Statistical Learning HW 9 - Unsupervised Learning

Christian Conroy

April 29, 2019

Part I: Bikeshare Ridership

The first part of the exam uses data on hourly ridership counts for the Capital Bikeshare system in Washington, DC for the years 2011 and 2012. Use the data frame cabi. The data frame contains time related variables and weather related variables, plus two numerical target variables. Each observation contains data for one hour during these two years, with a few gaps.

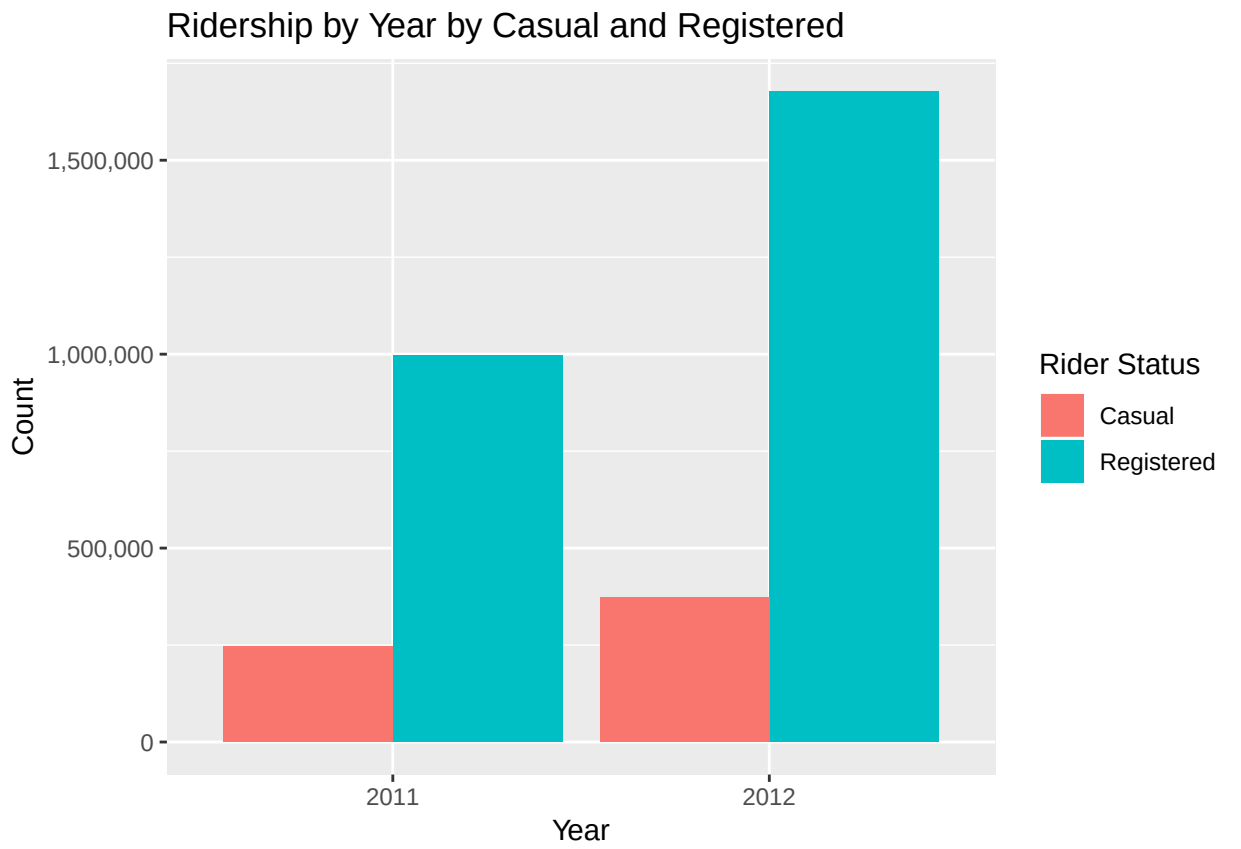
```
cabi <- read.csv("cabi.csv", stringsAsFactors = FALSE)
head(cabi)
```

```
##   X season year month wday hr temp atemp  hum windspeed weather casual
## 1 1      1 2011     1    0  0 0.24 0.288 0.81    0.0000       1      3
## 2 2      1 2011     1    0  1 0.22 0.273 0.80    0.0000       1      8
## 3 3      1 2011     1    0  2 0.22 0.273 0.80    0.0000       1      5
## 4 4      1 2011     1    0  3 0.24 0.288 0.75    0.0000       1      3
## 5 5      1 2011     1    0  4 0.24 0.288 0.75    0.0000       1      0
## 6 6      1 2011     1    0  5 0.24 0.258 0.75    0.0896       2      0
##   registered
## 1         13
## 2         32
## 3         27
## 4         10
## 5          1
## 6          1
```

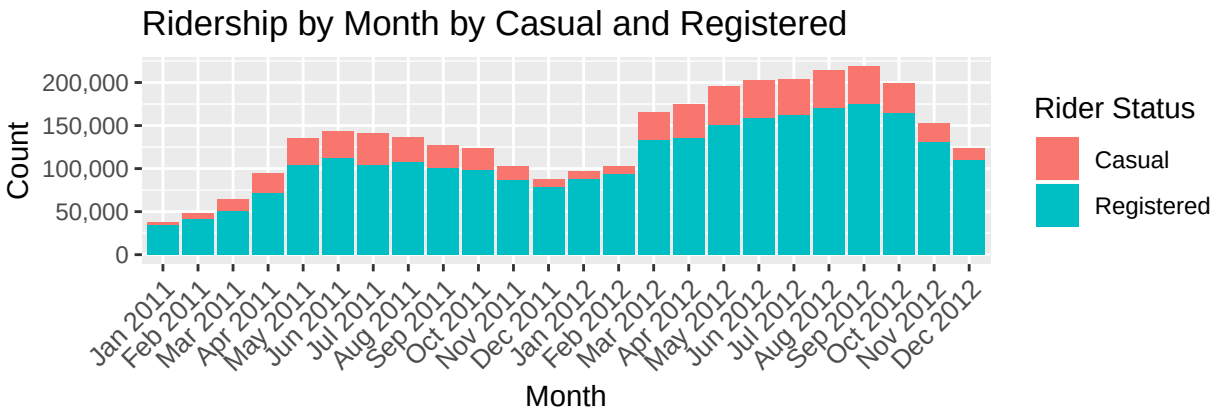
Problem 1 (20) Use numerical summaries, graphs, etc. to answer the following questions. No model fitting or other statistical procedures are required for this. Each graph should help answer one or more of these questions and should be accompanied by explanations.

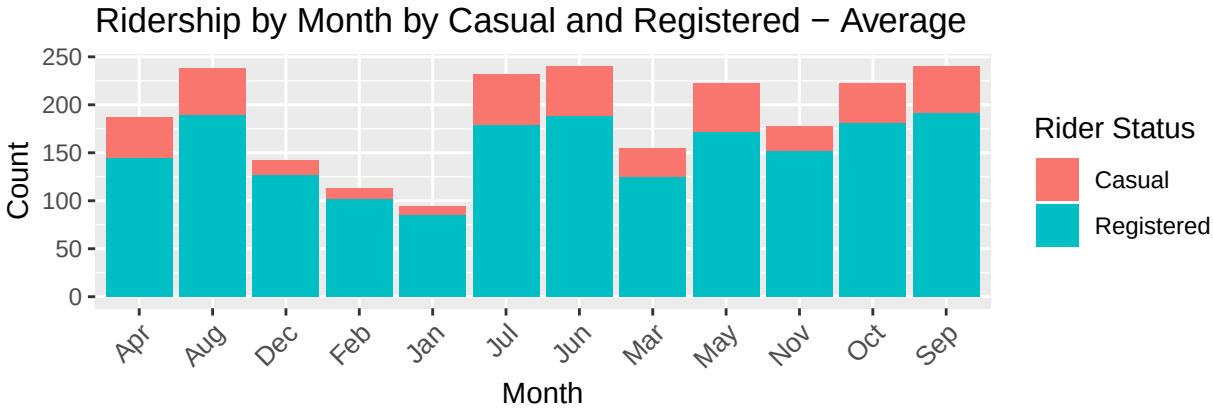
(a) How do ridership counts depend on the year? The month? The hour of the day? How do casual and registered riders differ in this respect?

```
# Year
yeardf <- cabi %>%
  group_by(year) %>%
  dplyr::summarize(Casual = sum(casual),
                   Registered = sum(registered))

yeardf <- melt(data.frame(yeardf), id.vars = 'year')

ggplot(yeardf, aes(x=factor(year), y=value, fill=variable)) + geom_bar(stat='identity', position='dodge
```

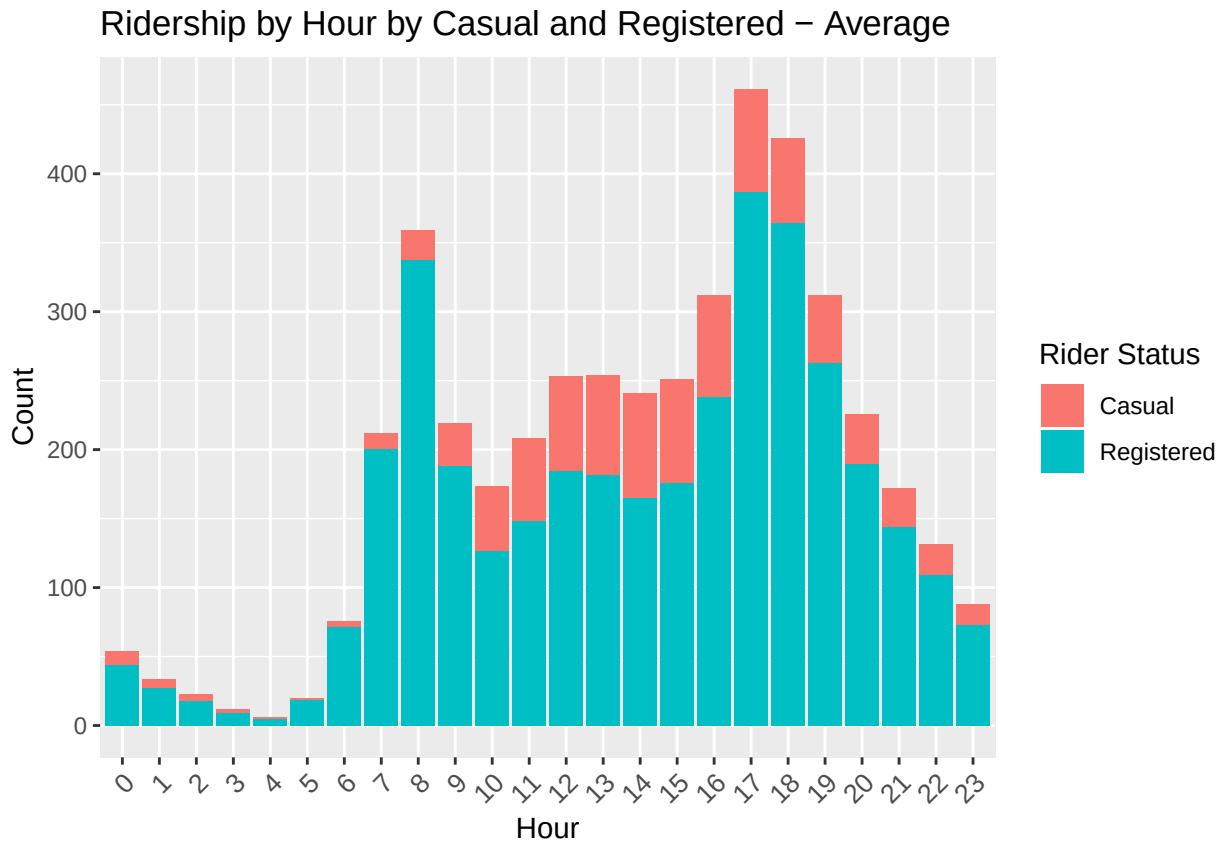



```
# Month
# MonthYear
cabi$month <- sprintf("%02d",cabi$month)
cabi$yearmonth <- zoo::as.yearmon(paste(cabi$year, cabi$month), "%Y %m")

yearmonthdf <- cabi %>%
  group_by(yearmonth) %>%
  dplyr::summarize(Casual = sum(casual),
                   Registered = sum(registered))

yearmonthdf <- melt(data.frame(yearmonthdf), id.vars = 'yearmonth')

a <- ggplot(yearmonthdf, aes(x=factor(yearmonth), y=value, fill=variable)) + geom_bar(stat='identity') +

# Avg. Month
monthdf <- cabi %>%
  group_by(month) %>%
  dplyr::summarize(Casual = mean(casual),
                   Registered = mean(registered))
monthdf$month <- month.abb[as.numeric(monthdf$month)]
monthdf <- melt(data.frame(monthdf), id.vars = 'month')

b <- ggplot(monthdf, aes(x=factor(month), y=value, fill=variable)) + geom_bar(stat='identity') + labs(t

grid.arrange(a, b, nrow = 2, ncol=1)
```





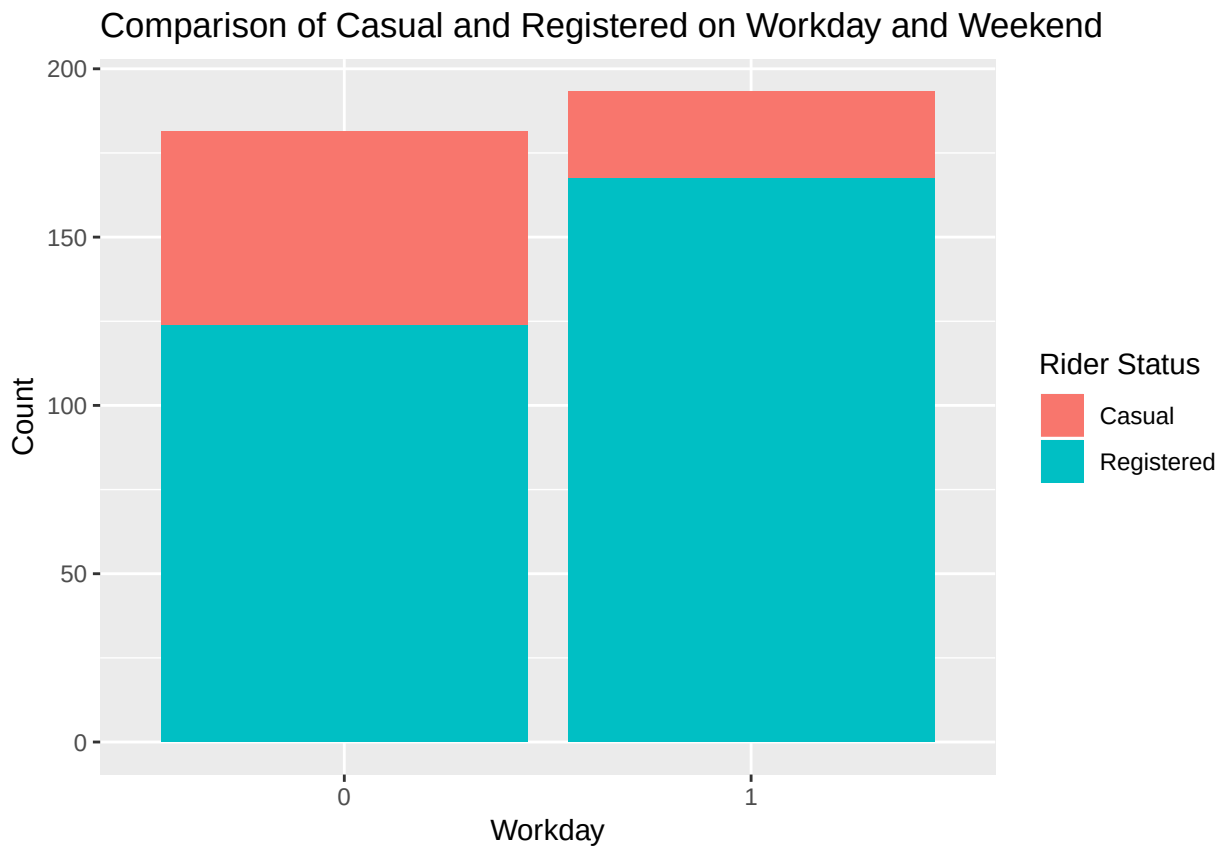

```
#dev.off()

# Avg. Hour of the day
hrdf <- cabi %>%
  group_by(hr) %>%
  dplyr::summarize(Casual = mean(casual),
                   Registered = mean(registered))

hrdf <- melt(data.frame(hrdf), id.vars = 'hr')

ggplot(hrdf, aes(x=factor(hr), y=value, fill=variable)) + geom_bar(stat='identity') + labs(title = "Rid
```



Ridership overall increased between 2011 and 2012, though as the monthly breakdown shows, the increase was concentrated in the warmer months of the respective years.

(b) How are casual and registered ridership counts related? Does this depend on the year? Does it depend on the type of day (working day or not)?

```
wddf <- cabi %>%
  group_by(wday) %>%
  dplyr::summarize(Casual = mean(casual),
                   Registered = mean(registered))

wddf <- melt(data.frame(wddf), id.vars = 'wday')

ggplot(wddf, aes(x=factor(wday), y=value, fill=variable)) + geom_bar(stat='identity') + labs(title = "C
```

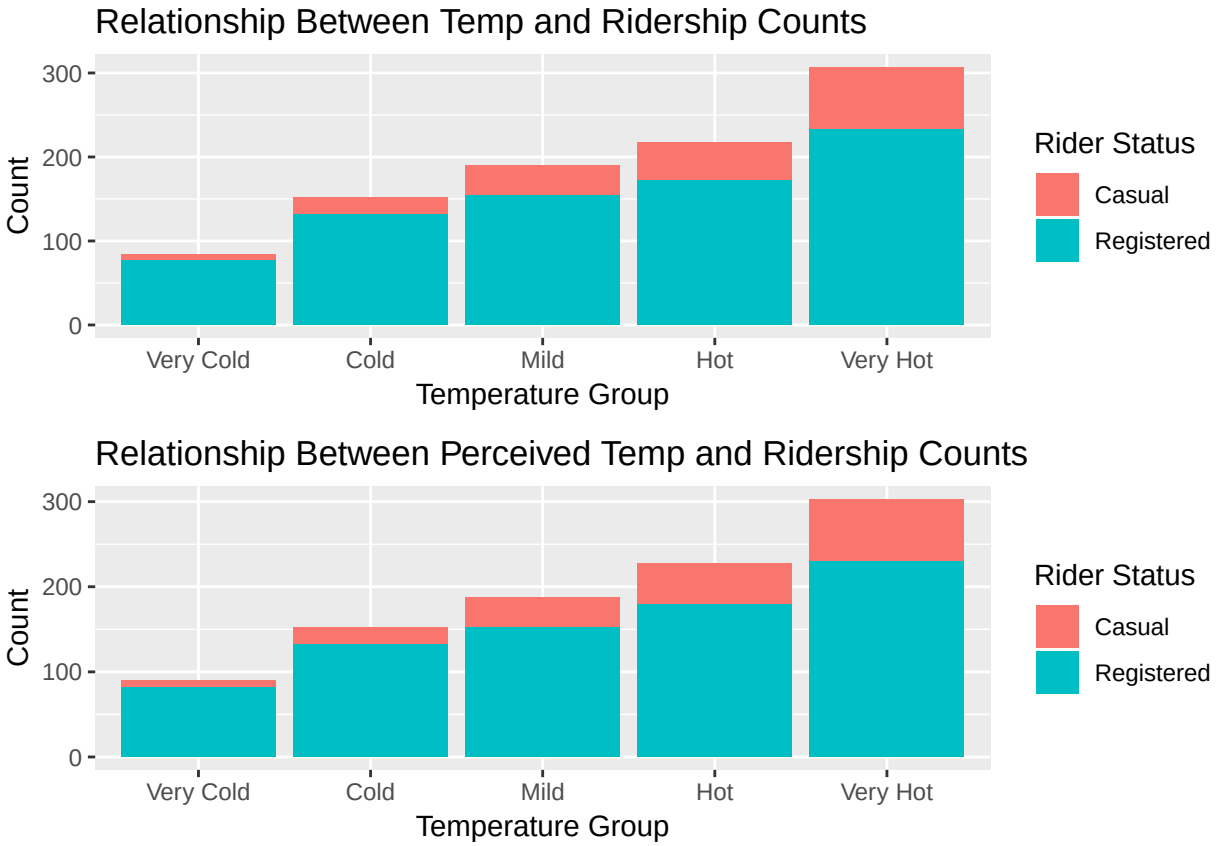


Registered riderships are higher than casual riderships at every year, month, and hour period. As one might guess however, the amount of casual riderships appears to increase during warm tourism months, hours in the middle of the day outside of primary commuting hours, and on the weekend suggesting that individuals riding for reasons other than commute are more likely to ride casual.

(c) Is there an association between the weather situation and ridership counts? For casual riders? For registered riders?

```
# Temperature
cabi$tempbin <- xtile(cabi$temp, 5)
cabi$tempbin <- ifelse(cabi$tempbin == 1, "Very Cold", ifelse(cabi$tempbin == 2, "Cold", ifelse(cabi$ter
cabi$tempbin <- factor(cabi$tempbin, levels = c("Very Cold","Cold","Mild", "Hot", "Very Hot"))

tempdf <- cabi %>%
  group_by(tempbin) %>%
  dplyr::summarize(Casual = mean(casual),
                   Registered = mean(registered))

tempdf <- melt(data.frame(tempdf), id.vars = 'tempbin')

c <- ggplot(tempdf, aes(x=factor(tempbin), y=value, fill=variable)) + geom_bar(stat='identity') + labs(

# Perceived Temperature
cabi$atempbin <- xtile(cabi$atemp, 5)
cabi$atempbin <- ifelse(cabi$atempbin == 1, "Very Cold", ifelse(cabi$atempbin == 2, "Cold", ifelse(cabi
cabi$atempbin <- factor(cabi$atempbin, levels = c("Very Cold","Cold","Mild", "Hot", "Very Hot"))
```

```r
atempdf <- cabi %>%
  group_by(atempbin) %>%
  dplyr::summarize(Casual = mean(casual),
                   Registered = mean(registered))

atempdf <- melt(data.frame(atempdf), id.vars = 'atempbin')

d <- ggplot(atempdf, aes(x=factor(atempbin), y=value, fill=variable)) + geom_bar(stat='identity') + labs

grid.arrange(c, d, nrow = 2, ncol=1)
```

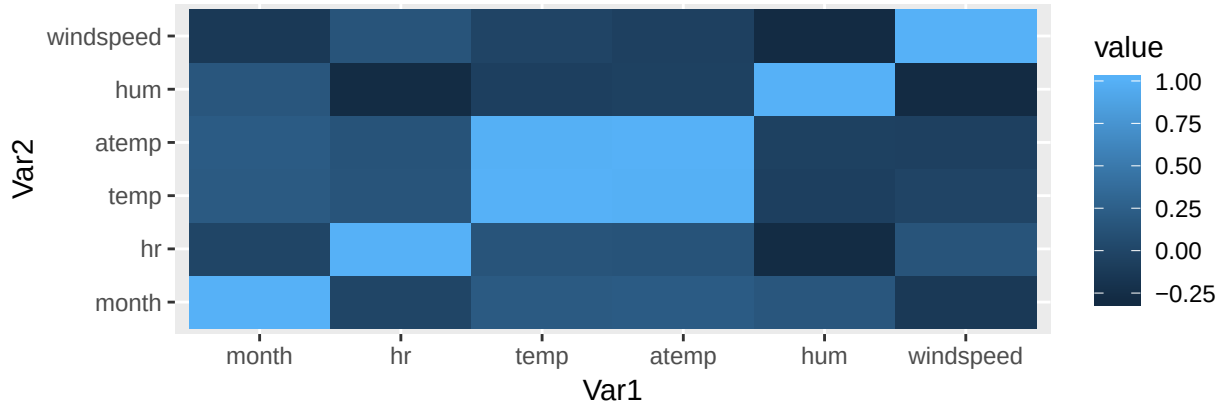


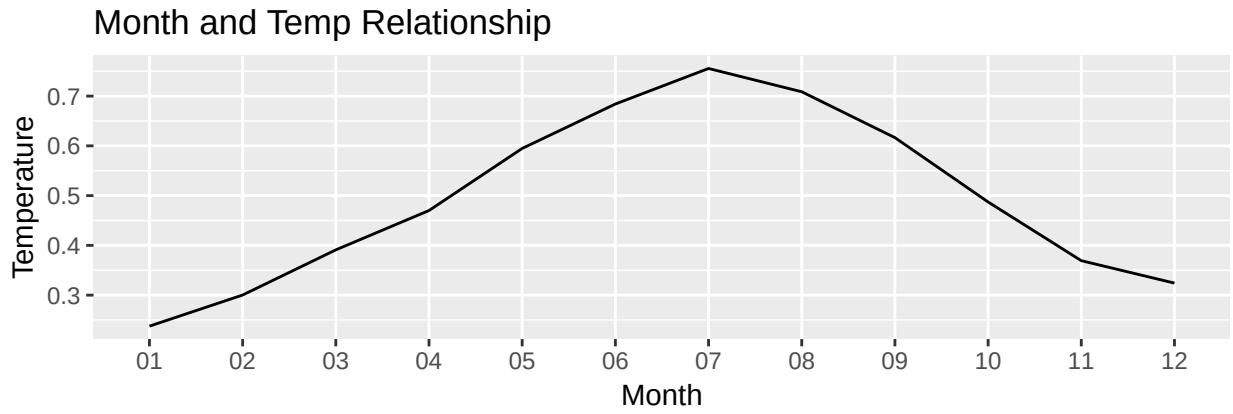
Unsurprisingly, there is clear positive relationship between the temperature and ridership.

(d) There are relations between time related predictors and weather related predictors. Demonstrate this with a few suitable graphs.

```r
# Corr Plot
timetemp <- data.frame(lapply(cabi[,c(4,6,7:10)], function(x) as.numeric(as.numeric(x))))
cortemp <- round(cor(timetemp),2)
melted_tempmonth <- melt(cortemp)
e <- ggplot(data = melted_tempmonth, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile()
# Lines
mttp <- cabi %>%
  group_by(month) %>%
  dplyr::summarize(temp = mean(temp))
```

```
# Line
f <- ggplot(data = mttp, aes(x = factor(month), y = temp, group=1)) +geom_line() + labs(title = "Month a

grid.arrange(e, f, nrow = 2, ncol=1)
```





For problems 2-4, split the data into a training set (70%) and a test set (30%).

```
# Split into train and test
cabi$yearmonth <- NULL
cabi$year <- as.factor(cabi$year)
cabi$season <- as.factor(cabi$season)
cabi$weather <- as.factor(cabi$weather)
cabi$tempbin <- NULL
cabi$atempbin <- NULL
set.seed(12345)
train <- sample(nrow(cabi),(nrow(cabi) * .70), replace = FALSE)

cabi_train <- cabi[train,]
cabi_test <- cabi[-train,]
```

Problem 2 (25) (a) Fit a multiple regression to predict registered ridership from the other variables (excluding casual ridership), using the training data. Identify the significant variables and comment on their coefficients.

```
cabireg <- lm(registered ~ . -X -casual, data = cabi_train)
summary(cabireg)
```

```
##
## Call:
```

```
## lm(formula = registered ~ . - X - casual, data = cabi_train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -270.0  -77.6  -25.2   46.6  615.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -36.312      7.707   -4.71  2.5e-06 ***
## season2       23.266      6.946    3.35  0.00081 ***
## season3       21.869      8.221    2.66  0.00782 **
## season4       67.235      6.912    9.73  < 2e-16 ***
## year2012      67.346      2.227   30.24  < 2e-16 ***
## month02        0.542      5.592    0.10  0.92277
## month03      -13.765      6.214   -2.22  0.02675 *
## month04      -20.070      9.253   -2.17  0.03010 *
## month05       -9.965      9.897   -1.01  0.31402
## month06      -22.682     10.041   -2.26  0.02391 *
## month07      -51.590     11.333   -4.55  5.4e-06 ***
## month08      -26.178     11.066   -2.37  0.01802 *
## month09        1.868      9.891    0.19  0.85024
## month10      -17.304      9.174   -1.89  0.05931 .
## month11      -26.487      8.820   -3.00  0.00268 **
## month12      -11.437      6.976   -1.64  0.10116
## wday          39.640      2.386   16.62  < 2e-16 ***
## hr             6.213      0.173   35.86  < 2e-16 ***
## temp         161.913     41.816    3.87  0.00011 ***
## atemp         86.255     43.800    1.97  0.04894 *
## hum         -126.740      7.284  -17.40  < 2e-16 ***
## windspeed     28.401     10.082    2.82  0.00486 **
## weather2       4.946      2.717    1.82  0.06874 .
## weather3     -31.114      4.516   -6.89  5.8e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 122 on 12141 degrees of freedom
## Multiple R-squared:  0.345,  Adjusted R-squared:  0.344
## F-statistic:  278 on 23 and 12141 DF,  p-value: <2e-16
```

A few findings are surprising. First, there is a highly statistically significant positive effect for temperature, but perceived temperature is only statistically significant at the 10% level. Windspeed also surprisngly shows a statistically significant positive effect. The coefficients on the season factor variable are all positive and statistically significant, indicating that ridership suffers most from January to March. While we could have left month in as a numeric variable, given that it is likely not a linear relationship, we left it in as a factor. Surprisingly, there is no statistically significant effect for any month compared to January.

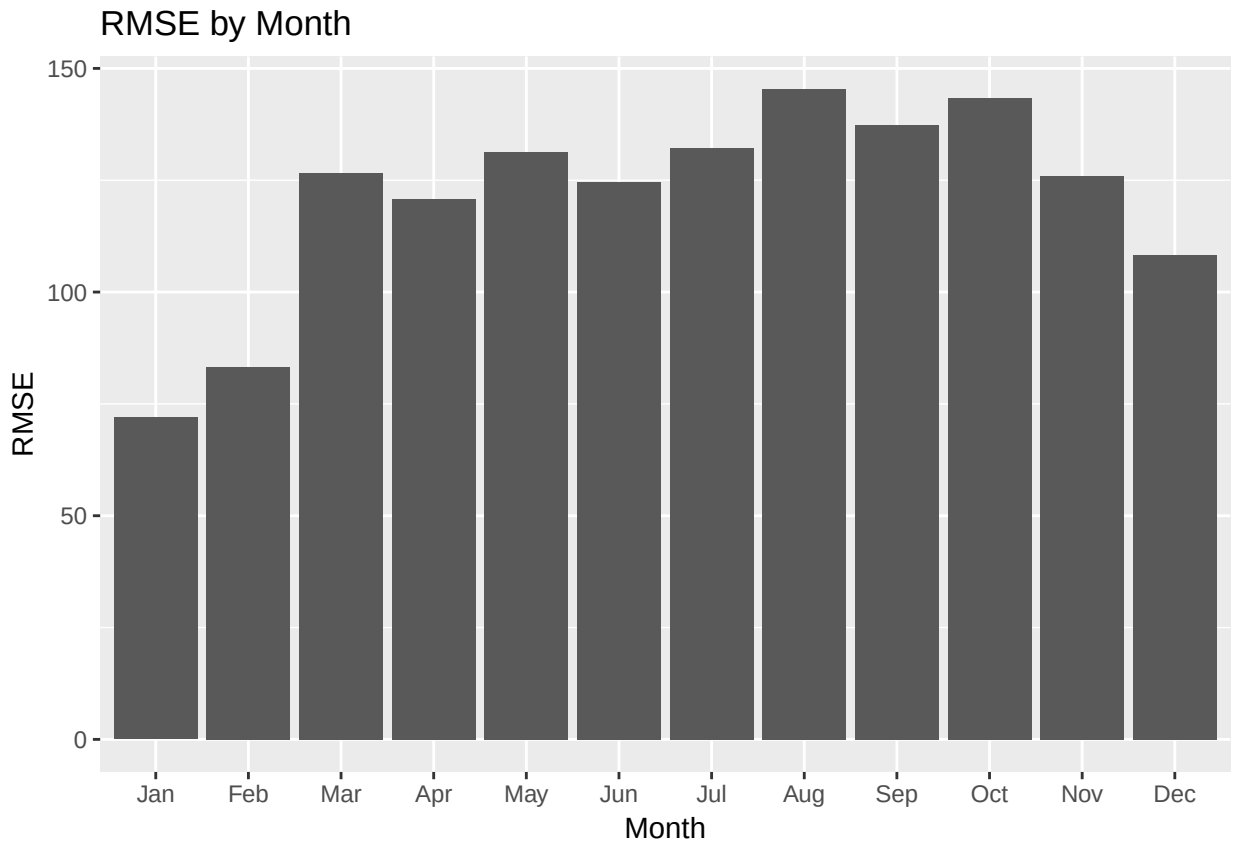
(b) Estimate the RMS prediction error of this model using the test set.

```
#Predict on Test
pred <- predict(cabireg, newdata = cabi_test)

# Calc RMSE
sqrt(mean((pred - cabi_test$registered)^2))
```

```
## [1] 124
```

(c) Does the RMS prediction error depend on the month? Answer this question using the test data and suitable tables or graphs.

```
subreg <- function(m) {
  # Run Model
  cabireg <- lm(registered ~ year + wday + hr + temp + atemp + hum + windspeed + weather, data = cabi_t
  #Predict on Test
  pred <- predict(cabireg, newdata = cabi_test[cabi_test$month == m,])
  # Calc RMSE
  a <- month.abb[as.numeric(m)]
  b <- sqrt(mean((pred - cabi_test[cabi_test$month == m,]$registered)^2))
  cbind(a,b)
}

mon <- unique(cabi$month)
data <- lapply(mon, subreg)
data1 <- do.call(rbind.data.frame, data)
data1$b <- as.numeric(as.character(data1$b))

ggplot(data=data1, aes(x=a, y=b)) + geom_bar(stat="identity") + labs(title = "RMSE by Month", x = "Mont
```



It looks like RMSE may be better during the summer months. GO BACK AND ASSESS WITH THE CORRLATION.

(d) Make copies of the training and test data in which hr is a categorical variable. Fit a multiple regression model. Compare the summary of this model to the one from part (a). Also estimate the RMS prediction error from the test set.

```
cabireghr <- lm(registered ~ year + wday + factor(hr) + temp + atemp + hum + windspeed + weather + month
summary(cabireghr)
```

```
##
## Call:
## lm(formula = registered ~ year + wday + factor(hr) + temp + atemp +
##     hum + windspeed + weather + month, data = cabi_train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -327.7  -48.9   -5.9   44.3  418.4
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -77.19       6.42  -12.02  < 2e-16 ***
## year2012        73.68       1.58   46.69  < 2e-16 ***
## wday            42.11       1.68   25.03  < 2e-16 ***
## factor(hr)1    -12.33       5.40   -2.28  0.02250 *
## factor(hr)2    -21.19       5.40   -3.92  8.8e-05 ***
## factor(hr)3    -30.87       5.43   -5.69  1.3e-08 ***
## factor(hr)4    -35.20       5.46   -6.45  1.2e-10 ***
## factor(hr)5    -19.75       5.44   -3.63  0.00028 ***
## factor(hr)6     34.71       5.38    6.45  1.1e-10 ***
## factor(hr)7    164.12       5.37   30.56  < 2e-16 ***
## factor(hr)8    292.14       5.37   54.41  < 2e-16 ***
## factor(hr)9    142.36       5.32   26.75  < 2e-16 ***
## factor(hr)10    76.34       5.44   14.04  < 2e-16 ***
## factor(hr)11    88.32       5.45   16.19  < 2e-16 ***
## factor(hr)12   122.61       5.54   22.11  < 2e-16 ***
## factor(hr)13   117.11       5.56   21.07  < 2e-16 ***
## factor(hr)14   100.79       5.58   18.08  < 2e-16 ***
## factor(hr)15   110.22       5.58   19.76  < 2e-16 ***
## factor(hr)16   171.16       5.57   30.75  < 2e-16 ***
## factor(hr)17   325.48       5.55   58.65  < 2e-16 ***
## factor(hr)18   299.45       5.46   54.87  < 2e-16 ***
## factor(hr)19   208.21       5.39   38.62  < 2e-16 ***
## factor(hr)20   137.41       5.39   25.48  < 2e-16 ***
## factor(hr)21    92.66       5.41   17.13  < 2e-16 ***
## factor(hr)22    59.01       5.38   10.97  < 2e-16 ***
## factor(hr)23    26.65       5.40    4.94  8.1e-07 ***
## temp            33.15      29.71    1.12  0.26446
## atemp          111.68      30.91    3.61  0.00030 ***
## hum            -53.36       5.58   -9.56  < 2e-16 ***
## windspeed      -19.15       7.13   -2.69  0.00722 **
## weather2        -5.80       1.94   -2.99  0.00282 **
## weather3       -53.97       3.25  -16.63  < 2e-16 ***
## month02          6.72       3.95    1.70  0.08912 .
## month03         13.38       4.07    3.29  0.00102 **
## month04         31.07       4.42    7.04  2.1e-12 ***
## month05         44.43       5.12    8.68  < 2e-16 ***
## month06         42.60       5.70    7.48  8.0e-14 ***
## month07         22.49       6.21    3.62  0.00029 ***
## month08         41.25       5.87    7.02  2.3e-12 ***
## month09         66.66       5.28   12.62  < 2e-16 ***
```

```
## month10         69.19       4.54   15.26  < 2e-16 ***
## month11         49.72       4.03   12.34  < 2e-16 ***
## month12         32.57       3.90    8.35  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 85.9 on 12122 degrees of freedom
## Multiple R-squared:  0.674,  Adjusted R-squared:  0.673
## F-statistic:  598 on 42 and 12122 DF,  p-value: <2e-16
```

```
#Predict on Test
pred <- predict(cabireghr, newdata = cabi_test)

# Calc RMSE
sqrt(mean((pred - cabi_test$registered)^2))
```

```
## [1] 87
```

When hour is included as a categorical variable, all months are statistically significant at some level but temperature becomes statistically insignificant.

Problem 3 (30) Use the original cabi data for this problem.

(a) Train artificial neural networks with various numbers of nodes in the hidden layer to predict registered ridership. Use the training data and only weather related variables. Recommend a suitable number of nodes, with explanation.

```
train_rmsr <- numeric()

set.seed(1)
for(i in 1:10) {
  nn_fit <- nnet(registered ~ temp + atemp + hum + windspeed + weather, data= cabi_train, maxit = 2000,
  train_rmsr[i] <- sqrt(mean((predict(nn_fit, type = "r") - cabi_train$registered)^2))
}
```

```
## # weights:  9
## initial  value 559679438.591218
## iter  10 value 236594108.658620
## iter  20 value 233959231.531219
## iter  30 value 233292935.553211
## iter  40 value 227883504.151855
## iter  50 value 226766259.420134
## final  value 226669247.271044
## converged
## # weights:  17
## initial  value 563601836.527366
## iter  10 value 274873918.707262
## final  value 274873877.622721
## converged
## # weights:  25
## initial  value 560094239.100099
## iter  10 value 263607511.364927
## iter  20 value 237199734.476425
## iter  30 value 232327717.862500
## iter  40 value 227225141.281221
## iter  50 value 224738882.220399
## iter  60 value 224710762.230007
```

```
## iter  70 value 224705118.829799
## iter  80 value 224557585.827711
## iter  90 value 224481442.938727
## iter 100 value 222780203.286256
## iter 110 value 222116102.067116
## iter 120 value 220461459.718498
## iter 130 value 219954477.224332
## iter 140 value 219862227.415370
## iter 150 value 219648500.856446
## iter 160 value 218834264.305926
## iter 170 value 218452817.326310
## iter 180 value 218406177.572562
## iter 190 value 218400971.694591
## iter 200 value 218396890.342975
## final  value 218394788.358424
## converged
## # weights:  33
## initial  value 558661893.022909
## iter  10 value 272343528.659037
## iter  20 value 255145717.196570
## iter  30 value 236728419.116635
## iter  40 value 232653667.247869
## iter  50 value 229012551.310406
## iter  60 value 225548337.373859
## iter  70 value 224664781.357112
## iter  80 value 224274490.618625
## iter  90 value 223893493.001550
## iter 100 value 222455472.355877
## iter 110 value 221503616.508663
## iter 120 value 221375876.461913
## iter 130 value 219023443.868334
## iter 140 value 218634014.101701
## iter 150 value 218539071.590517
## iter 160 value 218513677.134020
## iter 170 value 218470150.175976
## iter 180 value 218347386.617915
## iter 190 value 217228961.608378
## iter 200 value 216676527.276821
## iter 210 value 216572404.713528
## iter 220 value 216535833.274435
## iter 230 value 216535219.173235
## iter 240 value 216535030.731705
## iter 250 value 216533049.330721
## iter 260 value 216530410.635729
## iter 270 value 216530053.868439
## iter 280 value 216529019.536520
## iter 290 value 216523469.828397
## iter 300 value 216522105.631151
## iter 310 value 216520484.674068
## iter 320 value 216518765.293174
## iter 330 value 216516877.794731
## iter 340 value 216488129.545138
## final  value 216488092.084523
## converged
```

```
## # weights:  41
## initial  value 559481039.858424
## iter  10 value 234149719.339925
## iter  20 value 228488463.530849
## iter  30 value 224362385.602610
## iter  40 value 223018665.428103
## iter  50 value 221489501.831917
## iter  60 value 218887377.538493
## iter  70 value 218401139.967884
## iter  80 value 218370910.477334
## iter  90 value 218222891.847316
## iter 100 value 218162052.121412
## iter 110 value 218087952.110165
## iter 120 value 217983143.365077
## iter 130 value 217819190.272004
## iter 140 value 217781766.746645
## iter 150 value 217779112.918749
## iter 160 value 217771710.418937
## iter 170 value 217733917.889675
## iter 180 value 217587749.901859
## iter 190 value 217326893.332819
## iter 200 value 217186961.706141
## iter 210 value 217156370.046681
## iter 220 value 217141690.068974
## iter 230 value 217140253.637958
## iter 240 value 217113981.938700
## iter 250 value 217068567.147265
## iter 260 value 217017811.562782
## iter 270 value 216980058.073310
## iter 280 value 216965676.371292
## iter 290 value 216960909.190078
## iter 300 value 216957743.531818
## iter 310 value 216952883.690681
## iter 320 value 216937681.787581
## final  value 216937351.353675
## converged
## # weights:  49
## initial  value 560152098.287249
## iter  10 value 240220532.834982
## iter  20 value 231281000.128219
## iter  30 value 223332791.970564
## iter  40 value 220934932.454472
## iter  50 value 220720255.059314
## iter  60 value 218583199.071606
## iter  70 value 218046524.810998
## iter  80 value 217618673.904279
## iter  90 value 217489933.758479
## iter 100 value 217357041.363579
## iter 110 value 217275284.416437
## iter 120 value 217146191.992264
## iter 130 value 216744982.180209
## iter 140 value 216513757.176470
## iter 150 value 216438426.114365
## iter 160 value 216429227.796520
```

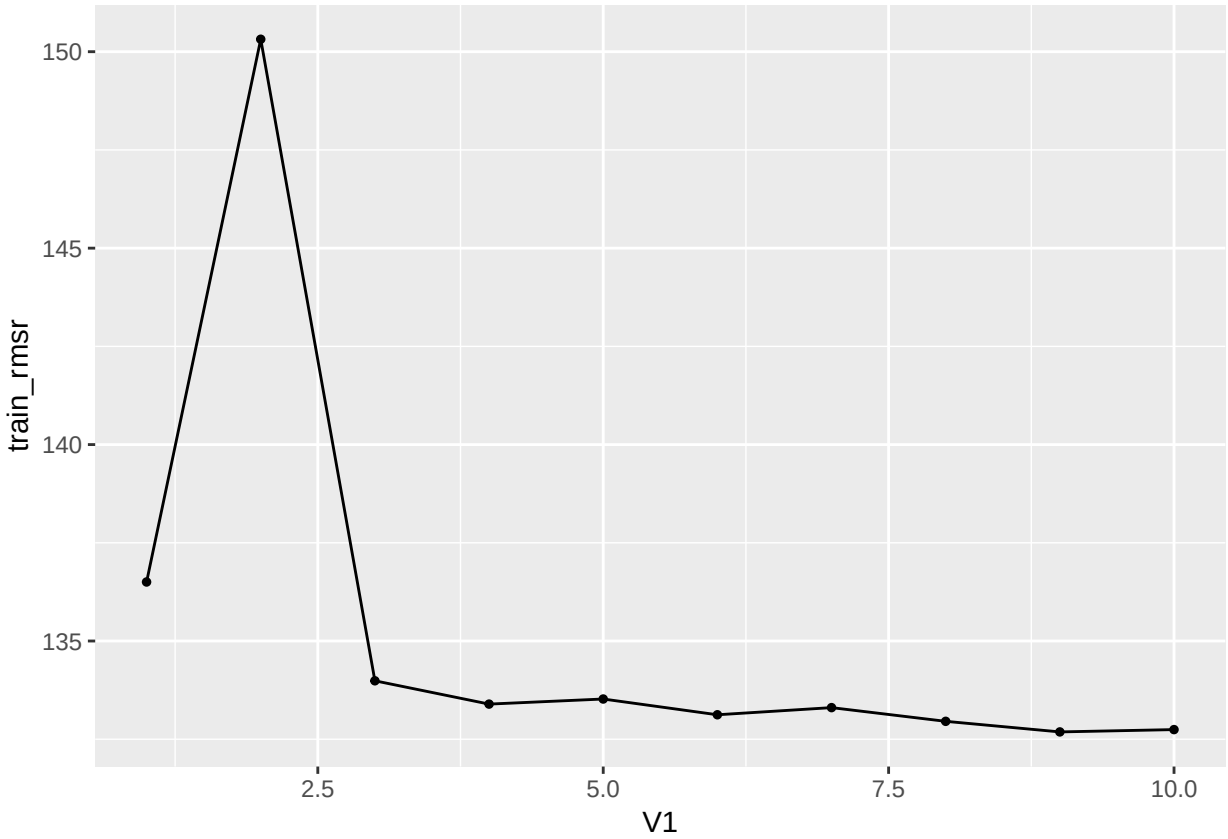
```
## iter 170 value 216418578.599046
## iter 180 value 216390650.683354
## iter 190 value 216357817.430497
## iter 200 value 216345850.544425
## iter 210 value 216331326.407121
## iter 220 value 216235009.985085
## iter 230 value 216046124.552268
## iter 240 value 215998504.569782
## iter 250 value 215972018.840852
## iter 260 value 215961660.724965
## iter 270 value 215946658.471564
## iter 280 value 215931098.208238
## iter 290 value 215923733.358308
## iter 300 value 215890979.953980
## iter 310 value 215729917.473060
## iter 320 value 215651248.729754
## iter 330 value 215618124.832223
## iter 340 value 215596858.141472
## iter 350 value 215590233.925735
## final  value 215590203.493999
## converged
## # weights:  57
## initial  value 560735625.299604
## iter  10 value 234981974.176518
## iter  20 value 231134831.606331
## iter  30 value 227543908.118941
## iter  40 value 226328921.911665
## iter  50 value 225055197.587587
## iter  60 value 223850767.058415
## iter  70 value 222953161.454796
## iter  80 value 221418965.871683
## iter  90 value 219966727.598920
## iter 100 value 218749999.209563
## iter 110 value 218367990.070501
## iter 120 value 217901074.094467
## iter 130 value 217793306.425612
## iter 140 value 217444156.253402
## iter 150 value 216757775.306021
## iter 160 value 216472235.095036
## iter 170 value 216393425.555556
## iter 180 value 216375572.942364
## iter 190 value 216371065.496214
## iter 200 value 216363253.421355
## iter 210 value 216354499.475632
## iter 220 value 216320052.754429
## iter 230 value 216225363.614929
## iter 240 value 216209043.892286
## iter 250 value 216204033.545844
## iter 250 value 216204031.917005
## iter 250 value 216204030.210138
## final  value 216204030.210138
## converged
## # weights:  65
## initial  value 563798463.306369
```

```
## iter  10 value 237329237.143807
## iter  20 value 231858290.905650
## iter  30 value 228918996.212840
## iter  40 value 224100461.659170
## iter  50 value 222951371.441482
## iter  60 value 222400302.378723
## iter  70 value 219131224.140482
## iter  80 value 218839123.964197
## iter  90 value 218741931.182608
## iter 100 value 218675684.034144
## iter 110 value 218659052.346032
## iter 120 value 218463294.453137
## iter 130 value 218344883.232311
## iter 140 value 218259544.668530
## iter 150 value 217830256.468919
## iter 160 value 216573536.161651
## iter 170 value 215795211.335119
## iter 180 value 215643883.680145
## iter 190 value 215618696.396465
## iter 200 value 215615752.006414
## iter 210 value 215609617.485065
## iter 220 value 215605238.715558
## iter 230 value 215549705.062175
## iter 240 value 215531290.734477
## iter 250 value 215402793.485000
## iter 260 value 215271457.153644
## iter 270 value 215179530.600807
## iter 280 value 215139641.356413
## iter 290 value 215133186.909113
## iter 300 value 215132179.034856
## iter 310 value 215131888.079104
## final  value 215131883.166687
## converged
## # weights:  73
## initial  value 561395052.965409
## iter  10 value 259273641.835805
## iter  20 value 230675146.008788
## iter  30 value 229115736.010459
## iter  40 value 223993733.577099
## iter  50 value 223525537.862132
## iter  60 value 223284057.371129
## iter  70 value 221547112.177259
## iter  80 value 220841743.170286
## iter  90 value 220016681.237165
## iter 100 value 218112848.608030
## iter 110 value 217766874.514770
## iter 120 value 217270913.108333
## iter 130 value 216970248.863245
## iter 140 value 216858438.236925
## iter 150 value 216740311.734267
## iter 160 value 216589152.368030
## iter 170 value 216306438.375331
## iter 180 value 216067188.937236
## iter 190 value 216012747.623135
```

```
## iter 200 value 215922342.246773
## iter 210 value 215864343.173877
## iter 220 value 215718311.675264
## iter 230 value 215575244.734522
## iter 240 value 215563060.000032
## iter 250 value 215548255.915185
## iter 260 value 215524288.078972
## iter 270 value 215493520.132790
## iter 280 value 215402576.071205
## iter 290 value 215152454.448100
## iter 300 value 215136490.761031
## iter 310 value 215126801.294091
## iter 320 value 215101886.030787
## iter 330 value 215079922.040527
## iter 340 value 215035429.796942
## iter 350 value 214847054.971130
## iter 360 value 214648570.946653
## iter 370 value 214608845.007911
## iter 380 value 214606438.486181
## iter 390 value 214602700.938967
## iter 400 value 214587328.500548
## iter 410 value 214559268.651012
## iter 420 value 214531924.365246
## iter 430 value 214515781.426996
## iter 440 value 214515406.536891
## iter 450 value 214514975.557983
## iter 460 value 214514127.154007
## iter 470 value 214512642.600990
## iter 480 value 214506017.831268
## iter 490 value 214475087.949967
## iter 500 value 214369806.959020
## iter 510 value 214315769.326573
## iter 520 value 214287793.779954
## iter 530 value 214256915.163662
## iter 540 value 214233933.047596
## iter 550 value 214231446.651017
## iter 560 value 214230257.781363
## iter 570 value 214229794.277862
## final  value 214229456.096156
## converged
## # weights:  81
## initial  value 562150571.812341
## iter  10 value 235188940.667604
## iter  20 value 230495944.773112
## iter  30 value 227268340.539611
## iter  40 value 224666104.601210
## iter  50 value 223098386.062223
## iter  60 value 222048402.085843
## iter  70 value 221691102.033255
## iter  80 value 221248178.821972
## iter  90 value 220961000.214849
## iter 100 value 220559239.940502
## iter 110 value 220460768.684782
## iter 120 value 219598889.527411
```

```
## iter 130 value 219040517.821947
## iter 140 value 218901170.902820
## iter 150 value 218781254.619805
## iter 160 value 218730296.262237
## iter 170 value 218472986.029772
## iter 180 value 218193103.990132
## iter 190 value 217844306.989297
## iter 200 value 217603515.712918
## iter 210 value 217575059.658279
## iter 220 value 217523551.797341
## iter 230 value 217437174.895158
## iter 240 value 217412905.720074
## iter 250 value 217380831.612885
## iter 260 value 217359979.957988
## iter 270 value 217336091.695345
## iter 280 value 217251721.467138
## iter 290 value 217068356.371661
## iter 300 value 216727681.700783
## iter 310 value 216566865.162249
## iter 320 value 216545772.609863
## iter 330 value 216541476.484232
## iter 340 value 216532608.441797
## iter 350 value 216519364.031042
## iter 360 value 216495413.071937
## iter 370 value 216425551.541948
## iter 380 value 216238293.735469
## iter 390 value 215736047.089480
## iter 400 value 215330487.750850
## iter 410 value 214957712.635261
## iter 420 value 214699025.084050
## iter 430 value 214587141.569233
## iter 440 value 214563106.037616
## iter 450 value 214550749.221745
## iter 460 value 214547951.115372
## iter 470 value 214542673.339362
## iter 480 value 214538789.872321
## iter 490 value 214535349.442714
## iter 500 value 214528304.654013
## iter 510 value 214520570.814584
## iter 520 value 214494695.351688
## iter 530 value 214466589.141054
## iter 540 value 214459015.417840
## iter 550 value 214456318.705438
## iter 560 value 214455133.166450
## final  value 214455113.554968
## converged
```

```
train_rmsr <- data.frame(cbind(1:10, train_rmsr))
ggplot(train_rmsr, aes(x = V1, y = train_rmsr, group = 1)) + geom_point(size = 1, alpha = 1, na.rm = TRU
```



While the RMSE continues to decline slightly through 10 nodes, there is only a tiny decrease in the RMSE between the model with 5 nodes and the model with 10 nodes, so we will choose the model with 5 nodes in order to get a model that both fits well and has a lower chance of overfitting.

(b) Repeat part (a), using only time related variables.

```
train_rmsr <- numeric()

set.seed(1)
for(i in 1:10) {
  nn_fit <- nnet(registered ~ year + season + wday + hr + month, data= cabi_train, maxit = 2000, decay =
  train_rmsr[i] <- sqrt(mean((predict(nn_fit, type = "r") - cabi_train$registered)^2))
}
```

```
## # weights:  20
## initial  value 560058924.171528
## final  value 274873730.399124
## converged
## # weights:  39
## initial  value 561190542.353287
## iter  10 value 270348996.913093
## final  value 268378792.937640
## converged
## # weights:  58
## initial  value 561303842.457560
## iter  10 value 261894700.844002
## iter  20 value 254595521.893932
## iter  30 value 237248225.129778
```

```
## iter  40 value 236581006.532755
## iter  50 value 235808241.535061
## iter  60 value 235044818.734402
## iter  70 value 234995342.346908
## iter  80 value 234972865.859668
## iter  90 value 234385506.242493
## iter 100 value 234240337.499215
## iter 110 value 223130260.341262
## iter 120 value 221510019.243014
## iter 130 value 218724015.565764
## iter 140 value 216662749.678966
## iter 150 value 212969711.280771
## iter 160 value 209583817.371654
## iter 170 value 192455389.375374
## iter 180 value 190828910.454085
## iter 190 value 189215296.158425
## iter 200 value 187980325.311748
## iter 210 value 185877683.432912
## iter 220 value 183676527.060108
## iter 230 value 180022060.943253
## iter 240 value 179620199.593663
## iter 250 value 177716062.696895
## iter 260 value 176871221.586714
## iter 270 value 176231973.079153
## iter 280 value 175292776.004136
## iter 290 value 175118111.051205
## iter 300 value 174454735.586740
## iter 310 value 174278164.998462
## iter 320 value 173665093.797607
## iter 330 value 172535384.996753
## iter 340 value 171832846.620246
## iter 350 value 171641582.630911
## iter 360 value 171536575.688987
## iter 370 value 171459403.373203
## iter 380 value 171292324.837175
## iter 390 value 171225231.954131
## iter 400 value 170919174.924165
## iter 410 value 170899135.943152
## iter 420 value 170770429.443466
## iter 430 value 170761363.021922
## iter 440 value 170748127.128551
## iter 450 value 170730085.213349
## iter 460 value 170724932.851558
## iter 470 value 170707321.206465
## iter 480 value 170650549.349669
## iter 490 value 170491222.307978
## iter 500 value 170474138.388964
## iter 510 value 170469080.949004
## iter 520 value 170433358.592336
## iter 530 value 170392374.159217
## iter 540 value 170391499.076862
## iter 550 value 170388302.148945
## iter 560 value 170333586.354755
## iter 570 value 170316381.591969
```

```
## iter 580 value 170315217.160363
## iter 590 value 170314854.348472
## iter 600 value 170308008.305590
## iter 610 value 170024723.511537
## iter 620 value 169924768.943905
## iter 630 value 169906409.111169
## iter 640 value 169850051.944758
## iter 650 value 169184803.969097
## iter 660 value 168814217.751936
## iter 670 value 168588612.560335
## iter 680 value 168037375.238398
## iter 690 value 168014081.025035
## iter 700 value 167891449.604536
## iter 710 value 167873032.827954
## iter 720 value 167867678.576390
## iter 730 value 167821863.129238
## iter 740 value 167812898.682826
## iter 750 value 167513987.642883
## iter 760 value 167228613.438428
## iter 770 value 167190495.621827
## iter 780 value 166087274.451821
## iter 790 value 165597017.740429
## iter 800 value 165511434.195554
## iter 810 value 163495987.750631
## iter 820 value 162629028.981513
## iter 830 value 162479549.791004
## iter 840 value 162256693.365520
## iter 850 value 162198062.983030
## iter 860 value 162156032.758486
## iter 870 value 162013951.995811
## iter 880 value 161922847.961865
## iter 890 value 161875791.946447
## iter 900 value 161860348.715049
## iter 910 value 161779933.729634
## iter 920 value 161755561.359383
## iter 930 value 161691826.061054
## iter 940 value 161682800.058242
## iter 950 value 161677490.143917
## iter 960 value 161633608.812508
## iter 970 value 161491198.574260
## iter 980 value 161437294.577872
## iter 990 value 161429451.974901
## iter1000 value 161422457.373082
## iter1010 value 160994412.201509
## iter1020 value 160792374.573817
## iter1030 value 160754831.820253
## iter1040 value 160681295.946087
## iter1050 value 160596439.161088
## iter1060 value 160543655.012620
## iter1070 value 160489825.388453
## iter1080 value 160481570.037485
## iter1090 value 160479144.894949
## iter1100 value 160449833.651236
## iter1110 value 160264279.272601
```

```
## iter1120 value 160161109.848266
## iter1130 value 160147682.582397
## iter1140 value 160144892.294077
## iter1150 value 160141801.861196
## iter1160 value 160140724.762823
## iter1170 value 160127315.260737
## final  value 160122666.024021
## converged
## # weights:  77
## initial  value 558960548.939488
## iter  10 value 222410468.233386
## iter  20 value 215895530.233566
## iter  30 value 213639277.492544
## iter  40 value 205996913.778181
## iter  50 value 204198058.342692
## iter  60 value 203042411.545947
## iter  70 value 201371419.153599
## iter  80 value 201096878.347305
## iter  90 value 200459112.575405
## iter 100 value 199751282.482397
## iter 110 value 199524881.160582
## iter 120 value 199167131.005569
## iter 130 value 199062995.367697
## iter 140 value 198509521.746745
## iter 150 value 198187889.277025
## iter 160 value 198175638.070772
## iter 170 value 198001361.396796
## iter 180 value 197891815.369033
## iter 190 value 197874339.229191
## iter 200 value 197774088.094917
## iter 210 value 197753357.665474
## iter 220 value 197720370.594047
## iter 230 value 197678591.104643
## iter 240 value 197677488.387154
## iter 250 value 197646972.779112
## iter 260 value 197639006.061167
## iter 270 value 195739581.428346
## iter 280 value 191213046.099890
## iter 290 value 187785404.855643
## iter 300 value 187525276.892276
## iter 310 value 187239771.347390
## iter 320 value 187164200.876282
## iter 330 value 187066446.689828
## iter 340 value 187062962.934628
## iter 350 value 187042915.296294
## iter 360 value 187035000.789858
## iter 370 value 186954198.533222
## iter 380 value 186915544.227009
## iter 390 value 186806047.625790
## iter 400 value 186782352.274274
## iter 410 value 186706864.466812
## iter 420 value 186679787.330785
## iter 430 value 186645297.055743
## iter 440 value 186562115.556660
```

```
## iter 450 value 186535399.045707
## iter 460 value 186370958.503200
## iter 470 value 186294506.013997
## iter 480 value 186006449.707683
## iter 490 value 185931672.391194
## iter 500 value 185913903.376266
## iter 510 value 185790752.253725
## iter 520 value 185684910.907506
## iter 530 value 185585570.259201
## iter 540 value 185571210.572841
## iter 550 value 185467119.385974
## iter 560 value 185156535.915195
## iter 570 value 185083165.752378
## iter 580 value 185057185.414974
## iter 590 value 185044059.248628
## iter 600 value 185010013.666018
## iter 610 value 184922066.611189
## iter 620 value 184875914.193327
## iter 630 value 184799012.570927
## iter 640 value 184769885.999104
## iter 650 value 184766928.281487
## iter 660 value 184764801.658754
## iter 670 value 184755929.703924
## iter 680 value 184746653.077899
## iter 690 value 184665347.434679
## iter 700 value 184640606.029679
## iter 710 value 184535958.506161
## iter 720 value 184379124.168169
## iter 730 value 184342548.439902
## iter 740 value 184340511.925701
## iter 750 value 184297183.160797
## iter 760 value 184273130.878900
## iter 770 value 184271319.806519
## iter 780 value 184264723.661018
## iter 790 value 184261226.402115
## iter 800 value 184217178.115303
## iter 810 value 184211603.979809
## iter 820 value 184210054.186049
## iter 830 value 184207562.934675
## iter 840 value 184204554.813659
## iter 850 value 184176347.385494
## iter 860 value 184120306.915864
## iter 870 value 184068464.617171
## iter 880 value 183919869.787223
## iter 890 value 183867106.156883
## iter 900 value 183843559.493124
## iter 910 value 183669589.064939
## iter 920 value 183652962.566693
## iter 930 value 183651236.825579
## iter 940 value 183616063.190951
## iter 950 value 183609922.489473
## iter 960 value 183598633.662672
## iter 970 value 183592412.099974
## iter 980 value 183592078.133368
```

```
## iter 990 value 183589503.280597
## iter1000 value 183575392.028189
## iter1010 value 183526041.651978
## iter1020 value 183512741.264570
## iter1030 value 183506893.061396
## iter1040 value 183488337.733263
## iter1050 value 183453485.745883
## iter1060 value 183205168.351768
## iter1070 value 183163748.098284
## iter1080 value 183150965.384019
## iter1090 value 183129716.130274
## iter1100 value 183121539.963535
## iter1110 value 183119259.802182
## iter1120 value 183105825.741678
## iter1130 value 183071923.917048
## iter1140 value 183042691.223726
## iter1150 value 183023375.474106
## iter1160 value 183020821.388938
## iter1170 value 182751837.079002
## iter1180 value 182679643.205313
## iter1190 value 182668865.253864
## iter1200 value 182667439.551292
## iter1210 value 182655488.285155
## iter1220 value 182600127.948724
## iter1230 value 182594441.688346
## iter1240 value 182592733.205599
## iter1250 value 182586166.868016
## iter1260 value 182578942.131545
## iter1270 value 182577256.117578
## iter1280 value 182576690.617096
## iter1290 value 182573400.593249
## iter1300 value 182528260.396770
## iter1310 value 182521153.952185
## iter1320 value 182514773.524184
## iter1330 value 182504656.973381
## iter1340 value 182446318.503944
## iter1350 value 182400269.704376
## iter1360 value 182392882.613694
## iter1370 value 182392636.146126
## iter1380 value 182392520.713982
## iter1380 value 182392519.522037
## iter1380 value 182392518.363670
## final  value 182392518.363670
## converged
## # weights:  96
## initial  value 567101083.943242
## iter  10 value 236143835.544485
## iter  20 value 221518976.030647
## iter  30 value 220367364.099157
## iter  40 value 214969700.189857
## iter  50 value 203144983.139372
## iter  60 value 199608944.078500
## iter  70 value 198898713.217253
## iter  80 value 198758344.805323
```

```
## iter  90 value 198726731.576111
## iter 100 value 197366316.569819
## iter 110 value 195984713.469456
## iter 120 value 192225494.992498
## iter 130 value 190691708.948890
## iter 140 value 190470315.379626
## iter 150 value 189424867.090911
## iter 160 value 189289372.183423
## iter 170 value 188938172.164800
## iter 180 value 188903171.302650
## iter 190 value 188823072.017051
## iter 200 value 188736956.636386
## iter 210 value 188665192.089342
## iter 220 value 188648815.054225
## iter 230 value 188631331.892784
## iter 240 value 188630977.468710
## final  value 188630829.187186
## converged
## # weights:  115
## initial  value 556738289.045530
## iter  10 value 195723839.888442
## iter  20 value 192510045.028395
## iter  30 value 190655904.277068
## iter  40 value 189369384.511245
## iter  50 value 188320300.030829
## iter  60 value 188082650.922057
## iter  70 value 188015204.996372
## iter  80 value 187064635.780251
## iter  90 value 186805162.485626
## iter 100 value 180462790.640880
## iter 110 value 177183306.202329
## iter 120 value 170517969.856044
## iter 130 value 164382021.346670
## iter 140 value 161350807.147943
## iter 150 value 160552879.556896
## iter 160 value 160233781.318957
## iter 170 value 159715257.249855
## iter 180 value 159018560.466580
## iter 190 value 156999746.778001
## iter 200 value 155846034.308629
## iter 210 value 154738179.966976
## iter 220 value 153129789.949611
## iter 230 value 152786925.809130
## iter 240 value 152718286.306698
## iter 250 value 152584360.877991
## iter 260 value 152203357.368821
## iter 270 value 150583663.923606
## iter 280 value 150186345.975783
## iter 290 value 149557943.707905
## iter 300 value 148941742.758460
## iter 310 value 148623641.805337
## iter 320 value 148234842.102359
## iter 330 value 147912592.301056
## iter 340 value 147777273.716834
```

```
## iter 350 value 147538731.883026
## iter 360 value 147396635.564519
## iter 370 value 146798793.414890
## iter 380 value 146346814.417135
## iter 390 value 145984234.637742
## iter 400 value 145835540.232984
## iter 410 value 145732888.683419
## iter 420 value 145676697.804163
## iter 430 value 145594902.650630
## iter 440 value 145401142.584197
## iter 450 value 144936213.868508
## iter 460 value 144619211.695719
## iter 470 value 144476634.318643
## iter 480 value 144417390.913872
## iter 490 value 144364658.439744
## iter 500 value 144323526.623804
## iter 510 value 144209102.402176
## iter 520 value 143758131.091924
## iter 530 value 142266899.779976
## iter 540 value 140763178.275896
## iter 550 value 140499435.546722
## iter 560 value 140419575.450276
## iter 570 value 140392307.502001
## iter 580 value 140378121.654941
## iter 590 value 140328559.136408
## iter 600 value 140215389.679625
## iter 610 value 140176780.196299
## iter 620 value 140078170.092040
## iter 630 value 140046053.511744
## iter 640 value 140027180.946832
## iter 650 value 140018459.933578
## iter 660 value 140009640.486964
## iter 670 value 139985806.377807
## iter 680 value 139953410.593384
## iter 690 value 139932165.252090
## iter 700 value 139921638.228966
## iter 710 value 139918192.022484
## iter 720 value 139748221.565998
## iter 730 value 139689099.829172
## iter 740 value 139541577.029923
## iter 750 value 139485554.895805
## iter 760 value 139473194.322608
## iter 770 value 139468133.073453
## iter 780 value 139423313.104761
## iter 790 value 139187259.204634
## iter 800 value 138923741.349814
## iter 810 value 138849917.480894
## iter 820 value 138826484.596593
## iter 830 value 138699791.592788
## iter 840 value 138681498.462736
## iter 850 value 138670175.161577
## iter 860 value 138665078.109022
## iter 870 value 138653722.884998
## iter 880 value 138561472.272631
```

```
## iter 890 value 138440799.283264
## iter 900 value 138414089.852272
## iter 910 value 138394729.034093
## iter 920 value 138387787.949459
## iter 930 value 138376907.356160
## iter 940 value 138343884.154494
## iter 950 value 138333855.150479
## iter 960 value 138324078.830701
## iter 970 value 138318169.542295
## iter 980 value 138313182.818859
## iter 990 value 138307883.553367
## iter1000 value 138297264.749656
## iter1010 value 138290339.260488
## iter1020 value 138287983.129620
## iter1030 value 138287006.733374
## iter1040 value 138286685.321490
## iter1050 value 138278769.612359
## iter1060 value 138262446.680061
## iter1070 value 138255709.005858
## iter1080 value 138105498.646050
## iter1090 value 138015432.176497
## iter1100 value 137995082.300177
## iter1110 value 137986001.413457
## iter1120 value 137978036.748142
## iter1130 value 137969932.375719
## iter1140 value 137964979.232304
## iter1150 value 137939626.513377
## iter1160 value 137930280.198958
## iter1170 value 137921917.298138
## iter1180 value 137919773.156757
## iter1190 value 137876581.040874
## iter1200 value 137867119.989201
## iter1210 value 137857686.437381
## iter1220 value 137852314.332314
## iter1230 value 137849440.970177
## iter1240 value 137847090.669071
## iter1250 value 137829228.628055
## iter1260 value 137816514.212960
## iter1270 value 137786791.353434
## iter1280 value 137718011.095053
## iter1290 value 137676293.216113
## iter1300 value 137591630.751045
## iter1310 value 137547909.392093
## iter1320 value 137493907.487794
## iter1330 value 137411509.158764
## iter1340 value 137258007.175587
## iter1350 value 137086652.738375
## iter1360 value 136953270.968075
## iter1370 value 136872974.070960
## iter1380 value 136831343.232500
## iter1390 value 136779324.685941
## iter1400 value 136745998.775888
## iter1410 value 136725453.499595
## iter1420 value 136697382.124802
```

```
## iter1430 value 136674242.148029
## iter1440 value 136611291.967526
## iter1450 value 136545636.706770
## iter1460 value 136472997.815347
## iter1470 value 136453584.385181
## iter1480 value 136399722.053435
## iter1490 value 136396702.133769
## iter1500 value 136392366.134055
## iter1510 value 136389963.997391
## iter1520 value 136385701.354788
## iter1530 value 136383528.491212
## iter1540 value 136382005.227777
## iter1550 value 136379277.920071
## iter1560 value 136376519.915753
## iter1570 value 136370663.258733
## iter1580 value 136368545.406257
## iter1590 value 136367456.255740
## iter1600 value 136366220.686038
## iter1610 value 136365756.025179
## iter1620 value 136365204.493813
## iter1630 value 136362446.318023
## iter1640 value 136360407.901249
## iter1650 value 136357470.241682
## iter1660 value 136355909.217472
## iter1670 value 136355199.637623
## iter1680 value 136354823.236116
## iter1690 value 136350133.094295
## iter1700 value 136345782.891982
## iter1710 value 136344327.308867
## iter1720 value 136343559.222801
## iter1730 value 136343300.706133
## iter1740 value 136343037.833456
## iter1750 value 136342689.174088
## iter1760 value 136341524.451428
## iter1770 value 136340494.001248
## iter1780 value 136339064.373734
## iter1790 value 136338209.945537
## iter1800 value 136337568.687446
## iter1810 value 136337416.507618
## iter1820 value 136337228.595023
## iter1830 value 136336797.927356
## iter1840 value 136335965.878500
## iter1850 value 136335077.403601
## iter1860 value 136334120.785350
## iter1870 value 136330761.774735
## iter1880 value 136327559.740070
## iter1890 value 136326411.141096
## iter1900 value 136324451.694705
## iter1910 value 136323171.089210
## iter1920 value 136321544.969513
## iter1930 value 136320892.871732
## iter1940 value 136320293.964166
## iter1950 value 136319779.796672
## iter1960 value 136319631.705492
```

```
## iter1970 value 136319445.799765
## iter1980 value 136318982.816146
## iter1990 value 136318425.171601
## iter2000 value 136318202.256116
## final  value 136318202.256116
## stopped after 2000 iterations
## # weights:  134
## initial  value 572058086.614738
## iter  10 value 349676879.581329
## iter  20 value 303904043.624716
## iter  30 value 290343132.406788
## iter  40 value 274893982.418625
## iter  50 value 274443195.037907
## iter  60 value 226730329.808414
## iter  70 value 182272035.642666
## iter  80 value 176921550.686865
## iter  90 value 174308840.186028
## iter 100 value 173942208.858714
## iter 110 value 166083731.519616
## iter 120 value 158384924.045941
## iter 130 value 155335401.323799
## iter 140 value 152987080.332871
## iter 150 value 150622596.747724
## iter 160 value 145226439.999040
## iter 170 value 143844914.022613
## iter 180 value 142543948.693289
## iter 190 value 142072472.740011
## iter 200 value 141265697.269381
## iter 210 value 141162508.597059
## iter 220 value 141087128.823730
## iter 230 value 141052656.721694
## iter 240 value 140260075.964682
## iter 250 value 139757536.268964
## iter 260 value 139202344.107935
## iter 270 value 137963173.269714
## iter 280 value 137911647.168004
## iter 290 value 137853418.355725
## iter 300 value 137742931.139035
## iter 310 value 137658378.004364
## iter 320 value 137653030.604772
## iter 320 value 137653030.122030
## iter 330 value 137612287.483938
## iter 340 value 137607276.248266
## iter 350 value 137219037.483408
## iter 360 value 137110317.985175
## iter 370 value 137084756.208674
## iter 380 value 136844284.990543
## iter 390 value 136723935.843285
## iter 400 value 136710113.164015
## iter 410 value 136684047.213515
## iter 420 value 136607586.497734
## iter 430 value 136592853.731835
## iter 440 value 136427392.884257
## iter 450 value 135972220.505890
```

```
## iter 460 value 135955567.505128
## iter 470 value 135928076.426100
## iter 480 value 135909240.487717
## iter 490 value 135824606.338888
## iter 500 value 135541190.263021
## iter 510 value 135479043.671202
## iter 520 value 135452266.445577
## iter 530 value 135449321.895842
## iter 540 value 135400814.855583
## iter 550 value 135351615.663059
## iter 560 value 135348290.392557
## iter 570 value 135347236.076057
## iter 580 value 135345769.484962
## iter 590 value 135316903.133402
## iter 600 value 135313975.373576
## iter 610 value 135313345.823718
## iter 620 value 135312394.614225
## iter 630 value 135310676.661361
## iter 640 value 135290097.552841
## iter 650 value 135218828.574793
## iter 660 value 135209092.842033
## iter 670 value 135184590.467851
## iter 680 value 135165718.462220
## iter 690 value 135122384.645348
## iter 700 value 135121535.245754
## iter 710 value 135121432.113933
## iter 720 value 135120296.823868
## iter 730 value 135114230.060456
## iter 740 value 135113728.510375
## iter 750 value 135113481.680191
## iter 760 value 135113421.350440
## iter 760 value 135113421.260375
## iter 760 value 135113420.713006
## final  value 135113420.713006
## converged
## # weights:  153
## initial  value 561901489.611868
## iter  10 value 225346792.303492
## iter  20 value 211600669.417411
## iter  30 value 208707413.047529
## iter  40 value 199162736.592236
## iter  50 value 193521603.867138
## iter  60 value 190734019.921808
## iter  70 value 190040354.654863
## iter  80 value 189706205.892925
## iter  90 value 187619637.295095
## iter 100 value 186812216.083075
## iter 110 value 185065337.596235
## iter 120 value 184402489.737525
## iter 130 value 183934576.906161
## iter 140 value 183763356.512017
## iter 150 value 183106414.897205
## iter 160 value 182200213.878898
## iter 170 value 182141623.076570
```

```
## iter 180 value 182053900.081567
## iter 190 value 181884976.388022
## iter 200 value 181538016.016445
## iter 210 value 180970995.313393
## iter 220 value 180740582.922897
## iter 230 value 180722078.537241
## iter 240 value 180696761.171204
## iter 250 value 180651518.563966
## iter 260 value 180635249.902208
## iter 270 value 180598333.450884
## iter 280 value 180515112.571170
## iter 290 value 180447030.196218
## iter 300 value 180429008.250430
## iter 310 value 180191212.617026
## iter 320 value 180146632.070734
## iter 330 value 179893544.899326
## iter 340 value 179709780.800929
## iter 350 value 179532977.425161
## iter 360 value 179197731.441266
## iter 370 value 178968931.657028
## iter 380 value 178907565.801720
## iter 390 value 178894889.221665
## iter 400 value 178738112.533782
## iter 410 value 178530007.688356
## iter 420 value 178504033.850053
## iter 430 value 178493440.690738
## iter 440 value 178483527.405710
## iter 450 value 178391162.589330
## iter 460 value 177303412.484536
## iter 470 value 176224438.602065
## iter 480 value 175512338.455399
## iter 490 value 175350670.174907
## iter 500 value 175231303.923632
## iter 510 value 175134463.229803
## iter 520 value 175068446.750045
## iter 530 value 175003280.199310
## iter 540 value 174960094.554962
## iter 550 value 174907398.834679
## iter 560 value 174864911.635320
## iter 570 value 174673034.146135
## iter 580 value 174429101.643332
## iter 590 value 174335100.370676
## iter 600 value 174213069.391495
## iter 610 value 174201667.609178
## iter 620 value 174196465.947129
## iter 630 value 174191659.274088
## iter 640 value 174172185.898372
## iter 650 value 174154364.157294
## iter 660 value 174149278.414660
## iter 670 value 174146499.888467
## iter 680 value 174144084.495435
## iter 690 value 174114019.025630
## iter 700 value 174039244.832011
## iter 710 value 174029939.885015
```

```
## iter 720 value 174028586.645350
## iter 730 value 174026525.331913
## iter 740 value 174020359.397321
## iter 750 value 174013488.878597
## iter 760 value 174007235.968985
## iter 770 value 173988506.134492
## iter 780 value 173975956.530133
## iter 790 value 173972001.159256
## iter 800 value 173970077.481605
## iter 810 value 173959290.544193
## iter 820 value 173957125.724782
## iter 830 value 173956605.215802
## iter 840 value 173956316.765180
## iter 850 value 173944702.772019
## iter 860 value 173942432.765929
## iter 870 value 173941573.904195
## iter 880 value 173940393.764482
## iter 890 value 173940205.728345
## iter 900 value 173934405.822677
## iter 910 value 173923259.131936
## iter 920 value 173921616.416767
## iter 930 value 173920927.938055
## iter 940 value 173918812.138604
## iter 950 value 173918397.734472
## iter 960 value 173917484.085136
## iter 970 value 173917298.330598
## iter 980 value 173915633.211026
## iter 990 value 173912450.411220
## iter1000 value 173902874.964185
## iter1010 value 173897187.087749
## iter1020 value 173888427.871349
## iter1030 value 173885775.529797
## iter1040 value 173883970.390989
## iter1050 value 173883573.432412
## iter1060 value 173882972.914481
## iter1070 value 173881873.734282
## iter1080 value 173876409.440360
## iter1090 value 173875780.015747
## iter1100 value 173875498.614937
## iter1110 value 173875422.549090
## iter1120 value 173875276.970741
## final  value 173875246.494125
## converged
## # weights:  172
## initial  value 565319400.330178
## iter  10 value 232772340.228518
## iter  20 value 217369218.735848
## iter  30 value 190722354.820759
## iter  40 value 180401360.317802
## iter  50 value 176116429.251630
## iter  60 value 169256199.487083
## iter  70 value 168716221.439271
## iter  80 value 168253540.535410
## iter  90 value 167789458.258252
```

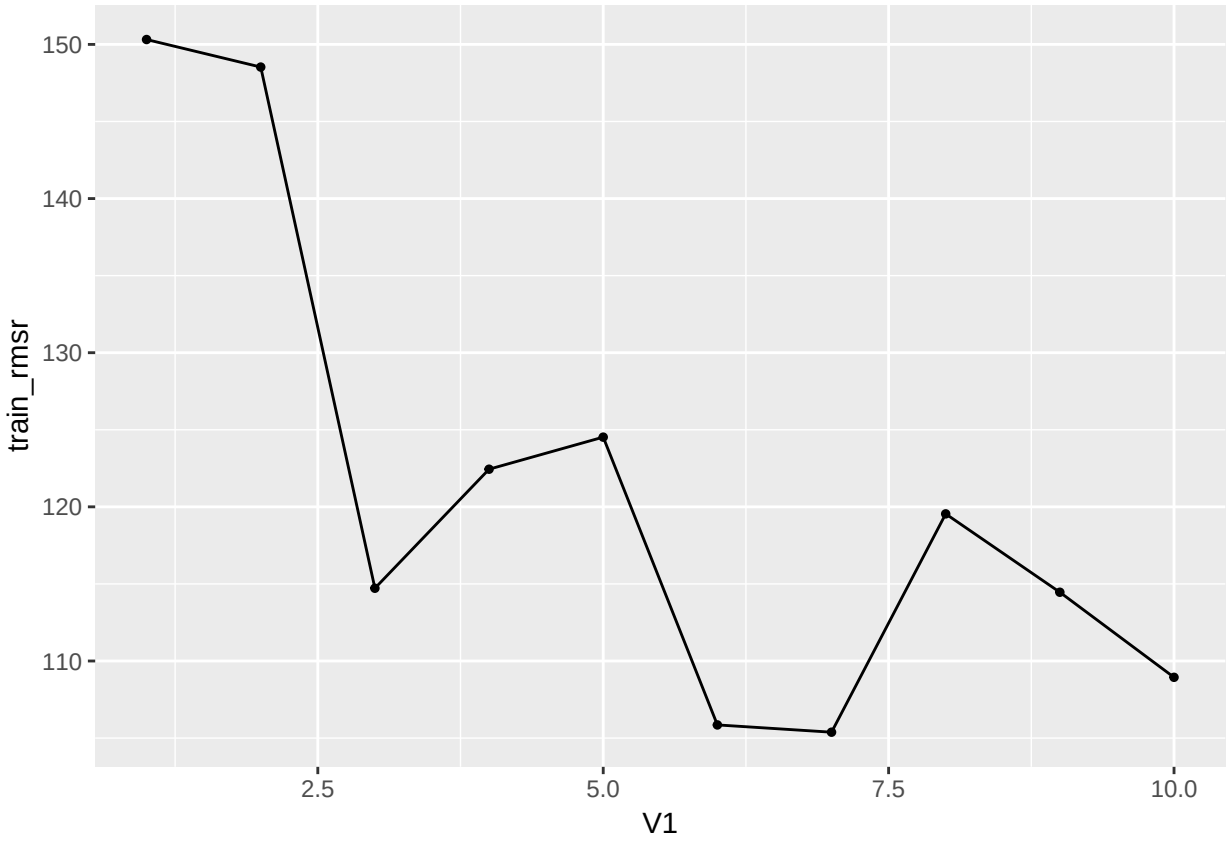
```
## iter 100 value 167619412.626344
## iter 110 value 167084947.746399
## iter 120 value 166969546.246414
## iter 130 value 166713843.839780
## iter 140 value 165926781.931532
## iter 150 value 165241975.162394
## iter 160 value 164769028.431212
## iter 170 value 164544522.812756
## iter 180 value 163738795.450577
## iter 190 value 162809641.756246
## iter 200 value 162030366.528539
## iter 210 value 161913786.279925
## iter 220 value 161809368.920469
## iter 230 value 161613617.787091
## iter 240 value 161491919.741107
## iter 250 value 161471538.687959
## iter 260 value 161323101.562713
## iter 270 value 160883360.051212
## iter 280 value 160524109.411720
## iter 290 value 160340516.673947
## iter 300 value 159971186.084920
## iter 310 value 159785528.529513
## iter 320 value 159744790.094757
## iter 330 value 159735382.152844
## iter 340 value 159719289.812088
## iter 350 value 159696266.067277
## iter 360 value 159666561.069099
## iter 370 value 159629124.536562
## iter 380 value 159595004.642027
## iter 390 value 159518237.996199
## iter 400 value 159482182.642712
## iter 410 value 159472210.782012
## iter 420 value 159463904.707984
## iter 430 value 159455501.240526
## iter 440 value 159449772.216661
## iter 450 value 159444193.255827
## iter 460 value 159423499.191754
## iter 470 value 159415222.634376
## iter 480 value 159406171.595469
## iter 490 value 159399637.494409
## iter 500 value 159394740.460859
## final  value 159392313.500032
## converged
## # weights:  191
## initial  value 561633003.980727
## iter  10 value 216103423.448484
## iter  20 value 198062755.350339
## iter  30 value 197095358.869978
## iter  40 value 196659242.596088
## iter  50 value 190683639.674090
## iter  60 value 190233501.336710
## iter  70 value 190087653.011203
## iter  80 value 189967303.409110
## iter  90 value 189877785.031046
```

```
## iter 100 value 189230898.114975
## iter 110 value 188843376.549593
## iter 120 value 188618837.477374
## iter 130 value 183715181.884257
## iter 140 value 176730860.502541
## iter 150 value 175384153.569825
## iter 160 value 174490481.173937
## iter 170 value 173721924.605959
## iter 180 value 173405478.554812
## iter 190 value 173217939.015835
## iter 200 value 172972951.887041
## iter 210 value 171781718.654623
## iter 220 value 171330188.599559
## iter 230 value 171114844.770416
## iter 240 value 170869860.336240
## iter 250 value 170709245.212989
## iter 260 value 170621694.308334
## iter 270 value 170471966.009759
## iter 280 value 170215821.726038
## iter 290 value 170122899.626032
## iter 300 value 170000549.792190
## iter 310 value 169841669.016859
## iter 320 value 169711720.395144
## iter 330 value 169565674.485387
## iter 340 value 169464588.396205
## iter 350 value 165040226.860908
## iter 360 value 162708102.838201
## iter 370 value 161665649.036965
## iter 380 value 160144616.899367
## iter 390 value 158396952.329143
## iter 400 value 153786314.260267
## iter 410 value 152182618.248059
## iter 420 value 151262943.466577
## iter 430 value 151015533.795134
## iter 440 value 150824569.473931
## iter 450 value 150243653.694171
## iter 460 value 148655504.075796
## iter 470 value 148178037.974314
## iter 480 value 147898828.368844
## iter 490 value 147702968.996351
## iter 500 value 147581228.530307
## iter 510 value 147528157.412243
## iter 520 value 147391363.935236
## iter 530 value 147269705.475136
## iter 540 value 147214803.550655
## iter 550 value 147063836.965573
## iter 560 value 146682950.968466
## iter 570 value 146546106.904127
## iter 580 value 146481138.784982
## iter 590 value 146450013.670185
## iter 600 value 146431611.716145
## iter 610 value 146383845.916668
## iter 620 value 146351618.334102
## iter 630 value 146282233.378362
```

```
## iter 640 value 146169919.336391
## iter 650 value 146056510.053723
## iter 660 value 146016818.241945
## iter 670 value 145914439.385065
## iter 680 value 145829089.308924
## iter 690 value 145797182.447391
## iter 700 value 145782875.319533
## iter 710 value 145774362.971405
## iter 720 value 145741138.024409
## iter 730 value 145713731.263433
## iter 740 value 145690722.514937
## iter 750 value 145637696.630790
## iter 760 value 145585023.322325
## iter 770 value 145558893.231320
## iter 780 value 145551717.224769
## iter 790 value 145533265.658901
## iter 800 value 145468319.951115
## iter 810 value 145451289.114893
## iter 820 value 145437326.514374
## iter 830 value 145430614.849597
## iter 840 value 145428316.971960
## iter 850 value 145425433.103232
## iter 860 value 145415049.176465
## iter 870 value 145404961.873524
## iter 880 value 145390012.713151
## iter 890 value 145385785.635786
## iter 900 value 145372437.036005
## iter 910 value 145357639.603439
## iter 920 value 145356137.068469
## iter 930 value 145355357.522629
## iter 940 value 145354669.501975
## iter 950 value 145354022.683023
## iter 960 value 145353722.730404
## iter 970 value 145093362.770847
## iter 980 value 144976162.625944
## iter 990 value 144923507.555791
## iter1000 value 144890637.844301
## iter1010 value 144861480.682621
## iter1020 value 144809131.261842
## iter1030 value 144712182.705917
## iter1040 value 144632210.949823
## iter1050 value 144554704.163336
## iter1060 value 144540028.593852
## iter1070 value 144530994.482350
## iter1080 value 144516186.569851
## iter1090 value 144474265.213374
## iter1100 value 144436757.462293
## iter1110 value 144428001.505223
## iter1120 value 144421259.257632
## iter1130 value 144419481.085131
## iter1140 value 144417505.238780
## iter1150 value 144406935.892247
## iter1150 value 144406935.725742
## final  value 144406935.725742
```

```
## converged
```

```
train_rmsr <- data.frame(cbind(1:10, train_rmsr))
ggplot(train_rmsr, aes(x = V1, y = train_rmsr, group = 1)) + geom_point(size = 1, alpha = 1, na.rm = TRU
```



While the RMSE continues to change throughout, there is only a tiny decrease in the RMSE between the model with 6 nodes and the model with 7 nodes, so we will choose the model with 6 nodes in order to get a model that both fits well and has a lower chance of overfitting.

(c) Repeat part (a), using two time related and two weather related variables. Explain your choice of variables.

```
train_rmsr <- numeric()

set.seed(1)
for(i in 1:10) {
  nn_fit <- nnet(registered ~ wday + month + temp + windspeed, data= cabi_train, maxit = 2000, decay =
  train_rmsr[i] <- sqrt(mean((predict(nn_fit, type = "r") - cabi_train$registered)^2))
}
```

```
## # weights:  17
## initial  value 560420429.512363
## iter  10 value 267960566.769188
## iter  20 value 267188460.319366
## iter  30 value 251481006.542679
## iter  40 value 251218517.165351
## iter  50 value 251218167.035904
## iter  60 value 251218038.113515
```

```
## iter  60 value 251218037.040066
## iter  60 value 251218035.727649
## final  value 251218035.727649
## converged
## # weights:  33
## initial  value 559759377.850141
## iter  10 value 266360626.786996
## iter  20 value 264841895.435228
## iter  30 value 234286810.789181
## iter  40 value 231266062.278660
## iter  50 value 230951459.259468
## iter  60 value 228171214.533844
## iter  70 value 226264024.957501
## iter  80 value 224622029.422517
## iter  90 value 224263124.918252
## iter 100 value 224171551.190108
## iter 110 value 224126730.776681
## iter 120 value 224093316.860373
## final  value 224015898.720749
## converged
## # weights:  49
## initial  value 558757480.956590
## iter  10 value 254132307.542546
## iter  20 value 244836707.584883
## iter  30 value 242357494.983544
## iter  40 value 239858479.266006
## iter  50 value 239092851.602062
## iter  60 value 236680728.911877
## iter  70 value 236517562.845258
## final  value 236517022.429727
## converged
## # weights:  65
## initial  value 561995186.010537
## iter  10 value 243094454.388583
## iter  20 value 232521279.662054
## iter  30 value 228046219.318199
## iter  40 value 226907723.388759
## iter  50 value 226112526.014757
## iter  60 value 225775364.488331
## iter  70 value 225233831.630658
## iter  80 value 223756203.551964
## iter  90 value 223374757.235519
## iter 100 value 223295215.465808
## iter 110 value 223048983.299473
## iter 120 value 222641200.549805
## iter 130 value 222466689.280955
## iter 140 value 222434669.683443
## iter 150 value 222422272.764007
## iter 160 value 222326767.439510
## iter 170 value 222186999.068250
## iter 180 value 221827723.351273
## iter 190 value 221794800.408772
## iter 200 value 221762459.365962
## iter 210 value 221751213.937189
```

```
## iter 220 value 221737316.895748
## iter 230 value 221713698.311744
## iter 240 value 221684689.239341
## iter 250 value 221675093.381634
## iter 260 value 221668169.273899
## iter 270 value 221664712.861379
## iter 280 value 221636417.421919
## iter 290 value 221635095.464544
## iter 300 value 221634288.929226
## iter 310 value 221633125.739631
## iter 320 value 221632965.954771
## iter 330 value 221632573.277425
## final  value 221632477.637558
## converged
## # weights:  81
## initial  value 562375808.029613
## final  value 274876841.744121
## converged
## # weights:  97
## initial  value 560865378.852913
## iter  10 value 274493645.522268
## iter  20 value 255001637.038386
## iter  30 value 250424858.320504
## iter  40 value 246891755.545472
## iter  50 value 246378711.856624
## iter  60 value 245981200.699355
## iter  70 value 245638372.498814
## iter  80 value 245483704.389092
## iter  90 value 243912286.461833
## iter 100 value 242538154.947233
## iter 110 value 242294902.133612
## iter 120 value 241281345.052850
## iter 130 value 240979829.572720
## iter 140 value 240948312.758994
## iter 150 value 240933635.451328
## iter 160 value 240782021.453199
## iter 170 value 240741913.917375
## iter 180 value 240741195.275838
## iter 190 value 240739919.806652
## iter 200 value 240313920.294646
## iter 210 value 240062909.368674
## iter 220 value 239926789.388290
## iter 230 value 230996841.526539
## iter 240 value 229964595.895397
## iter 250 value 229846944.439448
## iter 260 value 228915340.571161
## iter 270 value 227925482.559710
## iter 280 value 227625856.263275
## iter 290 value 227422809.523724
## iter 300 value 227384425.428849
## iter 310 value 226863867.760381
## iter 320 value 226694862.455117
## iter 330 value 226666000.854071
## iter 340 value 226556022.283256
```

```
## iter 350 value 226521740.863672
## iter 360 value 226411009.663817
## iter 370 value 226369153.173686
## iter 380 value 226352810.918007
## iter 390 value 226335911.633413
## iter 400 value 226255002.317376
## iter 410 value 226229034.621300
## iter 420 value 226208453.856979
## iter 430 value 226202144.838262
## iter 440 value 226120243.455997
## iter 450 value 226101347.567529
## iter 460 value 226097820.424014
## iter 470 value 225963610.912049
## iter 480 value 225943943.054512
## iter 490 value 225924626.947735
## iter 500 value 225900043.922597
## iter 510 value 225888598.516919
## iter 520 value 225800450.526821
## iter 530 value 225683371.260596
## iter 540 value 225662498.017886
## iter 550 value 225636767.311548
## iter 560 value 225632107.636935
## iter 570 value 225611059.372380
## iter 580 value 225590469.158310
## iter 590 value 225586748.340772
## iter 600 value 225532415.947595
## iter 610 value 225355177.583626
## iter 620 value 224553911.849898
## iter 630 value 224141807.198595
## iter 640 value 224070441.652799
## iter 650 value 223777815.602725
## iter 660 value 223731288.732262
## iter 670 value 223627596.691909
## iter 680 value 223434607.515853
## iter 690 value 222980817.202121
## iter 700 value 222858672.317820
## iter 710 value 222832149.188483
## iter 720 value 222806620.144992
## iter 730 value 222794734.029159
## iter 740 value 222660839.409862
## iter 750 value 222611402.747883
## iter 760 value 222504200.770605
## iter 770 value 222480428.373919
## iter 780 value 222473680.216103
## iter 790 value 222454162.835007
## iter 800 value 222423933.246594
## iter 810 value 222418256.024162
## iter 820 value 222373894.875413
## iter 830 value 222288617.441652
## iter 840 value 222276265.482787
## iter 850 value 222275227.539627
## iter 860 value 222274684.781560
## iter 870 value 222271571.910518
## iter 880 value 222268958.692123
```

```
## iter 890 value 222248818.720596
## iter 900 value 222226861.593452
## iter 910 value 222187997.984853
## iter 920 value 222185805.482566
## iter 930 value 222185104.463069
## iter 940 value 222183943.019928
## iter 950 value 222183394.212903
## iter 960 value 222182597.854211
## iter 970 value 222180502.853084
## iter 980 value 222135368.426985
## final  value 222134561.968161
## converged
## # weights:  113
## initial  value 561731973.473702
## iter  10 value 254122720.787994
## iter  20 value 250111043.583586
## iter  30 value 246907270.077521
## iter  40 value 237205683.406891
## iter  50 value 233672214.346385
## iter  60 value 232614224.751607
## iter  70 value 232485096.619593
## iter  80 value 231514810.418784
## iter  90 value 229918153.268546
## iter 100 value 228358889.538267
## iter 110 value 227742230.061376
## iter 120 value 226945733.485872
## iter 130 value 226451232.506613
## iter 140 value 224542658.580930
## iter 150 value 224186314.939494
## iter 160 value 224130761.906972
## iter 170 value 224082520.431732
## iter 180 value 223932809.753239
## iter 190 value 223909305.616734
## iter 200 value 223788857.367168
## iter 210 value 223732215.626746
## iter 220 value 223586112.479645
## iter 230 value 223464862.979314
## iter 240 value 222798250.709014
## iter 250 value 222435333.793110
## iter 260 value 222287076.179029
## iter 270 value 222266810.681502
## iter 280 value 222215971.627059
## iter 290 value 222180411.272915
## iter 300 value 222017070.434296
## iter 310 value 221962462.141680
## iter 320 value 221948652.211959
## iter 330 value 221929960.236804
## iter 340 value 221801379.383497
## iter 350 value 221749902.076450
## iter 360 value 221749188.207302
## iter 370 value 221748472.012806
## iter 380 value 221732548.774065
## iter 390 value 221676599.463820
## iter 400 value 221592388.649119
```

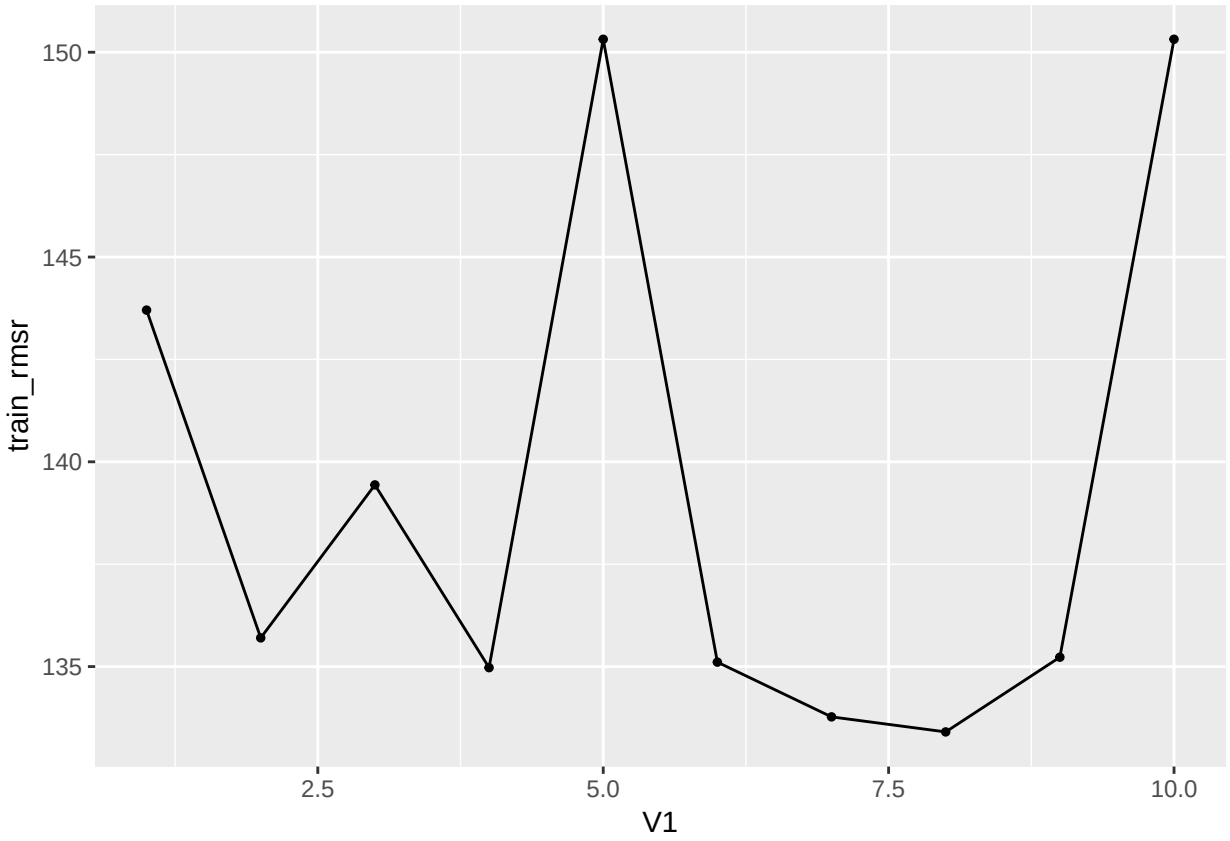
```
## iter 410 value 221412591.644251
## iter 420 value 221382314.722753
## iter 430 value 221370714.044608
## iter 440 value 221262896.097386
## iter 450 value 221140154.544916
## iter 460 value 221077565.705621
## iter 470 value 220634775.203133
## iter 480 value 220412031.096585
## iter 490 value 220361287.650406
## iter 500 value 220317523.782509
## iter 510 value 220249735.275182
## iter 520 value 220160787.451219
## iter 530 value 220040365.971081
## iter 540 value 219889909.192077
## iter 550 value 219761686.970245
## iter 560 value 219691656.635118
## iter 570 value 219662175.601432
## iter 580 value 219570800.454533
## iter 590 value 219466151.589199
## iter 600 value 219417056.630639
## iter 610 value 219316093.172904
## iter 620 value 219123002.338011
## iter 630 value 219025120.514533
## iter 640 value 219015922.874994
## iter 650 value 219003418.605291
## iter 660 value 218989240.568799
## iter 670 value 218974960.370229
## iter 680 value 218959297.063558
## iter 690 value 218933104.456378
## iter 700 value 218902549.960056
## iter 710 value 218873770.216995
## iter 720 value 218797178.555046
## iter 730 value 218582123.052027
## iter 740 value 218262445.360010
## iter 750 value 218098968.199318
## iter 760 value 218043554.452827
## iter 770 value 217982517.709096
## iter 780 value 217922136.509056
## iter 790 value 217883936.296966
## iter 800 value 217866465.724094
## iter 810 value 217842190.026402
## iter 820 value 217817711.524414
## iter 830 value 217813424.623253
## final  value 217813227.990316
## converged
## # weights:  129
## initial  value 559225273.266865
## iter  10 value 257196293.701282
## iter  20 value 253071270.676263
## iter  30 value 245693374.657564
## iter  40 value 242891775.462564
## iter  50 value 242141712.334032
## iter  60 value 238640359.414434
## iter  70 value 232167351.325617
```

```
## iter  80 value 231038421.705625
## iter  90 value 230021106.089983
## iter 100 value 228448938.232127
## iter 110 value 228353109.423143
## iter 120 value 228019635.250134
## iter 130 value 227616430.311059
## iter 140 value 227358271.762440
## iter 150 value 227245741.240883
## iter 160 value 227150267.519550
## iter 170 value 226094084.859575
## iter 180 value 225701507.085154
## iter 190 value 225638741.145541
## iter 200 value 225618898.013189
## iter 210 value 225612390.817550
## iter 220 value 224934435.759691
## iter 230 value 224866075.326817
## iter 240 value 224809443.692302
## iter 250 value 224707486.961571
## iter 260 value 224639180.987221
## iter 270 value 224595193.396377
## iter 280 value 224534546.496744
## iter 290 value 224446554.080284
## iter 300 value 224339318.904061
## iter 310 value 224300817.028834
## iter 320 value 224218524.804866
## iter 330 value 223884921.661652
## iter 340 value 223731671.390945
## iter 350 value 223383477.306808
## iter 360 value 223280324.074161
## iter 370 value 223191697.023983
## iter 380 value 223046427.903002
## iter 390 value 222976176.494933
## iter 400 value 222402072.517941
## iter 410 value 222199264.100725
## iter 420 value 221859838.602522
## iter 430 value 221797948.748559
## iter 440 value 221781665.938110
## iter 450 value 221604568.202373
## iter 460 value 221368153.434733
## iter 470 value 221327865.885588
## iter 480 value 221128291.407054
## iter 490 value 220103847.757475
## iter 500 value 219737062.749916
## iter 510 value 219660865.828607
## iter 520 value 219610064.330848
## iter 530 value 219501264.544654
## iter 540 value 219437347.939384
## iter 550 value 219404407.418850
## iter 560 value 219381009.894185
## iter 570 value 219290329.262139
## iter 580 value 219168921.201618
## iter 590 value 219044984.114906
## iter 600 value 218990483.605141
## iter 610 value 218913516.172435
```

```
## iter 620 value 218853054.211248
## iter 630 value 218838990.936339
## iter 640 value 218793600.649242
## iter 650 value 218719836.843181
## iter 660 value 218712453.852022
## iter 670 value 218635839.464791
## iter 680 value 218593871.664427
## iter 690 value 218427408.531671
## iter 700 value 218362707.927593
## iter 710 value 218356714.740119
## iter 720 value 218345656.905181
## iter 730 value 218340332.626304
## iter 740 value 218336103.620029
## iter 750 value 218330447.900100
## iter 760 value 218319295.285377
## iter 770 value 218303348.292513
## iter 780 value 218289566.737280
## iter 790 value 218269508.878001
## iter 800 value 218236123.436605
## iter 810 value 218215587.024901
## iter 820 value 218212584.736232
## iter 830 value 218209743.836138
## iter 840 value 218198810.911712
## iter 850 value 218114095.412742
## iter 860 value 217933571.467602
## iter 870 value 217680052.669299
## iter 880 value 217659412.500921
## iter 890 value 217649638.638385
## iter 900 value 217602864.139075
## iter 910 value 217487317.302185
## iter 920 value 217438373.176171
## iter 930 value 217401355.611353
## iter 940 value 217338625.229530
## iter 950 value 217287479.948598
## iter 960 value 217239547.899093
## iter 970 value 217157140.059245
## iter 980 value 217126992.440534
## iter 990 value 217087785.040798
## iter1000 value 217039710.126786
## iter1010 value 217013658.044391
## iter1020 value 217004046.359980
## iter1030 value 216972961.164037
## iter1040 value 216941820.398843
## iter1050 value 216889827.197767
## iter1060 value 216869065.848038
## iter1070 value 216841345.003795
## iter1080 value 216818380.777564
## iter1090 value 216732990.798613
## iter1100 value 216688030.022433
## iter1110 value 216650872.921668
## iter1120 value 216646178.209360
## iter1130 value 216637666.089629
## iter1140 value 216635126.379113
## iter1150 value 216635000.452590
```

```
## final  value 216634822.541653
## converged
## # weights:  145
## initial  value 558317610.773321
## iter  10 value 267969942.715862
## iter  20 value 243899300.701285
## iter  30 value 241063252.024437
## iter  40 value 239985968.884187
## iter  50 value 239059433.593696
## iter  60 value 238566626.938846
## iter  70 value 237667055.850197
## iter  80 value 234821562.867738
## iter  90 value 229214252.297149
## iter 100 value 228078238.062426
## iter 110 value 227280689.643361
## iter 120 value 226567814.430620
## iter 130 value 224406747.243236
## iter 140 value 224212783.007234
## iter 150 value 223664070.606473
## iter 160 value 223446159.989773
## iter 170 value 223398372.458676
## iter 180 value 222944070.684966
## iter 190 value 222859700.817455
## iter 200 value 222796573.639973
## iter 210 value 222755322.189215
## iter 220 value 222664453.099434
## iter 230 value 222630930.294322
## iter 240 value 222624498.165531
## iter 250 value 222611624.207972
## iter 260 value 222594495.551848
## iter 270 value 222507642.741056
## final  value 222462885.564079
## converged
## # weights:  161
## initial  value 553395759.604499
## final  value 274872758.976361
## converged
```

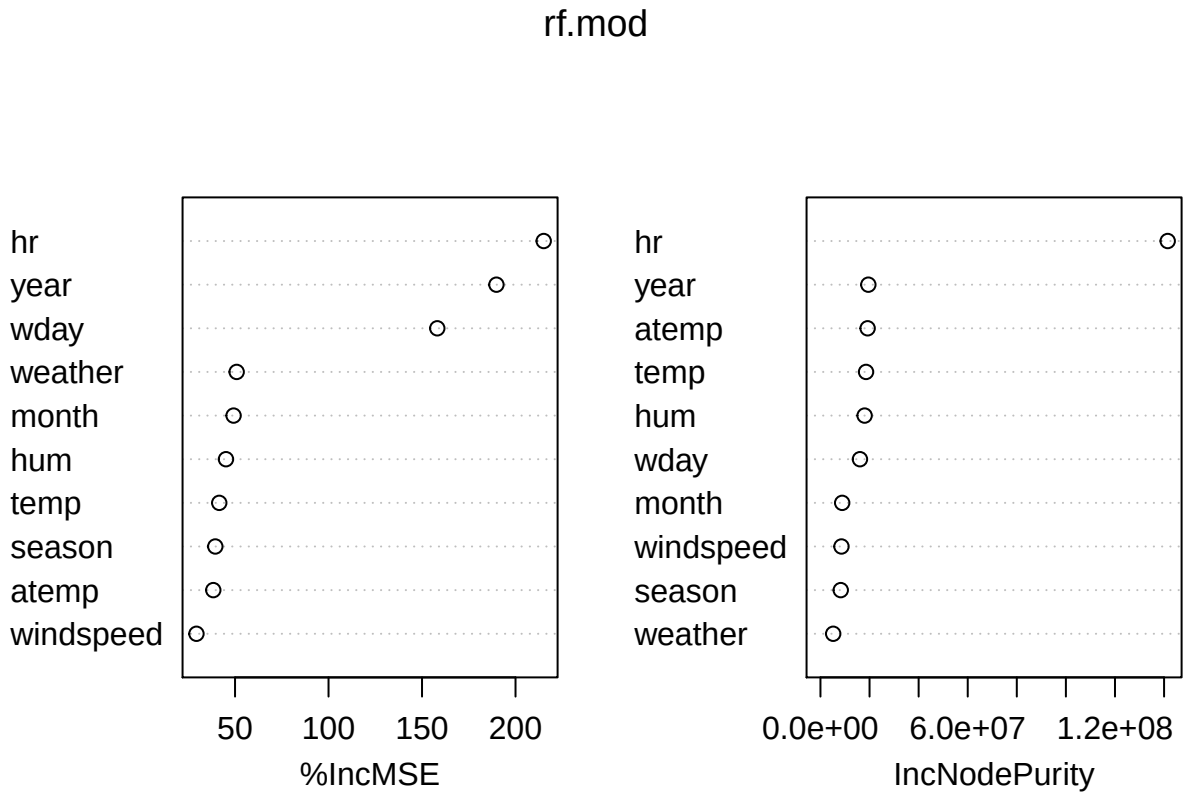
```
train_rmsr <- data.frame(cbind(1:10, train_rmsr))
ggplot(train_rmsr, aes(x = V1, y = train_rmsr, group = 1)) + geom_point(size = 1, alpha = 1, na.rm = TRU
```



I chose those four variables as they showed the highest correlation with ridership in our initial exploratory data analysis. While the RMSE continues to change throughout, there is only a tiny decrease in the RMSE between the model with 4 nodes and the model with 8 nodes, so we will choose the model with 4 nodes in order to get a model that both fits well and has a lower chance of overfitting.

Problem 4 (10) What do you think are six useful predictors? Use any method you want to answer this question.

```
rf.mod <- randomForest(registered ~ . -X -casual, data = cabi_train, importance =TRUE)
varImpPlot(rf.mod)
```



hr, year, and workday appear to be the most immportant as the mean decsrease in the mse resulting from the inclusion of the variables in the model is the smallest for those three variables.

Part II: Vegetation Cover

Problems 5 - 8 use data on vegetation cover. Use the data frames covtype.train and covtype.test. The original data are at https://archive.ics.uci.edu/ml/datasets/Covertype

Each data set contains 10,000 observations of 55 variables. These have been collected on 30m × 30m patches of hilly forest land by the US Forest Service.

```
covtype_train <- read.csv("covtype_train.csv", stringsAsFactors = FALSE , row.names = 1)
head(covtype_train)
```

```
##        elev aspect slope h_dist_hydro v_dist_hydro h_dist_road hillshade_9
## 278451 2380    323    19           30          10        1200         170
## 265496 2227    349    28          162         -38         815         158
## 41168  2724    239     1          127          -5        1570         216
## 355391 2766     60    10          120           0        2165         228
## 393250 3110    172    20          175           1        2407         228
## 347584 2780    153    26          360         160        1740         241
##        hillshade_12 hillshade_3 h_dist_fire wild1 wild2 wild3 wild4 soil1
## 278451          217         186         979     0     0     0     1     0
## 265496          183         156         366     0     0     0     1     0
## 41168           240         161        3957     1     0     0     0     0
## 355391          219         123         722     0     0     1     0     0
```

```
## 393250          245         139        3389     0     0     1     0     0
## 347584          230         102        1410     0     0     1     0     0
##        soil2 soil3 soil4 soil5 soil6 soil7 soil8 soil9 soil10 soil11
## 278451     0     0     0     0     0     0     0     0      1      0
## 265496     0     0     0     0     0     0     0     0      1      0
## 41168      0     0     0     0     0     0     0     0      0      0
## 355391     0     0     1     0     0     0     0     0      0      0
## 393250     0     0     0     0     0     0     0     0      0      0
## 347584     0     0     1     0     0     0     0     0      0      0
##        soil12 soil13 soil14 soil15 soil16 soil17 soil18 soil19 soil20
## 278451      0      0      0      0      0      0      0      0      0
## 265496      0      0      0      0      0      0      0      0      0
## 41168       1      0      0      0      0      0      0      0      0
## 355391      0      0      0      0      0      0      0      0      0
## 393250      0      0      0      0      0      0      0      0      0
## 347584      0      0      0      0      0      0      0      0      0
##        soil21 soil22 soil23 soil24 soil25 soil26 soil27 soil28 soil29
## 278451      0      0      0      0      0      0      0      0      0
## 265496      0      0      0      0      0      0      0      0      0
## 41168       0      0      0      0      0      0      0      0      0
## 355391      0      0      0      0      0      0      0      0      0
## 393250      0      0      0      0      0      0      0      0      0
## 347584      0      0      0      0      0      0      0      0      0
##        soil30 soil31 soil32 soil33 soil34 soil35 soil36 soil37 soil38
## 278451      0      0      0      0      0      0      0      0      0
## 265496      0      0      0      0      0      0      0      0      0
## 41168       0      0      0      0      0      0      0      0      0
## 355391      0      0      0      0      0      0      0      0      0
## 393250      0      0      0      1      0      0      0      0      0
## 347584      0      0      0      0      0      0      0      0      0
##        soil39 soil40 cover
## 278451      0      0     3
## 265496      0      0     3
## 41168       0      0     2
## 355391      0      0     2
## 393250      0      0     2
## 347584      0      0     2
```

```
covtype_test <- read.csv("covtype_test.csv", stringsAsFactors = FALSE, row.names = 1)
head(covtype_test)
```

```
##        elev aspect slope h_dist_hydro v_dist_hydro h_dist_road hillshade_9
## 455853 2934    233    22          395          114        1218         178
## 299983 3028     14     2          518            5        3000         217
## 271634 2233     23    22          150           43        1045         202
## 277247 2204    313    22          127           24        1140         157
## 521872 2948    236    24          190           62        3013         172
## 110862 2816      6     6          492           51        1879         212
##        hillshade_12 hillshade_3 h_dist_fire wild1 wild2 wild3 wild4 soil1
## 455853          253         206         150     0     0     1     0     0
## 299983          235         155        1230     0     0     1     0     0
## 271634          188         115         283     0     0     0     1     0
## 277247          218         198        1182     0     0     0     1     0
## 521872          252         211        2048     0     0     1     0     0
## 110862          227         153        2804     1     0     0     0     0
```

```
##        soil2 soil3 soil4 soil5 soil6 soil7 soil8 soil9 soil10 soil11
## 455853     0     0     0     0     0     0     0     0      0      0
## 299983     0     0     0     0     0     0     0     0      0      0
## 271634     0     0     0     0     0     0     0     0      1      0
## 277247     0     0     0     0     0     0     0     0      1      0
## 521872     0     0     0     0     0     0     0     0      0      0
## 110862     0     0     0     0     0     0     0     0      0      0
##        soil12 soil13 soil14 soil15 soil16 soil17 soil18 soil19 soil20
## 455853      0      1      0      0      0      0      0      0      0
## 299983      0      0      0      0      0      0      0      0      0
## 271634      0      0      0      0      0      0      0      0      0
## 277247      0      0      0      0      0      0      0      0      0
## 521872      0      1      0      0      0      0      0      0      0
## 110862      0      0      0      0      0      0      0      0      0
##        soil21 soil22 soil23 soil24 soil25 soil26 soil27 soil28 soil29
## 455853      0      0      0      0      0      0      0      0      0
## 299983      0      0      0      0      0      0      0      0      0
## 271634      0      0      0      0      0      0      0      0      0
## 277247      0      0      0      0      0      0      0      0      0
## 521872      0      0      0      0      0      0      0      0      0
## 110862      0      0      1      0      0      0      0      0      0
##        soil30 soil31 soil32 soil33 soil34 soil35 soil36 soil37 soil38
## 455853      0      0      0      0      0      0      0      0      0
## 299983      0      0      1      0      0      0      0      0      0
## 271634      0      0      0      0      0      0      0      0      0
## 277247      0      0      0      0      0      0      0      0      0
## 521872      0      0      0      0      0      0      0      0      0
## 110862      0      0      0      0      0      0      0      0      0
##        soil39 soil40 cover
## 455853      0      0     2
## 299983      0      0     2
## 271634      0      0     3
## 277247      0      0     3
## 521872      0      0     2
## 110862      0      0     2
```

```r
# Change dumies to 0 1 scale
covtype_train$cover <- ifelse(covtype_train$cover == 3, 1, ifelse(covtype_train$cover == 2, 0, NA))
covtype_test$cover <- ifelse(covtype_test$cover == 3, 1, ifelse(covtype_test$cover == 2, 0, NA))

# Go back to import in first part and do this with row names too
```

Problem 5 (20) Fit a logistic model to the training data in order to separate the classes. Choose a classification threshold so that sensitivity and specificity are approximately the same on the training data. Then report sensitivity, specificity, and overall error rate for the test data.

```r
# Using the Train Data
covfit <- glm(factor(cover) ~ ., data = covtype_train, family=binomial)


pred = predict(covfit, covtype_train, type = "response")
class_pred = predict(covfit, covtype_train, type = "response") > .17

#accuracy - which rows did we get the class prediction correctly equal to z
sum(class_pred == covtype_train$cover) / nrow(covtype_train)
```

```
## [1] 0.964
```

```
# confusion matrix
b <- table(covtype_train$cover, class_pred)

# Test whether the sensitivity and specificity are approximately the same
# Specificity
a <- b[1,1]/(b[1,1] + b[1,2])
# Sensitivity
c<- b[2,2]/(b[2,2] + b[2,1])
# Specificity minus sensitivity
a-c
```

```
## [1] 0.00132
```

```
# Using the Test Data

pred <- predict(covfit, newdata = covtype_test, type = 'response')
class_pred = predict(covfit, covtype_test, type = "response") > .17

#accuracy - which rows did we get the class prediction correctly equal to z
sum(class_pred == covtype_test$cover) / nrow(covtype_test)
```

```
## [1] 0.964
```

```
# confusion matrix
b <- table(covtype_test$cover, class_pred)

# Test whether the sensitivity and specificity are approximately the same
# Specificity
b[1,1]/(b[1,1] + b[1,2])
```

```
## [1] 0.964
```

```
# Sensitivity
b[2,2]/(b[2,2] + b[2,1])
```

```
## [1] 0.958
```

```
auc(covtype_test$cover, pred)
```

```
## Area under the curve: 0.994
```

If we use the same classification threshold for the test that we used for training the model (0.17), then we end up with a specificity of 0.964 and a sensitivity of 0.958. The model performs relatively well as indicated by the test error (auc) of 0.994.

Problem 6 (25) Fit a support vector machine with radial kernels in order to separate the classes. Tune the cost and gamma parameters so that cross validation gives the best performance on the training data. Then assess the resulting model on the test data. Report sensitivity, specificity, and overall error rate for training and test data.

```
# Eliminated several soil columns to make run time faster

covtype_train2 <-covtype_train[,c(1:15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55)]
tune_out_radial <- tune(svm, factor(cover) ~ ., data = covtype_train2, kernel = "radial", scale = TRUE,
```

```
tune_out_radial2  <- tune_out_radial$best.model

# Train error
```

```r
covtype_train2$Pred_Class <- predict(tune_out_radial2, covtype_train2)

confusionMatrix(covtype_train2$cover, covtype_train2$Pred_Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8885    0
##          1 1115    0
##
##                Accuracy : 0.888
##                  95% CI : (0.882, 0.895)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.888
##             Specificity :    NA
##          Pos Pred Value :    NA
##          Neg Pred Value :    NA
##              Prevalence : 1.000
##          Detection Rate : 0.888
##    Detection Prevalence : 0.888
##       Balanced Accuracy :    NA
##
##        'Positive' Class : 0
##
```

```r
# Test error

covtype_test$Pred_Class <- predict(tune_out_radial2, covtype_test)

confusionMatrix(covtype_test$cover, covtype_test$Pred_Class)
```

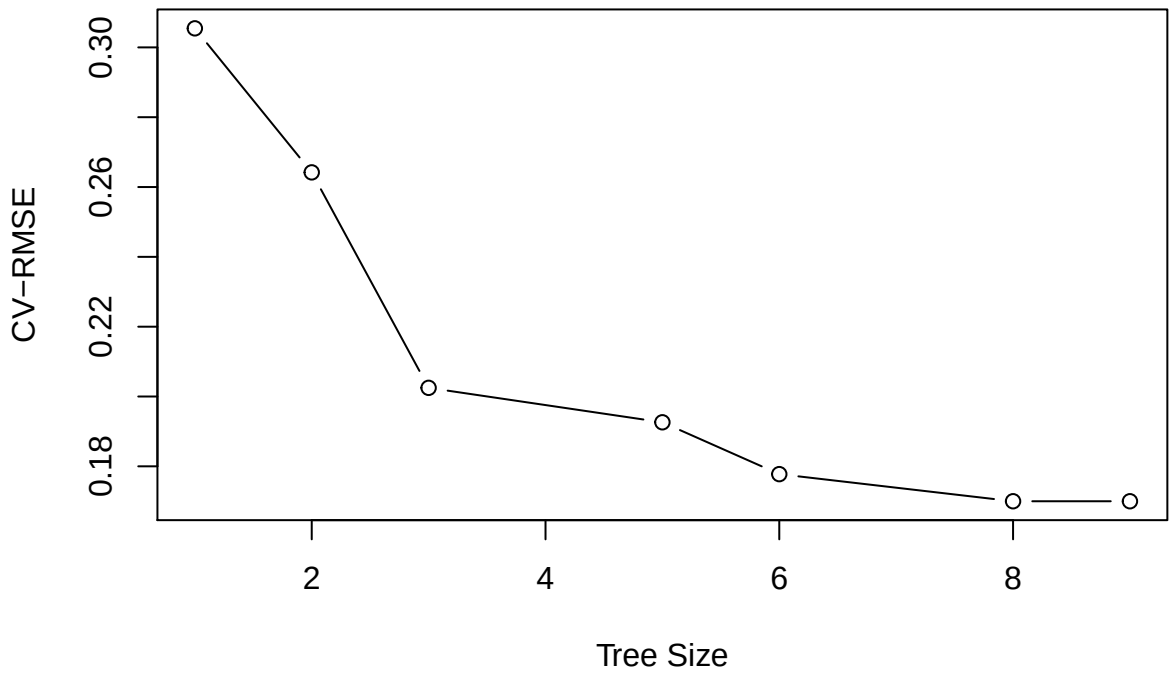
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8845    0
##          1 1155    0
##
##                Accuracy : 0.884
##                  95% CI : (0.878, 0.891)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.884
##             Specificity :    NA
```

```
##          Pos Pred Value :    NA
##          Neg Pred Value :    NA
##              Prevalence : 1.000
##          Detection Rate : 0.884
##    Detection Prevalence : 0.884
##       Balanced Accuracy :    NA
##
##        'Positive' Class : 0
##
```

```
# Initial class imbalance may have led model to only predict Os
```

Problem 7 (10) Fit a decision tree to the training data in order to separate the two classes. Prune the tree using cross validation and make sure that there are no redundant splits (i.e. splits that lead to leaves with the same classification). Then estimate the classification error rate for the pruned tree from the test data.

```
mytree <- tree(factor(cover) ~ ., data = covtype_train)

set.seed(3)
cv.mytree <- cv.tree(mytree ,FUN=prune.misclass)

plot(cv.mytree$size, sqrt(cv.mytree$dev / nrow(covtype_train)), type = "b",
     xlab = "Tree Size", ylab = "CV-RMSE")
```



Lowest error within one standard deviation rule is for a tree of size 3

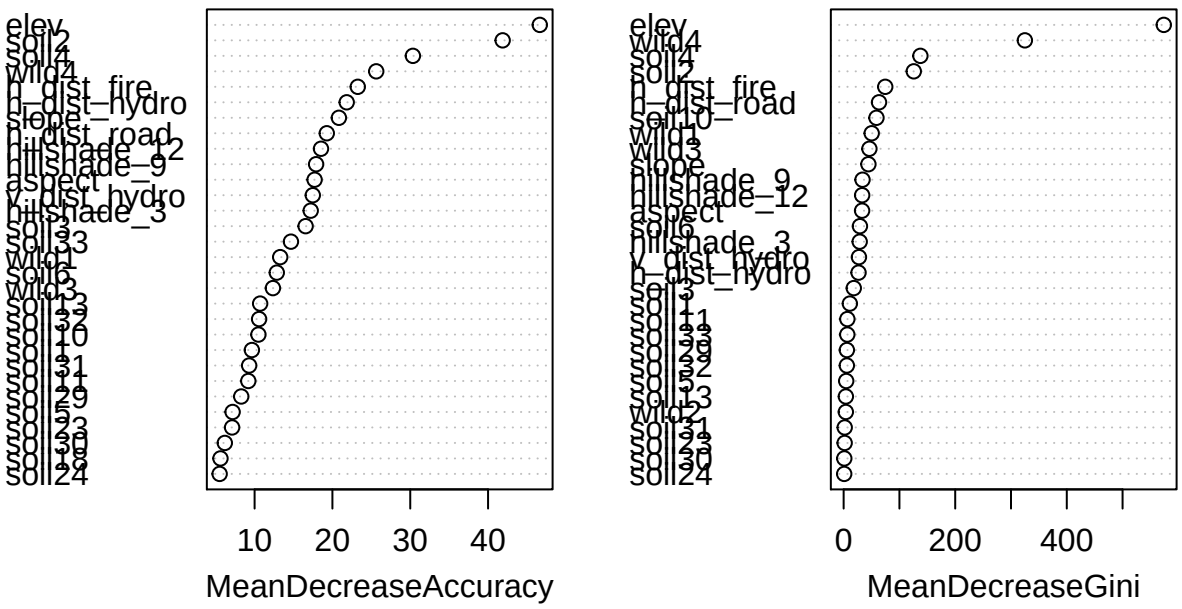
```
# So find the right tree length with cv and then pess through the prune.tree for the analysis
prune.mytree <- prune.tree(mytree, best=3)
```

```
y.pred <- predict(prune.mytree, newdata = covtype_test, type= "class")

confusionMatrix(y.pred, covtype_test$cover)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8776  410
##          1   69  745
##
##                Accuracy : 0.952
##                  95% CI : (0.948, 0.956)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.731
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.992
##             Specificity : 0.645
##          Pos Pred Value : 0.955
##          Neg Pred Value : 0.915
##              Prevalence : 0.884
##          Detection Rate : 0.878
##    Detection Prevalence : 0.919
##       Balanced Accuracy : 0.819
##
##        'Positive' Class : 0
##
```

Problem 8 (20) Fit a random forest model to the training data in order to separate the classes. Identify the ten most important variables and fit another random forest model, using only these variables. Use the test data to decide which model has better performance.

```
rf.fit <- randomForest(factor(cover) ~ ., data = covtype_train, importance =TRUE)
varImpPlot(rf.fit)
```




```
y.pred <- predict(rf.fit, newdata = covtype_test, type= "class")

confusionMatrix(y.pred, covtype_test$cover)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8752  110
##          1   93 1045
##
##                Accuracy : 0.98
##                  95% CI : (0.977, 0.982)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9
##  Mcnemar's Test P-Value : 0.261
##
##             Sensitivity : 0.989
##             Specificity : 0.905
##          Pos Pred Value : 0.988
##          Neg Pred Value : 0.918
##              Prevalence : 0.884
##          Detection Rate : 0.875
##    Detection Prevalence : 0.886
```

```
##       Balanced Accuracy : 0.947
##
##        'Positive' Class : 0
##
```

```
rf.fit2 <- randomForest(factor(cover) ~ elev + soil2 + soil4 + wild4 + h_dist_hydro + hillshade_12 + h_

y.pred2 <- predict(rf.fit2, newdata = covtype_test, type= "class")

confusionMatrix(y.pred2, covtype_test$cover)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8771  133
##          1   74 1022
##
##                Accuracy : 0.979
##                  95% CI : (0.976, 0.982)
##     No Information Rate : 0.884
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.896
##  Mcnemar's Test P-Value : 5.55e-05
##
##             Sensitivity : 0.992
##             Specificity : 0.885
##          Pos Pred Value : 0.985
##          Neg Pred Value : 0.932
##              Prevalence : 0.884
##          Detection Rate : 0.877
##    Detection Prevalence : 0.890
##       Balanced Accuracy : 0.938
##
##        'Positive' Class : 0
##
```

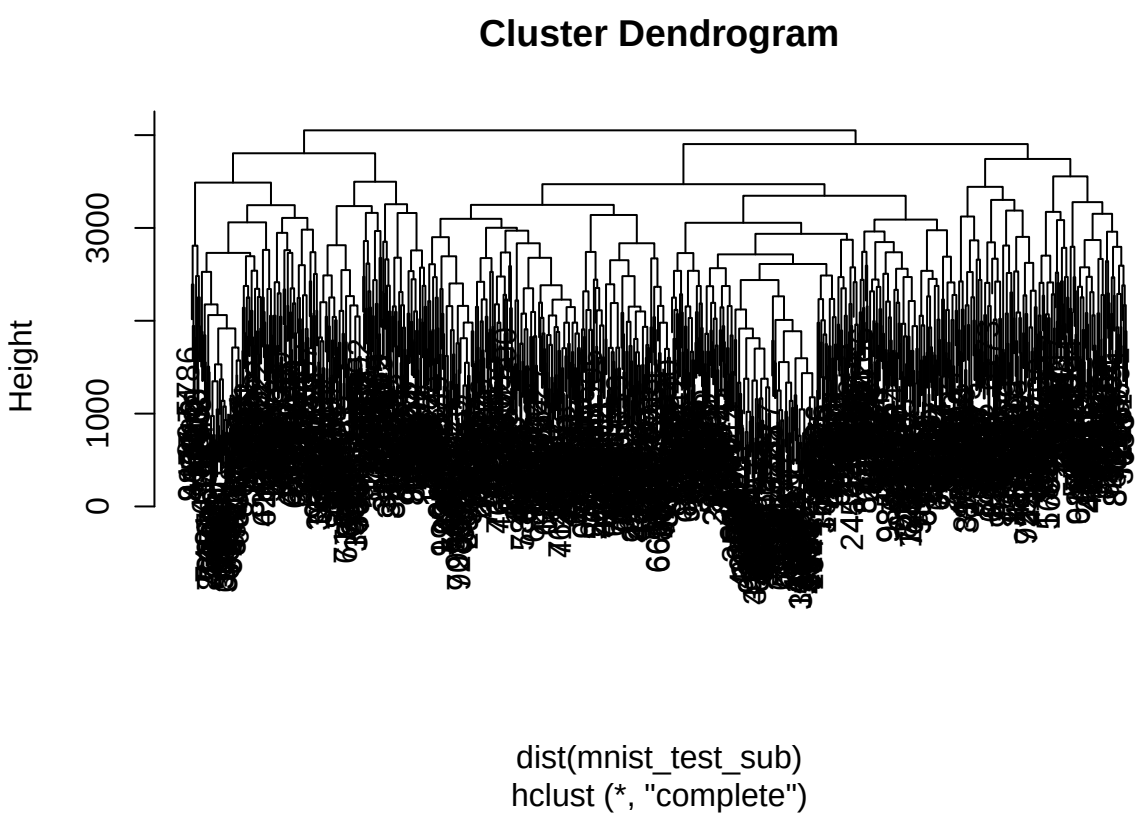
The model with only the top 10 variables included performs slightly better.

Part 3: MNIST Digit Data Problems 9 and 10 use the MNIST image classification data, available as mnist_all.RData in Canvas. We use only the test data (10,000 images).

```
mnist <-load('mnist_all.RData')
mnist_test <- data.frame(test$x, test$y)
```

Problem 9 (20) (a) Select a random subset of 1000 digits. Use hierarchical clustering with complete linkage on these images and visualize the dendrogram.

```
rsub <- sample(nrow(mnist_test), 1000, replace = FALSE)
mnist_test_sub <- mnist_test[rsub,]

clust.complete <- hclust(dist(mnist_test_sub),method="complete")
plot(clust.complete)
```




```
unique(test$y)
```

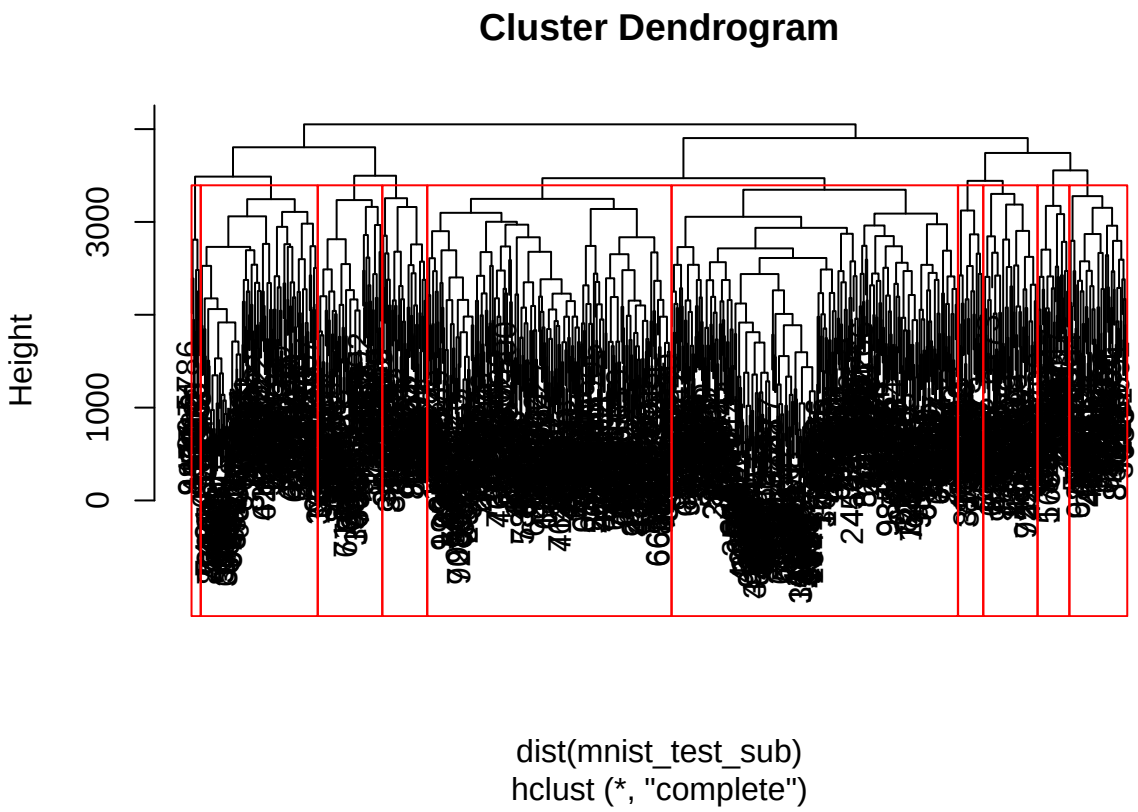
```
##  [1] 7 2 1 0 4 9 5 6 3 8
```

(b) Does the dendrogram provides compelling evidence about the "correct" number of clusters? Explain your answer.

The dendrogram does not give us the "correct" number of clusters, but it helps us better understand the ballpark. The lower in the tree that fusions occur, the more similar groups of observatios are to each other. In contrast, groups that fuse later can be quite different. In the above dendrogram, it looks like there is significant fusing that occurs lower in the tree. One single dendrogram can be used ot obtain any number of clusters though, and there are too many leaves above for us to sufficiently eyeball the correct number of clusters.

(c) Cut the dendrogram to generate a set of clusters that appears to be reasonable. There should be between 5 and 15 clusters. Then find a way to create a visual representation (i.e. a typical image) of each cluster. Explain and describe your approach.

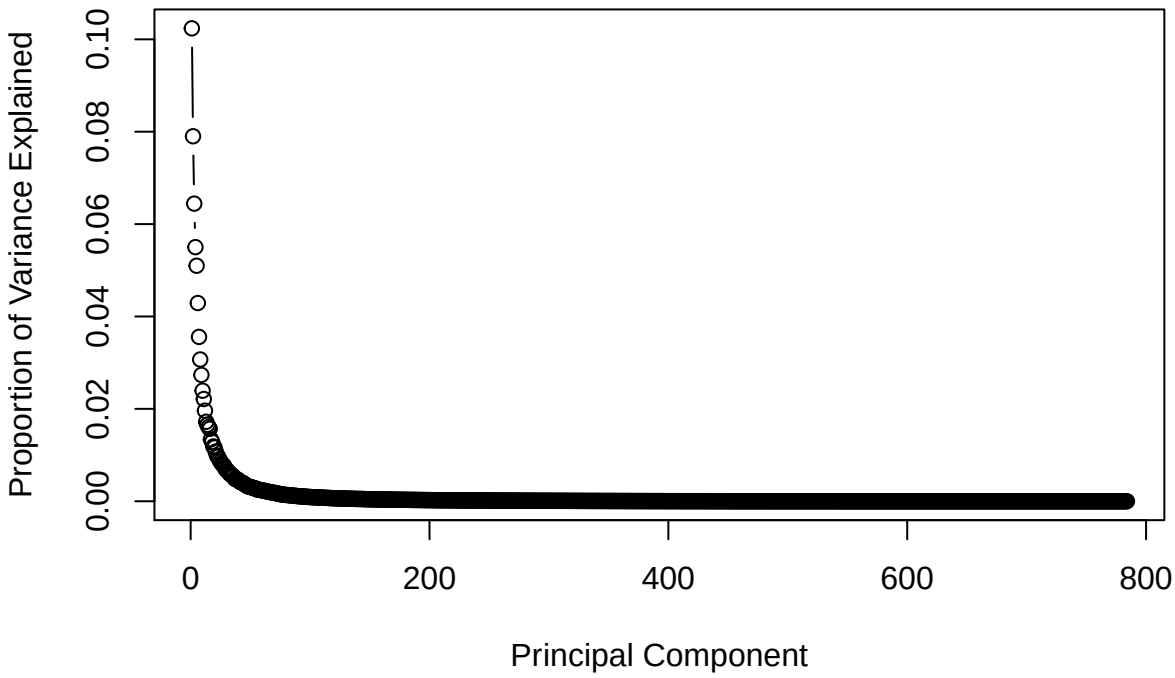
```
clust.complete.cut <- cutree(clust.complete,10)

plot(clust.complete)
rect.hclust(clust.complete, k=10, border="red")
```



Problem 10 (20) Use Principal Component Analysis on the MNIST images.

(a) Make a plot of the proportion of variance explained vs. number of principal components. Which fraction of the variance is explained by the first two principal components? Which fraction is explained by the first ten principal components?

```
# Create Matrix of Predictors
x <- model.matrix(test.y ~ .-1, mnist_test_sub)

# Compute PC
pr.out <- prcomp(x)

# Plot
std_dev <- pr.out$sdev
pr_var <- std_dev^2
prop_varex <- pr_var/sum(pr_var)
plot(prop_varex, xlab = "Principal Component",
             ylab = "Proportion of Variance Explained",
             type = "b")
```

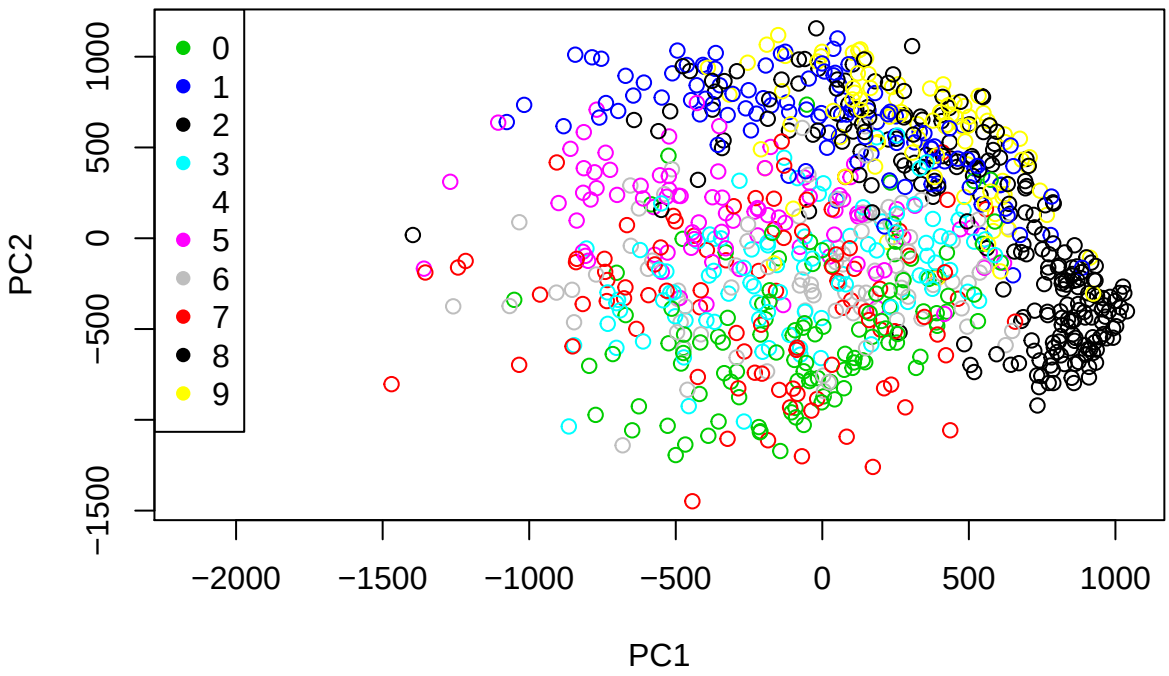



```
# Proportion explained by first ten
sum(prop_varex[1:10])
```

```
## [1] 0.512
```

(b) Plot the scores of the first two principal components of all digits against each other, color coded by the digit that is represented. Comment on the plot. Does it appear that digits may be separated by these scores?

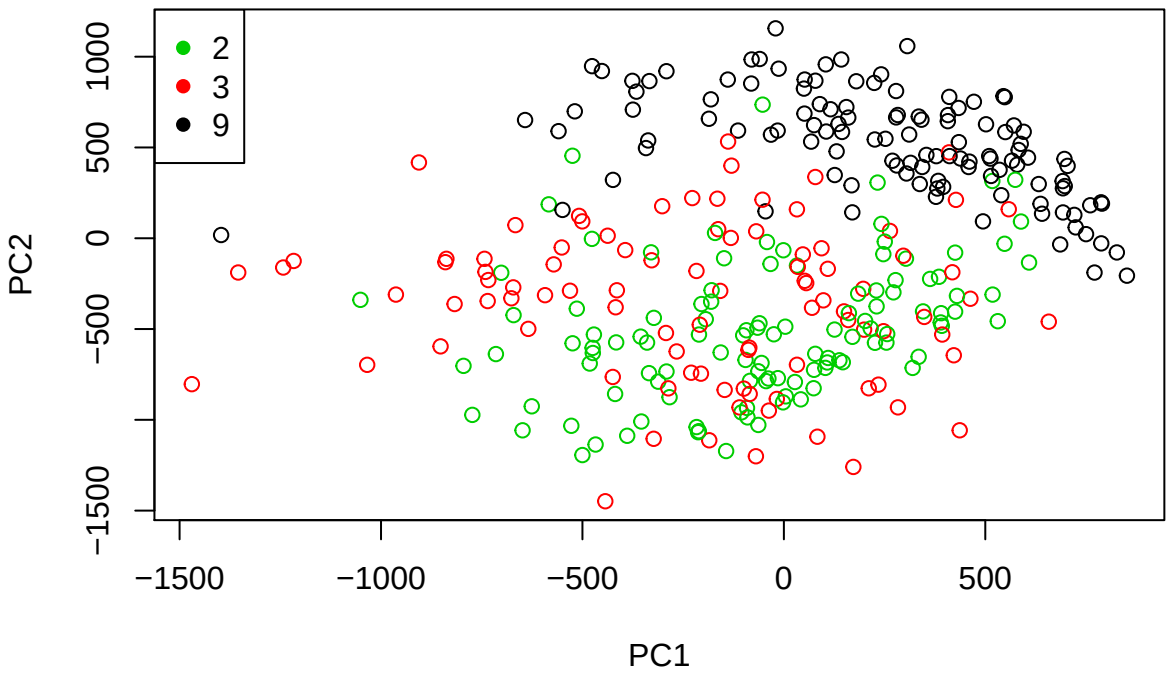
```
plot(pr.out$x[,1:2], col = mnist_test_sub$test.y)
legend("topleft", legend=levels(as.factor(mnist_test_sub$test.y)), pch=16, col=unique(mnist_test_sub$tes
```



It doea appear that some of the digits might be separated by the scores, though it is highly imperfect. As the above plot shows, for some numbers like those represented by black, blue, or yellow, there is something resembling clusters. For others like those represented by light blue or pink, it is not as clear.

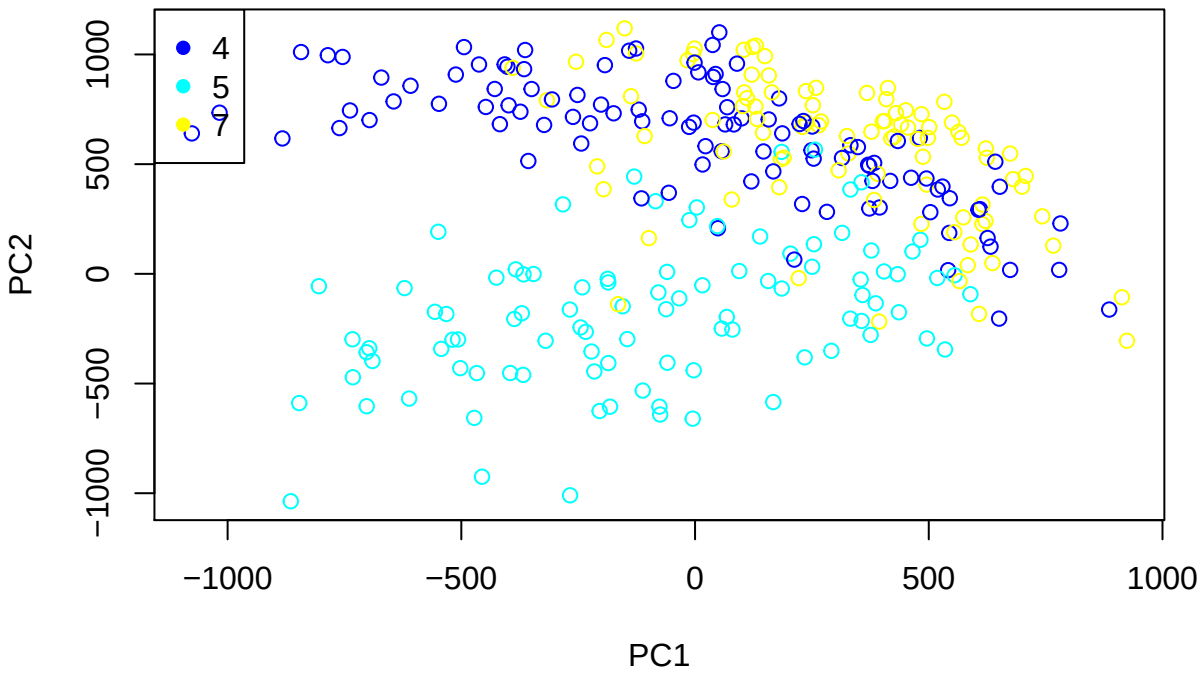
(c) Find three digits which are reasonably well separated by the plot that you made in part (b). Illustrate this with a color coded plot like the one in (b) for just these three digits. Don't expect perfect separation.

```
plot(pr.out$x[mnist_test_sub$test.y == 2 | mnist_test_sub$test.y == 3 | mnist_test_sub$test.y == 9,1:2]
legend("topleft", legend=levels(as.factor(mnist_test_sub[mnist_test_sub$test.y == 2 | mnist_test_sub$te
```



(d) Find three other digits which are not well separated by the plot that you made in part (b). Illustrate this with another color coded plot like the one in (b) for just these three digits.

```
plot(pr.out$x[mnist_test_sub$test.y == 4 | mnist_test_sub$test.y == 7 | mnist_test_sub$test.y == 5,1:2]
legend("topleft", legend=levels(as.factor(mnist_test_sub[mnist_test_sub$test.y == 4 | mnist_test_sub$te
```


4
5
7
PC2
1000
500
0
−500
−1000
−1000
−500
0
500
1000
PC1