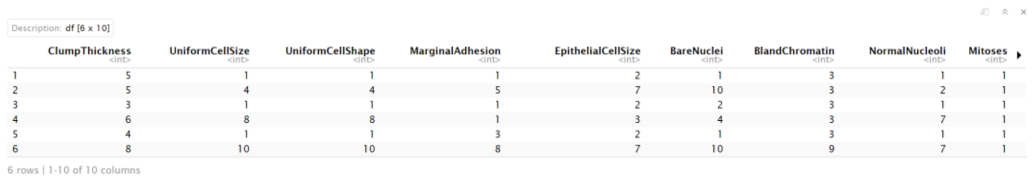
HOMEWORK 3

FUNDAMENTALS OF DATA SCIENCE

PROBLEM 1

For this problem, you will perform a straightforward training and evaluation of a decision tree, as well as generate rules by hand. Load the breast_cancer_updated.csv data. These data are visual features computed from samples of breast tissue being evaluated for cancer1. As a pre-processing step, remove the ID Number column and exclude rows with NA from the dataset.

~~~
# Installing Packages
```{r}
library(tidyr)
library(rattle)
library(tidyverse)
library(caret)
library(dplyr)
library(rpart)
library(GGally)
library(ggplot2)
library(e1071)
# Import the Dataset
library(readxl)
setwd("C:/Users/admin/Desktop")
breast_cancer_updated <- read.csv("breast_cancer_updated.csv")
# Remove ID Number
breast_cancer_updated <- breast_cancer_updated %>% select(-c("IDNumber"))
# Remove NA rows
breast_cancer_updated <- na.omit(breast_cancer_updated)
head(breast_cancer_updated)
```
~~~



Description: df [6 x 10]

	ClumpThickness <int>	UniformCellSize <int>	UniformCellShape <int>	MarginalAdhesion <int>	EpithelialCellSize <int>	BareNuclei <int>	BlandChromatin <int>	NormalNucleoli <int>	Mitoses <int>
1	5	1	1	1	2	1	3	1	1
2	5	4	4	5	7	10	3	2	1
3	3	1	1	1	2	2	3	1	1
4	6	8	8	1	3	4	3	7	1
5	4	1	1	3	2	1	3	1	1
6	8	10	10	8	7	10	9	7	1

6 rows | 1-10 of 10 columns

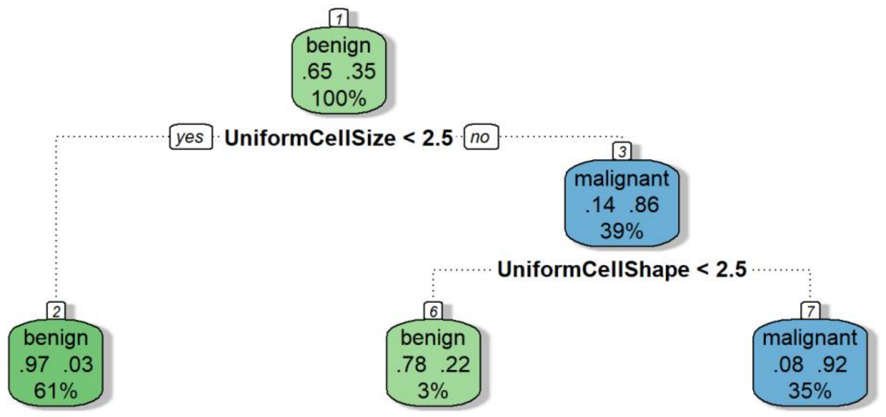
a. **Apply decision tree learning (use rpart) to the data to predict breast cancer malignancy (Class) and report the accuracy using 10-fold cross validation.**

~~~
```{r}
# Evaluation method using the 10-fold cross-validation
train_control = trainControl(method = "cv", number = 10)
# Fit the Model
tree1 <- train(Class ~., data = breast_cancer_updated, method = "rpart", trControl = train_control)
tree1
```
~~~

```
CART

683 samples
  9 predictor
  2 classes: 'benign', 'malignant'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 615, 615, 614, 614, 616, 615, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.02510460  0.9386390  0.8665123
  0.05439331  0.9268736  0.8419215
  0.79079498  0.8596516  0.6449787

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.0251046.
```

b. Generate a visualization of the decision tree.

~~~
```{r}
fancyRpartPlot(tree1$finalModel, caption = "")
```
~~~



c. Generate the full set of rules using IF-THEN statements

So based on the decision tree above, we're using the uniform cell shape and size to determine whether the tumor is benign or malignant. However, I noticed that my code's results differ from the actual dataset's results, as the dataset itself is likely using other variables to decide whether the tumor is benign (or) malignant. This is important to keep in mind as we compare our code's results to the original dataset.

```{r}
# n is the number of rows in the cell from 1 - 683
n = 1
for (n in 1:683)
  {
  x <- breast_cancer_updated$UniformCellSize[n]
  y <- breast_cancer_updated$UniformCellShape[n]

  # Print the number of the row
  print(n)

  # Adding the 1 to the previous n value
  n <- n+1
  if(x & y >= 2.5){
    print("malignant")
  } else if(x >= 2.5 & y < 2.5){
    print("Benign")
  }else {
    print("Benign")
  }
}
```

```
[1] 1
[1] "Benign"
[1] 2
[1] "malignant"
[1] 3
[1] "Benign"
[1] 4
[1] "malignant"
[1] 5
[1] "Benign"
[1] 6
[1] "malignant"
[1] 7
[1] "Benign"
[1] 8
[1] "Benign"
[1] 9
[1] "Benign"
[1] 10
[1] "Benign"
[1] 11
[1] "Benign"
[1] 12
[1] "Benign"
[1] 13
[1] "malignant"
[1] 14
[1] "Benign"
[1] 15
[1] "malignant"
[1] 16
[1] "malignant"
[1] 17
[1] "Benign"
[1] 18
[1] "Benign"
[1] 19
[1] "malignant"
[1] 20
```

```
[1] "Benign"
[1] 21
[1] "Benign"
[1] 22
[1] "malignant"
[1] 23
[1] "Benign"
[1] 24
[1] "Benign"
[1] 25
[1] "malignant"
[1] 26
[1] "Benign"
[1] 27
[1] "Benign"
[1] 28
[1] "Benign"
[1] 29
[1] "malignant"
[1] 30
[1] "Benign"
[1] 31
[1] "Benign"
[1] 32
[1] "malignant"
[1] 33
[1] "Benign"
[1] 34
[1] "Benign"
[1] 35
[1] "Benign"
[1] 36
[1] "malignant"
[1] 37
[1] "Benign"
[1] 38
[1] "malignant"
[1] 39
[1] "malignant"
```

```
[1] 42
[1] "malignant"
[1] 43
[1] "malignant"
[1] 44
[1] "Benign"
[1] 45
[1] "malignant"
[1] 46
[1] "Benign"
[1] 47
[1] "Benign"
[1] 48
[1] "malignant"
[1] 49
[1] "malignant"
[1] 50
[1] "malignant"
[1] 51
[1] "malignant"
[1] 52
[1] "malignant"
[1] 53
[1] "malignant"
[1] 54
[1] "malignant"
[1] 55
[1] "malignant"
[1] 56
[1] "malignant"
[1] 57
[1] "malignant"
[1] 58
[1] "malignant"
[1] 59
[1] "malignant"
[1] 60
[1] "Benign"
[1] 61
[1] "malignant"
[1] 62
```

```
[1] "malignant"
[1] 63
[1] "Benign"
[1] 64
[1] "Benign"
[1] 65
[1] "Benign"
[1] 66
[1] "malignant"
[1] 67
[1] "malignant"
[1] 68
[1] "Benign"
[1] 69
[1] "malignant"
[1] 70
[1] "Benign"
[1] 71
[1] "malignant"
[1] 72
[1] "malignant"
[1] 73
[1] "malignant"
[1] 74
[1] "Benign"
[1] 75
[1] "malignant"
[1] 76
[1] "Benign"
[1] 77
[1] "Benign"
[1] 78
[1] "Benign"
[1] 79
[1] "Benign"
[1] 80
[1] "Benign"
[1] 81
[1] "Benign"
[1] 82
```

```
[1] "Benign"
[1] 83
[1] "malignant"
[1] 84
[1] "malignant"
[1] 85
[1] "malignant"
[1] 86
[1] "malignant"
[1] 87
[1] "Benign"
[1] 88
[1] "Benign"
[1] 89
[1] "Benign"
[1] 90
[1] "Benign"
[1] 91
[1] "Benign"
[1] 92
[1] "Benign"
[1] 93
[1] "Benign"
[1] 94
[1] "Benign"
[1] 95
[1] "Benign"
[1] 96
[1] "Benign"
[1] 97
[1] "malignant"
[1] 98
[1] "malignant"
[1] 99
[1] "malignant"
[1] 100
[1] "malignant"
[1] 101
[1] "Benign"
[1] 102
```

```
[1] "malignant"
[1] 103
[1] "malignant"
[1] 104
[1] "malignant"
[1] 105
[1] "malignant"
[1] 106
[1] "malignant"
[1] 107
[1] "Benign"
[1] 108
[1] "malignant"
[1] 109
[1] "Benign"
[1] 110
[1] "malignant"
[1] 111
[1] "malignant"
[1] 112
[1] "malignant"
[1] 113
[1] "Benign"
[1] 114
[1] "Benign"
[1] 115
[1] "malignant"
[1] 116
[1] "malignant"
[1] 117
[1] "Benign"
[1] 118
[1] "Benign"
[1] 119
[1] "Benign"
[1] 120
[1] "Benign"
[1] 121
[1] "malignant"
[1] 122
```

```
[1] "malignant"
[1] 123
[1] "malignant"
[1] 124
[1] "Benign"
[1] 125
[1] "malignant"
[1] 126
[1] "Benign"
[1] 127
[1] "malignant"
[1] 128
[1] "Benign"
[1] 129
[1] "malignant"
[1] 130
[1] "Benign"
[1] 131
[1] "malignant"
[1] 132
[1] "Benign"
[1] 133
[1] "Benign"
[1] 134
[1] "Benign"
[1] 135
[1] "Benign"
[1] 136
[1] "Benign"
[1] 137
[1] "Benign"
[1] 138
[1] "Benign"
[1] 139
[1] "Benign"
[1] 140
[1] "malignant"
[1] 141
[1] "Benign"
[1] 142
[1] "Benign"
```

```
[1] 143
[1] "malignant"
[1] 144
[1] "Benign"
[1] 145
[1] "Benign"
[1] 146
[1] "malignant"
[1] 147
[1] "Benign"
[1] 148
[1] "malignant"
[1] 149
[1] "malignant"
[1] 150
[1] "Benign"
[1] 151
[1] "Benign"
[1] 152
[1] "malignant"
[1] 153
[1] "Benign"
[1] 154
[1] "Benign"
[1] 155
[1] "malignant"
[1] 156
[1] "malignant"
[1] 157
[1] "Benign"
[1] 158
[1] "Benign"
[1] 159
[1] "Benign"
[1] 160
[1] "Benign"
[1] 161
[1] "malignant"
[1] 162
[1] "malignant"
```

```
[1] 163
[1] "Benign"
[1] 164
[1] "Benign"
[1] 165
[1] "Benign"
[1] 166
[1] "Benign"
[1] 167
[1] "Benign"
[1] 168
[1] "malignant"
[1] 169
[1] "malignant"
[1] 170
[1] "malignant"
[1] 171
[1] "Benign"
[1] 172
[1] "malignant"
[1] 173
[1] "Benign"
[1] 174
[1] "malignant"
[1] 175
[1] "Benign"
[1] 176
[1] "Benign"
[1] 177
[1] "Benign"
[1] 178
[1] "malignant"
[1] 179
[1] "malignant"
[1] 180
[1] "Benign"
[1] 181
[1] "malignant"
[1] 182
[1] "malignant"
```

```
[1] 183
[1] "malignant"
[1] 184
[1] "malignant"
[1] 185
[1] "malignant"
[1] 186
[1] "malignant"
[1] 187
[1] "Benign"
[1] 188
[1] "Benign"
[1] 189
[1] "Benign"
[1] 190
[1] "Benign"
[1] 191
[1] "malignant"
[1] 192
[1] "Benign"
[1] 193
[1] "Benign"
[1] 194
[1] "Benign"
[1] 195
[1] "malignant"
[1] 196
[1] "malignant"
[1] 197
[1] "Benign"
[1] 198
[1] "Benign"
[1] 199
[1] "Benign"
[1] 200
[1] "malignant"
[1] 201
[1] "malignant"
[1] 202
[1] "Benign"
```

```
[1] 203
[1] "Benign"
[1] 204
[1] "Benign"
[1] 205
[1] "malignant"
[1] 206
[1] "malignant"
[1] 207
[1] "Benign"
[1] 208
[1] "malignant"
[1] 209
[1] "malignant"
[1] 210
[1] "malignant"
[1] 211
[1] "Benign"
[1] 212
[1] "Benign"
[1] 213
[1] "malignant"
[1] 214
[1] "malignant"
[1] 215
[1] "Benign"
[1] 216
[1] "malignant"
[1] 217
[1] "Benign"
[1] 218
[1] "malignant"
[1] 219
[1] "malignant"
[1] 220
[1] "Benign"
[1] 221
[1] "malignant"
[1] 222
[1] "malignant"
```

```
[1] 223
[1] "Benign"
[1] 224
[1] "malignant"
[1] 225
[1] "malignant"
[1] 226
[1] "malignant"
[1] 227
[1] "malignant"
[1] 228
[1] "malignant"
[1] 229
[1] "Benign"
[1] 230
[1] "malignant"
[1] 231
[1] "malignant"
[1] 232
[1] "malignant"
[1] 233
[1] "malignant"
[1] 234
[1] "malignant"
[1] 235
[1] "Benign"
[1] 236
[1] "Benign"
[1] 237
[1] "Benign"
[1] 238
[1] "Benign"
[1] 239
[1] "Benign"
[1] 240
[1] "malignant"
[1] 241
[1] "malignant"
[1] 242
[1] "Benign"
```

```
[1] 243
[1] "Benign"
[1] 244
[1] "malignant"
[1] 245
[1] "malignant"
[1] 246
[1] "malignant"
[1] 247
[1] "malignant"
[1] 248
[1] "malignant"
[1] 249
[1] "Benign"
[1] 250
[1] "Benign"
[1] 251
[1] "Benign"
[1] 252
[1] "malignant"
[1] 253
[1] "malignant"
[1] 254
[1] "malignant"
[1] 255
[1] "malignant"
[1] 256
[1] "malignant"
[1] 257
[1] "malignant"
[1] 258
[1] "malignant"
[1] 259
[1] "malignant"
[1] 260
[1] "malignant"
[1] 261
[1] "malignant"
[1] 262
[1] "Benign"
```

```
[1] 263
[1] "malignant"
[1] 264
[1] "Benign"
[1] 265
[1] "malignant"
[1] 266
[1] "malignant"
[1] 267
[1] "Benign"
[1] 268
[1] "Benign"
[1] 269
[1] "Benign"
[1] 270
[1] "Benign"
[1] 271
[1] "malignant"
[1] 272
[1] "Benign"
[1] 273
[1] "Benign"
[1] 274
[1] "malignant"
[1] 275
[1] "malignant"
[1] 276
[1] "malignant"
[1] 277
[1] "malignant"
[1] 278
[1] "malignant"
[1] 279
[1] "Benign"
[1] 280
[1] "malignant"
[1] 281
[1] "malignant"
[1] 282
[1] "Benign"
```

```
[1] 283
[1] "Benign"
[1] 284
[1] "malignant"
[1] 285
[1] "malignant"
[1] 286
[1] "malignant"
[1] 287
[1] "Benign"
[1] 288
[1] "Benign"
[1] 289
[1] "malignant"
[1] 290
[1] "Benign"
[1] 291
[1] "malignant"
[1] 292
[1] "Benign"
[1] 293
[1] "malignant"
[1] 294
[1] "malignant"
[1] 295
[1] "Benign"
[1] 296
[1] "Benign"
[1] 297
[1] "malignant"
[1] 298
[1] "Benign"
[1] 299
[1] "Benign"
[1] 300
[1] "Benign"
[1] 301
[1] "malignant"
[1] 302
[1] "Benign"
```

```
[1] 303
[1] "Benign"
[1] 304
[1] "malignant"
[1] 305
[1] "malignant"
[1] 306
[1] "Benign"
[1] 307
[1] "malignant"
[1] 308
[1] "malignant"
[1] 309
[1] "Benign"
[1] 310
[1] "malignant"
[1] 311
[1] "Benign"
[1] 312
[1] "Benign"
[1] 313
[1] "Benign"
[1] 314
[1] "Benign"
[1] 315
[1] "malignant"
[1] 316
[1] "malignant"
[1] 317
[1] "malignant"
[1] 318
[1] "Benign"
[1] 319
[1] "Benign"
[1] 320
[1] "malignant"
[1] 321
[1] "malignant"
[1] 322
[1] "Benign"
```

```
[1] 323
[1] "malignant"
[1] 324
[1] "Benign"
[1] 325
[1] "Benign"
[1] 326
[1] "malignant"
[1] 327
[1] "malignant"
[1] 328
[1] "Benign"
[1] 329
[1] "Benign"
[1] 330
[1] "Benign"
[1] 331
[1] "malignant"
[1] 332
[1] "Benign"
[1] 333
[1] "Benign"
[1] 334
[1] "Benign"
[1] 335
[1] "malignant"
[1] 336
[1] "malignant"
[1] 337
[1] "Benign"
[1] 338
[1] "Benign"
[1] 339
[1] "malignant"
[1] 340
[1] "malignant"
[1] 341
[1] "Benign"
```

```
[1] 342
[1] "Benign"
[1] 343
[1] "malignant"
[1] 344
[1] "malignant"
[1] 345
[1] "malignant"
[1] 346
[1] "malignant"
[1] 347
[1] "malignant"
[1] 348
[1] "malignant"
[1] 349
[1] "Benign"
[1] 350
[1] "malignant"
[1] 351
[1] "Benign"
[1] 352
[1] "Benign"
[1] 353
[1] "malignant"
[1] 354
[1] "malignant"
[1] 355
[1] "malignant"
[1] 356
[1] "malignant"
[1] 357
[1] "Benign"
[1] 358
[1] "malignant"
[1] 359
[1] "Benign"
[1] 360
[1] "Benign"
[1] 361
[1] "Benign"
```

```
[1] 362
[1] "Benign"
[1] 363
[1] "Benign"
[1] 364
[1] "Benign"
[1] 365
[1] "Benign"
[1] 366
[1] "malignant"
[1] 367
[1] "Benign"
[1] 368
[1] "malignant"
[1] 369
[1] "Benign"
[1] 370
[1] "Benign"
[1] 371
[1] "Benign"
[1] 372
[1] "Benign"
[1] 373
[1] "malignant"
[1] 374
[1] "malignant"
[1] 375
[1] "Benign"
[1] 376
[1] "Benign"
[1] 377
[1] "Benign"
[1] 378
[1] "malignant"
[1] 379
[1] "Benign"
[1] 380
[1] "Benign"
[1] 381
[1] "malignant"
```

```
[1] 382
[1] "Benign"
[1] 383
[1] "Benign"
[1] 384
[1] "Benign"
[1] 385
[1] "Benign"
[1] 386
[1] "malignant"
[1] 387
[1] "malignant"
[1] 388
[1] "Benign"
[1] 389
[1] "malignant"
[1] 390
[1] "Benign"
[1] 391
[1] "Benign"
[1] 392
[1] "Benign"
[1] 393
[1] "Benign"
[1] 394
[1] "Benign"
[1] 395
[1] "Benign"
[1] 396
[1] "Benign"
[1] 397
[1] "Benign"
[1] 398
[1] "malignant"
[1] 399
[1] "Benign"
[1] 400
[1] "malignant"
[1] 401
[1] "Benign"
```

```
[1] 402
[1] "malignant"
[1] 403
[1] "Benign"
[1] 404
[1] "Benign"
[1] 405
[1] "Benign"
[1] 406
[1] "Benign"
[1] 407
[1] "malignant"
[1] 408
[1] "malignant"
[1] 409
[1] "malignant"
[1] 410
[1] "Benign"
[1] 411
[1] "malignant"
[1] 412
[1] "malignant"
[1] 413
[1] "malignant"
[1] 414
[1] "Benign"
[1] 415
[1] "Benign"
[1] 416
[1] "Benign"
[1] 417
[1] "Benign"
[1] 418
[1] "Benign"
[1] 419
[1] "Benign"
[1] 420
[1] "malignant"
[1] 421
[1] "malignant"
```

```
[1] 422
[1] "malignant"
[1] 423
[1] "Benign"
[1] 424
[1] "malignant"
[1] 425
[1] "Benign"
[1] 426
[1] "malignant"
[1] 427
[1] "Benign"
[1] 428
[1] "Benign"
[1] 429
[1] "Benign"
[1] 430
[1] "Benign"
[1] 431
[1] "Benign"
[1] 432
[1] "Benign"
[1] 433
[1] "Benign"
[1] 434
[1] "Benign"
[1] 435
[1] "malignant"
[1] 436
[1] "Benign"
[1] 437
[1] "Benign"
[1] 438
[1] "Benign"
[1] 439
[1] "malignant"
[1] 440
[1] "Benign"
[1] 441
[1] "Benign"
```

```
[1] 442
[1] "malignant"
[1] 443
[1] "malignant"
[1] 444
[1] "Benign"
[1] 445
[1] "malignant"
[1] 446
[1] "Benign"
[1] 447
[1] "Benign"
[1] 448
[1] "Benign"
[1] 449
[1] "Benign"
[1] 450
[1] "Benign"
[1] 451
[1] "malignant"
[1] 452
[1] "malignant"
[1] 453
[1] "malignant"
[1] 454
[1] "Benign"
[1] 455
[1] "Benign"
[1] 456
[1] "Benign"
[1] 457
[1] "Benign"
[1] 458
[1] "Benign"
[1] 459
[1] "Benign"
[1] 460
[1] "Benign"
[1] 461
[1] "Benign"
```

```
[1] 462
[1] "Benign"
[1] 463
[1] "Benign"
[1] 464
[1] "Benign"
[1] 465
[1] "malignant"
[1] 466
[1] "Benign"
[1] 467
[1] "Benign"
[1] 468
[1] "malignant"
[1] 469
[1] "malignant"
[1] 470
[1] "Benign"
[1] 471
[1] "Benign"
[1] 472
[1] "Benign"
[1] 473
[1] "malignant"
[1] 474
[1] "malignant"
[1] 475
[1] "Benign"
[1] 476
[1] "Benign"
[1] 477
[1] "malignant"
[1] 478
[1] "Benign"
[1] 479
[1] "malignant"
[1] 480
```

```
[1] "Benign"
[1] 481
[1] "Benign"
[1] 482
[1] "Benign"
[1] 483
[1] "Benign"
[1] 484
[1] "Benign"
[1] 485
[1] "Benign"
[1] 486
[1] "Benign"
[1] 487
[1] "Benign"
[1] 488
[1] "Benign"
[1] 489
[1] "Benign"
[1] 490
[1] "Benign"
[1] 491
[1] "Benign"
[1] 492
[1] "malignant"
[1] 493
[1] "Benign"
[1] 494
[1] "Benign"
[1] 495
[1] "Benign"
[1] 496
[1] "Benign"
[1] 497
[1] "Benign"
[1] 498
[1] "Benign"
[1] 499
[1] "Benign"
[1] 500
```

```
[1] "malignant"
[1] 501
[1] "malignant"
[1] 502
[1] "Benign"
[1] 503
[1] "Benign"
[1] 504
[1] "Benign"
[1] 505
[1] "malignant"
[1] 506
[1] "Benign"
[1] 507
[1] "Benign"
[1] 508
[1] "malignant"
[1] 509
[1] "malignant"
[1] 510
[1] "Benign"
[1] 511
[1] "Benign"
[1] 512
[1] "Benign"
[1] 513
[1] "Benign"
[1] 514
[1] "malignant"
[1] 515
[1] "Benign"
[1] 516
[1] "malignant"
[1] 517
[1] "Benign"
[1] 518
[1] "Benign"
[1] 519
[1] "Benign"
[1] 520
```

```
[1] "Benign"
[1] 521
[1] "malignant"
[1] 522
[1] "Benign"
[1] 523
[1] "Benign"
[1] 524
[1] "Benign"
[1] 525
[1] "Benign"
[1] 526
[1] "Benign"
[1] 527
[1] "Benign"
[1] 528
[1] "Benign"
[1] 529
[1] "Benign"
[1] 530
[1] "malignant"
[1] 531
[1] "Benign"
[1] 532
[1] "malignant"
[1] 533
[1] "Benign"
[1] 534
[1] "Benign"
[1] 535
[1] "malignant"
[1] 536
[1] "Benign"
[1] 537
[1] "Benign"
[1] 538
[1] "Benign"
[1] 539
[1] "Benign"
[1] 540
```

```
[1] "Benign"
[1] 541
[1] "Benign"
[1] 542
[1] "Benign"
[1] 543
[1] "Benign"
[1] 544
[1] "Benign"
[1] 545
[1] "Benign"
[1] 546
[1] "Benign"
[1] 547
[1] "Benign"
[1] 548
[1] "Benign"
[1] 549
[1] "Benign"
[1] 550
[1] "Benign"
[1] 551
[1] "malignant"
[1] 552
[1] "Benign"
[1] 553
[1] "Benign"
[1] 554
[1] "malignant"
[1] 555
[1] "malignant"
[1] 556
[1] "malignant"
[1] 557
[1] "malignant"
[1] 558
[1] "Benign"
[1] 559
[1] "Benign"
[1] 560
```

```
[1] "malignant"
[1] 561
[1] "Benign"
[1] 562
[1] "Benign"
[1] 563
[1] "Benign"
[1] 564
[1] "Benign"
[1] 565
[1] "Benign"
[1] 566
[1] "Benign"
[1] 567
[1] "malignant"
[1] 568
[1] "malignant"
[1] 569
[1] "Benign"
[1] 570
[1] "Benign"
[1] 571
[1] "Benign"
[1] 572
[1] "malignant"
[1] 573
[1] "Benign"
[1] 574
[1] "malignant"
[1] 575
[1] "Benign"
[1] 576
[1] "malignant"
[1] 577
[1] "malignant"
[1] 578
[1] "malignant"
[1] 579
[1] "Benign"
[1] 580
```

```
[1] "malignant"
[1] 581
[1] "Benign"
[1] 582
[1] "Benign"
[1] 583
[1] "malignant"
[1] 584
[1] "Benign"
[1] 585
[1] "malignant"
[1] 586
[1] "Benign"
[1] 587
[1] "Benign"
[1] 588
[1] "Benign"
[1] 589
[1] "malignant"
[1] 590
[1] "Benign"
[1] 591
[1] "malignant"
[1] 592
[1] "Benign"
[1] 593
[1] "Benign"
[1] 594
[1] "malignant"
[1] 595
[1] "Benign"
[1] 596
[1] "malignant"
[1] 597
[1] "malignant"
[1] 598
[1] "malignant"
[1] 599
[1] "Benign"
[1] 600
```

```
[1] "Benign"
[1] 601
[1] "malignant"
[1] 602
[1] "Benign"
[1] 603
[1] "Benign"
[1] 604
[1] "Benign"
[1] 605
[1] "Benign"
[1] 606
[1] "malignant"
[1] 607
[1] "Benign"
[1] 608
[1] "Benign"
[1] 609
[1] "Benign"
[1] 610
[1] "malignant"
[1] 611
[1] "malignant"
[1] 612
[1] "Benign"
[1] 613
[1] "Benign"
[1] 614
[1] "Benign"
[1] 615
[1] "malignant"
[1] 616
[1] "Benign"
[1] 617
[1] "Benign"
[1] 618
[1] "malignant"
[1] 619
[1] "Benign"
[1] 620
```

```
[1] "malignant"
[1] 621
[1] "malignant"
[1] 622
[1] "malignant"
[1] 623
[1] "Benign"
[1] 624
[1] "Benign"
[1] 625
[1] "Benign"
[1] 626
[1] "Benign"
[1] 627
[1] "Benign"
[1] 628
[1] "Benign"
[1] 629
[1] "Benign"
[1] 630
[1] "Benign"
[1] 631
[1] "Benign"
[1] 632
[1] "Benign"
[1] 633
[1] "malignant"
[1] 634
[1] "Benign"
[1] 635
[1] "Benign"
[1] 636
[1] "Benign"
[1] 637
[1] "Benign"
[1] 638
[1] "Benign"
[1] 639
[1] "Benign"
[1] 640
```

```
[1] "Benign"
[1] 641
[1] "Benign"
[1] 642
[1] "malignant"
[1] 643
[1] "malignant"
[1] 644
[1] "Benign"
[1] 645
[1] "Benign"
[1] 646
[1] "Benign"
[1] 647
[1] "malignant"
[1] 648
[1] "malignant"
[1] 649
[1] "Benign"
[1] 650
[1] "Benign"
[1] 651
[1] "Benign"
[1] 652
[1] "Benign"
[1] 653
[1] "malignant"
[1] 654
[1] "malignant"
[1] 655
[1] "malignant"
[1] 656
[1] "Benign"
[1] 657
[1] "Benign"
[1] 658
[1] "Benign"
[1] 659
[1] "Benign"
[1] 660
```

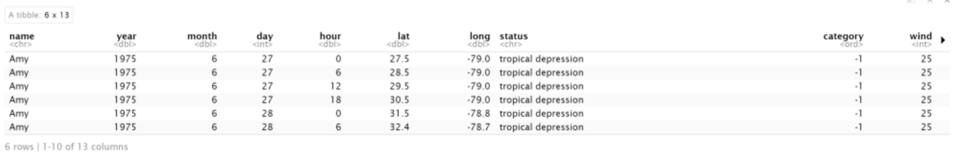
```
[1] "malignant"
[1] 661
[1] "Benign"
[1] 662
[1] "Benign"
[1] 663
[1] "Benign"
[1] 664
[1] "Benign"
[1] 665
[1] "malignant"
[1] 666
[1] "malignant"
[1] 667
[1] "Benign"
[1] 668
[1] "Benign"
[1] 669
[1] "Benign"
[1] 670
[1] "Benign"
[1] 671
[1] "Benign"
[1] 672
[1] "Benign"
[1] 673
[1] "Benign"
[1] 674
[1] "Benign"
[1] 675
[1] "Benign"
[1] 676
[1] "malignant"
[1] 677
[1] "Benign"
[1] 678
[1] "Benign"
[1] 679
[1] "Benign"
[1] 680
[1] "Benign"
[1] 681
```

```
[1] "malignant"
[1] 682
[1] "malignant"
[1] 683
[1] "malignant"
```

PROBLEM 2

In this problem you will generate decision trees with a set of parameters. You will be using the storms data, a subset of the NOAA Atlantic hurricane database2, which includes the positions and attributes of 198 tropical storms (potential hurricanes), measured every six hours during the lifetime of a storm. It is part of the dplyr library, so load the library and you will be able to access it. As a preprocessing step, view the data and make sure the target variable (category) is converted to a factor (as opposed to character string).

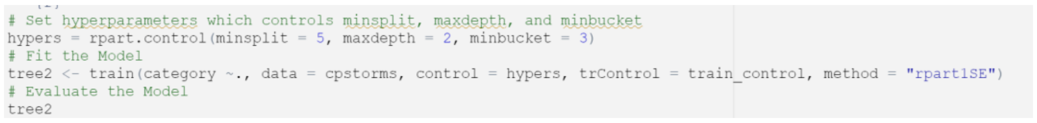
```
# View storms dataset from the dplye library
head(storms)
```



```
A tibble: 6 x 13
name   year  month  day  hour  lat   long   status               category  wind
<chr>  <dbl> <dbl>  <int> <dbl> <dbl> <dbl> <chr>                <ord>     <int>
Amy    1975  6      27   0     27.5  -79.0  tropical depression  -1        25
Amy    1975  6      27   6     28.5  -79.0  tropical depression  -1        25
Amy    1975  6      27   12    29.5  -79.0  tropical depression  -1        25
Amy    1975  6      27   18    30.5  -79.0  tropical depression  -1        25
Amy    1975  6      28   0     31.5  -78.8  tropical depression  -1        25
Amy    1975  6      28   6     32.4  -78.7  tropical depression  -1        25
6 rows | 1-10 of 13 columns
```

```
# Create a copy
cpstorms <- storms
# Convert category to a factor
cpstorms$category <- as.factor(cpstorms$category)
# Check if it was done correctly
is.factor(cpstorms$category)
# Now to remove any rows with NAs
cpstorms <- na.omit(cpstorms)
```
```

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

a. Build a decision tree using the following hyperparameters, maxdepth=2, minsplit=5 and minbucket=3. Be careful to use the right method of training so that you are not automatically tuning the cp parameter, but you are controlling the aforementioned parameters specifically. Use cross validation to report your accuracy score. These parameters will result in a relatively small tree.

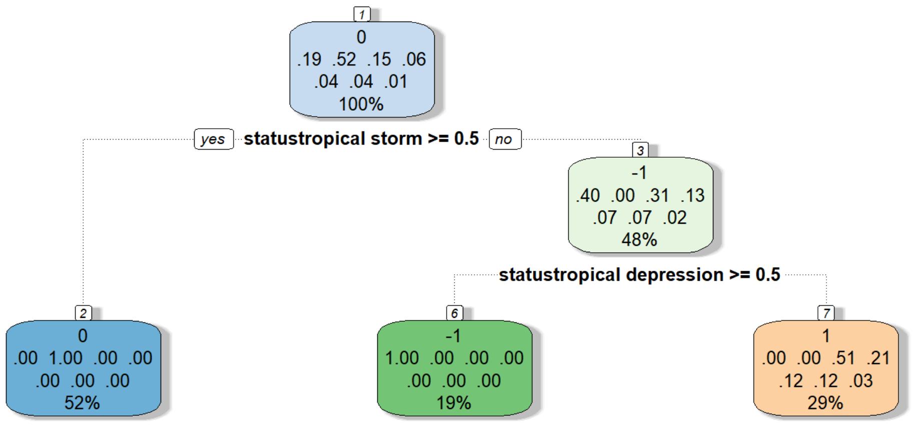


```
# Set hyperparameters which controls minsplit, maxdepth, and minbucket
hypers = rpart.control(minsplit = 5, maxdepth = 2, minbucket = 3)
# Fit the Model
tree2 <- train(category ~., data = cpstorms, control = hypers, trControl = train_control, method = "rpart1SE")
# Evaluate the Model
tree2
```

```
CART

5350 samples
  12 predictor
   7 classes: '-1', '0', '1', '2', '3', '4', '5'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 4816, 4815, 4813, 4815, 4815, 4815, ...
Resampling results:

  Accuracy   Kappa
  0.8594406  0.7842288
```

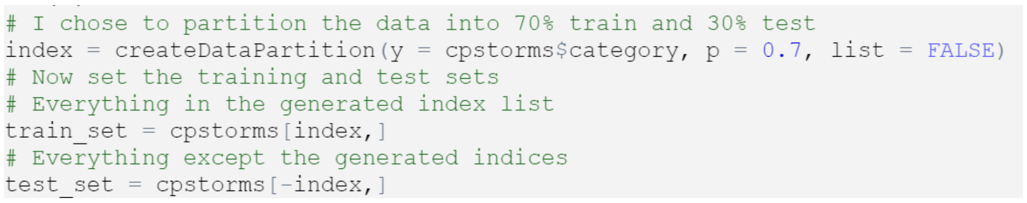
We can see that the accuracy and kappa are both quite high, reporting at 86% and 79% accordingly. Now, let's create a decision tree based on this data.

```
# Visualize the Decision Tree
fancyRpartPlot(tree2$finalModel, caption = "")
```



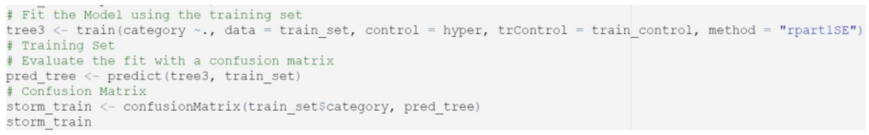
This resulted in a small tree, which is good to note.

b. To see how this performed with respect to the individual classes, we could use a confusion matrix. We also want to see if that aspect of performance is different on the train versus the test set. Create a train/test partition. Train on the training set. By making predictions with that model on the train set and on the test set separately, use the outputs to create two separate confusion matrices, one for each partition. Remember, we are testing if the model built with the training data performs differently on data used to train it (train set) as opposed to new data (test set). Compare the confusion matrices and report which classes it has problem classifying. Do you think that both are performing similarly and what does that suggest about overfitting for the model?

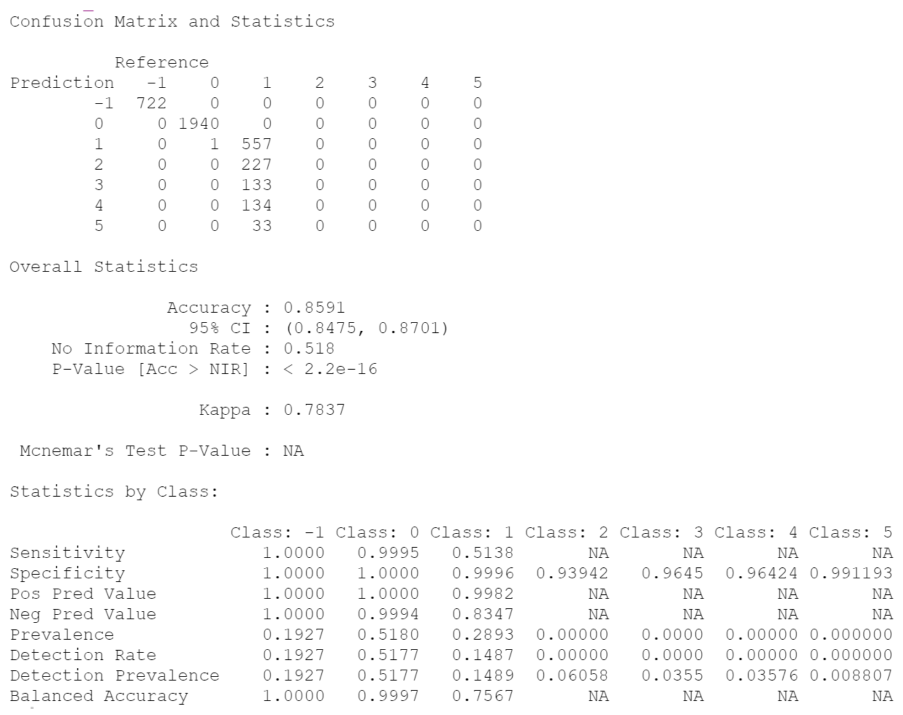


```
# I chose to partition the data into 70% train and 30% test
index = createDataPartition(y = cpstorms$category, p = 0.7, list = FALSE)
# Now set the training and test sets
# Everything in the generated index list
train_set = cpstorms[index,]
# Everything except the generated indices
test_set = cpstorms[-index,]
```

Now, I'm going to evaluate the fit for both training set and test set, then create a confusion matrix for each, then send the accuracy scores to a table (since we're mainly concerned about the accuracy comparison here).

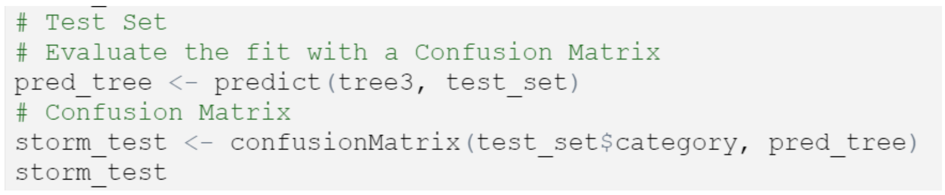


```
# Fit the Model using the training set
tree3 <- train(category ~., data = train_set, control = hyper, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree3, train_set)
# Confusion Matrix
storm_train <- confusionMatrix(train_set$category, pred_tree)
storm_train
```

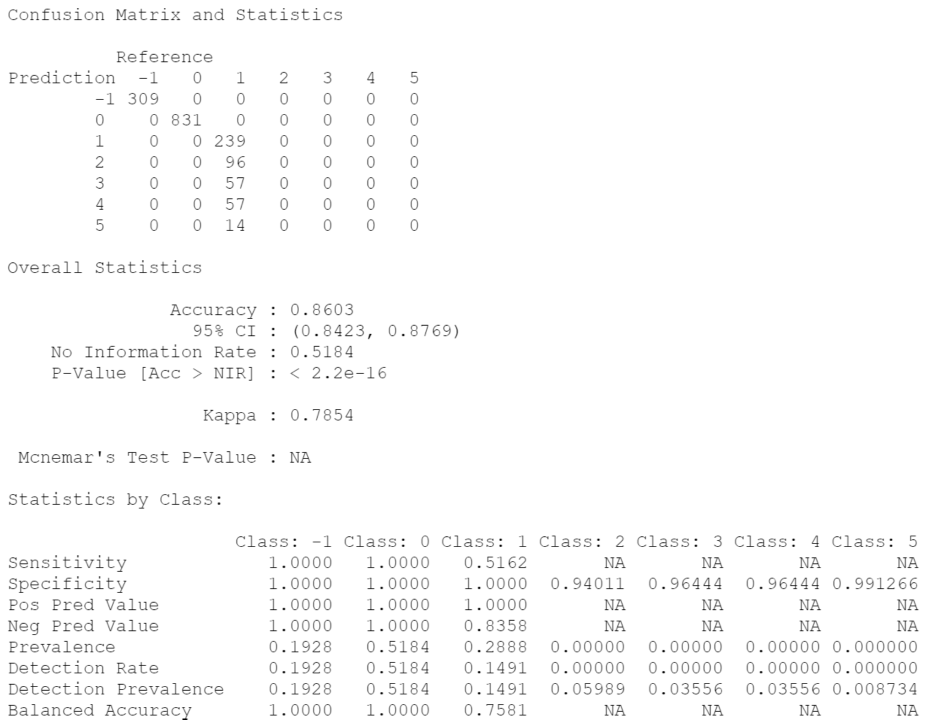


```
Confusion Matrix and Statistics

          Reference
Prediction   -1    0    1    2    3    4    5
        -1  722    0    0    0    0    0    0
        0     0 1940    0    0    0    0    0
        1     0    1  557    0    0    0    0
        2     0    0  227    0    0    0    0
        3     0    0  133    0    0    0    0
        4     0    0  134    0    0    0    0
        5     0    0   33    0    0    0    0

Overall Statistics

               Accuracy : 0.8591
                 95% CI : (0.8475, 0.8701)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7837

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: -1 Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity             1.0000   0.9995   0.5138       NA       NA       NA       NA
Specificity             1.0000   1.0000   0.9996  0.93942   0.9645  0.96424 0.991193
Pos Pred Value          1.0000   1.0000   0.9982       NA       NA       NA       NA
Neg Pred Value          1.0000   0.9994   0.8347       NA       NA       NA       NA
Prevalence              0.1927   0.5180   0.2893  0.00000   0.0000  0.00000 0.000000
Detection Rate          0.1927   0.5177   0.1487  0.00000   0.0000  0.00000 0.000000
Detection Prevalence    0.1927   0.5177   0.1489  0.06058   0.0355  0.03576 0.008807
Balanced Accuracy       1.0000   0.9997   0.7567       NA       NA       NA       NA
```

The training set model here seems to be struggling with classes 2, 3, 4, and 5. Accuracy is being reported at 86%, with a kappa value of ~79%. Now, let's take a look at the test set model.

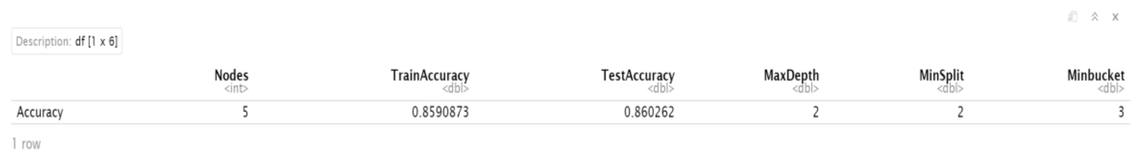


```
# Test Set
# Evaluate the fit with a Confusion Matrix
pred_tree <- predict(tree3, test_set)
# Confusion Matrix
storm_test <- confusionMatrix(test_set$category, pred_tree)
storm_test
```



```
Confusion Matrix and Statistics

          Reference
Prediction  -1   0   1   2   3   4   5
        -1 309   0   0   0   0   0   0
        0    0 831   0   0   0   0   0
        1    0   0 239   0   0   0   0
        2    0   0  96   0   0   0   0
        3    0   0  57   0   0   0   0
        4    0   0  57   0   0   0   0
        5    0   0  14   0   0   0   0

Overall Statistics

               Accuracy : 0.8603
                 95% CI : (0.8423, 0.8769)
    No Information Rate : 0.5184
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7854

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: -1 Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity             1.0000   1.0000   0.5162       NA       NA       NA       NA
Specificity             1.0000   1.0000   1.0000  0.94011  0.96444  0.96444 0.991266
Pos Pred Value          1.0000   1.0000   1.0000       NA       NA       NA       NA
Neg Pred Value          1.0000   1.0000   0.8358       NA       NA       NA       NA
Prevalence              0.1928   0.5184   0.2888  0.00000  0.00000  0.00000 0.000000
Detection Rate          0.1928   0.5184   0.1491  0.00000  0.00000  0.00000 0.000000
Detection Prevalence    0.1928   0.5184   0.1491  0.05989  0.03556  0.03556 0.008734
Balanced Accuracy       1.0000   1.0000   0.7581       NA       NA       NA       NA
```

The test set based model is also struggling with classes 2, 3, 4, and 5. Accuracy looks comparable at 86%, with a kappa value of ~78%. Now, I created a small table to better visualize what went into these models, and how the accuracies compare.



Description: df [1 x 6]

	Nodes <int>	TrainAccuracy <dbl>	TestAccuracy <dbl>	MaxDepth <dbl>	MinSplit <dbl>	Minbucket <dbl>
Accuracy	5	0.8590873	0.860262	2	2	3

1 row

```
# Get the training accuracy
a_train <- storm_train$overall[1]
# Get the testing accuracy
a_test <- storm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree3$finalModel$frame)
# Form the table
comp_tbl <- data.frame("Nodes" = nodes, "TrainAccuracy" = a_train, "TestAccuracy" = a_test, "MaxDepth" = 2, "MinSplit" = 2, "Minbucket" = 3)
comp_tbl
```

Both training and test sets have very close accuracy scores. Judging by this we can safely suspect that there might not be any overfitting in the data.

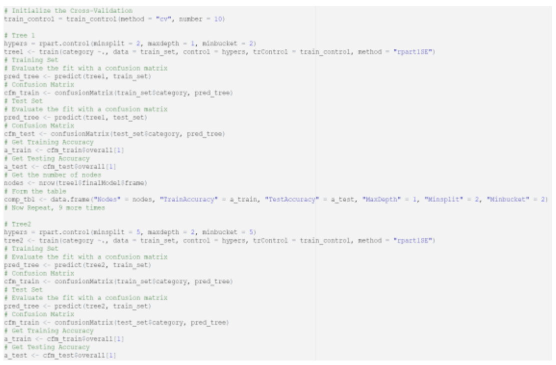
PROBLEM 3

This is will be an extension of Problem 2, using the same data and class. Here you will build many decision trees, manually tuning the parameters to gain intuition about the tradeoffs and how these tree parameters affect the complexity and quality of the model. The goal is to find the best tree model, which means it should be accurate but not too complex that the model overfits the training data. We will achieve this by using multiple sets of parameters and creating a graph of accuracy versus complexity for the training and the test sets (refer to the tutorial). This problem may require a significant amount of effort because you will need to train a substantial number of trees (at least 10).

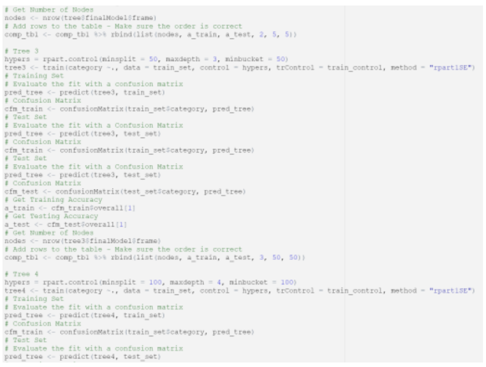
a. Partition your data into 80% for training and 20% for the test data set

```
# Partition the data at 80% for training, which is the 0.8 here
index = createDataPartition(y = cpstorms$category, p = 0.8, list = FALSE)
# Everything in the generated index list
train_set = cpstorms[index,]
# Everything except the generated indices
test_set = cpstorms[-index,]
```

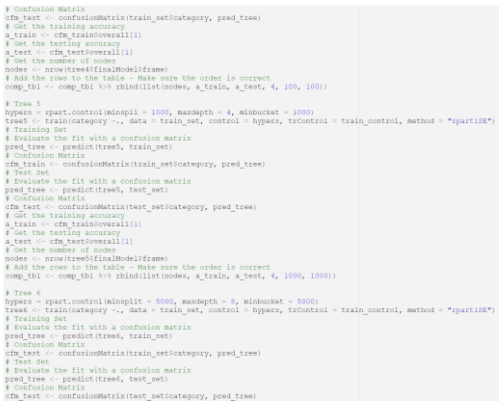
b. Train at least 10 trees using different sets of parameters, through you made need more. Create the graph described above such that you can identify the inflection point where the tree is overfitting and pick a high-quality decision tree. Your strategy should be to make at least one very simple model and at least one very complex model and work towards the center by changing different parameters. Generate a table that contains all of the parameters (maxdepth, minsplit, minbucket, etc) used along with the number of nodes created, and the training and testing set accuracy values. The number of rows will be equal to the number of sets of parameters used. You will use the data in the table to generate the graph. The final results to be reported for this problem are the table and graph.



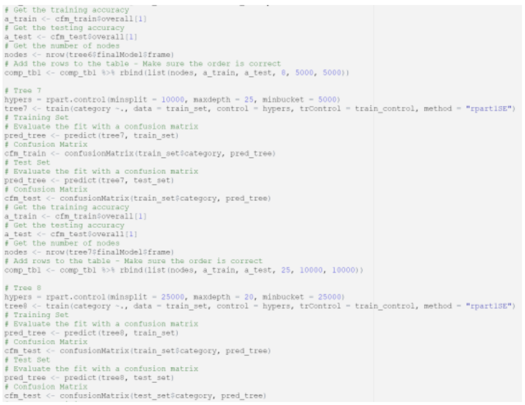
```r
# Initialize the Cross-Validation
train_control = train_control(method = "cv", number = 10)

# Tree 1
hypers = rpart.control(minsplit = 2, maxdepth = 1, minbucket = 2)
tree1 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree1, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree1, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(test_set$category, pred_tree)
# Get Training Accuracy
a_train <- cfm_train$overall[1]
# Get Testing Accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree1$finalModel$frame)
# Form the table
comp_tbl <- data.frame("Nodes" = nodes, "TrainAccuracy" = a_train, "TestAccuracy" = a_test, "MaxDepth" = 1, "Minsplit" = 2, "Minbucket" = 2)
# Now Repeat, 9 more times

# Tree2
hypers = rpart.control(minsplit = 5, maxdepth = 2, minbucket = 5)
tree2 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree2, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree2, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(test_set$category, pred_tree)
# Get Training Accuracy
a_train <- cfm_train$overall[1]
# Get Testing Accuracy
a_test <- cfm_test$overall[1]
```



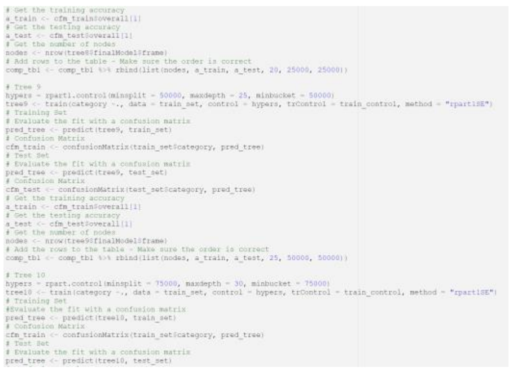
```r
# Get Number of Nodes
nodes <- nrow(tree$finalModel$frame)
# Add rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 2, 5, 5))

# Tree 3
hypers = rpart.control(minsplit = 50, maxdepth = 3, minbucket = 50)
tree3 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree3, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a Confusion Matrix
pred_tree <- predict(tree3, test_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a Confusion Matrix
pred_tree <- predict(tree3, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(test_set$category, pred_tree)
# Get Training Accuracy
a_train <- cfm_train$overall[1]
# Get Testing Accuracy
a_test <- cfm_test$overall[1]
# Get Number of Nodes
nodes <- nrow(tree3$finalModel$frame)
# Add rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 3, 50, 50))

# Tree 4
hypers = rpart.control(minsplit = 100, maxdepth = 4, minbucket = 100)
tree4 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree4, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree4, test_set)
```



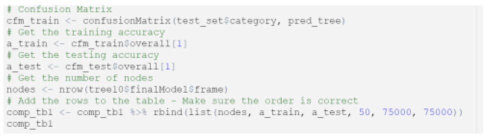
```r
# Confusion Matrix
cfm_test <- confusionMatrix(train_set$category, pred_tree)
# Get the training accuracy
a_train <- cfm_train$overall[1]
# Get the testing accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree4$finalModel$frame)
# Add the rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 4, 100, 100))

# Tree 5
hypers = rpart.control(minspli = 1000, maxdepth = 4, minbucket = 1000)
tree5 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree5, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree5, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(test_set$category, pred_tree)
# Get the training accuracy
a_train <- cfm_train$overall[1]
# Get the testing accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree5$finalModel$frame)
# Add the rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 4, 1000, 1000))

# Tree 6
hypers = rpart.control(minsplit = 5000, maxdepth = 8, minbucket = 5000)
tree6 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree6, train_set)
# Confusion Matrix
cfm_test <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree6, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(test_set$category, pred_tree)
```



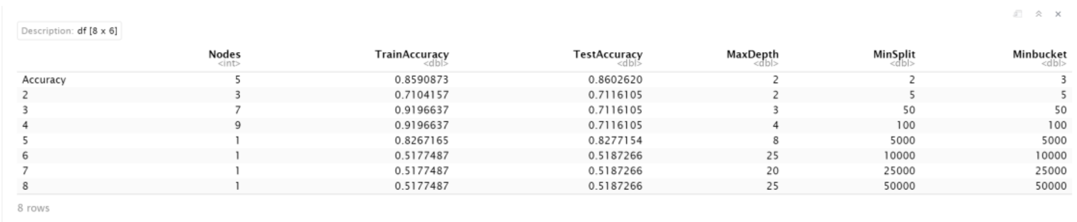
```r
# Get the training accuracy
a_train <- cfm_train$overall[1]
# Get the testing accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree6$finalModel$frame)
# Add the rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 8, 5000, 5000))

# Tree 7
hypers = rpart.control(minsplit = 10000, maxdepth = 25, minbucket = 5000)
tree7 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree7, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree7, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(train_set$category, pred_tree)
# Get the training accuracy
a_train <- cfm_train$overall[1]
# Get the testing accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree7$finalModel$frame)
# Add rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 25, 10000, 10000))

# Tree 8
hypers = rpart.control(minsplit = 25000, maxdepth = 20, minbucket = 25000)
tree8 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree8, train_set)
# Confusion Matrix
cfm_test <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree8, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(test_set$category, pred_tree)
```



```r
# Get the training accuracy
a_train <- cfm_train$overall[1]
# Get the testing accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree8$finalModel$frame)
# Add rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 20, 25000, 25000))

# Tree 9
hypers = rpart1.control(minsplit = 50000, maxdepth = 25, minbucket = 50000)
tree9 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree9, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree9, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(test_set$category, pred_tree)
# Get the training accuracy
a_train <- cfm_train$overall[1]
# Get the testing accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree9$finalModel$frame)
# Add the rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 25, 50000, 50000))

# Tree 10
hypers = rpart.control(minsplit = 75000, maxdepth = 30, minbucket = 75000)
tree10 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
#Evaluate the fit with a confusion matrix
pred_tree <- predict(tree10, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree10, test_set)
```



```r
# Confusion Matrix
cfm_train <- confusionMatrix(test_set$category, pred_tree)
# Get the training accuracy
a_train <- cfm_train$overall[1]
# Get the testing accuracy
a_test <- cfm_test$overall[1]
# Get the number of nodes
nodes <- nrow(tree10$finalModel$frame)
# Add the rows to the table - Make sure the order is correct
comp_tbl <- comp_tbl %>% rbind(list(nodes, a_train, a_test, 50, 75000, 75000))
comp_tbl
```



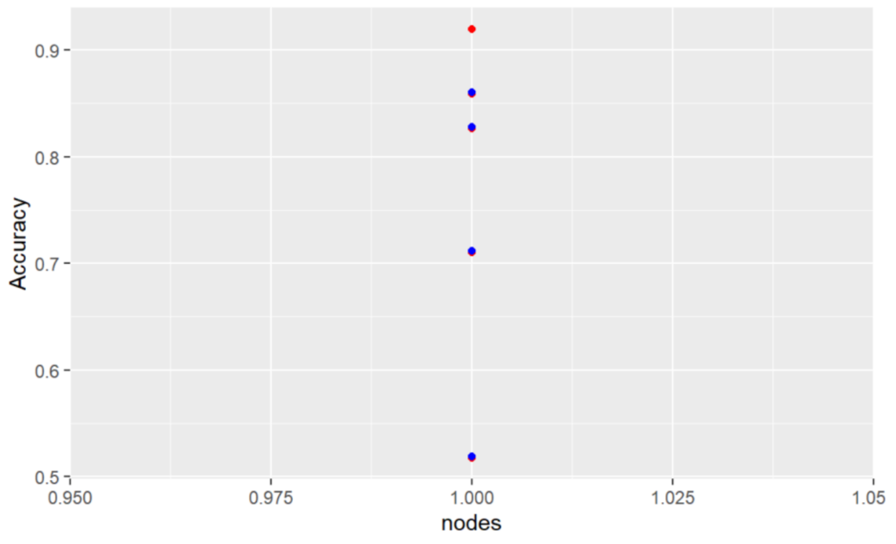
Description: df [8 x 6]

	Nodes <int>	TrainAccuracy <dbl>	TestAccuracy <dbl>	MaxDepth <dbl>	MinSplit <dbl>	Minbucket <dbl>
Accuracy	5	0.8590873	0.8602620	2	2	3
2	3	0.7104157	0.7116105	2	5	5
3	7	0.9196637	0.7116105	3	50	50
4	9	0.9196637	0.7116105	4	100	100
5	1	0.8267165	0.8277154	8	5000	5000
6	1	0.5177487	0.5187266	25	10000	10000
7	1	0.5177487	0.5187266	20	25000	25000
8	1	0.5177487	0.5187266	25	50000	50000

8 rows

Now, we're visualizing this data in the scatter plot with the nodes on the x-axis.

```
# Visualize with the scatter plot
ggplot(comp_tbl, aes(x = nodes)) +
  geom_point(aes(y = TrainAccuracy), color = "red") +
  geom_point(aes(y = TestAccuracy), color = "blue") +
  ylab("Accuracy")
```



c. **Identify the final choice of model, list it parameters and evaluate with a confusion matrix to make sure that it gets balanced performance over classes. Also get a better accuracy estimate for this tree using cross validation.**

While the 4th tree had the best accuracy values, with ~96% reported for both test and training sets, it had 9 nodes. This is quite complex. However, I think in this case that the 2nd tree would be the best model as it has a test and training set accuracy of 86% with only 5 nodes. This is far less complex with high accuracy, which makes for a decent model. Another thing to note is that the 6th, 7th, 8th, 9th, and 10th trees provided us with insignificant information and only 1 node each. Therefore, the 2nd tree we modelled would be the best model, with accuracies being similar, and 5 nodes, in which we can confidently say this is a balanced dataset. The 2nd tree had the following hyper parameters: minsplit value of 5, maxdepth of 2, and minbucket of 5.

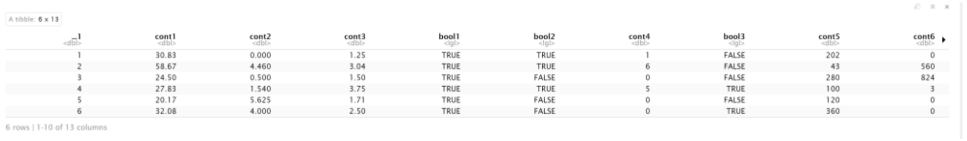
```r
hypers = rpart.control(minsplit = 5, maxdepth = 2, minbucket = 5)
tree2 <- train(category ~., data = train_set, control = hypers, trControl = train_control, method = "rpart1SE")
# Training Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree2, train_set)
# Confusion Matrix
cfm_train <- confusionMatrix(train_set$category, pred_tree)
# Test Set
# Evaluate the fit with a confusion matrix
pred_tree <- predict(tree2, test_set)
# Confusion Matrix
cfm_test <- confusionMatrix(test_set$category, pred_tree)
cfm_train
```

```
Confusion Matrix and Statistics

          Reference
Prediction   -1    0    1    2    3    4    5
        -1  825    0    0    0    0    0    0
        0     0 2217    0    0    0    0    0
        1     0    1  637    0    0    0    0
        2     0    0  259    0    0    0    0
        3     0    0  152    0    0    0    0
        4     0    0  153    0    0    0    0
        5     0    0   38    0    0    0    0

Overall Statistics
                                          
               Accuracy : 0.8592          
                 95% CI : (0.8484, 0.8695)
    No Information Rate : 0.518           
    P-Value [Acc > NIR] : < 2.2e-16       
                                          
                  Kappa : 0.7839          
                                          
 Mcnemar's Test P-Value : NA              

Statistics by Class:

                     Class: -1 Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity             1.0000   0.9995   0.5141       NA       NA       NA
Specificity             1.0000   1.0000   0.9997  0.93951   0.9645  0.96427
Pos Pred Value          1.0000   1.0000   0.9984       NA       NA       NA
Neg Pred Value          1.0000   0.9995   0.8348       NA       NA       NA
Prevalence              0.1927   0.5180   0.2894  0.00000   0.0000  0.00000
Detection Rate          0.1927   0.5177   0.1488  0.00000   0.0000  0.00000
Detection Prevalence    0.1927   0.5177   0.1490  0.06049   0.0355  0.03573
Balanced Accuracy       1.0000   0.9998   0.7569       NA       NA       NA
                     Class: 5
Sensitivity                NA
Specificity          0.991126
Pos Pred Value             NA
Neg Pred Value             NA
Prevalence           0.000000
Detection Rate       0.000000
Detection Prevalence 0.008874
Balanced Accuracy          NA
```

PROBLEM 4

In this problem you will identify the most important independent variables used in a classification model. Use the Bank_Modified.csv data. As a pre-processing step, remove the ID column and make sure to convert the target variable, approval, from a string to a factor.

```r
# Import dataset
library(readxl)
Bank_Modified <- read_csv("C:/Users/admin/Desktop/Bank_Modified.csv")
# Remove ID column
Bank_Modified <- Bank_Modified %>% select(-c("ID"))
# Convert approval to factor
Bank_Modified$approval <- as.factor(Bank_Modified$approval)
# Remove NA's
Bank_Modified <- na.omit(Bank_Modified)
head(Bank_Modified)
```



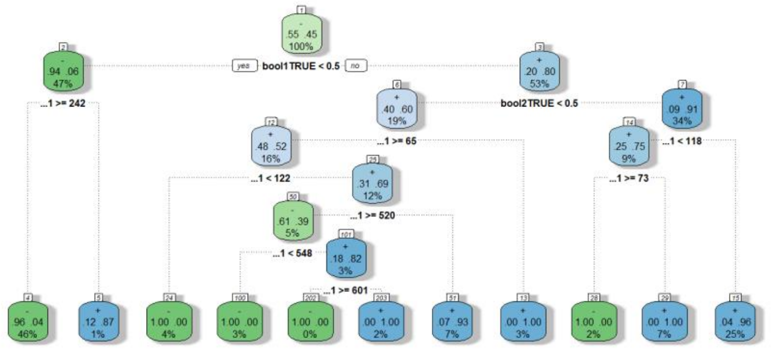
A tibble: 6 x 13

...1 <dbl>	cont1 <dbl>	cont2 <dbl>	cont3 <dbl>	bool1 <lgl>	bool2 <lgl>	cont4 <dbl>	bool3 <lgl>	cont5 <dbl>	cont6 <dbl>
1	30.83	0.000	1.25	TRUE	TRUE	1	FALSE	202	0
2	58.67	4.460	3.04	TRUE	TRUE	6	FALSE	43	560
3	24.50	0.500	1.50	TRUE	FALSE	0	FALSE	280	824
4	27.83	1.540	3.75	TRUE	TRUE	5	TRUE	100	3
5	20.17	5.625	1.71	TRUE	FALSE	0	FALSE	120	0
6	32.08	4.000	2.50	TRUE	FALSE	0	TRUE	360	0

6 rows | 1-10 of 13 columns

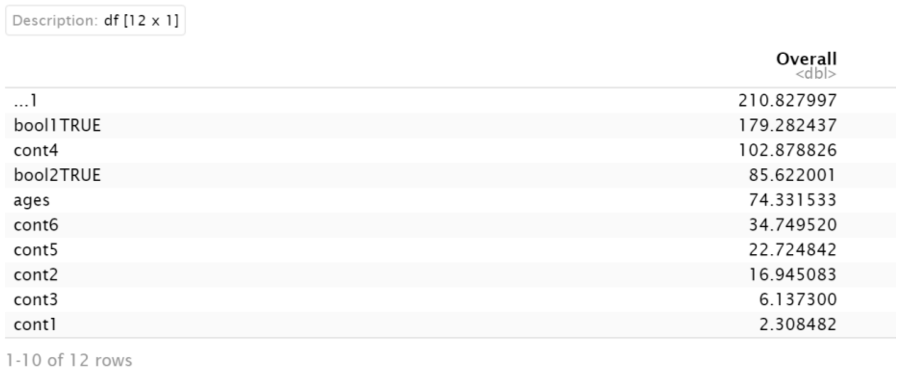
a. Build your initial decision tree model with minsplit=10 and maxdepth=20

```
# Set the hyperparameters
hypers = rpart.control(minsplit = 10, maxdepth = 20)
# Fit the Model
tree1 <- train(approval ~., data = Bank_Modified, control = hypers, trControl = train_control, method = "rpart1SE")
# Visualize the decision tree
fancyRpartPlot(tree1$finalModel, caption = "")
```



b. Run variable importance analysis on the model and print the result.

```
# Fit the Model
tree1 <- train(approval ~., data = Bank_Modified, method = "rpart1SE", trControl = train_control)
# View the variable importance scores using the varImp function
var_imp <- varImp(tree1, scale = FALSE)
print(var_imp)
```



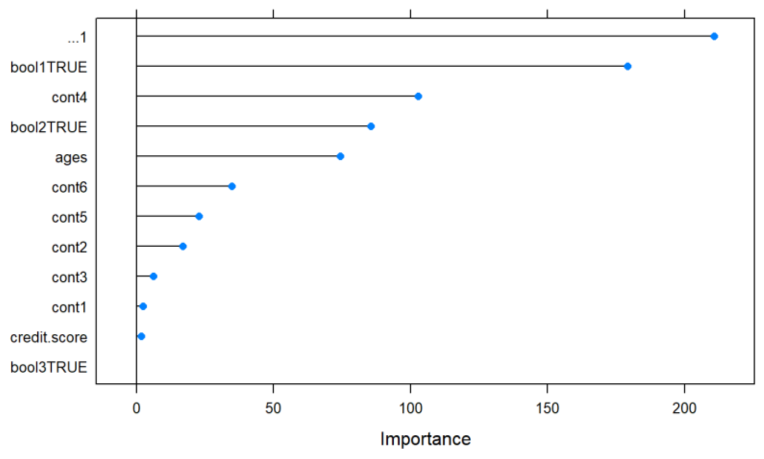
Description: df [12 x 1]

	Overall <dbl>
...1	210.827997
bool1TRUE	179.282437
cont4	102.878826
bool2TRUE	85.622001
ages	74.331533
cont6	34.749520
cont5	22.724842
cont2	16.945083
cont3	6.137300
cont1	2.308482

1-10 of 12 rows

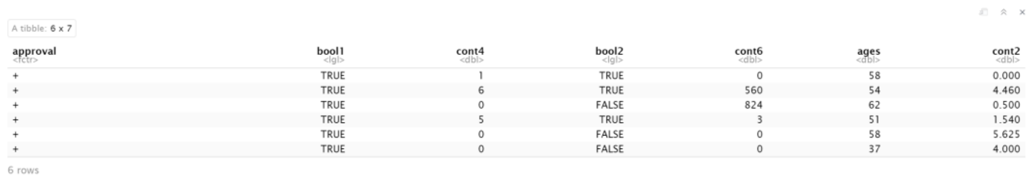
c. Generate a plot to visualize the variables by importance.

```
plot(var_imp)
```



d. Rebuild your model with the top six variables only, based on the variable relevance analysis. Did this change have an effect on the accuracy?

```
# Make a copy with approval and 6 top predictors
newBank_Modified <- Bank_Modified %>% select(c("approval", "bool1", "cont4", "bool2", "cont6", "ages", "cont2"))
head(newBank_Modified)
```

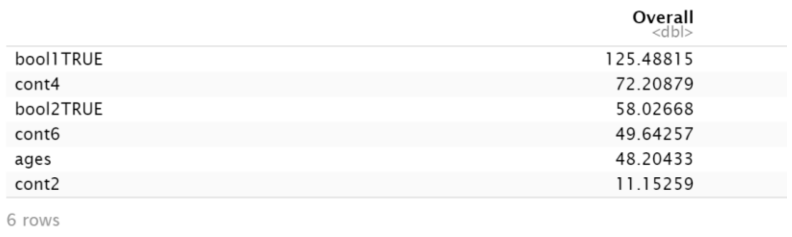

A tibble: 6 x 7

approval <fctr>	bool1 <lgl>	cont4 <dbl>	bool2 <lgl>	cont6 <dbl>	ages <dbl>	cont2 <dbl>
+	TRUE	1	TRUE	0	58	0.000
+	TRUE	6	TRUE	560	54	4.460
+	TRUE	0	FALSE	824	62	0.500
+	TRUE	5	TRUE	3	51	1.540
+	TRUE	0	FALSE	0	58	5.625
+	TRUE	0	FALSE	0	37	4.000

6 rows

Now, partition the data into training and test sets to create tree2 based on the 6 predictors, then get the importance variable

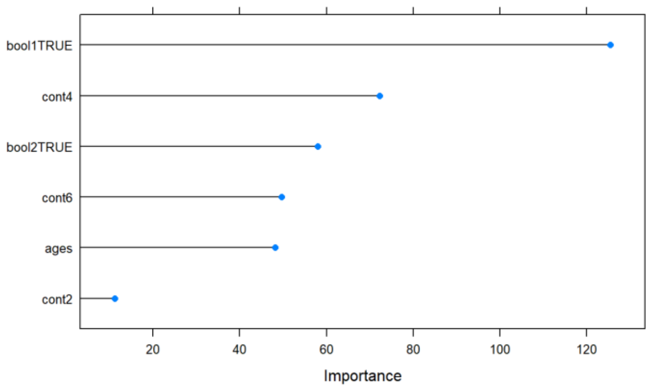
```
# Partition into training and test sets
index = createDataPartition(y = newBank_Modified$approval, p=0.7, list = FALSE)
train_set = newBank_Modified[index,]
test_set = newBank_Modified[-index,]
tree2 <- train(approval ~., data = train_set, method = "rpart1SE", trControl = train_control)
# View the variable importance scores using the varImp function
var_imp <- varImp(tree2, scale = FALSE)
print(var_imp)
```



	Overall <dbl>
bool1TRUE	125.48815
cont4	72.20879
bool2TRUE	58.02668
cont6	49.64257
ages	48.20433
cont2	11.15259

6 rows

```
plot(var_imp)
```



Viewing tree 2 composed of the full dataset:

```
tree2
```

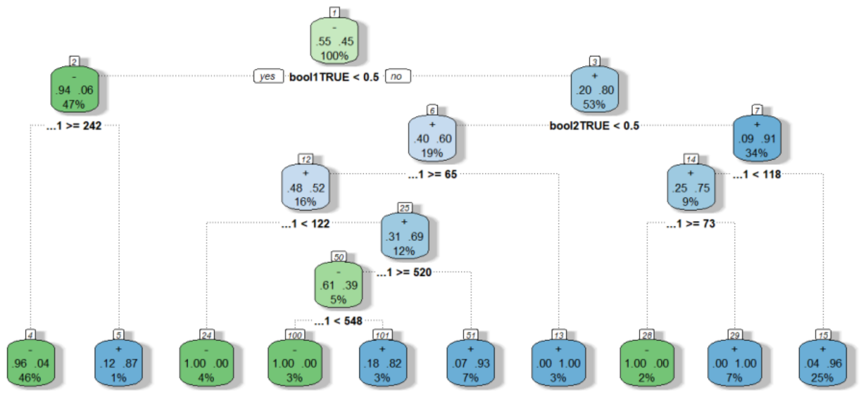
```
CART

484 samples
  6 predictor
  2 classes: '-', '+'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 435, 436, 437, 435, 435, 435, ...
Resampling results:

  Accuracy   Kappa
  0.8537325  0.7027476
```

We can see here that the accuracy and kappa both slightly increased as we used only the 6 predictors. This method helped in removal of some of the noise caused by other, possible irrelevant, predictors.

e. Visualize the trees from (a) and (d) and report if reducing the number of variables had an effect on the size of the tree?

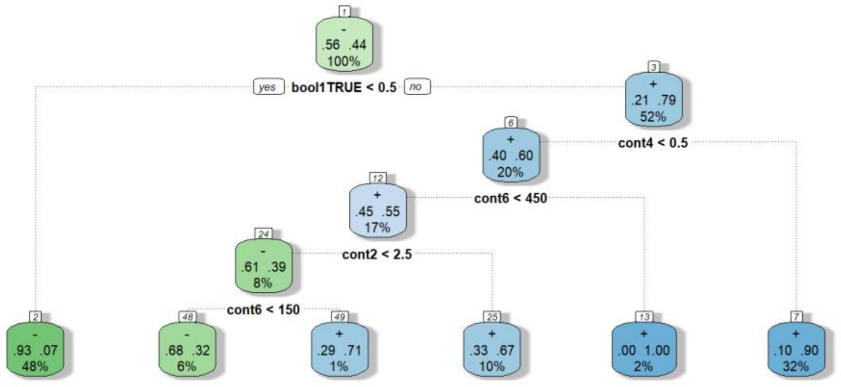
Tree 1:

```
fancyRpartPlot(tree1$finalModel, caption = "")
```



Tree 2:

```
fancyRpartPlot(tree2$finalModel, caption = "")
```



We can see here that there is 13 nodes when using the 6 predictors only, with 15 nodes when using the full dataset. There is also far more decision steps in Tree 1 compared to Tree 2. Therefore, based on the number of nodes, accuracy, and the importance variable, we can confidently say that using the 6 predictors alone may provide us with a more accurate model compared to using the complete dataset.