using `rpart.plot`. What is the accuracy and the confusion matrix on the testing dataset?

"using default parameters, create a decision tree"

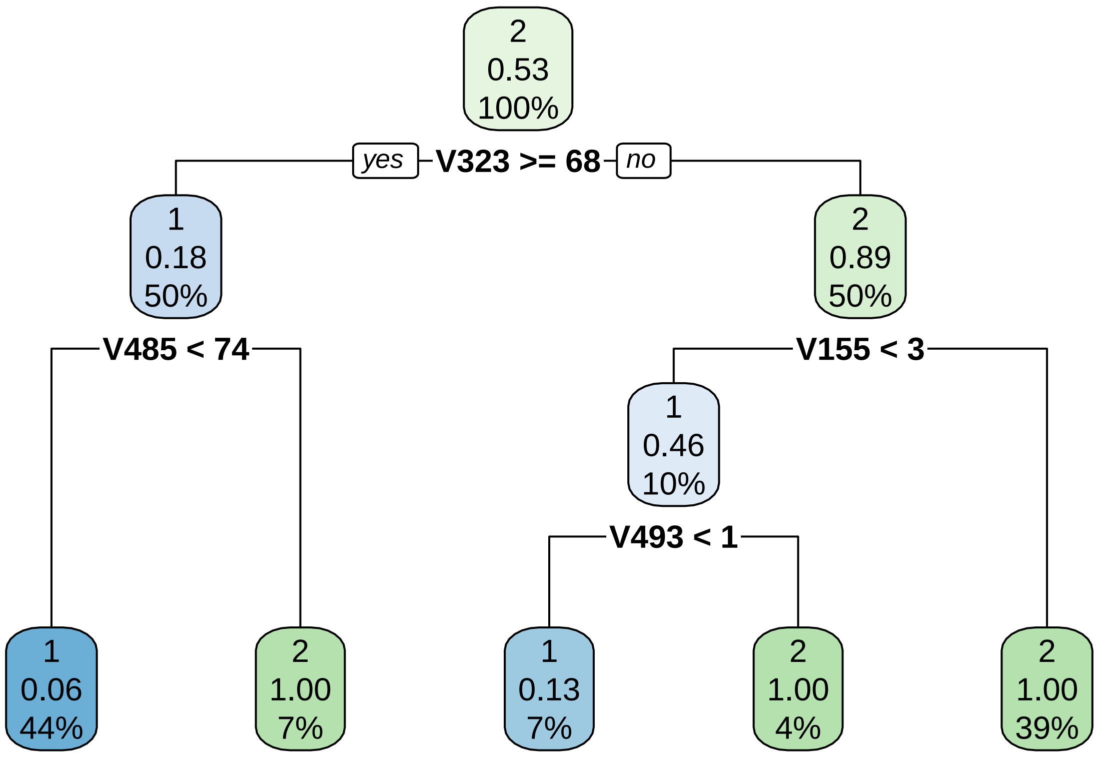
```
digit.model <-
  decision_tree() %>%#  skip tree_depth and cost_complexity
  set_mode("classification") %>%
  set_engine("rpart")

digit.recipe <- recipe(digit ~., data = train.12.tbl)

digit.wflow <- workflow() %>%
  add_recipe(digit.recipe) %>%
  add_model(digit.model)

digit.fit <- fit(digit.wflow, train.12.tbl)

augment(digit.fit, test.12.tbl) %>%
  accuracy(truth = digit, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.84
```

```
augment(digit.fit, test.12.tbl) %>%
  conf_mat(truth = digit, estimate = .pred_class)
```

```
##             Truth
## Prediction  1  2
##          1 13  4
##          2  0  8
```

```
digit.fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint = FALSE)
```



.metric .estimator .estimate 1 accuracy binary 0.949

Note: with the smaller dataset above, accuracy is now 0.84.

```
      Truth
```

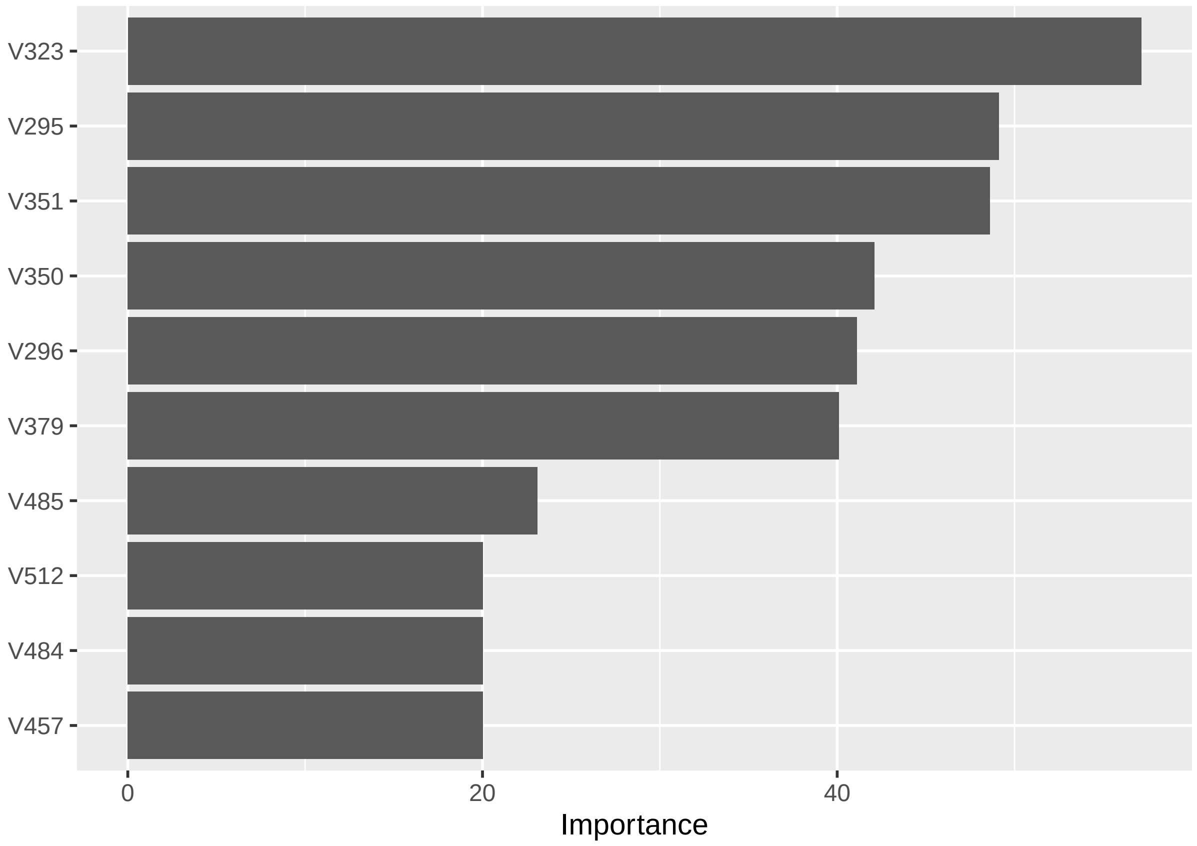
Prediction 1 2 1 122 4 2 8 100

1 gets misclassified as 2 more often than 2 gets misclassified as 1.

In decision trees we can quantify the importance of variables in the following. At each node a single variables is used to partition the data into two homogeneous groups and in doing so maximizes some measure of improvement. The importance of a variable x is the sum of the squared improvements over all internal nodes of the tree for which x was chosen as the partitioning variable.

Notice that in R we can use the `vip` library to calculate the importance in the following manner. Notice that we can get the information as a tibble using the function `vip:vi()`

```r
library(vip)

digit.fit %>%
  extract_fit_engine() %>%
  vip::vip()
```

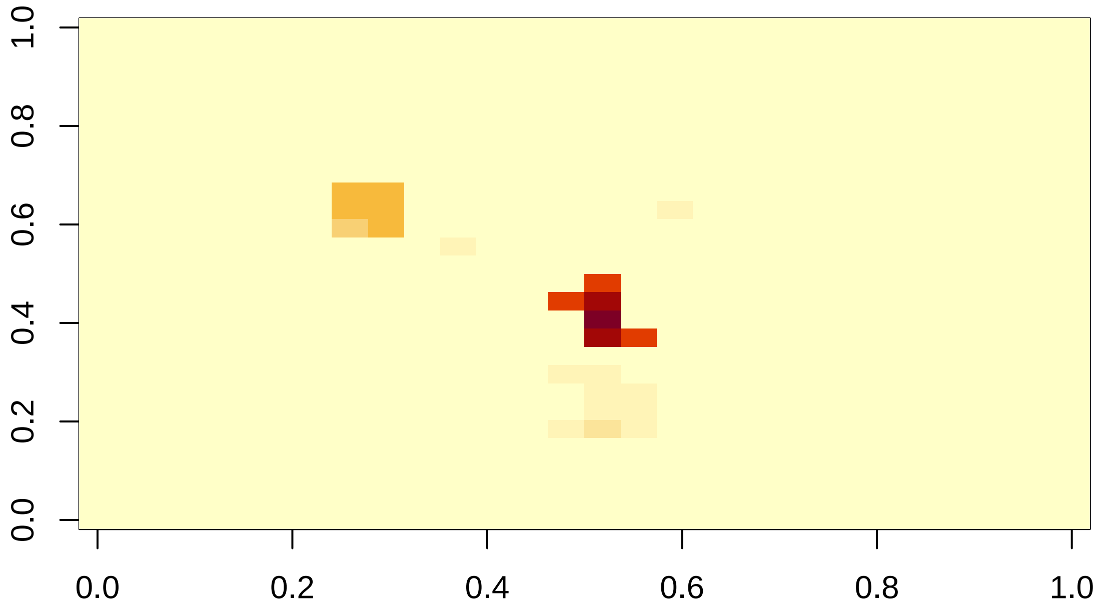


```r
imp.tbl <- digit.fit %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

```
## # A tibble: 24 x 2
##    Variable Importance
##    <chr>         <dbl>
##  1 V323           57.1
##  2 V295           49.1
##  3 V351           48.6
##  4 V350           42.1
##  5 V296           41.1
##  6 V379           40.1
##  7 V485           23.1
##  8 V457           20.0
##  9 V484           20.0
## 10 V512           20.0
## # ... with 14 more rows
```

Finally we can create an image that will allow us to visualize the importance of those pixels (features)

```r
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))#turning it into a number

mat <- rep(0, 28*28)
mat[imp.tbl$col] <- imp.tbl$Importance
image(matrix(mat, 28, 28))
```



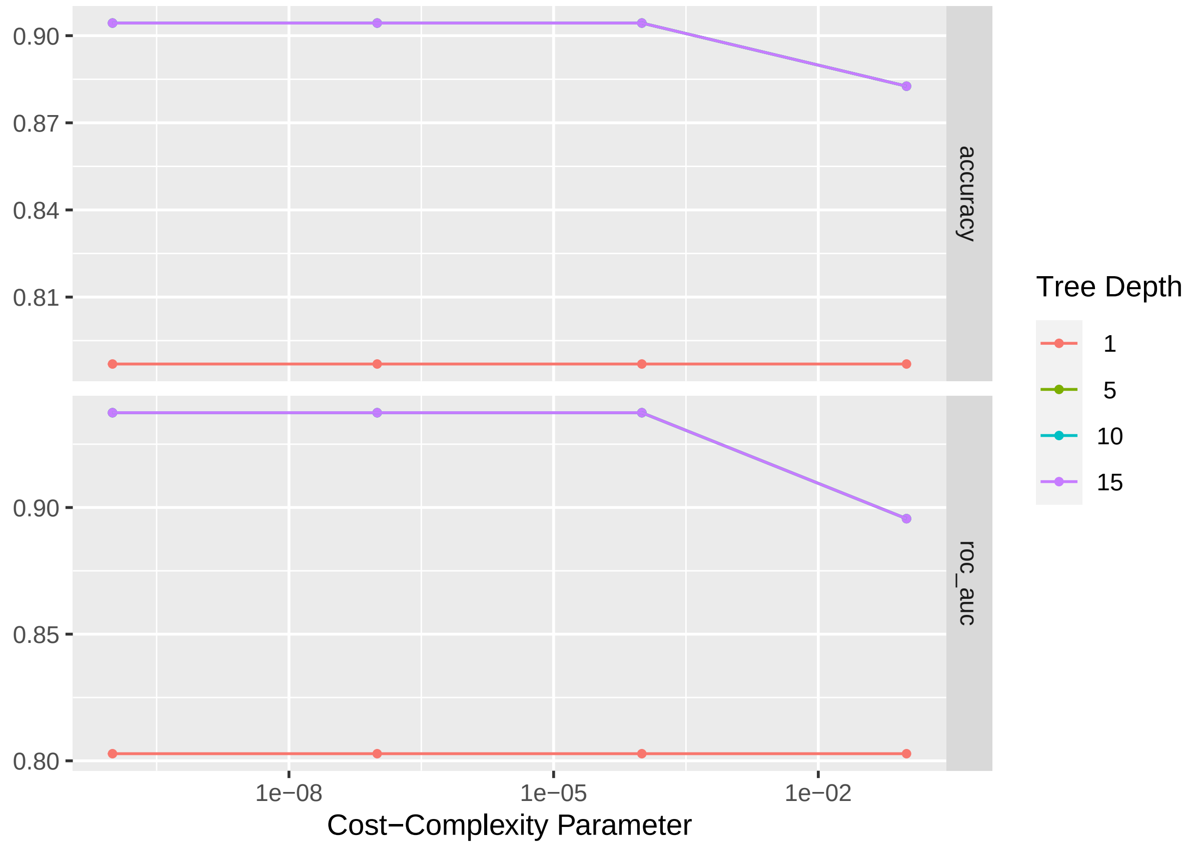
5. Find the optimal `cp` and `tree_depth` using 10-fold cross-validation and the one standard-error rule. What is your accuracy using your testing dataset? Create an image with most important features used by your model.

```r
#general model stuff
digits.model <- decision_tree(cost_complexity = tune(), tree_depth = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

digits.recipe <- recipe(digit ~., data = train.12.tbl)

digits.wflow <- workflow() %>%
  add_recipe(digits.recipe) %>%
  add_model(digits.model)
```

```r
#cross validation stuff
param_grid <- grid_regular(cost_complexity(), tree_depth(), levels = 4)# this is what I could get to ru

# Create the cross-validation dataset
set.seed(31416)
```

```r
digits.folds <- vfold_cv(train.12.tbl, v = 10)
```

```r
#calculate MSE
tune_res <- tune_grid(
  digits.wflow,
  resamples = digits.folds,
  grid = param_grid#,
  #metrics = metric_set(accuracy)#classification?
)
autoplot(tune_res)
```



```r
show_best(tune_res, metric = "accuracy")
```

```
## # A tibble: 5 x 8
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1    0.0000000001          5 accuracy binary     0.904    10  0.0203 Preprocess~
## 2    0.0000001             5 accuracy binary     0.904    10  0.0203 Preprocess~
## 3    0.0001                5 accuracy binary     0.904    10  0.0203 Preprocess~
## 4    0.0000000001         10 accuracy binary     0.904    10  0.0203 Preprocess~
## 5    0.0000001            10 accuracy binary     0.904    10  0.0203 Preprocess~
```

cost_complexity tree_depth .metric .estimator mean n std_err .config

1 0.0000000001 5 accuracy binary 0.904 10 0.0203 Preprocessor1_Model05 2 0.0000001 5 accuracy binary 0.904 10 0.0203 Preprocessor1_Model06 3 0.0001 5 accuracy binary 0.904 10 0.0203 Preprocessor1_Model07 4 0.0000000001 10 accuracy binary 0.904 10 0.0203 Preprocessor1_Model09 5 0.0000001 10 accuracy binary

0.904 10 0.0203 Preprocessor1_Model10

```
#find the optimal `cp` and tree_depth using the "one-standard-error"
best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", -cost_complexity, -tree_depth)#, -
best.penalty
```

```
## # A tibble: 1 x 10
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1          0.0001         15 accuracy binary     0.904    10  0.0203 Preprocess~
## # ... with 2 more variables: .best <dbl>, .bound <dbl>
```

cost_complexity tree_depth .metric .estimator mean n std_err .config .best .bound 1 0.0001 15 accuracy binary 0.904 10 0.0203 Preproces… 0.904 0.884

tree depth = 15, cost_complexity is 0.0001

```
digits.final.wf <- finalize_workflow(digits.wflow, best.penalty)
digits.final.fit <- fit(digits.final.wf, train.12.tbl)

#not sure about this part. trying to create a split object.
custom_split <- make_splits(train.12.tbl, assessment = test.12.tbl)

digits.final.rs <- last_fit(digits.final.wf, custom_split)
collect_metrics(digits.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <fct>
## 1 accuracy binary         0.84  Preprocessor1_Model1
## 2 roc_auc  binary         0.904 Preprocessor1_Model1
```

.metric .estimator .estimate .config

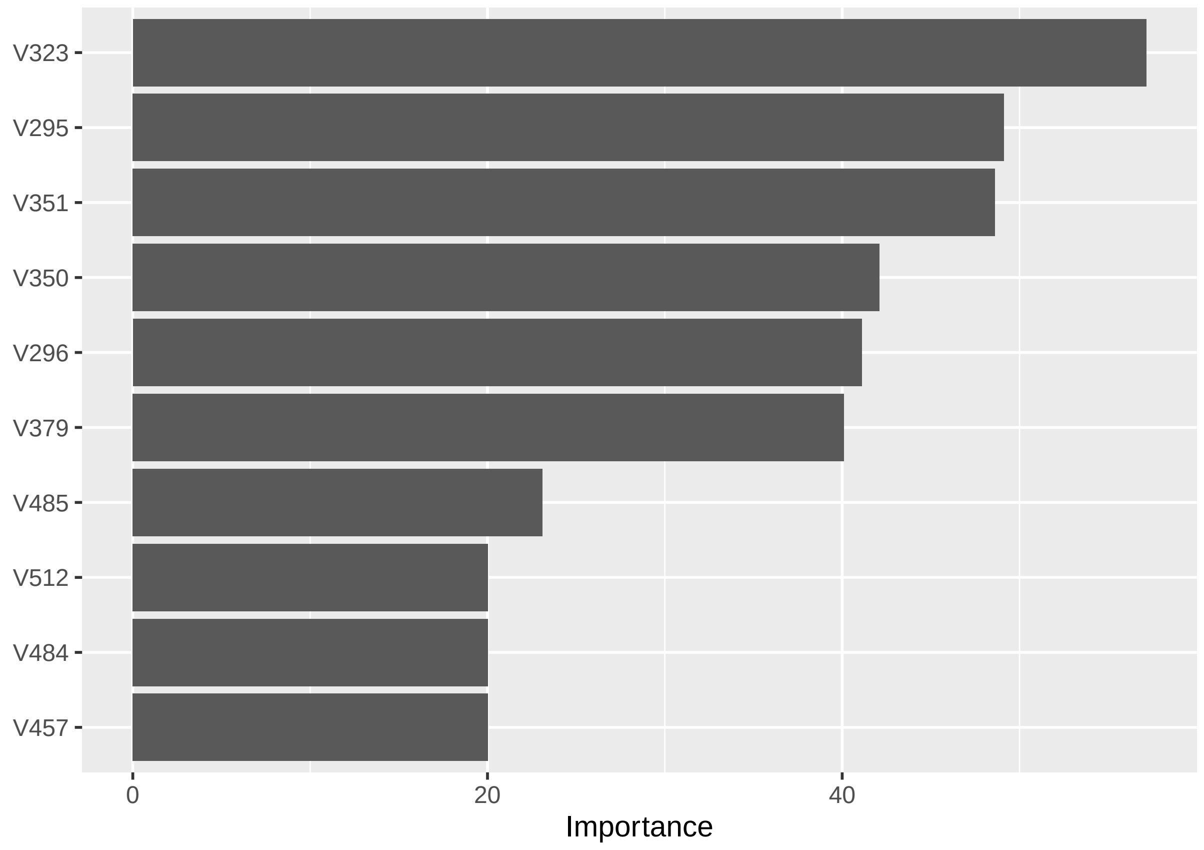
1 accuracy binary 0.84 Preprocessor1_Model1 2 roc_auc binary 0.904 Preprocessor1_Model1

accuracy is 0.84 which is the same as #4 when run with the same size data. This might be due to the fairly small dataset I am using since my computer is not very powerful.

I feel like this number should be higher, but I did use a smaller dataset and a smaller levels value in grid_regular(. It seems the dataset size has a great effect on accuracy.

Create an image with most important features used by your model:

```
digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vip()
```



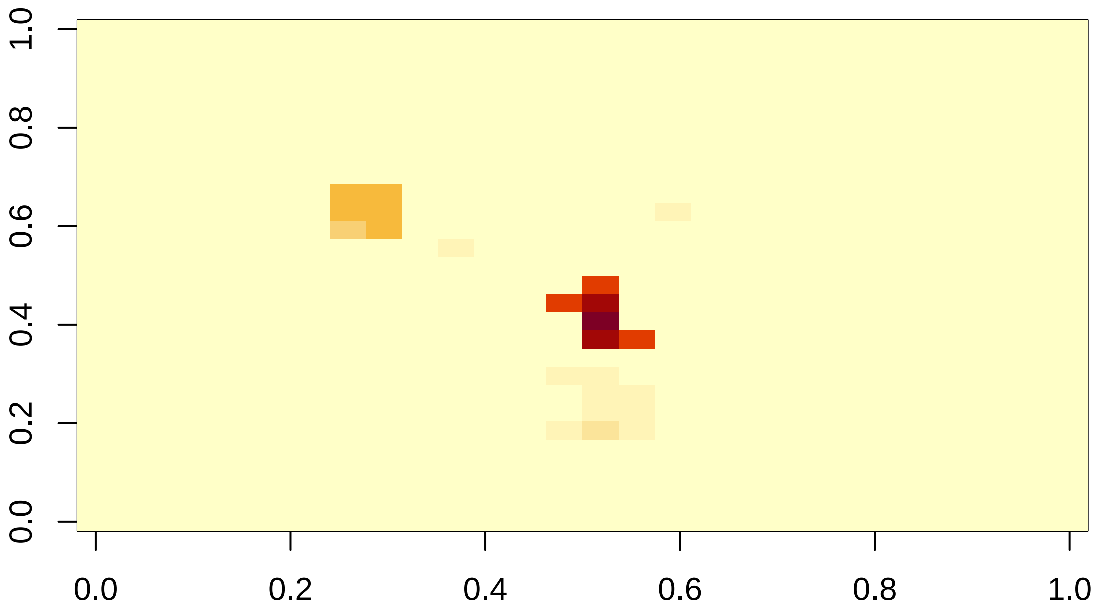
```
imp.tbl <- digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

```
## # A tibble: 24 x 2
##    Variable Importance
##    <chr>         <dbl>
##  1 V323           57.1
##  2 V295           49.1
##  3 V351           48.6
##  4 V350           42.1
##  5 V296           41.1
##  6 V379           40.1
##  7 V485           23.1
##  8 V457           20.0
##  9 V484           20.0
## 10 V512           20.0
## # ... with 14 more rows
```

Variable Importance 1 V323 57.1 2 V295 49.1 3 V351 48.6 4 V350 42.1 5 V296 41.1 6 V379 40.1 7 V485 23.1 8 V457 20.0 9 V484 20.0 10 V512 20.0

```
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))#turning it into a number

mat <- rep(0, 28*28)
```

```
mat[imp.tbl$col] <- imp.tbl$Importance
image(matrix(mat, 28, 28))
```



Comparing to the previous problem's image for #4, the first 7 variables are the same in importance. After that, variables start to differ from importance with #4 (when run with the smaller dataset).

6. Create an optimal decision tree (e.g. by optimizing `cp` and `tree_depth` for the pair of digits that you were given in your first challenge). What is your accuracy and confusion matrix using your testing dataset? Plot a couple of digits that get missclassified. Create an image with most important features used by your model.

Create testing and training for 0 and 7:

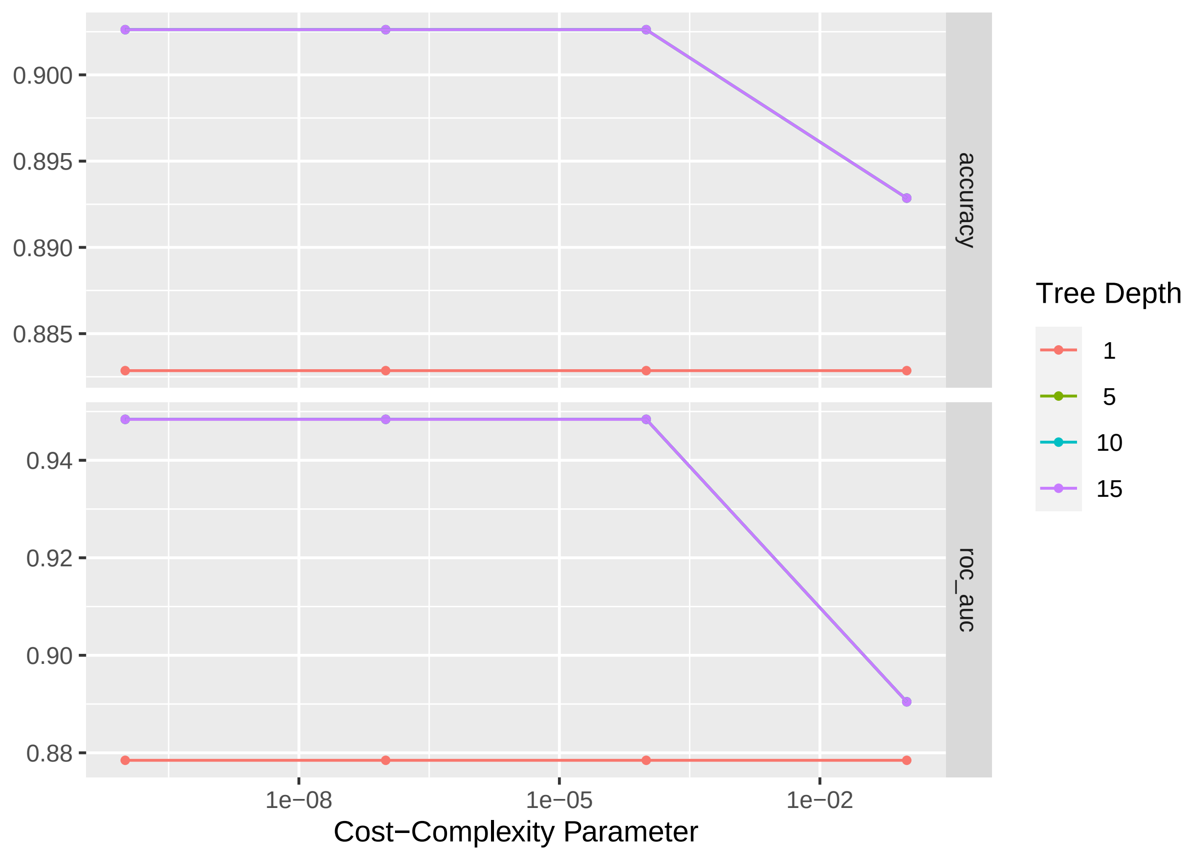
```
mnist <- read_mnist()
set.seed(2023)
index <- sample(nrow(mnist$train$images), 1000)#was 10000
train.tbl <- as_tibble (mnist$train$images[index,]) %>%
  mutate(digit = factor(mnist$train$labels[index]))
index <- sample(nrow(mnist$test$images), 100)#was 1000
test.tbl <- as_tibble (mnist$test$images[index,]) %>%
  mutate(digit = factor(mnist$test$labels[index]))

digits = c(0,7)
train.12.tbl = train.tbl %>%
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))
test.12.tbl = test.tbl %>%
```

```r
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))
```

```r
#general model stuff
digits.model <- decision_tree(cost_complexity = tune(), tree_depth = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

digits.recipe <- recipe(digit ~., data = train.12.tbl)

digits.wflow <- workflow() %>%
  add_recipe(digits.recipe) %>%
  add_model(digits.model)
```

```r
#cross validation stuff
param_grid <- grid_regular(cost_complexity(), tree_depth(), levels = 4)
# Create the cross-validation dataset
set.seed(31416)
digits.folds <- vfold_cv(train.12.tbl, v = 10)
```

```r
#calculate MSE
tune_res <- tune_grid(
  digits.wflow,
  resamples = digits.folds,
  grid = param_grid#,
  #metrics = metric_set(accuracy)#classification?
)
autoplot(tune_res)
```



```
#show_best(tune_res, metric = "accuracy")
```

```
#find the optimal `cp` and tree_depth using the "one-standard-error"
best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", -cost_complexity, -tree_depth)#, -
best.penalty
```

```
## # A tibble: 1 x 10
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1             0.1         15 accuracy binary     0.893    10  0.0231 Preprocess~
## # ... with 2 more variables: .best <dbl>, .bound <dbl>
```

tree depth = 15, cost_complxity = 0.1

```
digits.final.wf <- finalize_workflow(digits.wflow, best.penalty)
digits.final.fit <- fit(digits.final.wf, train.12.tbl)

#not sure about this part. trying to create a split object.
custom_split <- make_splits(train.12.tbl, assessment = test.12.tbl)

digits.final.rs <- last_fit(digits.final.wf, custom_split)
collect_metrics(digits.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <fct>
## 1 accuracy binary         0.947 Preprocessor1_Model1
## 2 roc_auc  binary         0.95  Preprocessor1_Model1
```

accuracy = 0.947

(is this the same as the accuracy displayed below? how is last_fit different from fit?)

```r
augment(digits.final.fit, test.12.tbl) %>%
  accuracy(truth = digit, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.947
```

```r
augment(digits.final.fit, test.12.tbl) %>%
  conf_mat(truth = digit, estimate = .pred_class)
```

```
##           Truth
## Prediction 0 7
##          0 9 1
##          7 0 9
```

.metric .estimator .estimate 1 accuracy binary 0.947

```
      Truth
```

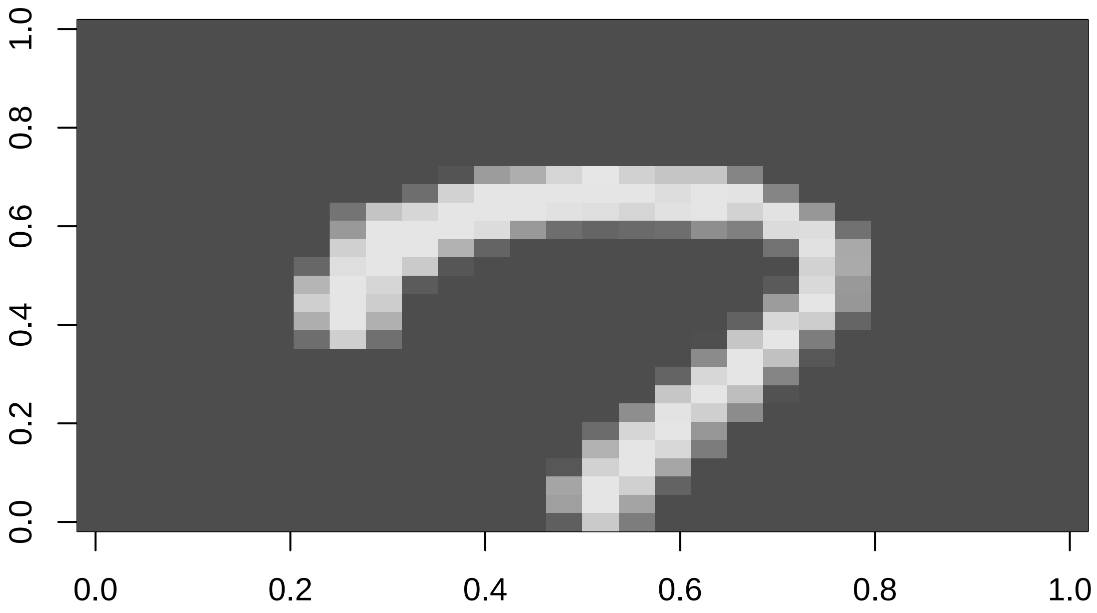
Prediction 0 7 0 9 1 7 0 9

Plot a few digits that get misclassified:

```r
#7s that got labeled as 0s
index <- augment(digits.final.fit, test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(.pred_class=="0" & digit == "7") %>%
  arrange(desc(n)) %>%
  pull(n)
```

18

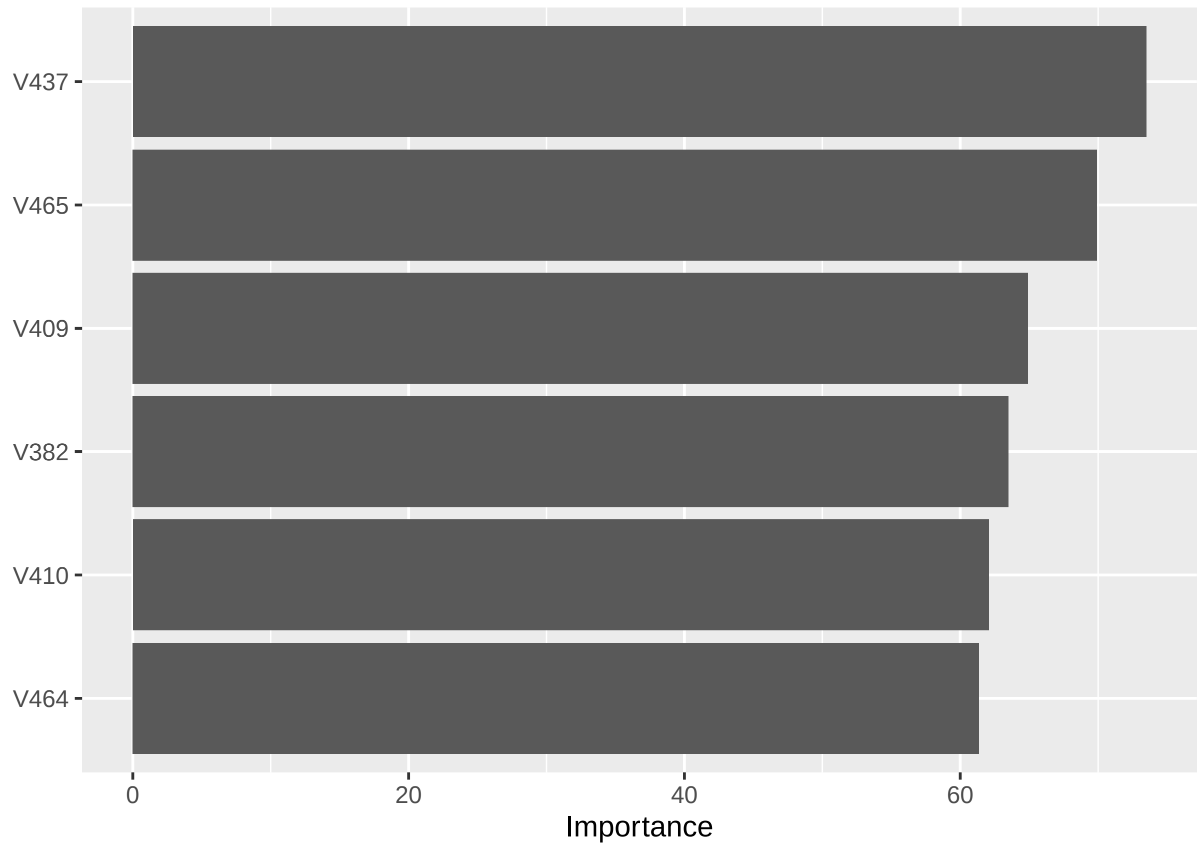
```r
plot_row(test.12.tbl[18,])
```



Due to limited data I have just one graph I can plot. (confusion matrix shows one misclassified value)

Create an image with most important features used by your model:

```
digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vip()
```



```r
imp.tbl <- digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

```
## # A tibble: 6 x 2
##   Variable Importance
##   <chr>         <dbl>
## 1 V437           73.5
## 2 V465           69.9
## 3 V409           64.9
## 4 V382           63.5
## 5 V410           62.1
## 6 V464           61.4
```

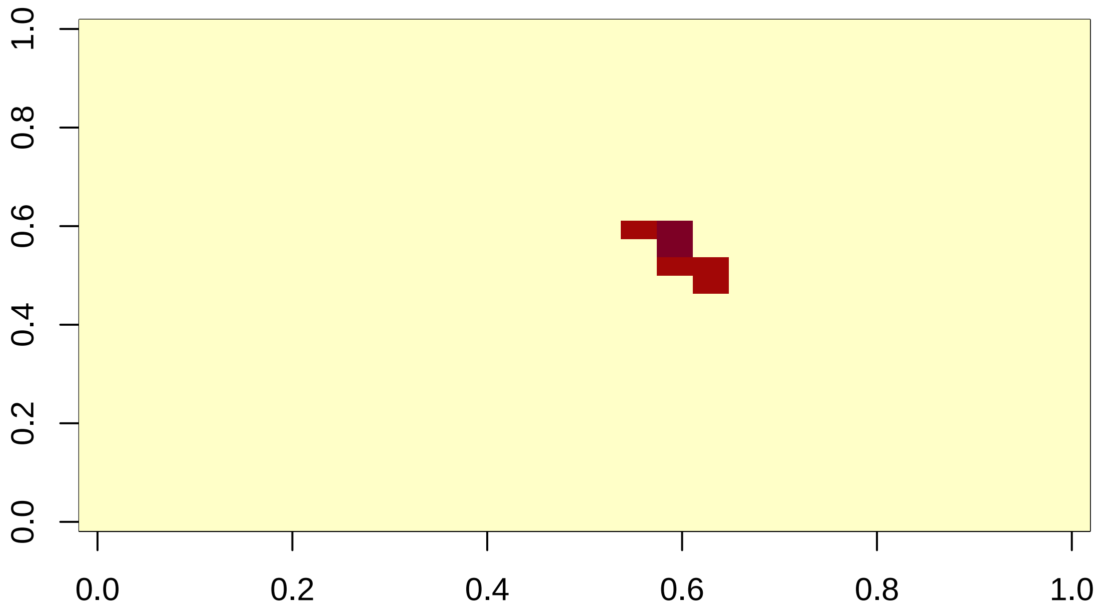
A tibble: 6 × 2

Variable Importance 1 V437 73.5 2 V465 69.9 3 V409 64.9 4 V382 63.5 5 V410 62.1 6 V464 61.4

(extra graph:)

```r
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))#turning it into a number

mat <- rep(0, 28*28)
```

```
mat[imp.tbl$col] <- imp.tbl$Importance
image(matrix(mat, 28, 28))
```



7. Create a new dataset by adding 5s to the mix (or another digit, in case 5 was in your original pair of digits). Repeat the steps outlined in exercise 6 for this new dataset.

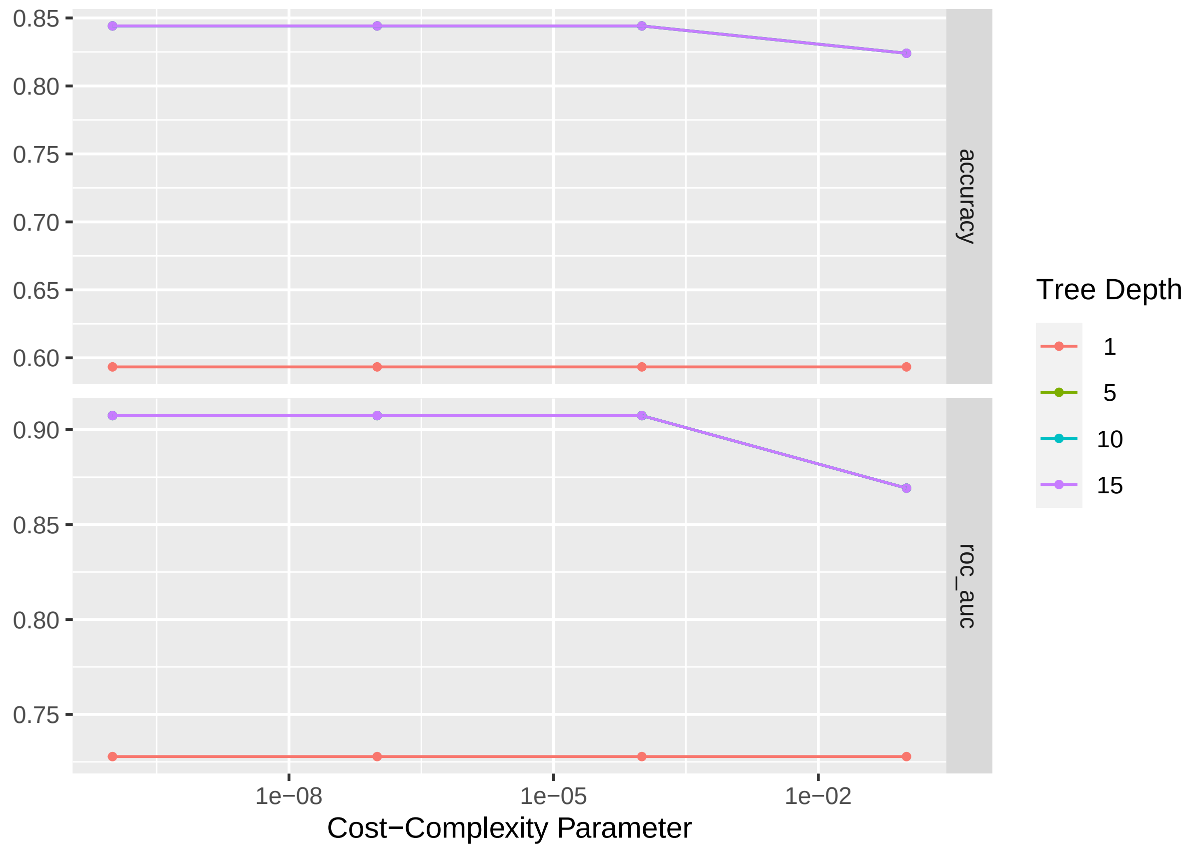
Create testing and training for 0 and 7 and 5:

```
mnist <- read_mnist()
set.seed(2023)
index <- sample(nrow(mnist$train$images), 1000)#was 10000
train.tbl <- as_tibble (mnist$train$images[index,]) %>%
  mutate(digit = factor(mnist$train$labels[index]))
index <- sample(nrow(mnist$test$images), 100)#was 1000
test.tbl <- as_tibble (mnist$test$images[index,]) %>%
  mutate(digit = factor(mnist$test$labels[index]))

digits = c(0,7, 5)
train.12.tbl = train.tbl %>%
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))
test.12.tbl = test.tbl %>%
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))
```

```
#general model stuff
digits.model <- decision_tree(cost_complexity = tune(), tree_depth = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

digits.recipe <- recipe(digit ~., data = train.12.tbl)

digits.wflow <- workflow() %>%
  add_recipe(digits.recipe) %>%
  add_model(digits.model)
```

```
#cross validation stuff
param_grid <- grid_regular(cost_complexity(), tree_depth(), levels = 4)
# Create the cross-validation dataset
set.seed(31416)
digits.folds <- vfold_cv(train.12.tbl, v = 10)
```

```
#calculate MSE
tune_res <- tune_grid(
  digits.wflow,
  resamples = digits.folds,
  grid = param_grid#,
  #metrics = metric_set(accuracy)#classification?
)
autoplot(tune_res)
```



```
#show_best(tune_res, metric = "accuracy")
```

```
#find the optimal `cp` and tree_depth using the "one-standard-error"
best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", -cost_complexity, -tree_depth)#, -
best.penalty
```

```
## # A tibble: 1 x 10
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1          0.0001         15 accuracy multiclass 0.844    10  0.0195 Preprocess~
## # ... with 2 more variables: .best <dbl>, .bound <dbl>
```

tree_depth = 15, cost_complexity = 0.0001

```
digits.final.wf <- finalize_workflow(digits.wflow, best.penalty)
digits.final.fit <- fit(digits.final.wf, train.12.tbl)

#not sure about this part. trying to create a split object.
custom_split <- make_splits(train.12.tbl, assessment = test.12.tbl)

digits.final.rs <- last_fit(digits.final.wf, custom_split)
collect_metrics(digits.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <fct>
## 1 accuracy multiclass     0.759 Preprocessor1_Model1
## 2 roc_auc  hand_till      0.893 Preprocessor1_Model1
```

accuracy = 0.759

accuracy is lower.

```
augment(digits.final.fit, test.12.tbl) %>%
  accuracy(truth = digit, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.759
```

```
augment(digits.final.fit, test.12.tbl) %>%
  conf_mat(truth = digit, estimate = .pred_class)
```

```
##           Truth
## Prediction 0 7 5
##          0 7 2 3
##          7 1 8 0
##          5 1 0 7
```

.metric .estimator .estimate 1 accuracy binary 0.759

```
      Truth
```

Prediction 0 7 5 0 7 2 3 7 1 8 0 5 1 0 7

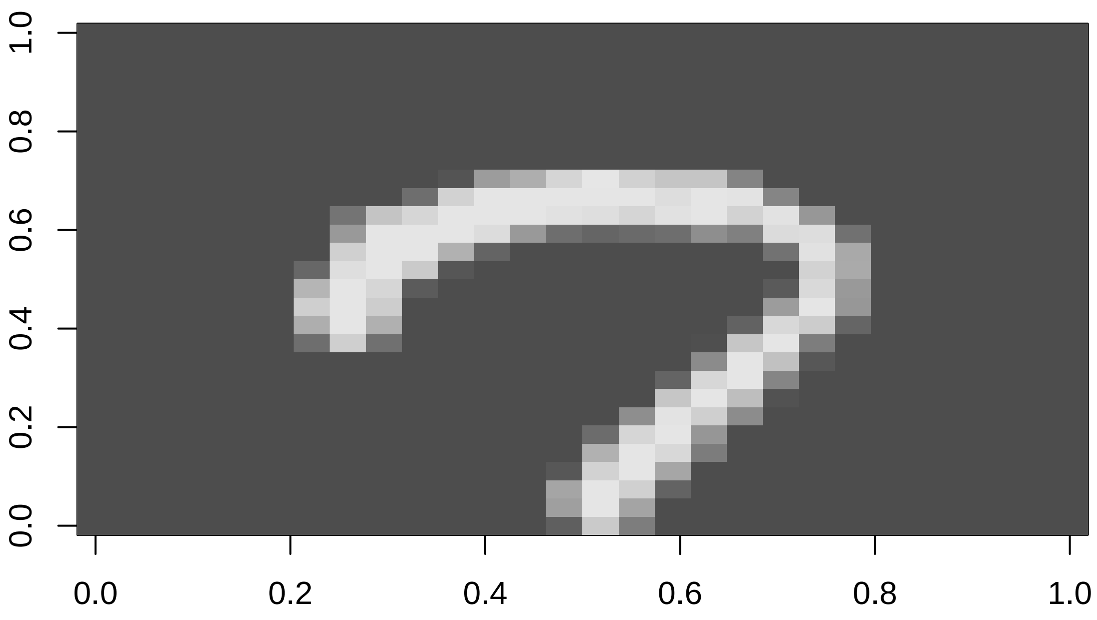
Plot a few digits that get misclassified:

```
#7 that got labeled as 0
index <- augment(digits.final.fit, test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(.pred_class=="0" & digit == "7") %>%
  arrange(desc(n)) %>%
  pull(n)
index
```

```
## [1] 28 11
```

28, 11

```
plot_row(test.12.tbl[28,])
```

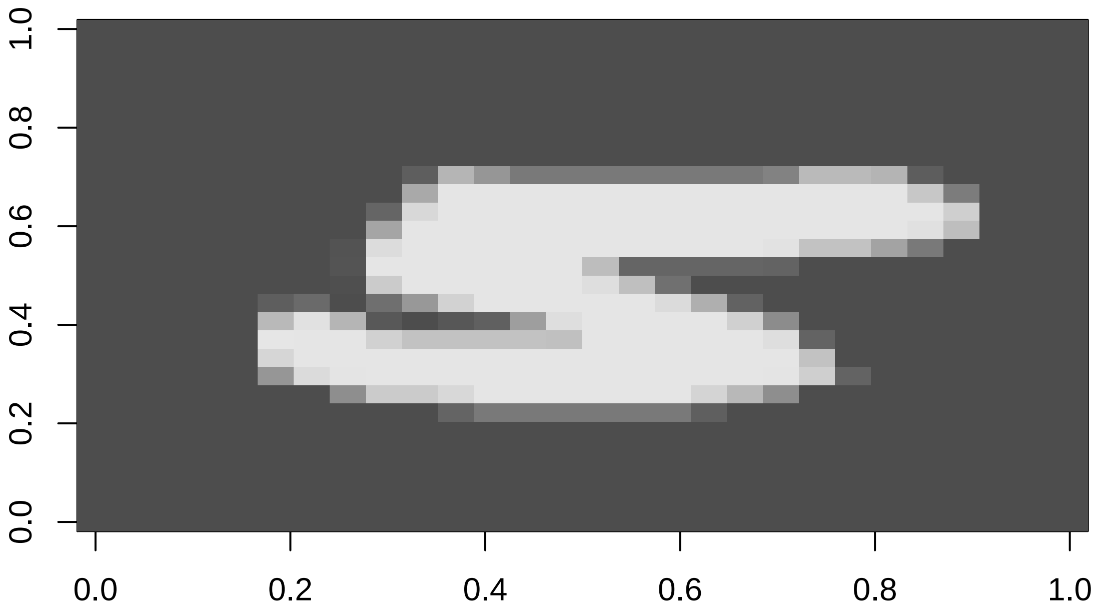


```
#5 that got labelled as a 0
index <- augment(digits.final.fit, test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(.pred_class=="0" & digit == "5") %>%
  arrange(desc(n)) %>%
  pull(n)
index
```

```
## [1] 15  9  6
```

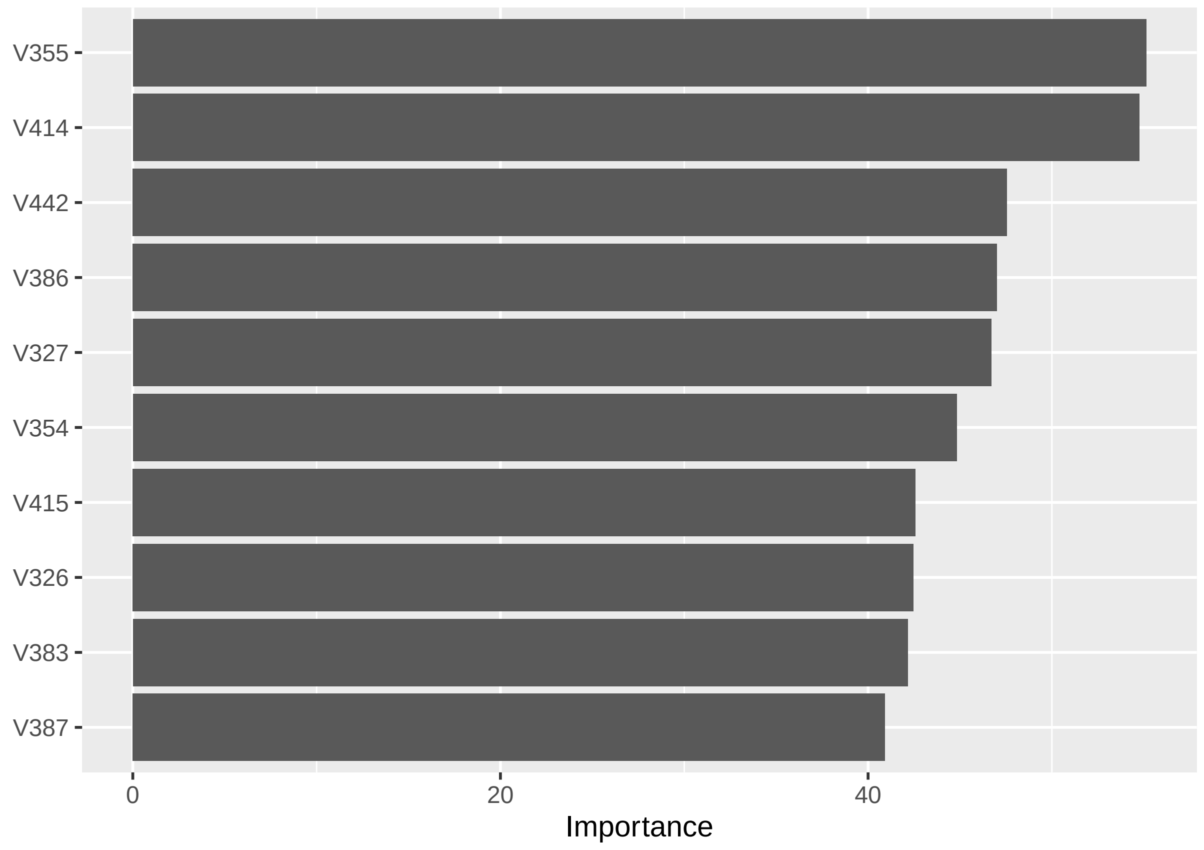
15, 9, 6

```
plot_row(test.12.tbl[15,])
```



Create an image with most important features used by your model:

```
digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vip()
```



```
imp.tbl <- digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

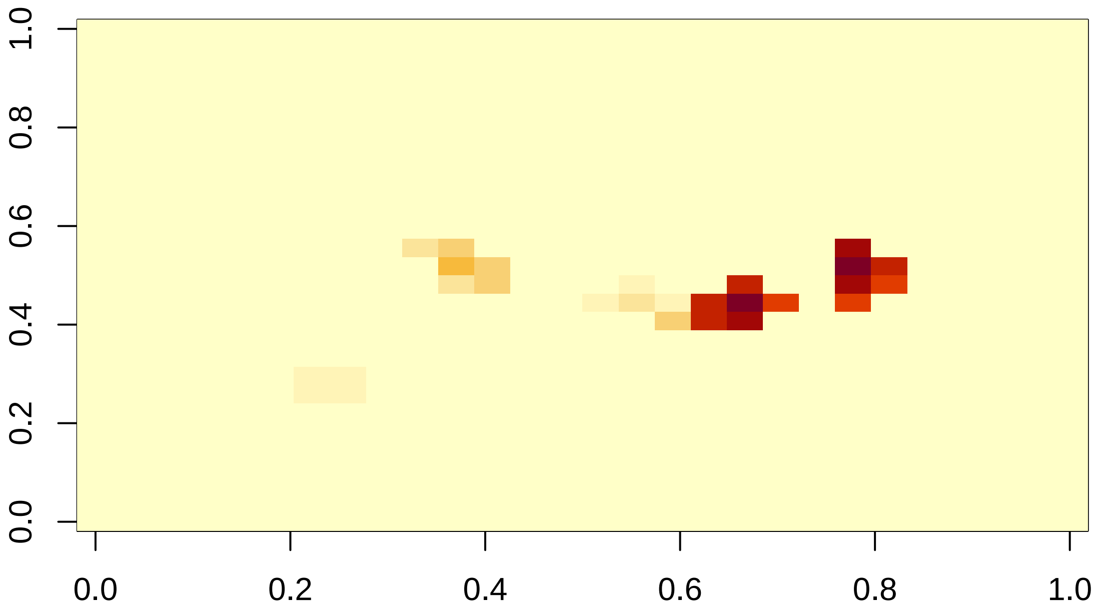
```
## # A tibble: 33 x 2
##    Variable Importance
##    <chr>         <dbl>
##  1 V355           55.1
##  2 V414           54.8
##  3 V442           47.6
##  4 V386           47.0
##  5 V327           46.7
##  6 V354           44.8
##  7 V415           42.6
##  8 V326           42.5
##  9 V383           42.2
## 10 V387           40.9
## # ... with 23 more rows
```

A tibble: 33 $\times$ 2

Variable Importance 1 V355 55.1 2 V414 54.8 3 V442 47.6 4 V386 47.0 5 V327 46.7 6 V354 44.8 7 V415 42.6 8 V326 42.5 9 V383 42.2 10 V387 40.9 # … with 23 more rows

(extra graph:)

```
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))#turning it into a number

mat <- rep(0, 28*28)
mat[imp.tbl$col] <- imp.tbl$Importance
image(matrix(mat, 28, 28))
```



8. This time train an optimal classification tree using `train.tbl` and evaluate using `test.tbl` for identifying the 10 digits by repeating the steps from exercise 6. What pairs of digits get confused the most? Plot a couple of them.

Create testing and training for 0 and 7 and 5:

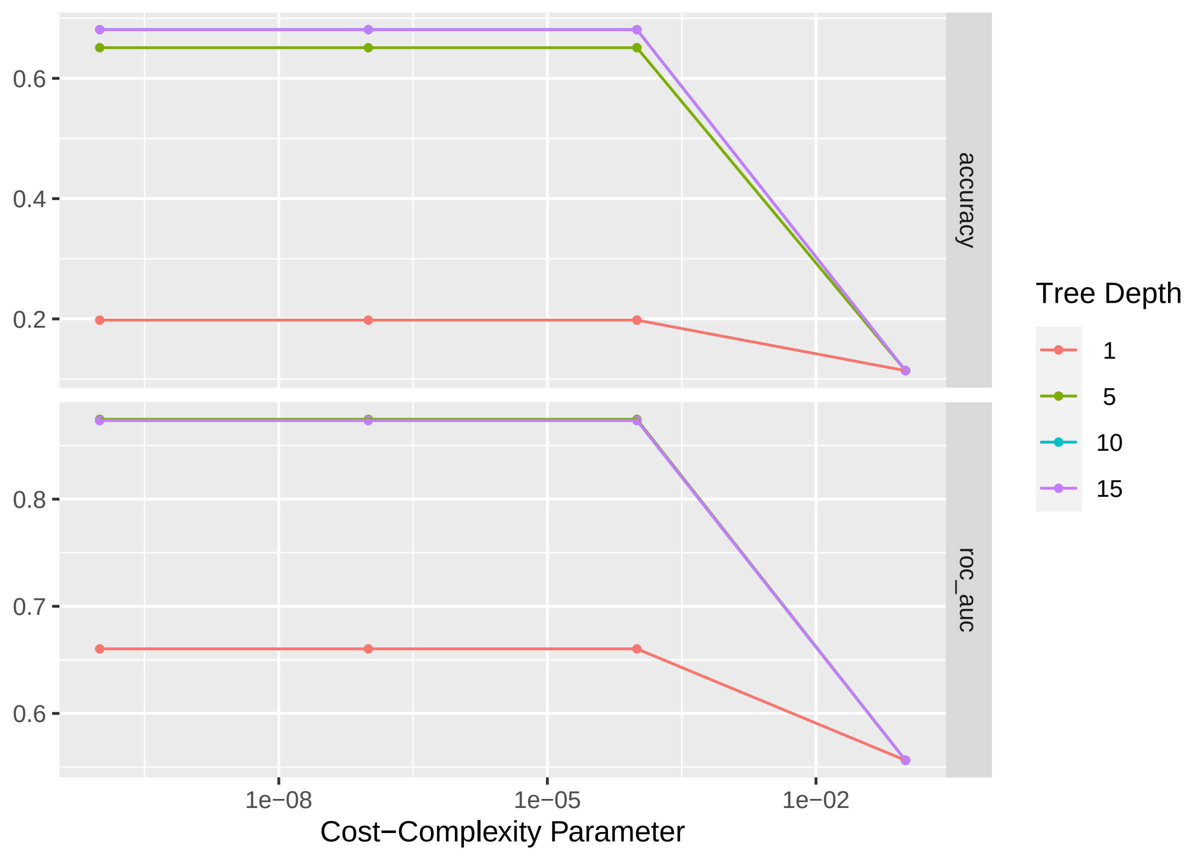
```
mnist <- read_mnist()
set.seed(2023)
index <- sample(nrow(mnist$train$images), 1000)#was 10000
train.tbl <- as_tibble (mnist$train$images[index,]) %>%
  mutate(digit = factor(mnist$train$labels[index]))
index <- sample(nrow(mnist$test$images), 100)#was 1000
test.tbl <- as_tibble (mnist$test$images[index,]) %>%
  mutate(digit = factor(mnist$test$labels[index]))

digits = c(0,1,2,3,4,5,6,7,8,9)
train.12.tbl = train.tbl %>%
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))
```

```r
test.12.tbl = test.tbl %>%
  filter(digit %in% digits) %>%
  mutate(digit = factor(digit, levels=digits))
```

```r
#general model stuff
digits.model <- decision_tree(cost_complexity = tune(), tree_depth = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

digits.recipe <- recipe(digit ~., data = train.12.tbl)

digits.wflow <- workflow() %>%
  add_recipe(digits.recipe) %>%
  add_model(digits.model)
```

```r
#cross validation stuff
param_grid <- grid_regular(cost_complexity(), tree_depth(), levels = 4)
# Create the cross-validation dataset
set.seed(31416)
digits.folds <- vfold_cv(train.12.tbl, v = 10)
```

```r
#calculate MSE
tune_res <- tune_grid(
  digits.wflow,
  resamples = digits.folds,
  grid = param_grid#,
  #metrics = metric_set(accuracy)#classification?
)
autoplot(tune_res)
```



```
#show_best(tune_res, metric = "accuracy")
```

```
#find the optimal `cp` and tree_depth using the "one-standard-error"
best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", -cost_complexity, -tree_depth)#, -
best.penalty
```

```
## # A tibble: 1 x 10
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1          0.0001         15 accuracy multiclass 0.681    10  0.0173 Preprocess~
## # ... with 2 more variables: .best <dbl>, .bound <dbl>
```

tree_depth = 15, cost_complexity = 0.0001

```
digits.final.wf <- finalize_workflow(digits.wflow, best.penalty)
digits.final.fit <- fit(digits.final.wf, train.12.tbl)

#not sure about this part. trying to create a split object.
custom_split <- make_splits(train.12.tbl, assessment = test.12.tbl)

digits.final.rs <- last_fit(digits.final.wf, custom_split)
collect_metrics(digits.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <fct>
## 1 accuracy multiclass     0.71  Preprocessor1_Model1
## 2 roc_auc  hand_till      0.886 Preprocessor1_Model1
```

accuracy = 0.71

accuracy is lower than with just 0 and 7 and 5.

```
augment(digits.final.fit, test.12.tbl) %>%
  accuracy(truth = digit, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass      0.71
```

```
augment(digits.final.fit, test.12.tbl) %>%
  conf_mat(truth = digit, estimate = .pred_class)
```

```
##           Truth
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0  6  0  1  0  0  0  0  1  0  0
##          1  0 12  0  0  0  0  0  0  1  0
##          2  0  0  3  0  0  0  1  1  0  0
##          3  2  1  0  5  0  1  0  0  0  0
##          4  0  0  0  0  9  2  1  0  0  1
##          5  0  0  0  3  0  4  0  0  0  0
##          6  1  0  1  0  0  1 10  0  1  0
##          7  0  0  0  0  0  0  0  8  0  0
##          8  0  1  0  3  0  1  1  0  9  0
##          9  0  0  0  0  0  1  1  0  1  5
```

.metric .estimator .estimate 1 accuracy binary 0.759

```
      Truth
```

Prediction 0 1 2 3 4 5 6 7 8 9 0 6 0 1 0 0 0 0 1 0 0 1 0 12 0 0 0 0 0 0 1 0 2 0 0 3 0 0 0 1 1 0 0 3 2 1 0 5 0 1 0 0 0 0 4 0 0 0 0 9 2 1 0 0 1 5 0 0 0 3 0 4 0 0 0 0 6 1 0 1 0 0 1 10 0 1 0 7 0 0 0 0 0 0 0 8 0 0 8 0 1 0 3 0 1 1 0 9 0 9 0 0 0 0 0 1 1 0 1 5

commonly confused pairs: 3 confused as 8, 3 confused as 5, 0 confused as 3.

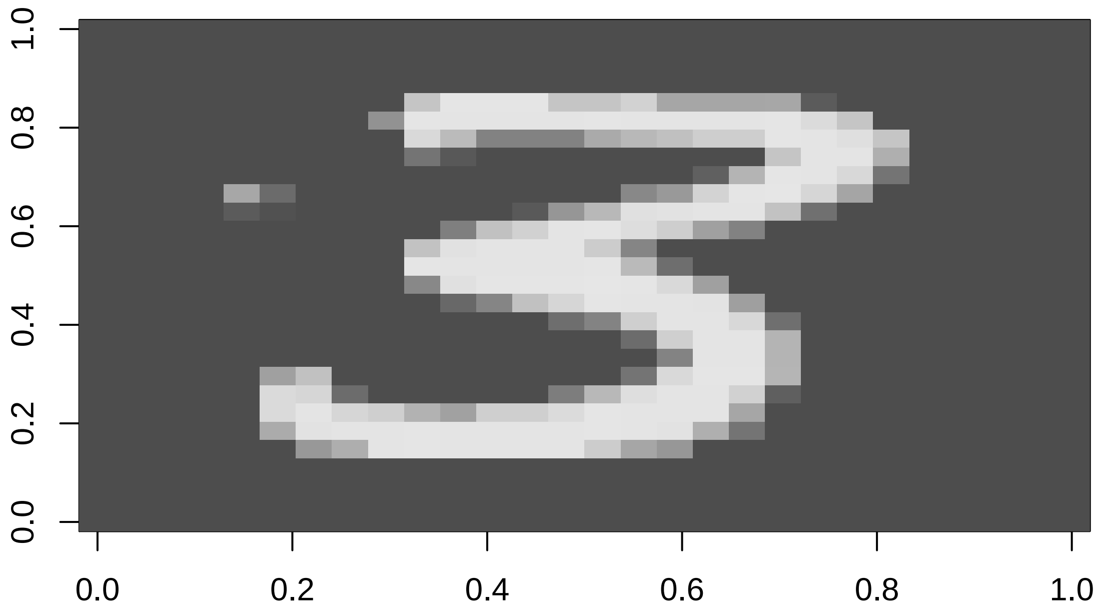
Plot a few digits that get misclassified:

```
#3 that got mislabelled as not 3 (which in the confusion matrix aobve is 5 or 8)
index <- augment(digits.final.fit, test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(.pred_class!=digit & digit == "3") %>%
  #filter(digit == "3" | digit == "8" | digit == "5") %>%
  arrange(desc(n)) %>%
  pull(n)
index
```

```
## [1] 85 59 49 38 12  3
```

85 59 49 38 12 3

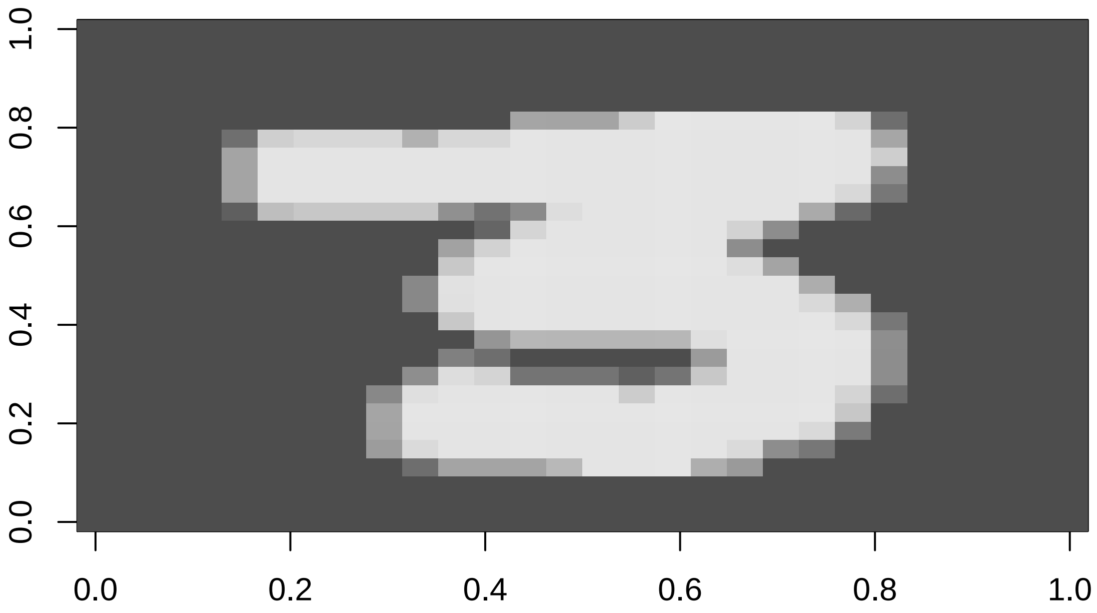
```
#3 mistaken for 8 (the image looks like a 3 but pred_class is 8)
plot_row(test.12.tbl[3,])
```



```
augment(digits.final.fit, test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(n == 3) %>%
  pull(.pred_class)
```

```
## [1] 8
## Levels: 0 1 2 3 4 5 6 7 8 9
```

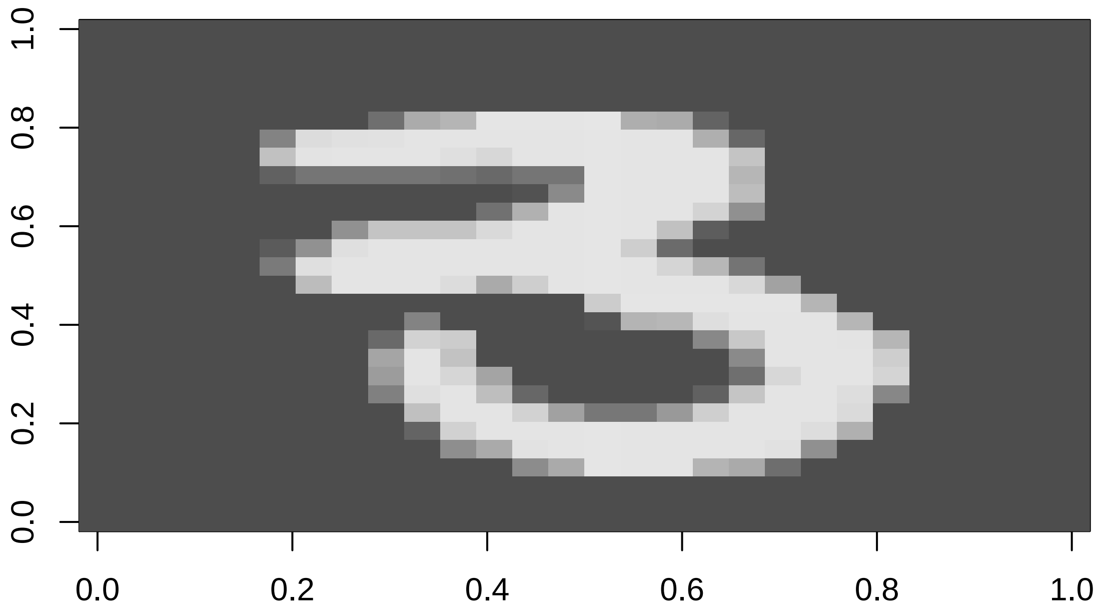
```
#3 mistaken for 8
plot_row(test.12.tbl[12,])
```



```
augment(digits.final.fit, test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(n == 12) %>%
  pull(.pred_class)
```

```
## [1] 8
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
#3 mistaken for 5
plot_row(test.12.tbl[38,])
```

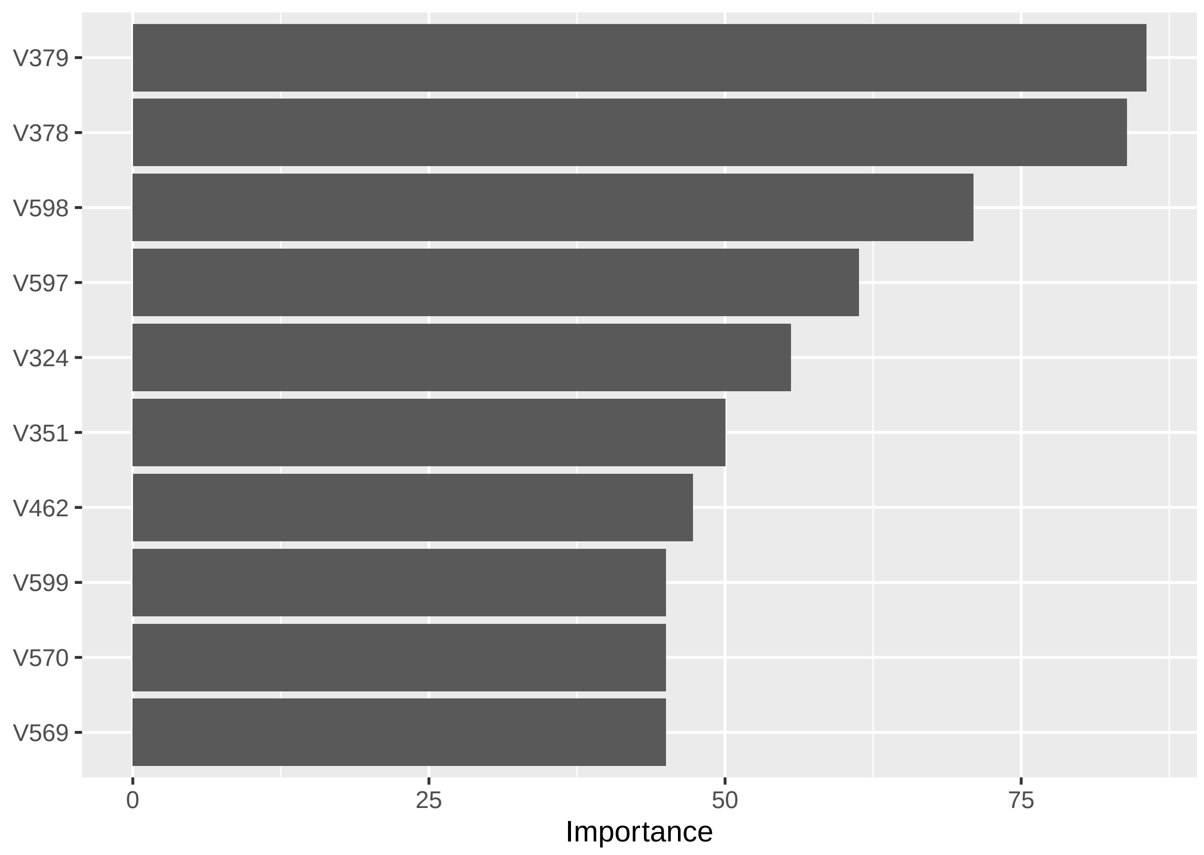


```r
augment(digits.final.fit, test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(n == 38) %>%
  pull(.pred_class)
```

```
## [1] 5
## Levels: 0 1 2 3 4 5 6 7 8 9
```

Create an image with most important features used by your model:

```r
digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vip()
```



```r
imp.tbl <- digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

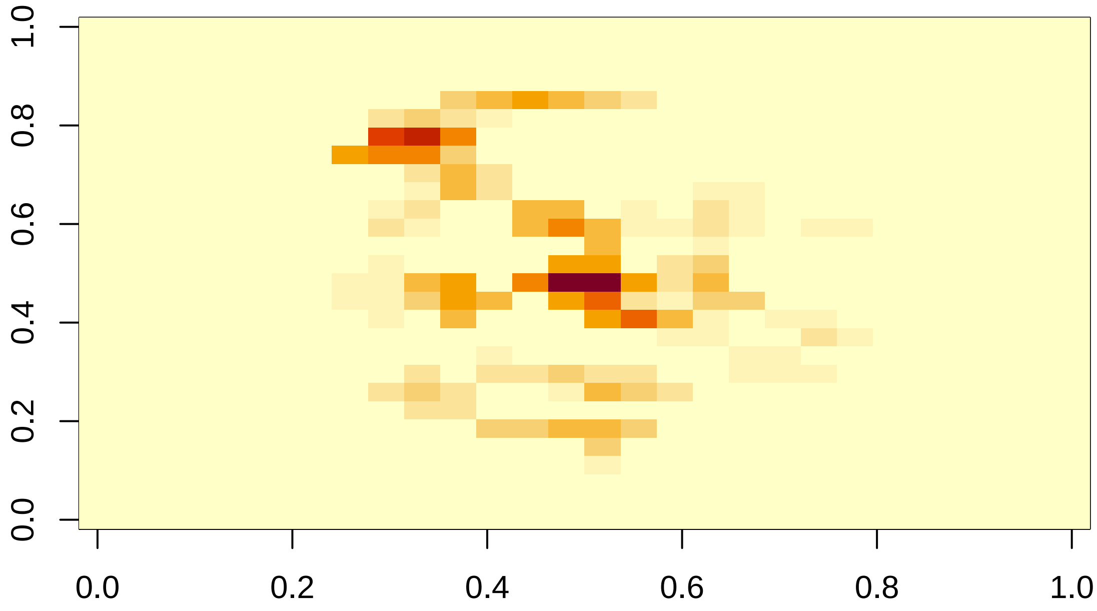
```
## # A tibble: 161 x 2
##    Variable Importance
##    <chr>         <dbl>
##  1 V379           85.6
##  2 V378           83.9
##  3 V598           71.0
##  4 V597           61.3
##  5 V324           55.6
##  6 V351           50.0
##  7 V462           47.3
##  8 V569           45.0
##  9 V570           45.0
## 10 V599           45.0
## # ... with 151 more rows
```

A tibble: 161 $\times$ 2 Variable Importance 1 V379 85.6 2 V378 83.9 3 V598 71.0 4 V597 61.3 5 V324 55.6 6 V351 50.0 7 V462 47.3 8 V569 45.0 9 V570 45.0 10 V599 45.0 … with 151 more rows

(extra graph:)

```r
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))#turning it into a number
```

```r
mat <- rep(0, 28*28)
mat[imp.tbl$col] <- imp.tbl$Importance
image(matrix(mat, 28, 28))
```



9. Same as exercise 8, but this time try a ridge model. Don't forget to optimize the penalty parameter.

Notes for myself (code box may or may not be knitted into the pdf)

```r
#ridge.model <- linear_reg(penalty = tune(), mixture = 0) %>%
#  set_mode("regression") %>%
#  set_engine("lm")#glmnet
#*regression? I notice lasso can use multinom reg, logistic_reg, etc. (3_lasso_classification.key). Rid
#*I also notice lasso has classification and regression modes. Ridge only has regression.*
#think of these as the same model (ridge and lasso). Only difference is the value mixture takes on. The

#difference between glmnet and lm? (explanation is in the help box. basically, glmnet is a little more
```

Use 10-fold cross-validation to find the optimal λ for this model: Create a 10 fold cross validation tibble

```r
digits.fold <- vfold_cv(train.12.tbl, v = 10)
```

Create a ridge model specifying that the `penalty` will be **tuned**

```r
ridge.model <-
  multinom_reg(mixture = 0, penalty=tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

```

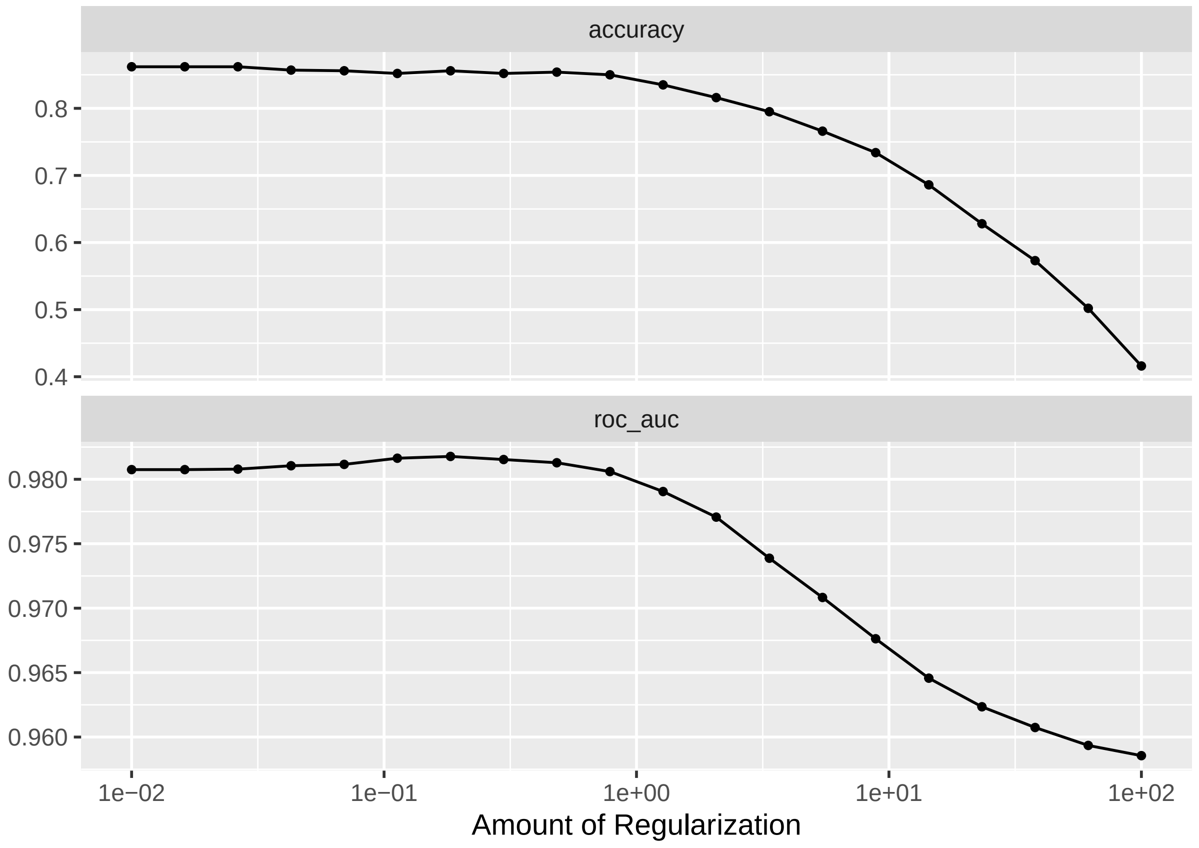
```
ridge.recipe <-
  recipe(formula = digit ~ ., data = train.12.tbl) %>%
  step_zv(all_predictors())
  #step_dummy(all_nominal_predictors()) %>%
  #step_normalize(all_predictors())

ridge.wf <- workflow() %>%
  add_recipe(ridge.recipe) %>%
  add_model(ridge.model)
```

Create a penalty grid that takes 20 values between -2 to 2

```
penalty.grid <-
  grid_regular(penalty(range = c(-2, 2)), levels = 20)
```

Use `tune_grid()` to evaluate the model on the different parameters of the grid and plot the effect of the parameter on the fit of the model.

```
tune_res <- tune_grid(
  ridge.wf,
  resamples = digits.fold,
  grid = penalty.grid
)
autoplot(tune_res)
```



Use 10-fold cross validation and `select_by_one_std_err()` to determine the optimal penalty.

```
#Note: class example from 3_lasso_classification.key uses 5 levels, not 10.

(best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", desc(penalty)))
```

```
## # A tibble: 1 x 9
##   penalty .metric  .estimator  mean     n std_err .config             .best .bound
##     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>               <dbl>  <dbl>
## 1   0.483 accuracy multiclass 0.854    10 0.00581 Preprocessor1_Mo~ 0.862  0.853
```

```
digit.final.wf <- finalize_workflow(ridge.wf, best.penalty)

digit.final.fit <- fit(digit.final.wf, data = train.12.tbl)

augment(digit.final.fit, new_data = test.12.tbl) %>%
  accuracy(truth = digit, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass      0.83
```

```
augment(digit.final.fit, new_data = test.12.tbl) %>%
  conf_mat(truth = digit, estimate = .pred_class)
```

```
##           Truth
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0  9  0  0  0  0  0  0  0  0  0
##          1  0 13  0  0  0  0  0  0  1  0
##          2  0  0  5  0  1  0  0  1  0  0
##          3  0  1  0 10  0  0  0  0  1  0
##          4  0  0  0  0  7  0  1  1  0  0
##          5  0  0  0  1  0  6  0  0  2  0
##          6  0  0  0  0  0  0 12  0  0  0
##          7  0  0  0  0  1  0  0  8  0  0
##          8  0  0  0  0  0  2  1  0  7  0
##          9  0  0  0  0  0  2  0  0  1  6
```

optimal penalty: lambda = 0.298 (mean accuracy for folds is 0.854 I think based on the output for best.penalty).

accuracy on testing dataset is 0.83

```
      Truth
```

Prediction 0 1 2 3 4 5 6 7 8 9 0 9 0 0 0 0 0 0 0 0 0 1 0 13 0 0 0 0 0 0 1 0 2 0 0 5 0 1 0 0 1 0 0 3 0 1 0 10 0 0 0 0 0 0 4 0 0 0 0 8 0 1 1 0 0 5 0 0 0 1 0 6 0 0 2 0 6 0 0 0 0 0 0 12 0 0 0 7 0 0 0 0 0 0 0 8 0 0 8 0 0 0 0 0 2 1 0 8 0 9 0 0 0 0 0 2 0 0 1 6

Which pairs of digits get confused the most: Most commonly confused: 5 is confused for 8, and 5 is confused for 9. 8 is confused as a 5. (9 is not confused for a 5). There are other pairs that are confused once. These pairs are confused two times.

Plot a few digits that get misclassified:

```
#5 that got mislabelled as not 5 (which in the confusion matrix aobve is 8 or 9)
index <- augment(digit.final.fit, new_data = test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(.pred_class!=digit & digit == "5") %>%
  filter(digit == "5" | digit == "8" | digit == "9") %>%
  arrange(desc(n)) %>%
  pull(n)
index
```

```
## [1] 89 39 28 20
```

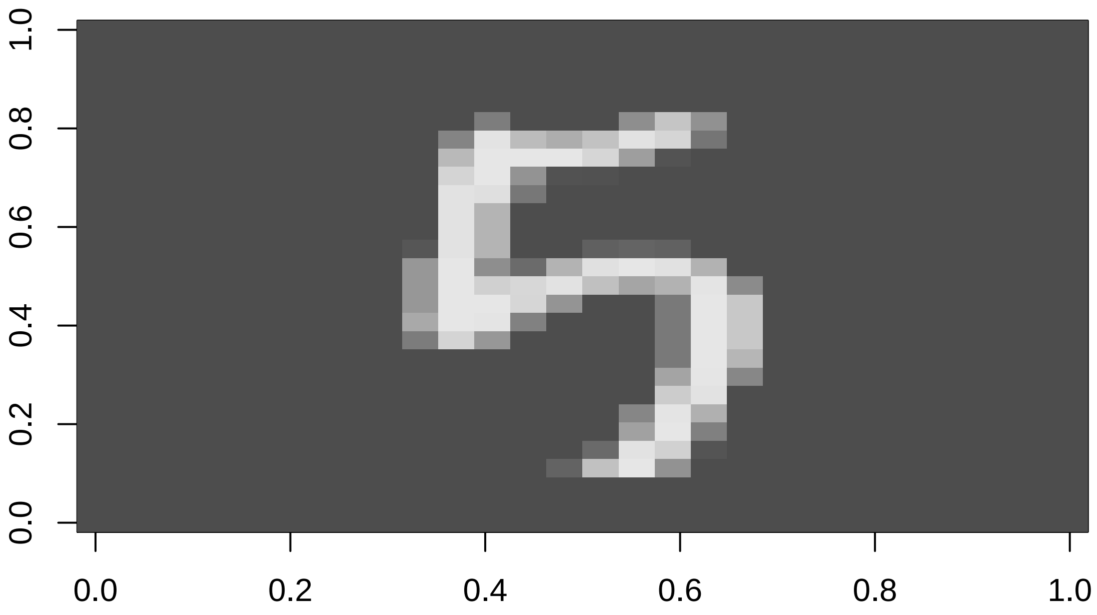
89 39 28 20

```
augment(digit.final.fit, new_data = test.12.tbl)%>%
  mutate(n = row_number()) %>%
  filter(n == 20) %>%
  select(.pred_class, digit)
```

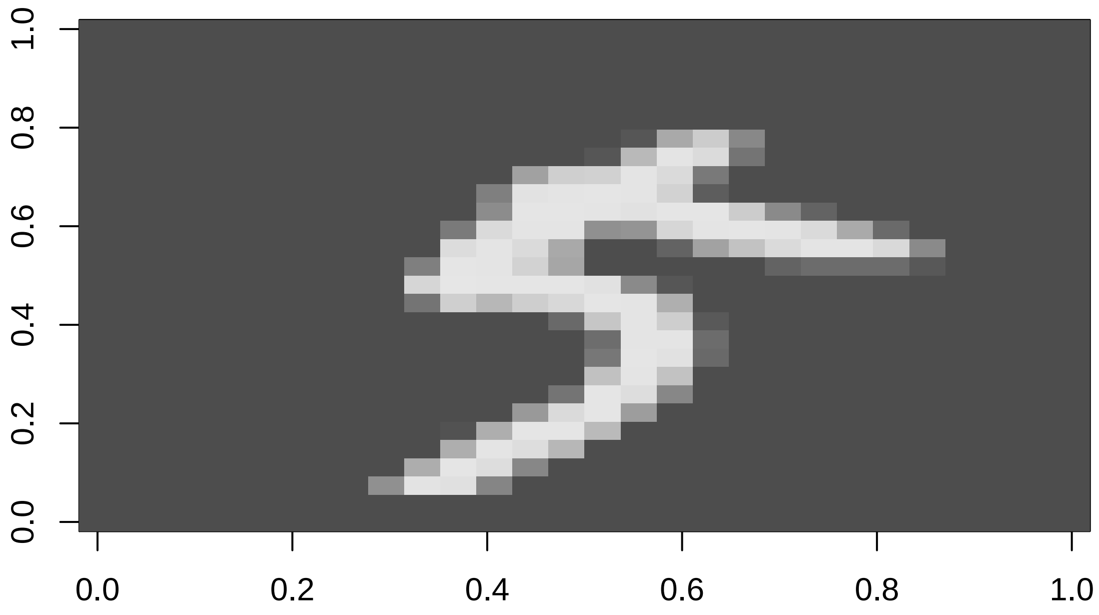
```
## # A tibble: 1 x 2
##   .pred_class digit
##   <fct>       <fct>
## 1 9           5
```

for n = 89: digit is 5 and .pred_class is 9. for n = 39: digit is 5 and .pred_class is 8. for n = 28: digit is 5 and .pred_class is 8. for n = 20: digit is 5 and .pred_class is 9.

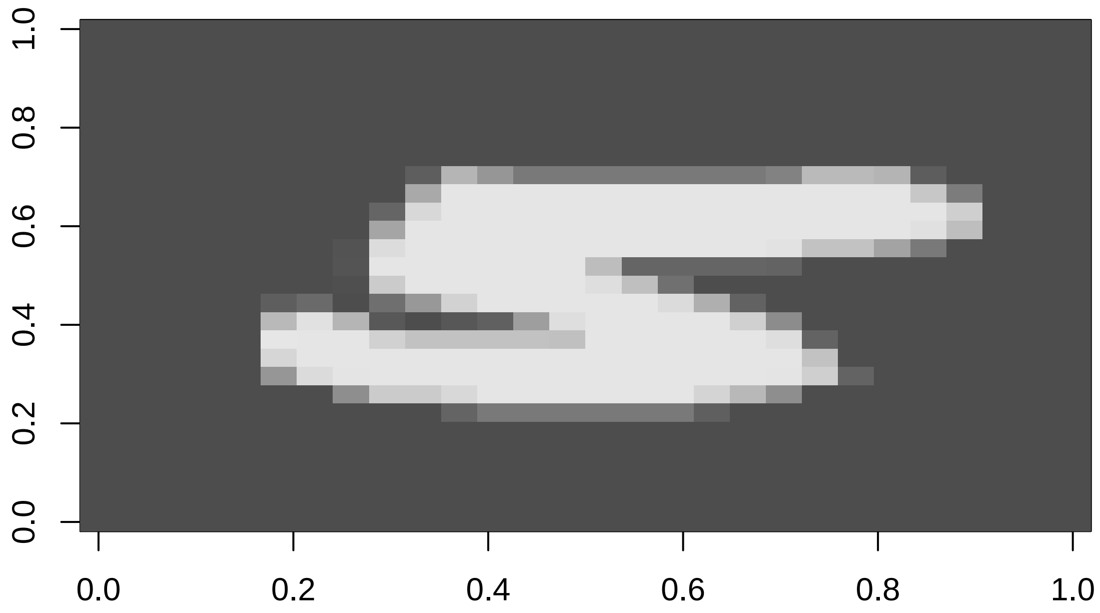
```
#digit is 5 and .pred_class is 9.
plot_row(test.12.tbl[89,])
```



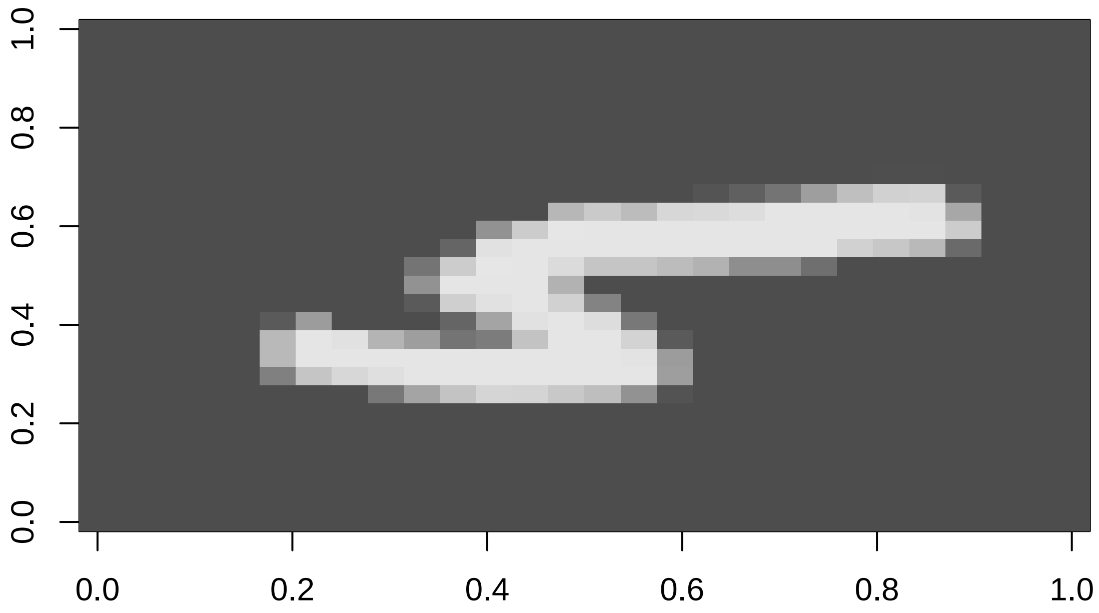
```
#digit is 5 and .pred_class is 9.
plot_row(test.12.tbl[20,])
```



```
#digit is 5 and .pred_class is 8.
plot_row(test.12.tbl[39,])
```



```
#digit is 5 and .pred_class is 8.
plot_row(test.12.tbl[28,])
```



Create an image with most important features used by your model: (and get accuracy again)

```
(best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", desc(penalty)))#same as earlier a
```

```
## # A tibble: 1 x 9
##   penalty .metric  .estimator  mean     n std_err .config           .best .bound
##     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>             <dbl>  <dbl>
## 1   0.483 accuracy multiclass 0.854    10 0.00581 Preprocessor1_Mo~ 0.862  0.853
```

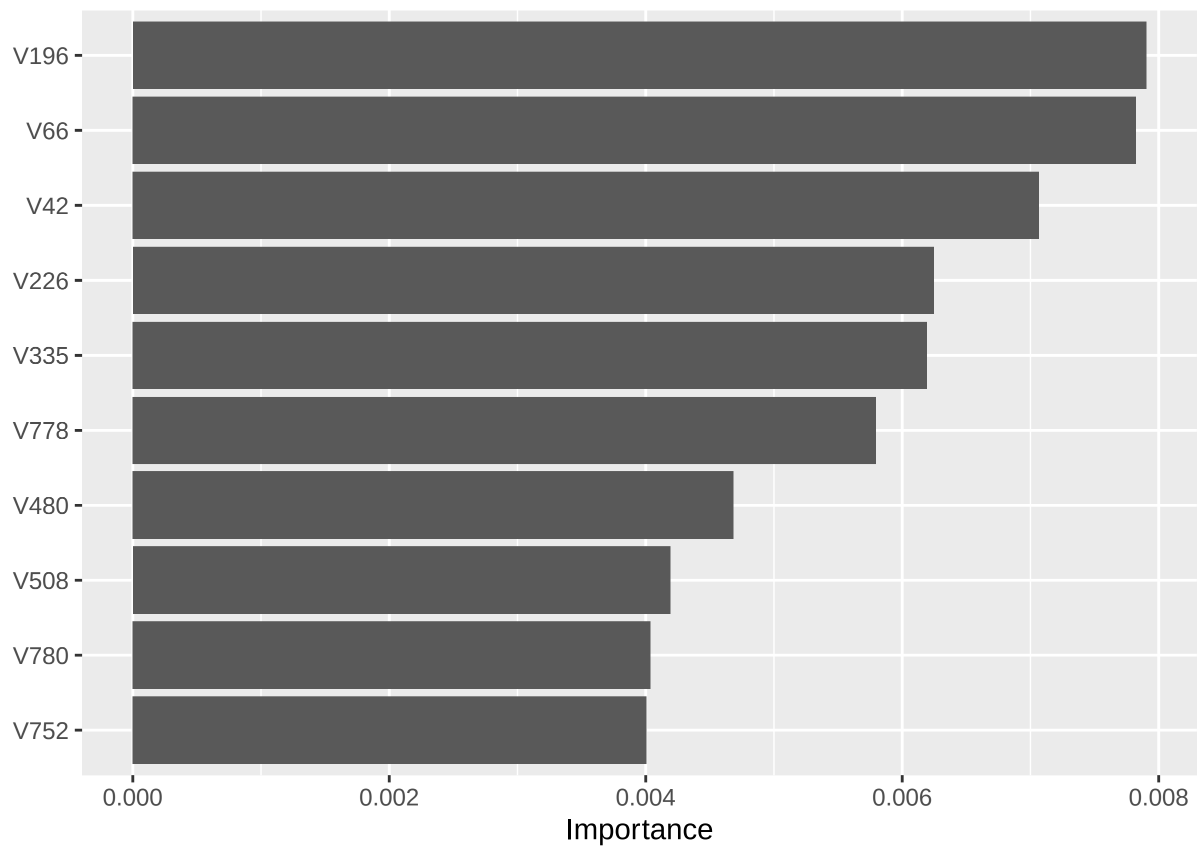
```
digits.final.wf <- finalize_workflow(ridge.wf, best.penalty)#same as earlier accuracy calculation
digits.final.fit <- fit(digits.final.wf, train.12.tbl)#same as earlier accuracy calculation

#not sure about this part. trying to create a split object.
custom_split <- make_splits(train.12.tbl, assessment = test.12.tbl)

digits.final.rs <- last_fit(digits.final.wf, custom_split)
collect_metrics(digits.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <fct>
## 1 accuracy multiclass     0.83  Preprocessor1_Model1
## 2 roc_auc  hand_till      0.970 Preprocessor1_Model1
```

accuracy on test dataset is 0.83.

Create an image with most important features used by your model:

```
digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vip()
```



```
imp.tbl <- digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

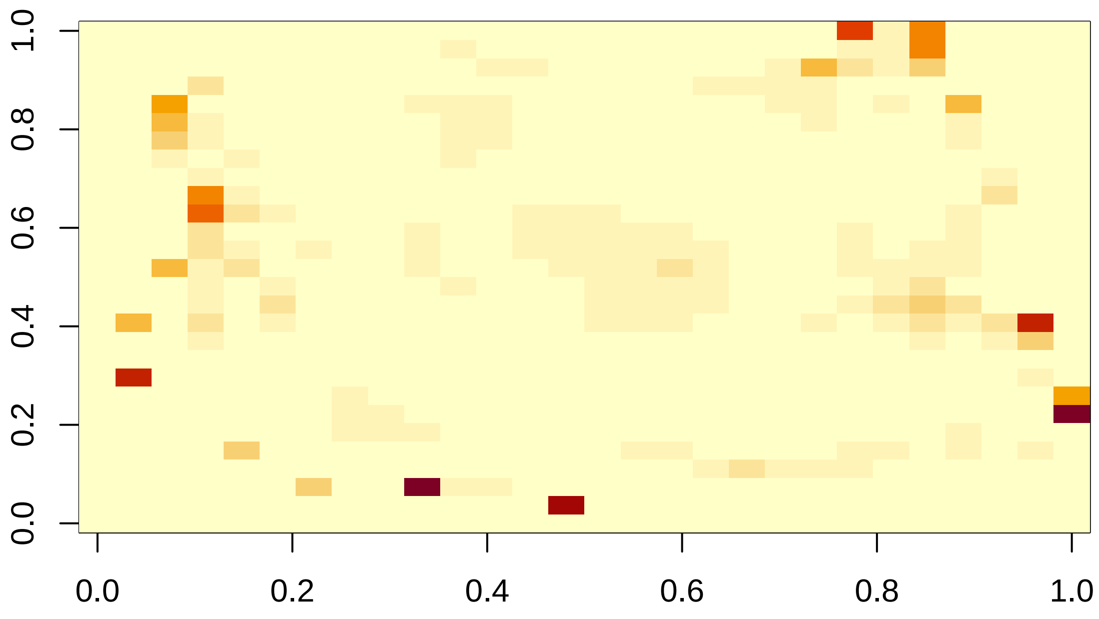
```
## # A tibble: 617 x 3
##    Variable Importance Sign
##    <chr>         <dbl> <chr>
##  1 V196        0.00790 POS
##  2 V66         0.00782 NEG
##  3 V42         0.00707 NEG
##  4 V226        0.00625 NEG
##  5 V335        0.00619 NEG
##  6 V778        0.00579 NEG
##  7 V480        0.00468 NEG
##  8 V508        0.00419 NEG
##  9 V780        0.00404 NEG
## 10 V752        0.00401 NEG
## # ... with 607 more rows
```

Variable Importance Sign 1 V196 0.00790 POS
2 V66 0.00782 NEG
3 V42 0.00707 NEG

```
4 V226 0.00625 NEG
5 V335 0.00619 NEG
6 V778 0.00579 NEG
7 V480 0.00468 NEG
8 V508 0.00419 NEG
9 V780 0.00404 NEG
10 V752 0.00401 NEG
... with 607 more rows
```

(extra graph:)

```r
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))#turning it into a number

mat <- rep(0, 28*28)
mat[imp.tbl$col] <- imp.tbl$Importance
image(matrix(mat, 28, 28))
```



Much more variation in which variables are the most important.

10. Same as exercises 8 and 9, but this time use a LASSO model. Compare and contrast the accuracy of the 3 approaches and the images corresponding to the most important features for the 3 approaches.

Use 10-fold cross-validation to find the optimal λ for this model: Create a 10 fold cross validation tibble

```r
digits.fold <- vfold_cv(train.12.tbl, v = 10)
```

Create a ridge model specifying that the `penalty` will be **tuned**

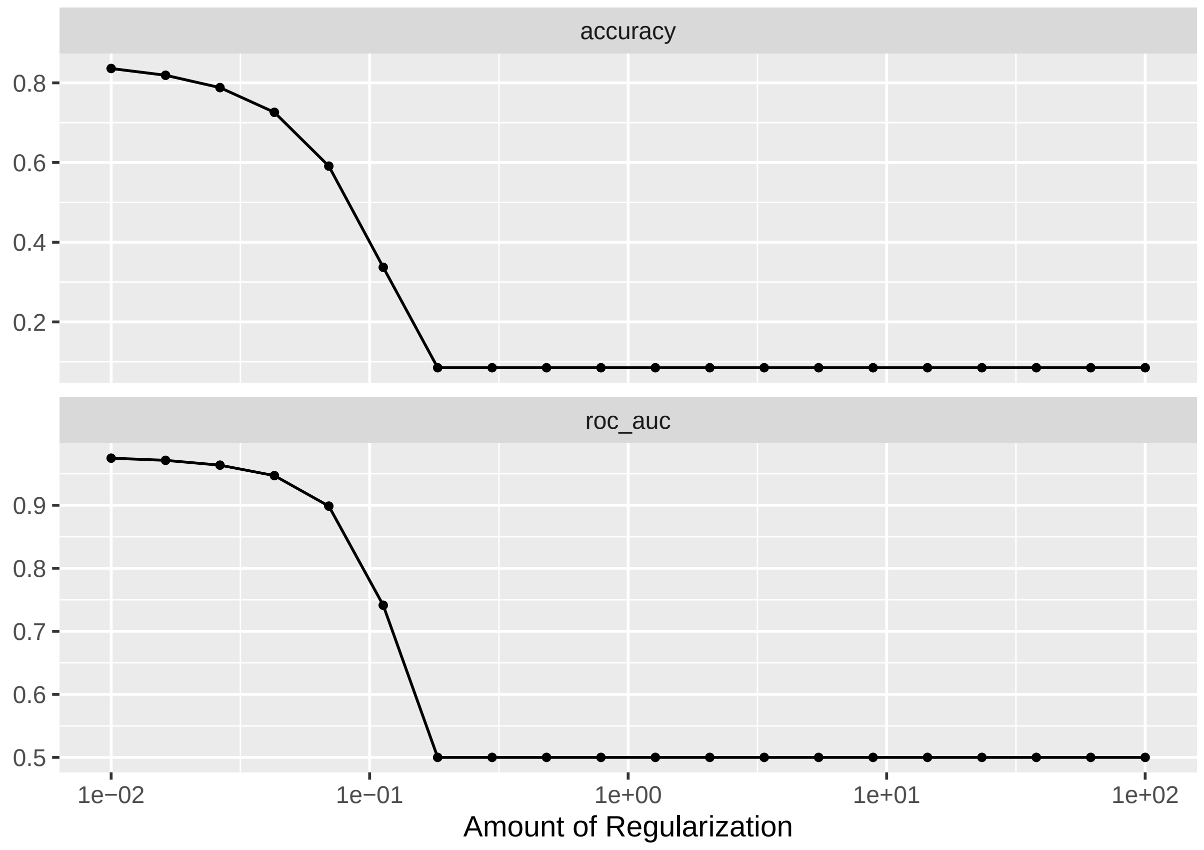
```
ridge.model <-
  multinom_reg(mixture = 1, penalty=tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

ridge.recipe <-
  recipe(formula = digit ~ ., data = train.12.tbl) %>%
  step_zv(all_predictors())
  #step_dummy(all_nominal_predictors()) %>%
  #step_normalize(all_predictors())

ridge.wf <- workflow() %>%
  add_recipe(ridge.recipe) %>%
  add_model(ridge.model)
```

Create a penalty grid that takes 20 values between -2 to 2

```
penalty.grid <-
  grid_regular(penalty(range = c(-2, 2)), levels = 20)
```

Use `tune_grid()` to evaluate the model on the different parameters of the grid and plot the effect of the parameter on the fit of the model.

```
tune_res <- tune_grid(
  ridge.wf,
  resamples = digits.fold,
  grid = penalty.grid
)
autoplot(tune_res)
```



Use 10-fold cross validation and `select_by_one_std_err()` to determine the optimal penalty.

```
#Note: class example from 3_lasso_classification.key uses 5 levels, not 10.

(best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", desc(penalty)))
```

```
## # A tibble: 1 x 9
##   penalty .metric  .estimator  mean     n std_err .config            .best .bound
##     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>              <dbl>  <dbl>
## 1  0.0162 accuracy multiclass 0.819    10  0.0202 Preprocessor1_Mo~ 0.836  0.815
```

```
digit.final.wf <- finalize_workflow(ridge.wf, best.penalty)

digit.final.fit <- fit(digit.final.wf, data = train.12.tbl)

augment(digit.final.fit, new_data = test.12.tbl) %>%
  accuracy(truth = digit, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass      0.84
```

```
augment(digit.final.fit, new_data = test.12.tbl) %>%
  conf_mat(truth = digit, estimate = .pred_class)
```

```
##            Truth
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0  9  0  1  0  0  1  0  0  0  0
##          1  0 13  0  0  0  0  0  0  1  0
##          2  0  0  4  0  0  0  0  1  0  0
##          3  0  1  0 10  0  0  0  0  1  0
##          4  0  0  0  0  9  1  1  1  0  1
##          5  0  0  0  1  0  6  0  0  1  0
##          6  0  0  0  0  0  0 12  0  0  0
##          7  0  0  0  0  0  0  0  8  0  0
##          8  0  0  0  0  0  2  1  0  8  0
##          9  0  0  0  0  0  0  0  0  1  5
```

optimal penalty: lambda = 0.01 (mean accuracy for folds is 0.833 I think based on the output for best.penalty).

accuracy on testing dataset is 0.85

```
      Truth
```

Prediction 0 1 2 3 4 5 6 7 8 9 0 9 0 1 0 0 1 0 0 0 0 1 0 13 0 0 0 0 0 0 1 0 2 0 0 4 0 0 0 0 1 0 0 3 0 1 0 10 0 0 0 0 1 0 4 0 0 0 0 9 1 1 1 0 1 5 0 0 0 1 0 6 0 0 1 0 6 0 0 0 0 0 0 13 0 0 0 7 0 0 0 0 0 0 0 8 0 0 8 0 0 0 0 0 2 0 0 8 0 9 0 0 0 0 0 0 0 0 1 5

Which pairs of digits get confused the most: 5 gets confused as an 8 twice. The other numbers get confused once.

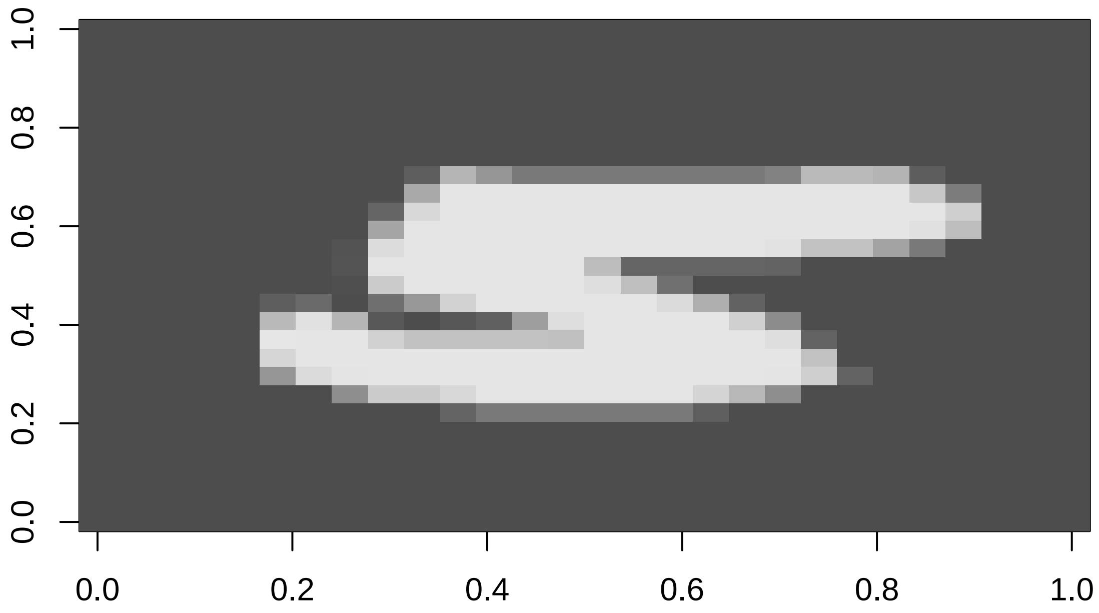
Plot a few digits that get misclassified:

```
#5 that got mislabelled as an 8
index <- augment(digit.final.fit, new_data = test.12.tbl) %>%
  mutate(n = row_number()) %>%
  filter(.pred_class!=digit & digit == "5") %>%
  filter(.pred_class == "8") %>%
  arrange(desc(n)) %>%
  pull(n)
index
```

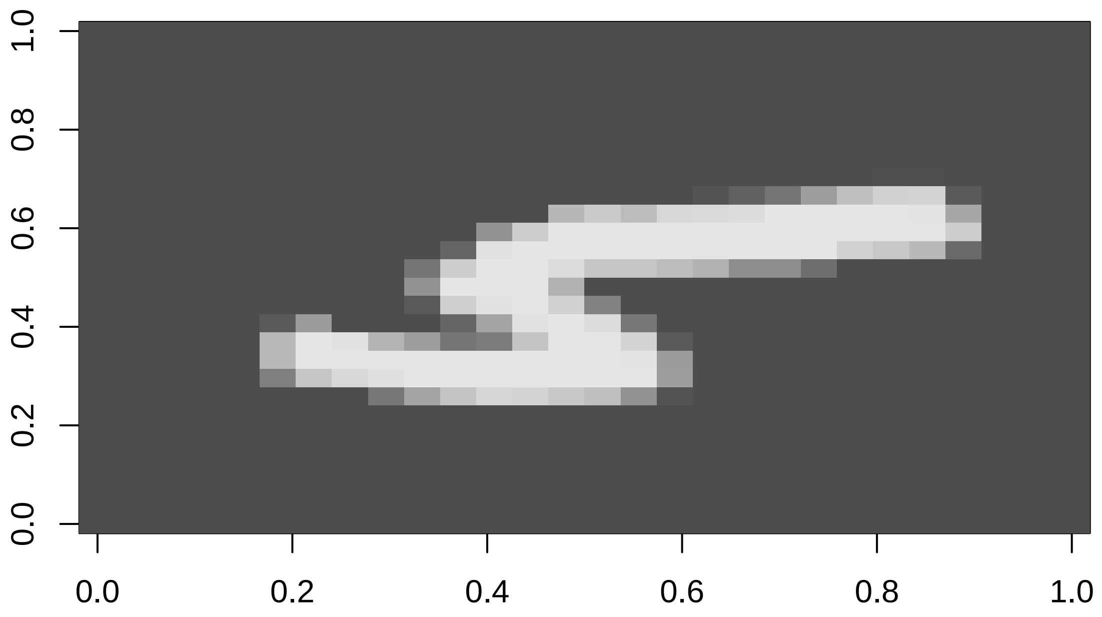
```
## [1] 39 28
```

39 28

```
plot_row(test.12.tbl[39,])
```



```
plot_row(test.12.tbl[28,])
```



Create an image with most important features used by your model: (and get accuracy again)

```
(best.penalty <- select_by_one_std_err(tune_res, metric = "accuracy", desc(penalty)))#same as earlier a
```

```
## # A tibble: 1 x 9
##   penalty .metric  .estimator  mean     n std_err .config              .best .bound
##     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>                <dbl>  <dbl>
## 1  0.0162 accuracy multiclass 0.819    10  0.0202 Preprocessor1_Mo~ 0.836  0.815
```

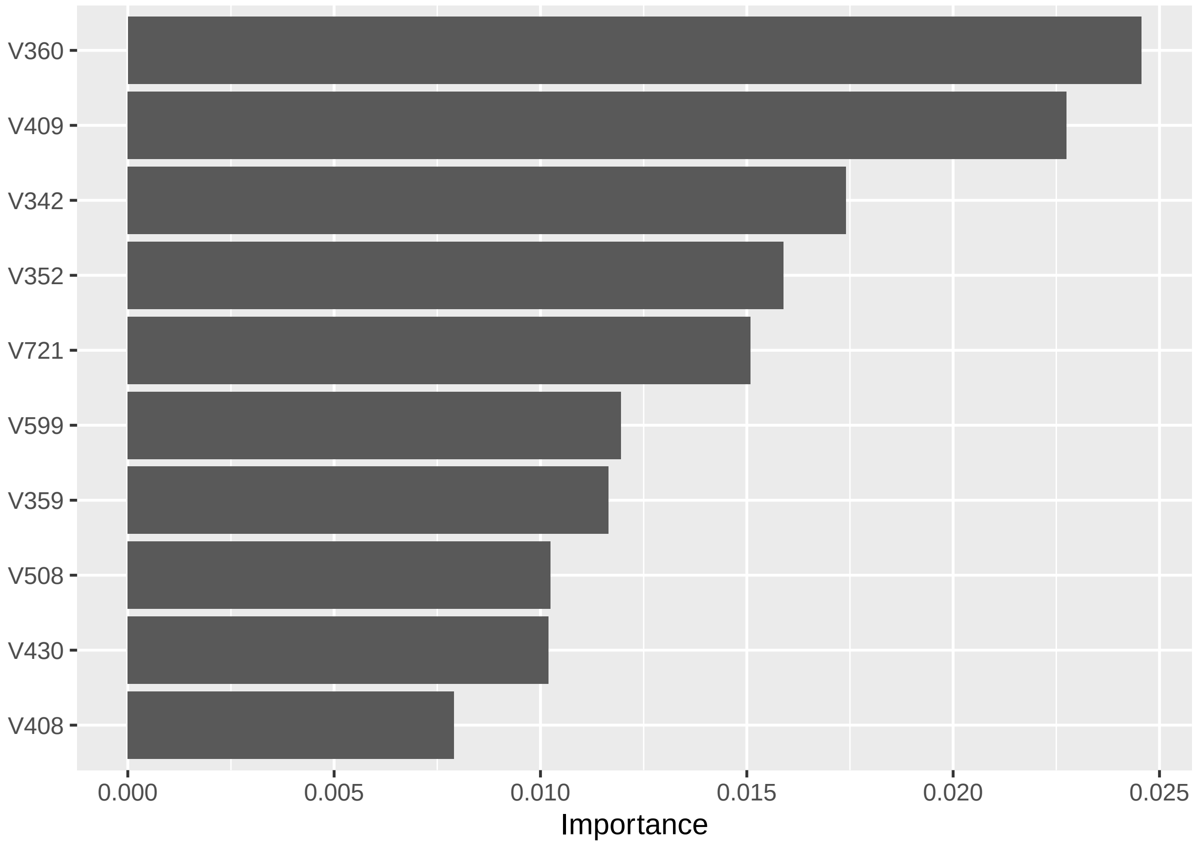
```
digits.final.wf <- finalize_workflow(ridge.wf, best.penalty)#same as earlier accuracy calculation
digits.final.fit <- fit(digits.final.wf, train.12.tbl)#same as earlier accuracy calculation

#not sure about this part. trying to create a split object.
custom_split <- make_splits(train.12.tbl, assessment = test.12.tbl)

digits.final.rs <- last_fit(digits.final.wf, custom_split)
collect_metrics(digits.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <fct>
## 1 accuracy multiclass     0.84  Preprocessor1_Model1
## 2 roc_auc  hand_till      0.966 Preprocessor1_Model1
```

accuracy on test dataset is 0.85

Create an image with most important features used by your model:

```
digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vip()
```



```
imp.tbl <- digits.final.rs %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

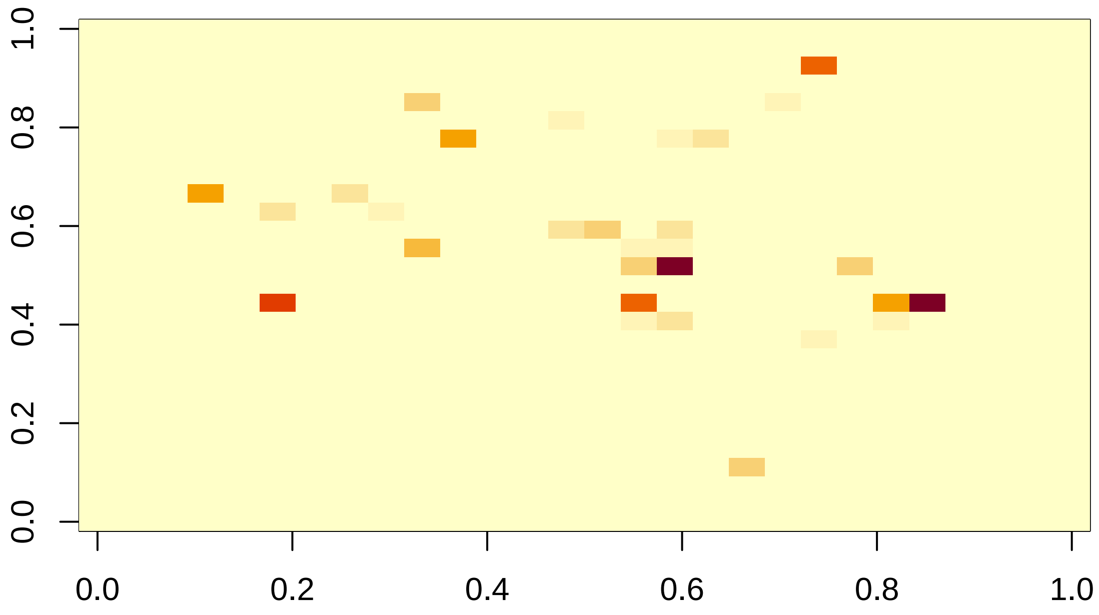
```
## # A tibble: 617 x 3
##    Variable Importance Sign
##    <chr>         <dbl> <chr>
##  1 V360        0.0246  POS
##  2 V409        0.0227  NEG
##  3 V342        0.0174  POS
##  4 V352        0.0159  NEG
##  5 V721        0.0151  NEG
##  6 V599        0.0120  POS
##  7 V359        0.0117  POS
##  8 V508        0.0102  NEG
##  9 V430        0.0102  POS
## 10 V408        0.00790 NEG
## # ... with 607 more rows
```

Variable Importance Sign 1 V360 0.0246 POS
2 V409 0.0227 NEG
3 V342 0.0174 POS

```
4 V352 0.0159 NEG
5 V721 0.0151 NEG
6 V599 0.0120 POS
7 V359 0.0117 POS
8 V508 0.0102 NEG
9 V430 0.0102 POS
10 V408 0.00790 NEG
... with 607 more rows
```

(extra graph:)

```r
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))#turning it into a number

mat <- rep(0, 28*28)
mat[imp.tbl$col] <- imp.tbl$Importance
image(matrix(mat, 28, 28))
```



Still more variation in which variables are the most important, but not as much as in #9.

Accuracy on test dataset for 8, 9, and 10:

8: 0.759

9: 0.83

10: .85

The most accurate model for classifying numbers is the lasso model (with multinom_reg). The accuracy of each model (and which model is best) might change if I run each model with more data, but my computer is very slow so I had to use small data samples.

Most important features for models in 8, 9, and 10:

The most important variables seem to fluctuate a good deal. In fact, the top-10 table for #8 shares only one value with the top-10 table for #10.

The tables for #9 and #10 share only two variables in their top 10.

8: A tibble: 161 × 2 Variable Importance 1 V379 85.6 2 V378 83.9 3 V598 71.0 4 V597 61.3 5 V324 55.6 6 V351 50.0 7 V462 47.3 8 V569 45.0 9 V570 45.0 10 V599 45.0 … with 151 more rows

9: Variable Importance Sign 1 V196 0.00790 POS 2 V66 0.00782 NEG
3 V42 0.00707 NEG
4 V226 0.00625 NEG
5 V335 0.00619 NEG
6 V778 0.00579 NEG
7 V480 0.00468 NEG
8 V508 0.00419 NEG
9 V780 0.00404 NEG
10 V752 0.00401 NEG
… with 607 more rows

10: Variable Importance Sign 1 V360 0.0246 POS
2 V409 0.0227 NEG
3 V342 0.0174 POS
4 V352 0.0159 NEG
5 V721 0.0151 NEG
6 V599 0.0120 POS
7 V359 0.0117 POS
8 V508 0.0102 NEG
9 V430 0.0102 POS
10 V408 0.00790 NEG