Introduction to Regression Trees

Jaime Davila

4/20/2021

Introduction

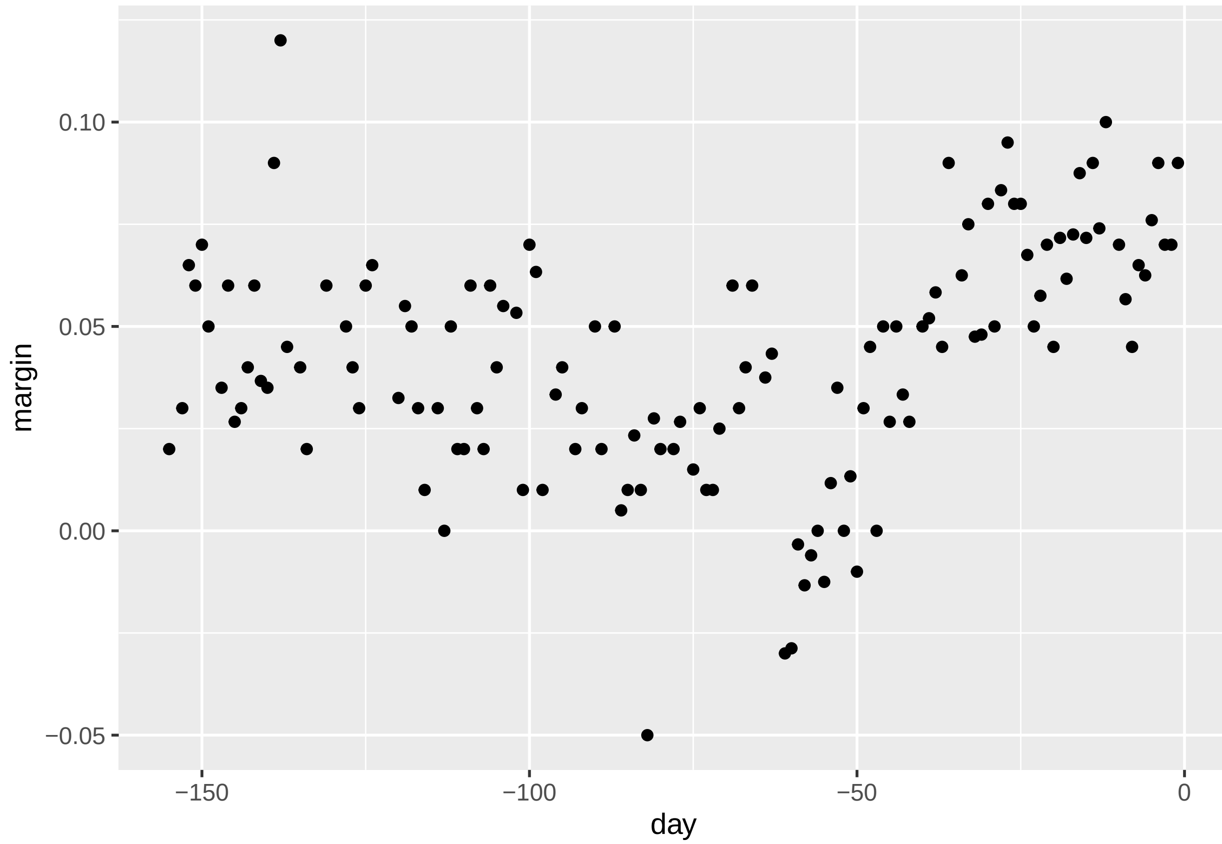
Today we will be using data from the presidential polls for the 2008 election (Obama vs McCain). Let's start by loading the dataset

```
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. 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)
```

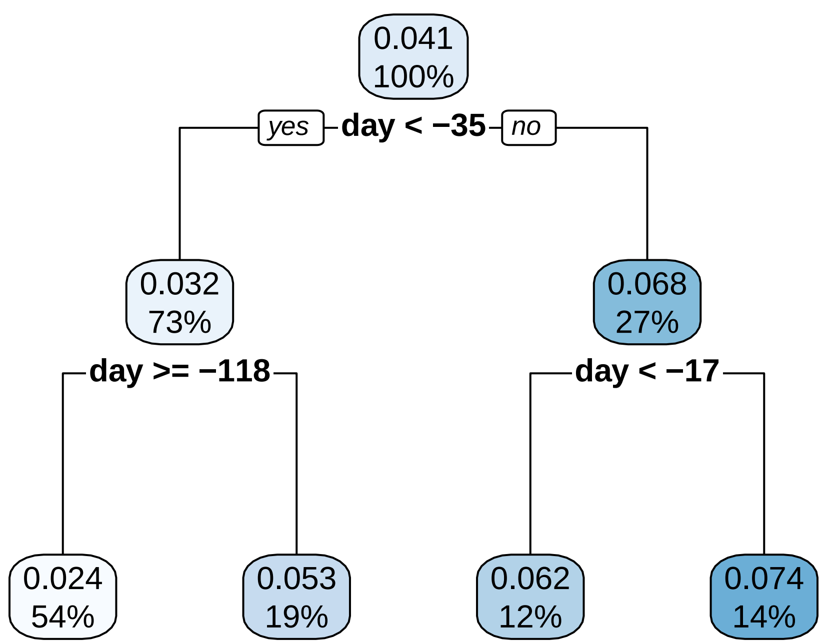
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0274
```

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

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

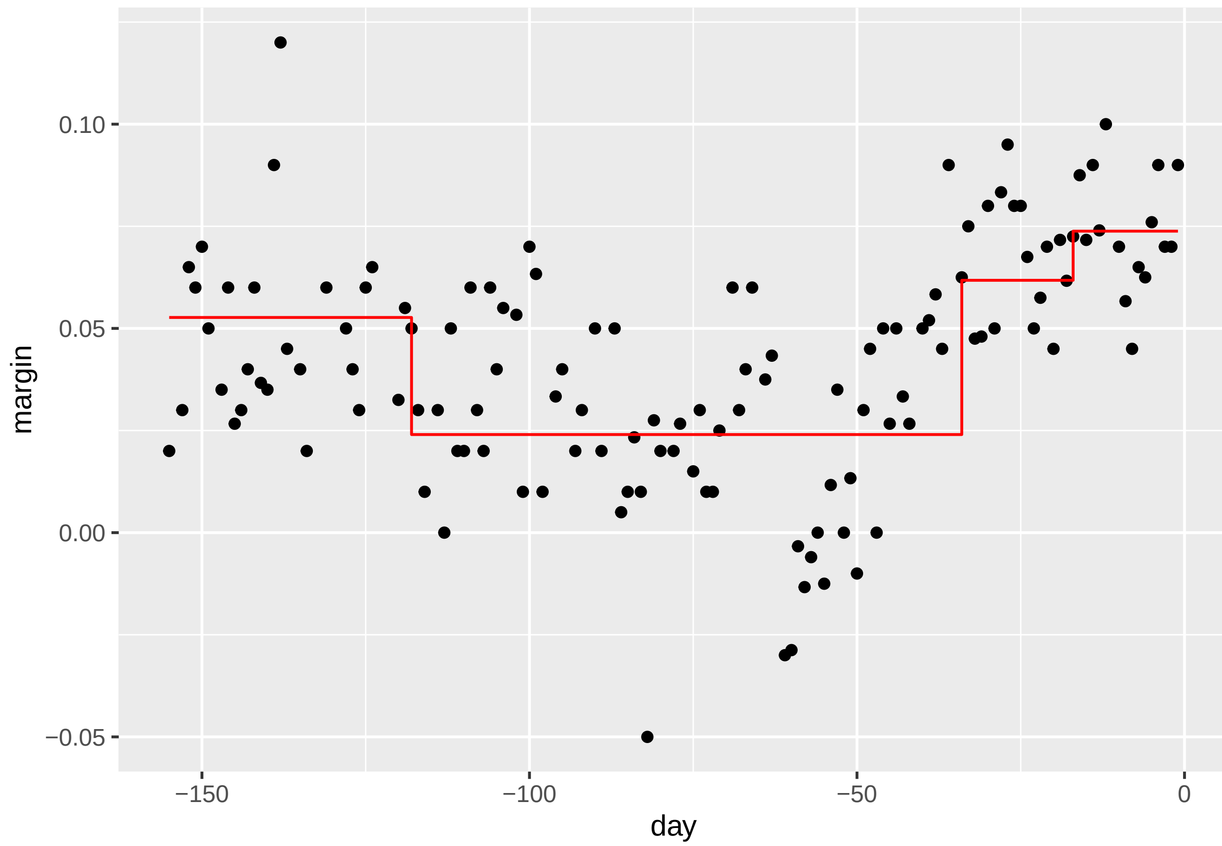
We can visualize our regression tree as a tree

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



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

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



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.

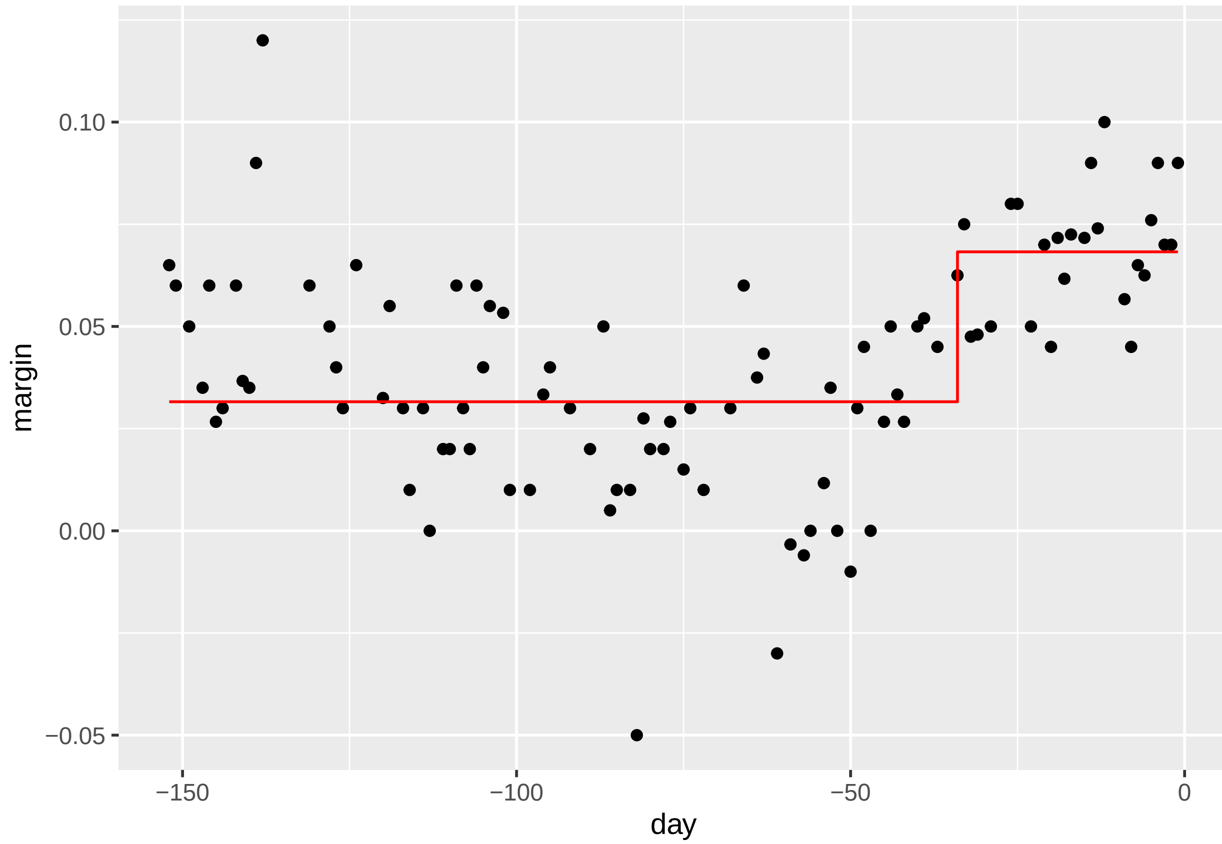
1. Fill out the blanks of the function `calc_mse_tree` that receives two parameters, `tree_depth` and `cost_complexity`, creates a regression tree with such parameters and calculates the *mse* on the training data (yes that's correct, the *training* dataset). Test your function using `tree_depth`=1,2,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

```r
calc_mse <- function(tree_depth, cost_complexity) {
   # Train your model

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

  # Calculate and output the rmse
```

```
}
```

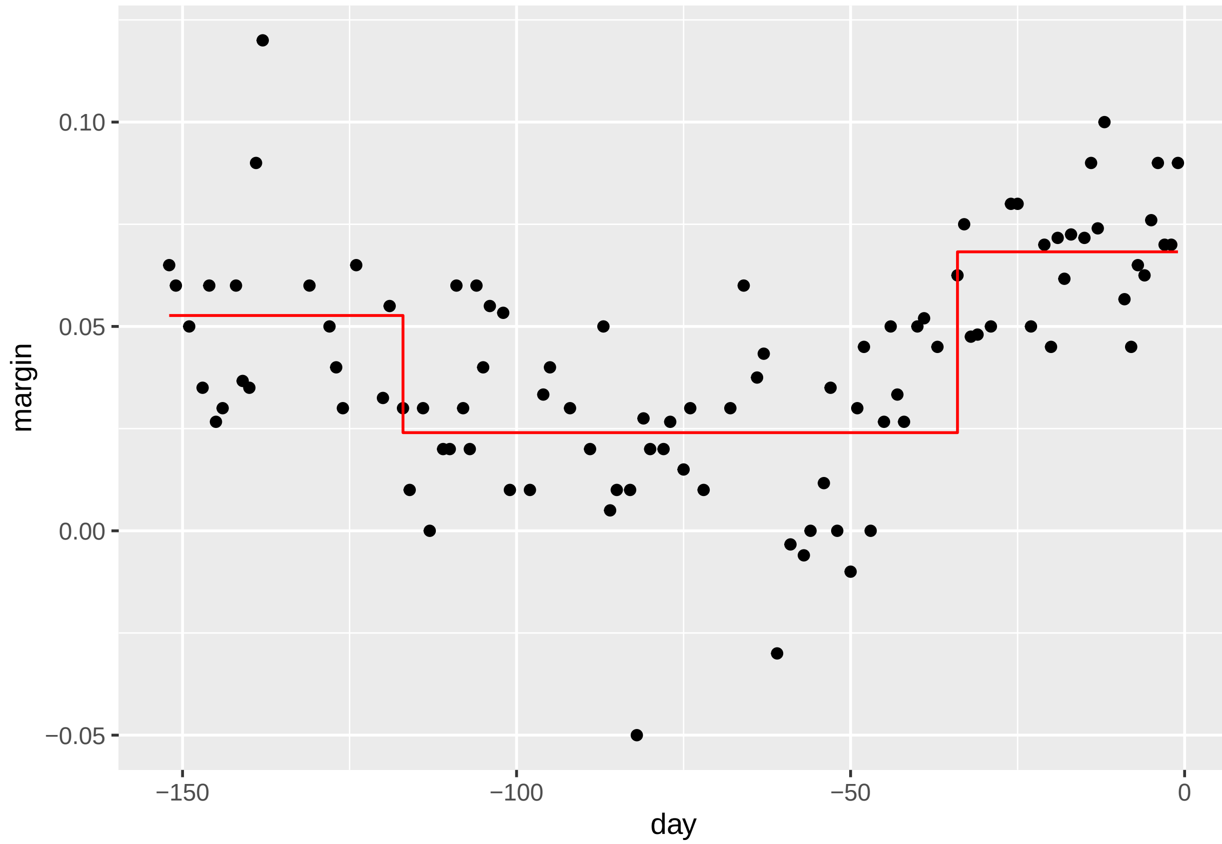
```
calc_mse <- function(tree_depth, cost_complexity) {
  poll.model <-
  decision_tree(tree_depth=tree_depth, cost_complexity=cost_complexity) %>%
  set_mode("regression") %>%
  set_engine("rpart")

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

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

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

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

  rmse <- augment(poll.fit, poll.train.tbl) %>%
    rmse(margin, .pred) %>%
    pull(.estimate)
  rmse^2
}
```

```
calc_mse(1,0.1)
```




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

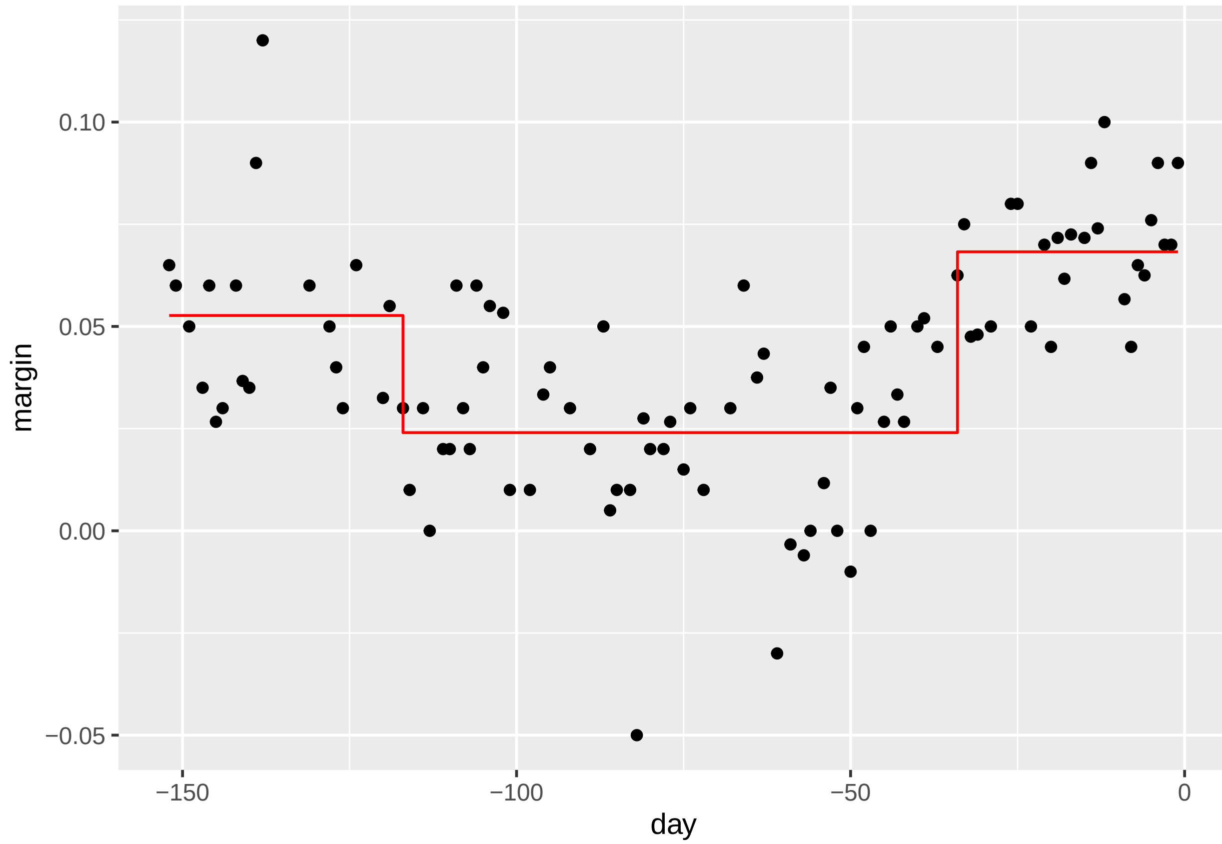
```r
calc_mse(2,0.1)
```




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

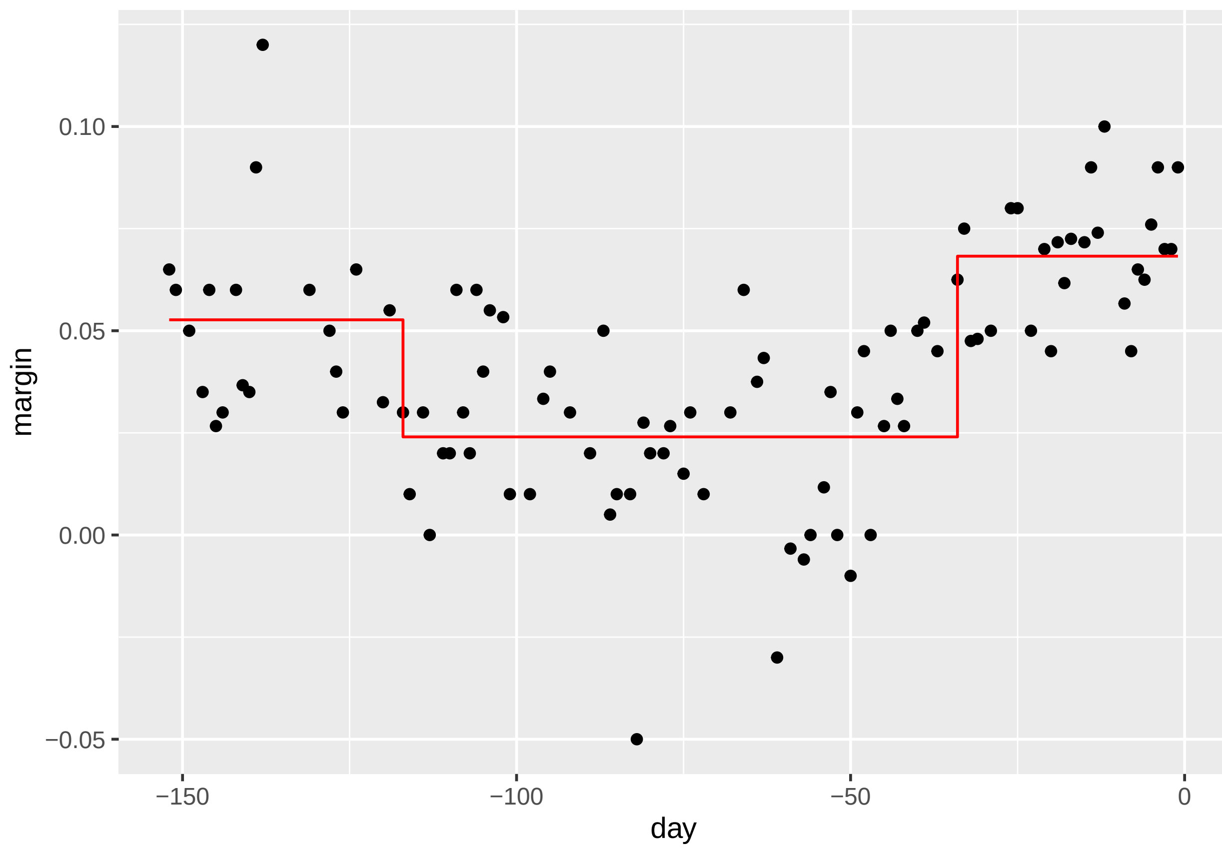
2. In principle, every time that we add a level to our tree we can decrease our RSS. If we continue this approach indefinitely we could end up with a tree where every leaf is a single point which is a clear case of overfitting. The `complexity_parameter` (`cp`) controls the number of recursive splits 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(2,0.1)
```



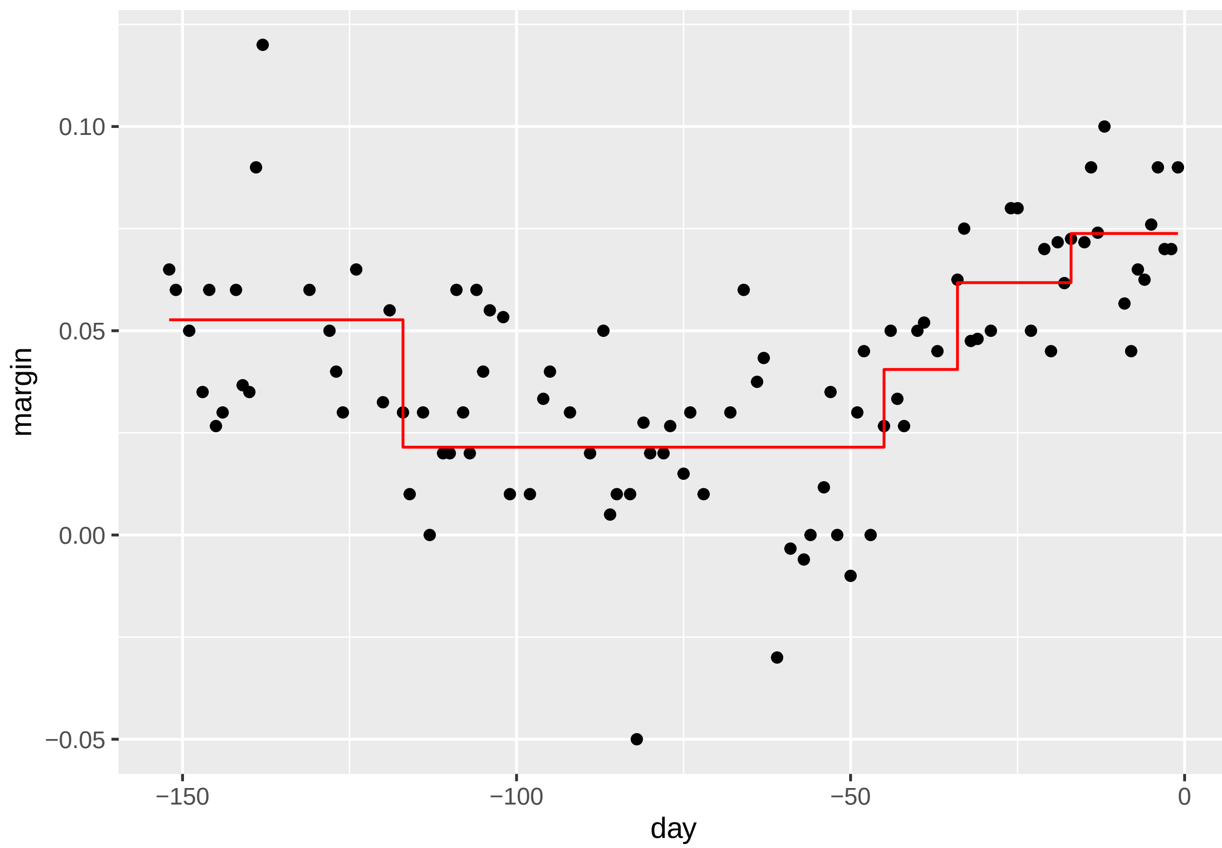

```
## [1] 0.0004262214
```

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



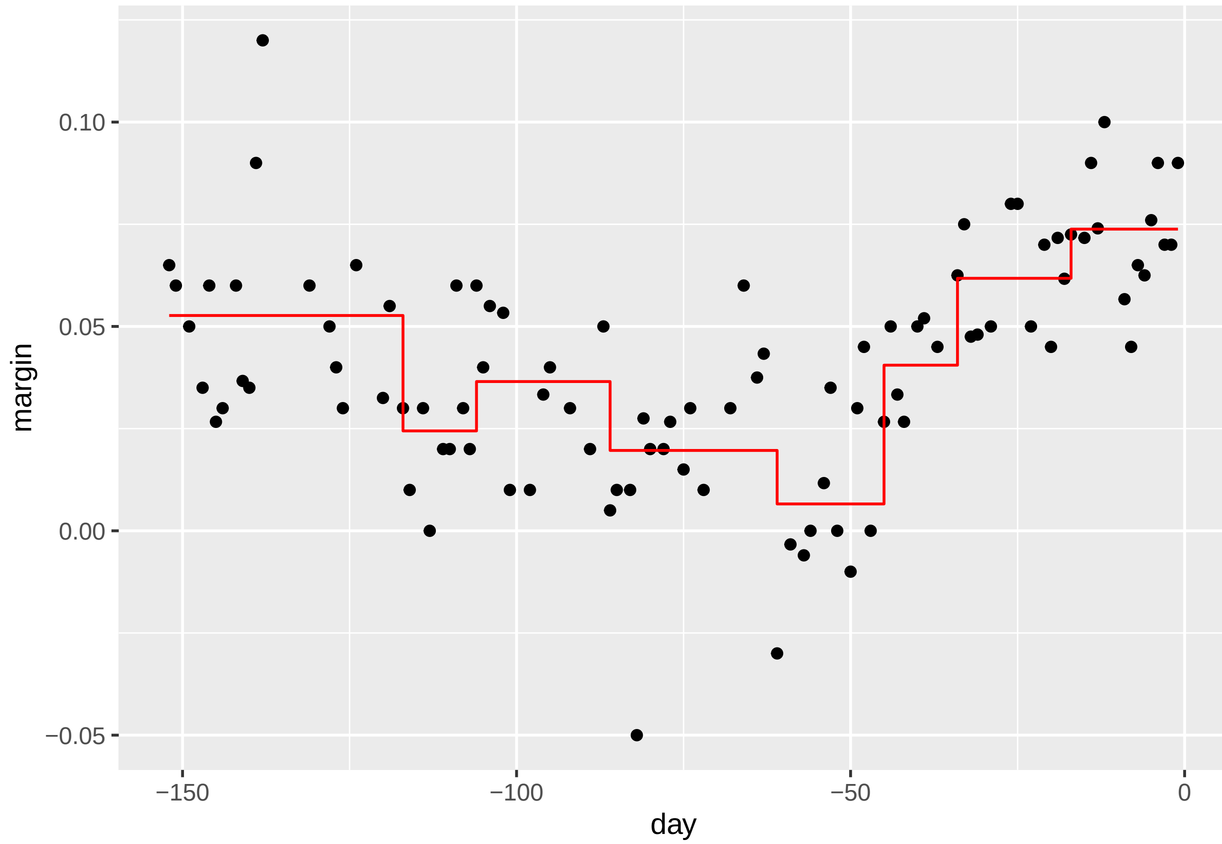

```
## [1] 0.0004262214
```

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




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

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



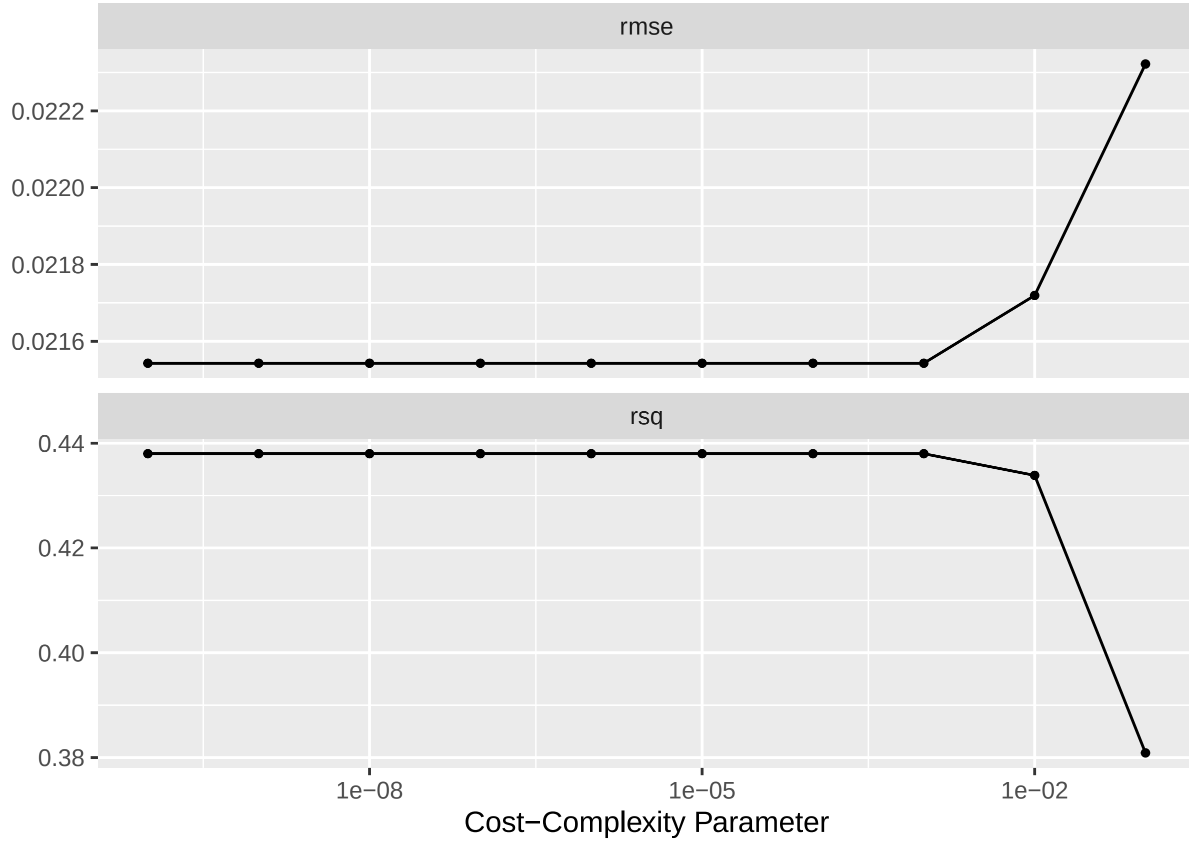

```
## [1] 0.000342623
```

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)
```

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

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

poll.grid <-
  grid_regular(cost_complexity(), levels = 10)

poll.res <-
  tune_grid(
    poll.wflow,
    resamples = poll.folds,
    grid = poll.grid)
```

```r
autoplot(poll.res)
```




```r
show_best(poll.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_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
```

```r
(best.penalty <- select_by_one_std_err(poll.res,
                                       metric = "rmse",
                                       -cost_complexity))
```

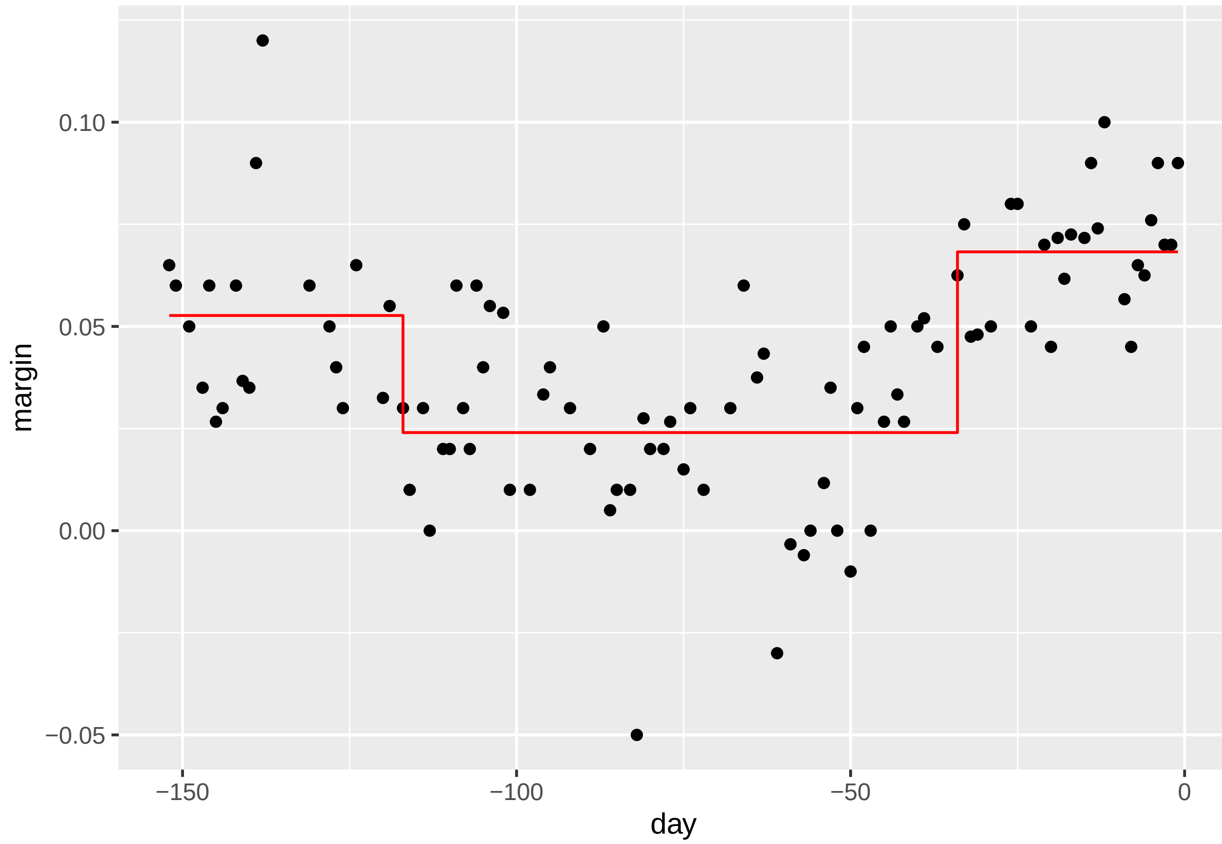
```
## # A tibble: 1 x 9
##   cost_complexity .metric .estimator   mean     n std_err .config   .best .bound
##             <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <fct>     <dbl>  <dbl>
## 1               0.1 rmse    standard   0.0223    10 0.00169 Preproc~ 0.0215 0.0232
```

```r
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)
```

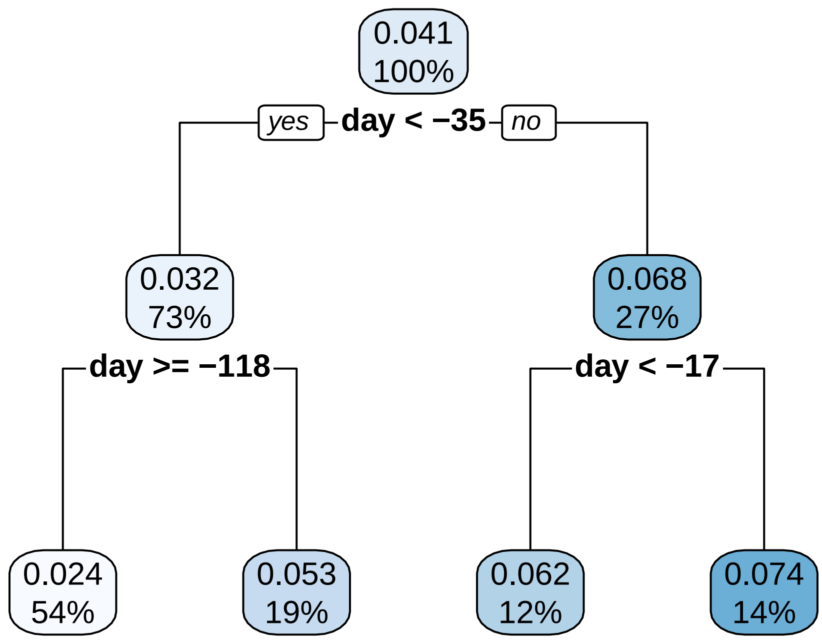
```
## # A tibble: 2 x 4
```

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

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




```
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), 10000)
train.tbl <- as_tibble (mnist$train$images[index,]) %>%
  mutate(digit = factor(mnist$train$labels[index]))

index <- sample(nrow(mnist$test$images), 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))
}
```

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?

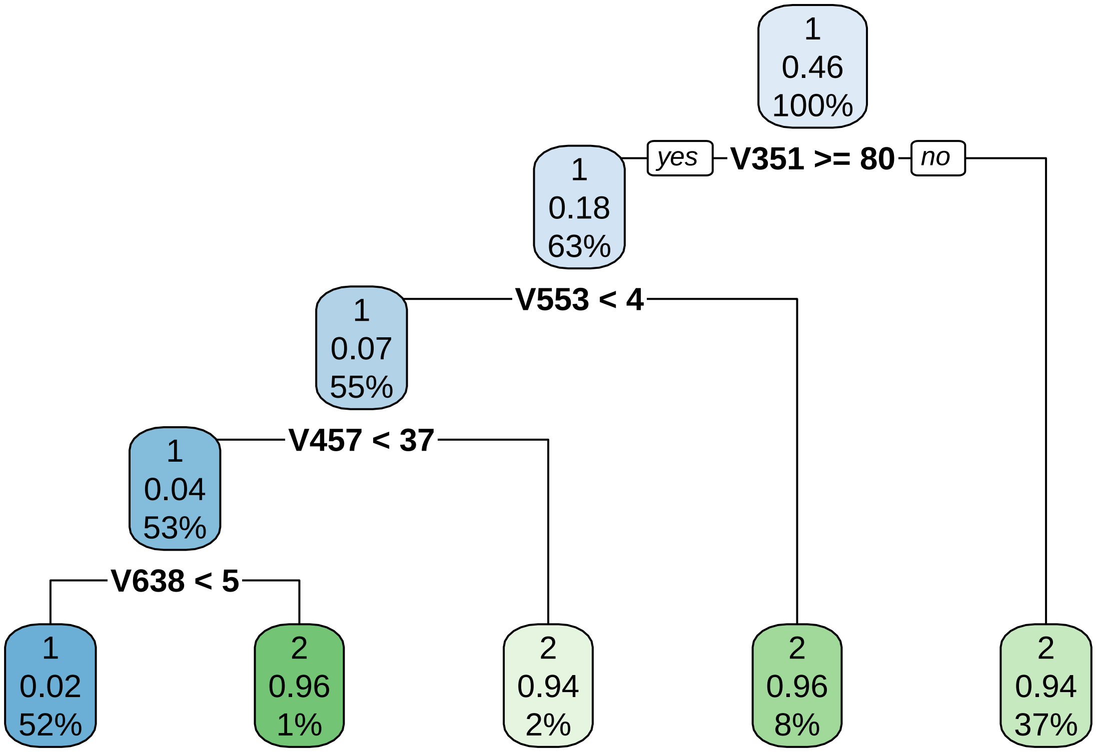
```
digit.model <-
  decision_tree() %>%
  set_mode("classification") %>%
  set_engine("rpart")

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         0.949
```

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

```
##           Truth
## Prediction   1   2
##          1 122   4
##          2   8 100
```

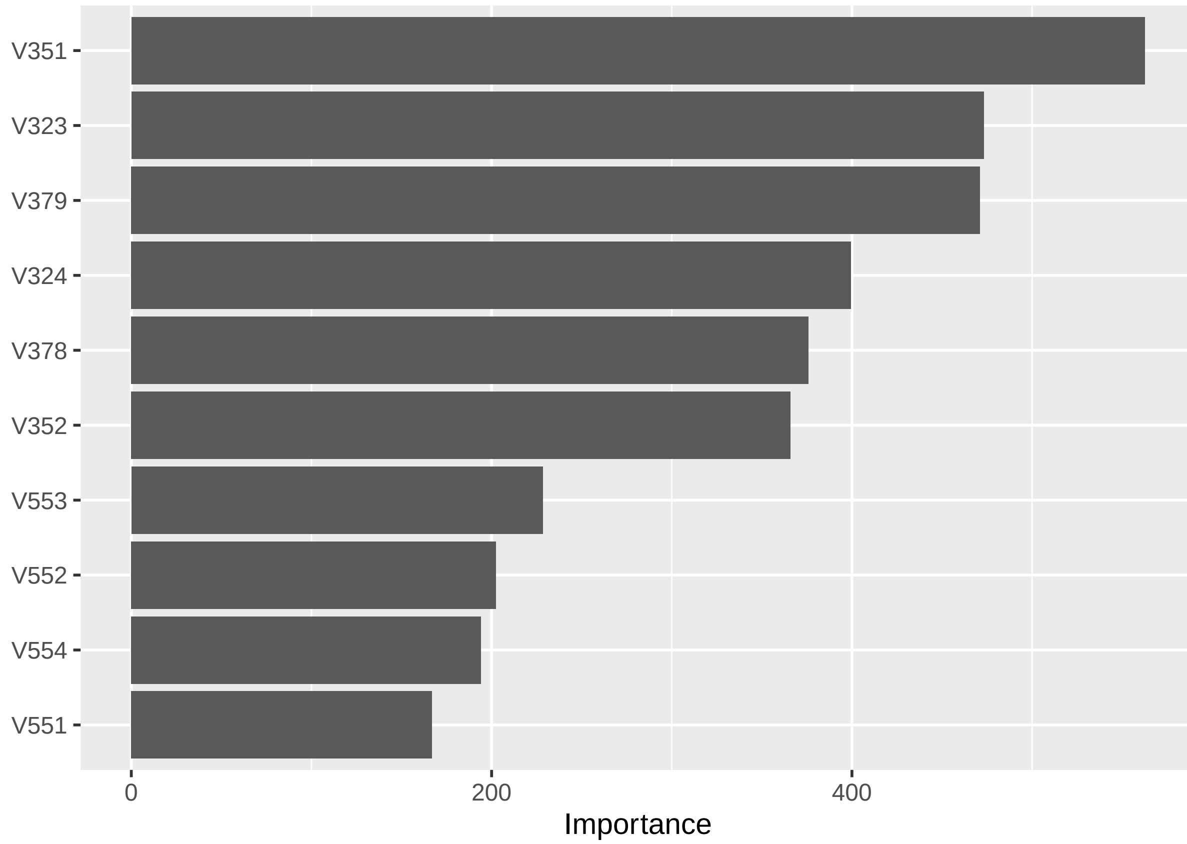
```
digit.fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



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()`

```
library(vip)

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



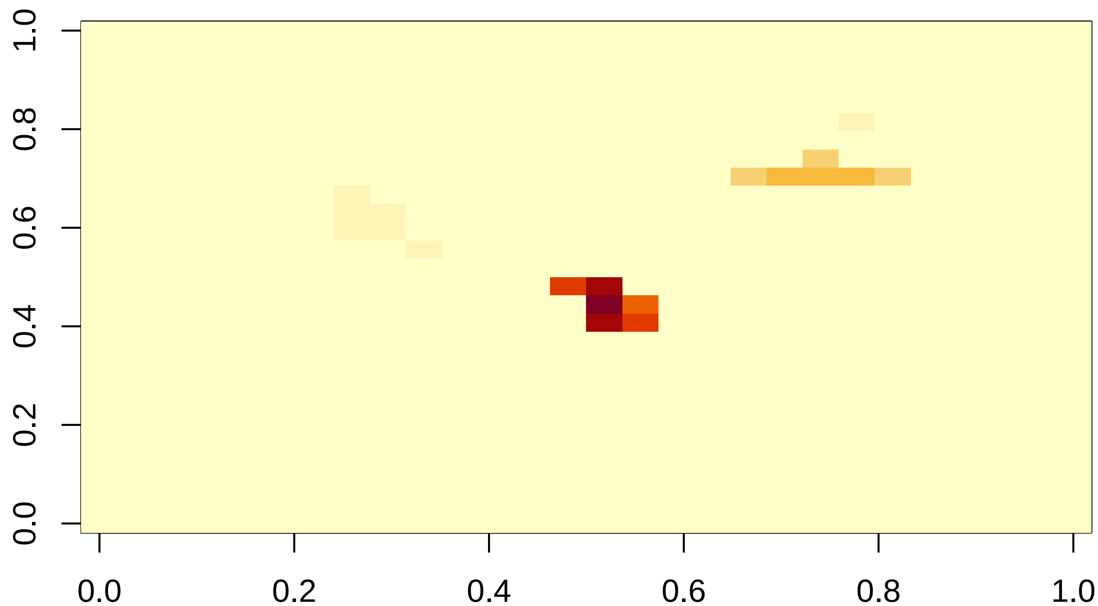

```
imp.tbl <- digit.fit %>%
  extract_fit_engine() %>%
  vip::vi()
imp.tbl
```

```
## # A tibble: 24 x 2
##    Variable Importance
##    <chr>         <dbl>
##  1 V351           563.
##  2 V323           473.
##  3 V379           471.
##  4 V324           400.
##  5 V378           376.
##  6 V352           366.
##  7 V553           228.
##  8 V552           202.
##  9 V554           194.
## 10 V551           167.
## # ... with 14 more rows
```

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

```
imp.tbl <- imp.tbl %>%
  mutate(col=as.double(str_remove(Variable,"V")))

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

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