Assignment template

Name

Date

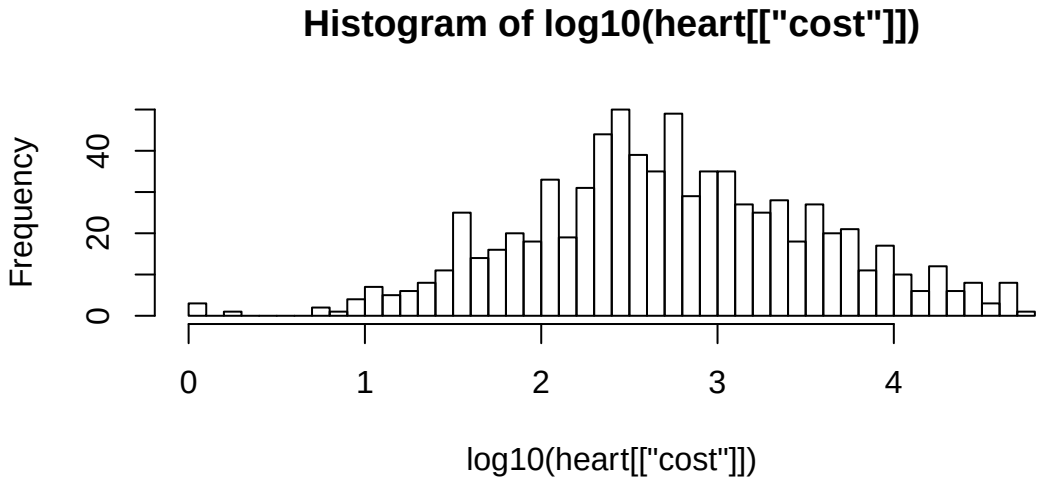
Problem 1

Reading in data

```
New names:
* `` -> ...1
```

```
# A tibble: 6 x 10
   ...1   cost   age  gend intvn drugs ervis  comp comorb   dur
  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>
1     1   179.    63     0     2     1     4     0      3   300
2     2   319     59     0     2     0     6     0      0   120
3     3  9311.    62     0    17     0     2     0      5   353
4     4   281.    60     1     9     0     7     0      2   332
5     5 18727.    55     0     5     2     7     0      0    18
6     6   453.    66     0     1     0     3     0      4   296
```

log10 transforming the response variable and plotting a histogram of the result.



(a) - Fitting a linear model

```
Call:
lm(formula = log10_cost ~ ., data = heart.fit1.data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.44852 -0.30093  0.01049  0.28276  1.72581
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.2228290  0.1698975  13.083  < 2e-16 ***
age         -0.0044135  0.0028817  -1.532   0.1260
gend        -0.0669173  0.0460024  -1.455   0.1462
intvn        0.0878065  0.0038090  23.053  < 2e-16 ***
drugs       -0.0257198  0.0213709  -1.203   0.2291
ervis        0.0224358  0.0090588   2.477   0.0135 *
comp         0.3270883  0.0794497   4.117 4.25e-05 ***
comorb       0.0228849  0.0037393   6.120 1.48e-09 ***
dur          0.0012181  0.0001874   6.501 1.43e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5373 on 779 degrees of freedom
Multiple R-squared:  0.5831, Adjusted R-squared:  0.5789
F-statistic: 136.2 on 8 and 779 DF,  p-value: < 2.2e-16
```

We see that there are some variable above, which are not significant. Under normal circumstances, it makes sense to run the lm fit without them.
In this case, however, I will keep them.

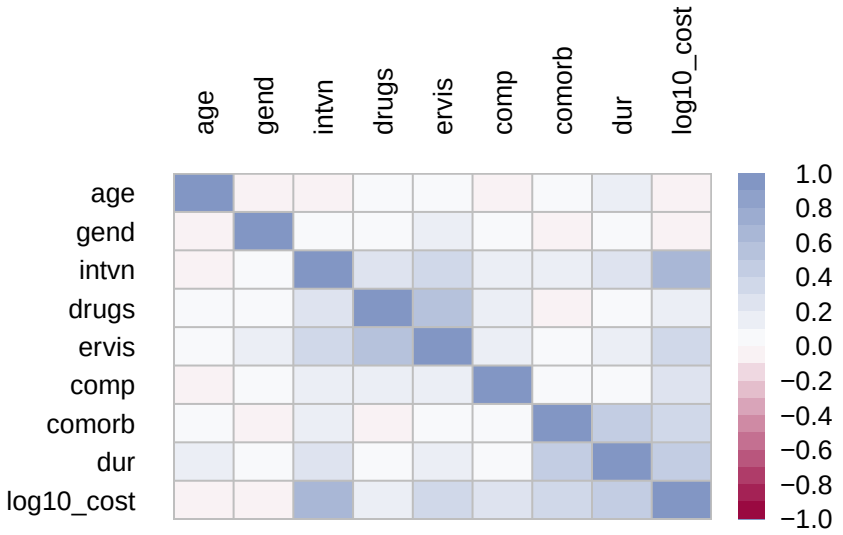
I calculate the Variance Inflation Factors as a check for multicollinearity. There doesn't appear to be a VIF>10, so don't suspect multicollinearity.

```
     age     gend    intvn    drugs    ervis     comp   comorb      dur
1.032749 1.018121 1.238005 1.409274 1.556264 1.058985 1.349956 1.399433
```

Calculating Correlation Matrix for the predictors as a further check on multicollinearity

```
               age        gend       intvn       drugs      ervis
age     1.00000000 -0.01804302 -0.03167889  0.02673140 0.06215487
gend   -0.01804302  1.00000000  0.03285323  0.03011970 0.11064303
intvn  -0.03167889  0.03285323  1.00000000  0.22953160 0.36676833
drugs   0.02673140  0.03011970  0.22953160  1.00000000 0.52771831
ervis   0.06215487  0.11064303  0.36676833  0.52771831 1.00000000
comp   -0.04271363  0.03317535  0.19610253  0.14876413 0.16093851
comorb  0.09367482 -0.04115477  0.14899747 -0.05093854 0.03078128
dur     0.13551739  0.02355227  0.22637435  0.06477961 0.14470949
              comp      comorb        dur
age    -0.04271363  0.09367482 0.13551739
gend    0.03317535 -0.04115477 0.02355227
intvn   0.19610253  0.14899747 0.22637435
drugs   0.14876413 -0.05093854 0.06477961
ervis   0.16093851  0.03078128 0.14470949
comp    1.00000000  0.03228381 0.07589605
comorb  0.03228381  1.00000000 0.49518636
dur     0.07589605  0.49518636 1.00000000
```

Visualizing the correlation matrix so this is easier to read

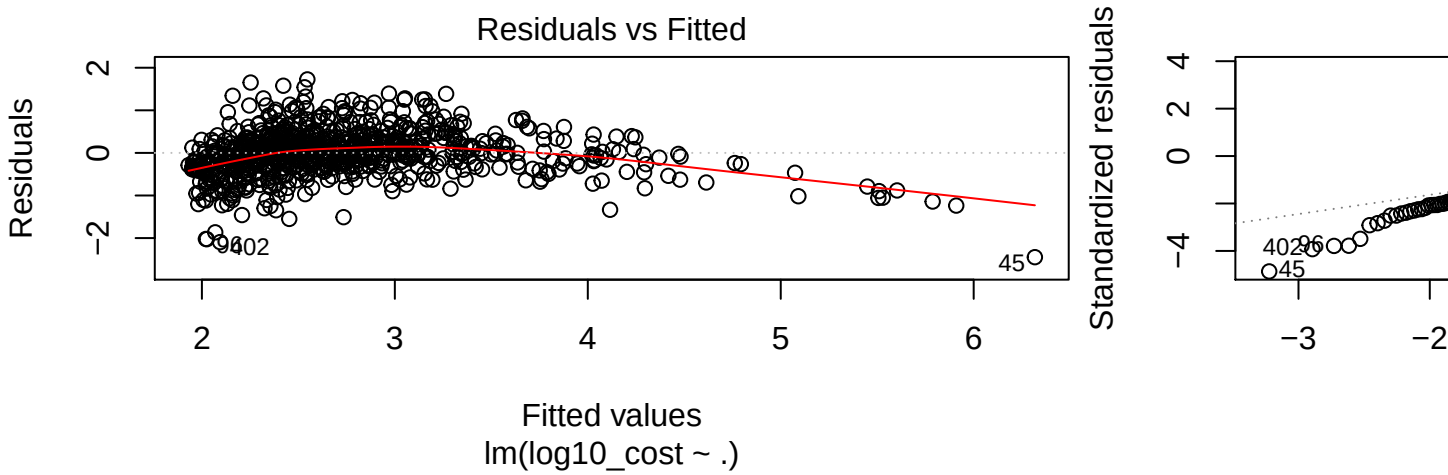


/2020−02−14

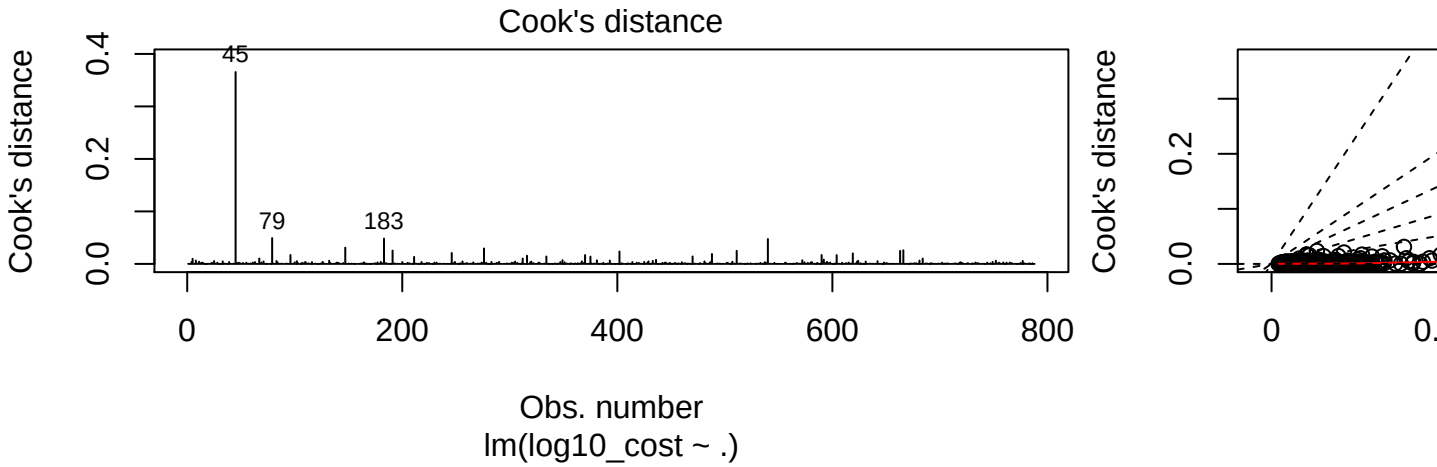
Checking for any correlation > .6 There doesn't appear to be any multicollinearity between our predictor variables

```
          age  gend intvn drugs ervis  comp comorb   dur
age      TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE
gend    FALSE  TRUE FALSE FALSE FALSE FALSE  FALSE FALSE
intvn   FALSE FALSE  TRUE FALSE FALSE FALSE  FALSE FALSE
drugs   FALSE FALSE FALSE  TRUE FALSE FALSE  FALSE FALSE
ervis   FALSE FALSE FALSE FALSE  TRUE FALSE  FALSE FALSE
comp    FALSE FALSE FALSE FALSE FALSE  TRUE  FALSE FALSE
comorb  FALSE FALSE FALSE FALSE FALSE FALSE   TRUE FALSE
dur     FALSE FALSE FALSE FALSE FALSE FALSE  FALSE  TRUE
```

I'll make a few plots to check other diagnostics on our model.
The below two graphs show us that our normality and homoscedasticity assumptions are somewhat satisfied



The two plots below show us that there are 3 observations (45, 79, and 183 which are outliers & infuential for our model. Might be a good idea to look at those more closely)



Diagnostics on our model are promising. However, the adjusted R-squared is 0.5789, which means there is a lot of variability in the data that our model is not capturing.

Let's try cross validation as a measure of the predictive power of our model. First I've standardized my data. Then I use the train() function in the caret package to run 10-fold cross, which I initialize 50 times. Note that I've set the intercept argument to FALSE, because we are working with standardized data.

```
Linear Regression

788 samples
  8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 50 times)
Summary of sample sizes: 710, 708, 711, 709, 709, 710, ...
Resampling results:

  RMSE       Rsquared   MAE
  0.6525244  0.5849761  0.4842181

Tuning parameter 'intercept' was held constant at a value of FALSE
```

The R-squared from CV for the linear model is 0.5849761 And this is the final model which produced the lowest RMSE / highest Rsquared

```
Call:
lm(formula = .outcome ~ 0 + ., data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.93887 -0.35127  0.02971  0.35554  2.10383

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
age    -0.03581    0.02350  -1.524   0.1279
gend   -0.06211    0.04869  -1.276   0.2025
intvn   0.59342    0.02573  23.063  < 2e-16 ***
```

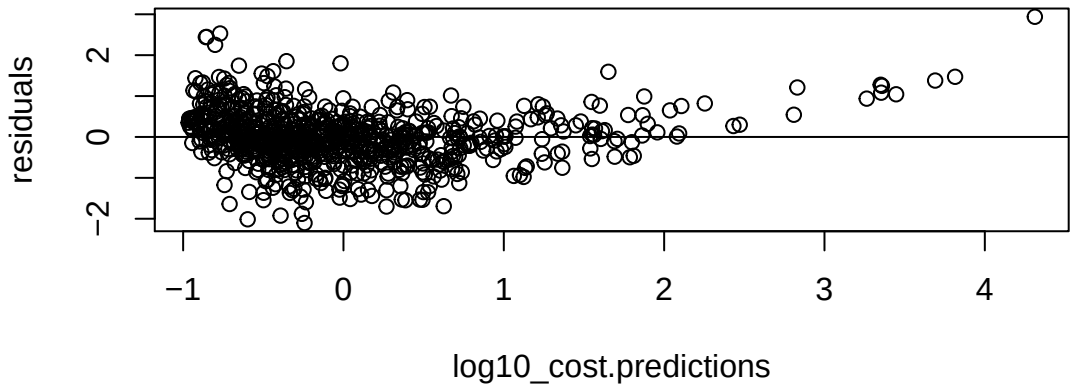
```
drugs  -0.03269    0.02745  -1.191   0.2340
ervis   0.07043    0.02881   2.445   0.0147 *
comp    0.09786    0.02380   4.112 4.33e-05 ***
comorb  0.16501    0.02686   6.144 1.28e-09 ***
dur     0.17755    0.02735   6.491 1.51e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6487 on 780 degrees of freedom
Multiple R-squared:  0.5829, Adjusted R-squared:  0.5786
F-statistic: 136.2 on 8 and 780 DF,  p-value: < 2.2e-16
```

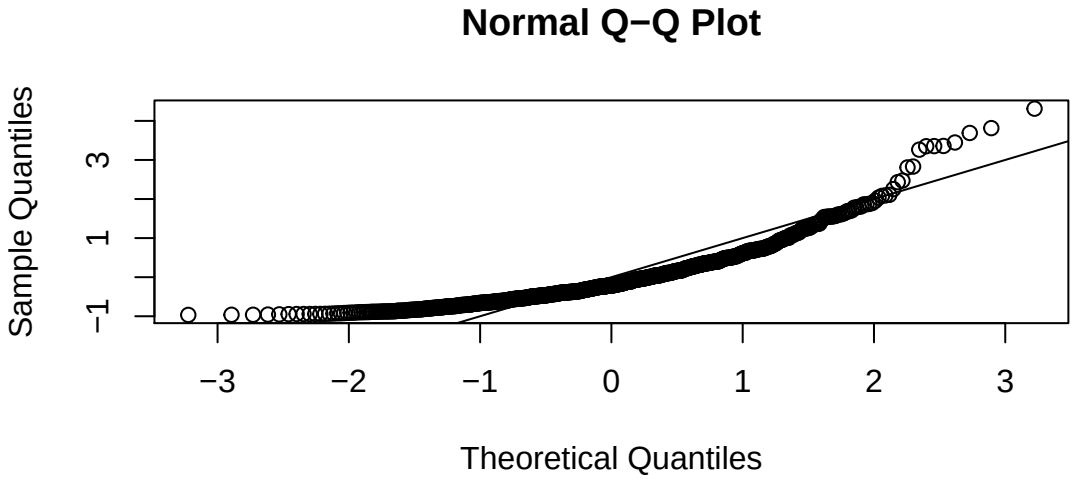
b)

Examining the above output, I see that: - number of interventions (intvn) has the highest standardized & significant coefficient.
Therefore, it is the variable with the most influence on the cost of the patient.
- After that, with a much lower coefficient, are # of chronic comorbid conditions (comorb) and duration of treatment (dur).
- Finally, the number of complications (comp) appears to have weight on the cost as well.
- Number of ER visits is somewhat significant, but it's coefficient is rather small.

c) - Diagnostics

The notes I made about the original model pretty much apply in the same way to our cross-validated model Heteroscedasticity appears to be an issue based on the residuals vs fitted values plot. The q-q plot shows that normality is not a good assumptions.





Most importantly, the adjusted R-squared never exceeds 60%. This probably means that our linear model does not correctly represent the real relationship between the variables. Let's try a Neural Net in the next Problem!

Problem 2

First, I set up tuning parameters for decay and size of hidden layer. I'm looking at combinations of size = 1-10 and decay = 0 - 5 (in increments of .5) In this way I will have to fit approximately 1000 neural nets.

```
   decay size
1   0.0    1
2   0.5    1
3   1.0    1
4   1.5    1
5   2.0    1
6   2.5    1
```

It turns out the best parameter is decay = 0 and size = 5.

```
Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
trainInfo, : There were missing values in resampled performance measures.

Warning in train.default(x, y, weights = w, ...): missing values found in
aggregated results

Multi-Layer Perceptron

788 samples
  8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 709, 710, 708, 708, 710, 709, ...
Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared   MAE
```

0.0	1	0.8105585	0.4924178	0.6264534
0.0	2	0.5808662	0.6752735	0.4404824
0.0	3	0.5769100	0.6801605	0.4412746
0.0	4	0.5846051	0.6825920	0.4387775
0.0	5	0.5761923	0.6899818	0.4311652
0.0	6	0.5871789	0.6787895	0.4401156
0.0	7	0.5913655	0.6741130	0.4490458
0.0	8	0.5774084	0.6899381	0.4362354
0.0	9	0.5989328	0.6832976	0.4534055
0.0	10	0.5906554	0.6718772	0.4440841
0.5	1	0.9993831	0.1857077	0.7954320
0.5	2	1.0062930	0.2293844	0.8020911
0.5	3	1.0025956	0.1394482	0.7995332
0.5	4	1.0297966	0.2166023	0.8219219
0.5	5	1.0287008	0.2066056	0.8180134
0.5	6	1.0504911	0.1395201	0.8465633
0.5	7	1.0839054	0.1620754	0.8641875
0.5	8	1.0698205	0.1616306	0.8621301
0.5	9	1.1053022	0.2246708	0.8903888
0.5	10	1.0855319	0.1772860	0.8703042
1.0	1	0.9996727	0.1816959	0.7952383
1.0	2	0.9991514	0.2077509	0.7969495
1.0	3	1.0039565	0.2081189	0.8023947
1.0	4	1.0158827	0.1880691	0.8060845
1.0	5	1.0310956	0.1981738	0.8319192
1.0	6	1.0719047	0.1352439	0.8604250
1.0	7	1.0326501	0.1520675	0.8207080
1.0	8	1.0609778	0.1188253	0.8461104
1.0	9	1.0916010	0.2148399	0.8736855
1.0	10	1.1690471	0.1847911	0.9520821
1.5	1	0.9988313	0.1413606	0.7951443
1.5	2	1.0051746	0.1909550	0.8002403
1.5	3	1.0158757	0.2077256	0.8155847
1.5	4	1.0391211	0.2973586	0.8291157
1.5	5	1.1147867	0.1497543	0.9085403
1.5	6	1.0745950	0.2972080	0.8633672
1.5	7	1.1658458	0.1923964	0.9530006
1.5	8	1.6874879	0.1628573	1.4594132
1.5	9	NaN	NaN	NaN
1.5	10	NaN	NaN	NaN
2.0	1	NaN	NaN	NaN
2.0	2	NaN	NaN	NaN
2.0	3	NaN	NaN	NaN
2.0	4	NaN	NaN	NaN
2.0	5	NaN	NaN	NaN
2.0	6	NaN	NaN	NaN
2.0	7	NaN	NaN	NaN
2.0	8	NaN	NaN	NaN
2.0	9	NaN	NaN	NaN
2.0	10	NaN	NaN	NaN
2.5	1	NaN	NaN	NaN
2.5	2	NaN	NaN	NaN
2.5	3	NaN	NaN	NaN
2.5	4	NaN	NaN	NaN

2.5	5	NaN	NaN	NaN
2.5	6	NaN	NaN	NaN
2.5	7	NaN	NaN	NaN
2.5	8	NaN	NaN	NaN
2.5	9	NaN	NaN	NaN
2.5	10	NaN	NaN	NaN
3.0	1	NaN	NaN	NaN
3.0	2	NaN	NaN	NaN
3.0	3	NaN	NaN	NaN
3.0	4	NaN	NaN	NaN
3.0	5	NaN	NaN	NaN
3.0	6	NaN	NaN	NaN
3.0	7	NaN	NaN	NaN
3.0	8	NaN	NaN	NaN
3.0	9	NaN	NaN	NaN
3.0	10	NaN	NaN	NaN
3.5	1	NaN	NaN	NaN
3.5	2	NaN	NaN	NaN
3.5	3	NaN	NaN	NaN
3.5	4	NaN	NaN	NaN
3.5	5	NaN	NaN	NaN
3.5	6	NaN	NaN	NaN
3.5	7	NaN	NaN	NaN
3.5	8	NaN	NaN	NaN
3.5	9	NaN	NaN	NaN
3.5	10	NaN	NaN	NaN
4.0	1	NaN	NaN	NaN
4.0	2	NaN	NaN	NaN
4.0	3	NaN	NaN	NaN
4.0	4	NaN	NaN	NaN
4.0	5	NaN	NaN	NaN
4.0	6	NaN	NaN	NaN
4.0	7	NaN	NaN	NaN
4.0	8	NaN	NaN	NaN
4.0	9	NaN	NaN	NaN
4.0	10	NaN	NaN	NaN
4.5	1	NaN	NaN	NaN
4.5	2	NaN	NaN	NaN
4.5	3	NaN	NaN	NaN
4.5	4	NaN	NaN	NaN
4.5	5	NaN	NaN	NaN
4.5	6	NaN	NaN	NaN
4.5	7	NaN	NaN	NaN
4.5	8	NaN	NaN	NaN
4.5	9	NaN	NaN	NaN
4.5	10	NaN	NaN	NaN
5.0	1	NaN	NaN	NaN
5.0	2	NaN	NaN	NaN
5.0	3	NaN	NaN	NaN
5.0	4	NaN	NaN	NaN
5.0	5	NaN	NaN	NaN
5.0	6	NaN	NaN	NaN
5.0	7	NaN	NaN	NaN
5.0	8	NaN	NaN	NaN

```
  5.0     9          NaN        NaN        NaN
  5.0    10          NaN        NaN        NaN

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 5 and decay = 0.
```

I will rerun with sizes 3-7 and decay 0-1 in .1 increments. Overall, this fits another 500 neural nets. It turns out that decay = 0 and size = 6 are best.

```
Multi-Layer Perceptron

788 samples
  8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 709, 709, 710, 710, 711, 709, ...
Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared    MAE
  0.0    3     0.5849132  0.68472085  0.4404708
  0.0    4     0.5768466  0.67846022  0.4351733
  0.0    5     0.5782513  0.67634261  0.4298722
  0.0    6     0.5794947  0.68426065  0.4381552
  0.0    7     0.5815350  0.66929941  0.4306910
  0.1    3     1.0260177  0.23587069  0.8221234
  0.1    4     1.0195718  0.37979585  0.8165636
  0.1    5     1.0734206  0.26080410  0.8535168
  0.1    6     1.0433394  0.18369787  0.8386110
  0.1    7     1.0465640  0.21890115  0.8332339
  0.2    3     1.0170988  0.23444611  0.8102142
  0.2    4     1.0141056  0.24401526  0.8127645
  0.2    5     1.0368460  0.23503329  0.8340846
  0.2    6     1.0305206  0.18951292  0.8181694
  0.2    7     1.0485089  0.18319477  0.8401089
  0.3    3     1.0070213  0.17236283  0.7980931
  0.3    4     1.0177729  0.17413763  0.8121181
  0.3    5     1.0462266  0.15916684  0.8266636
  0.3    6     1.0214618  0.11215058  0.8154598
  0.3    7     1.0481248  0.18591598  0.8379640
  0.4    3     1.0141547  0.12224737  0.8110912
  0.4    4     1.0156901  0.19063943  0.8142186
  0.4    5     1.0061627  0.12192822  0.7998721
  0.4    6     1.0240294  0.15334966  0.8187304
  0.4    7     1.0553418  0.17496354  0.8504499
  0.5    3     1.0156129  0.14618132  0.8147917
  0.5    4     1.0148980  0.10872365  0.8067633
  0.5    5     1.0188319  0.16430154  0.8077896
  0.5    6     1.0760049  0.14634932  0.8656043
  0.5    7     1.0534859  0.12910019  0.8460941
  0.6    3     1.0113962  0.26278183  0.8085673
  0.6    4     1.0100713  0.25920087  0.8068957
  0.6    5     1.0151973  0.14751985  0.8100910
  0.6    6     1.0223025  0.28301562  0.8093287
```

```
  0.6    7     1.0709915  0.14524499  0.8513348
  0.7    3     1.0111874  0.24182421  0.8057600
  0.7    4     1.0122557  0.12103743  0.8069013
  0.7    5     1.0157660  0.15991439  0.8051777
  0.7    6     1.0390904  0.17873232  0.8332810
  0.7    7     1.0630995  0.26580959  0.8442040
  0.8    3     0.9995773  0.14876027  0.7962432
  0.8    4     1.0242787  0.26544934  0.8151238
  0.8    5     1.0149396  0.23021261  0.8082441
  0.8    6     1.0392670  0.28150204  0.8309758
  0.8    7     1.0659572  0.15582946  0.8561687
  0.9    3     1.0093988  0.22381877  0.8043317
  0.9    4     1.0203700  0.22158128  0.8191685
  0.9    5     1.0188352  0.21010136  0.8138502
  0.9    6     1.0337347  0.15548889  0.8271148
  0.9    7     1.0255970  0.12380710  0.8172755
  1.0    3     1.0077369  0.09910975  0.8060565
  1.0    4     1.0104517  0.17749732  0.8023532
  1.0    5     1.0271724  0.20244987  0.8197068
  1.0    6     1.0214673  0.10101321  0.8197106
  1.0    7     1.0502990  0.12975675  0.8339320

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 4 and decay = 0.
```

I will limit decay to >0, between .01 and .5 in .01 increments. I'll test size 5 and 6. Overall, this fits another 1000 neural nets. The best combination turns out to be size = 6 and decay = .01

```
Multi-Layer Perceptron

788 samples
  8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 710, 710, 710, 710, 709, 709, ...
Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared    MAE
  0.01   5     0.7222593  0.57169637  0.5513367
  0.01   6     0.7502461  0.57618924  0.5846992
  0.02   5     0.8202775  0.55873995  0.6465376
  0.02   6     0.8399601  0.54470081  0.6624144
  0.03   5     0.8838612  0.51916929  0.7027528
  0.03   6     0.9061509  0.53980511  0.7291126
  0.04   5     1.0756001  0.38019674  0.8607515
  0.04   6     1.0371339  0.48800770  0.8288473
  0.05   5     1.0554639  0.23744312  0.8438385
  0.05   6     1.0374061  0.24330703  0.8272987
  0.06   5     1.0765745  0.21927839  0.8653583
  0.06   6     1.0437617  0.21311973  0.8327170
  0.07   5     1.0718311  0.26312226  0.8554948
  0.07   6     1.0938589  0.25375325  0.8904048
  0.08   5     1.0321892  0.32367443  0.8295449
```

0.08	6	1.1438035	0.24391370	0.9267698
0.09	5	1.0542462	0.25843405	0.8386049
0.09	6	1.0923277	0.15276063	0.8848889
0.10	5	1.0776126	0.18922517	0.8586316
0.10	6	1.0602172	0.23476607	0.8504505
0.11	5	1.0555203	0.34766682	0.8522993
0.11	6	1.1091175	0.26892885	0.8870906
0.12	5	1.0178144	0.30385517	0.8158307
0.12	6	1.1129284	0.12258990	0.9006861
0.13	5	1.0254249	0.16879654	0.8149260
0.13	6	1.0598766	0.23984371	0.8445152
0.14	5	1.0495940	0.25177513	0.8501401
0.14	6	1.0301237	0.30319535	0.8276102
0.15	5	1.0420356	0.22537421	0.8458199
0.15	6	1.0128566	0.23399775	0.8094561
0.16	5	1.0400200	0.13624626	0.8300519
0.16	6	1.0515138	0.18038757	0.8311006
0.17	5	1.0244655	0.19581346	0.8198505
0.17	6	1.1093189	0.20933430	0.8834764
0.18	5	1.0460706	0.21563730	0.8300628
0.18	6	1.0635010	0.20320252	0.8573544
0.19	5	1.0468133	0.20172491	0.8364529
0.19	6	1.0270712	0.21643164	0.8188108
0.20	5	1.0164651	0.29023558	0.8187550
0.20	6	1.0457875	0.23568588	0.8366176
0.21	5	1.0216340	0.21937975	0.8122568
0.21	6	1.0210029	0.18273282	0.8066800
0.22	5	1.0684750	0.27640445	0.8615301
0.22	6	1.0713984	0.13584963	0.8501761
0.23	5	1.0327849	0.18330055	0.8193444
0.23	6	1.0293250	0.25669674	0.8171516
0.24	5	1.0278670	0.13762024	0.8266176
0.24	6	1.0281555	0.19770468	0.8193255
0.25	5	1.0164529	0.20030657	0.8168252
0.25	6	1.0591796	0.21468560	0.8442834
0.26	5	1.0350987	0.17026417	0.8235901
0.26	6	1.0468393	0.36179575	0.8336712
0.27	5	1.0555713	0.16820294	0.8470922
0.27	6	1.0137739	0.22535878	0.8092977
0.28	5	1.0232117	0.16773615	0.8223153
0.28	6	1.0680092	0.16812922	0.8680306
0.29	5	1.0208574	0.17092076	0.8129526
0.29	6	1.0578348	0.20894577	0.8504511
0.30	5	1.0390986	0.25004467	0.8384522
0.30	6	1.0593482	0.23179061	0.8546471
0.31	5	1.0137099	0.21382948	0.8042170
0.31	6	1.0505711	0.18034841	0.8406724
0.32	5	1.0338515	0.14400622	0.8307035
0.32	6	1.0249256	0.23952141	0.8156123
0.33	5	1.0333765	0.30681069	0.8179178
0.33	6	1.0406755	0.19982810	0.8388564
0.34	5	1.0556016	0.21020531	0.8396735
0.34	6	1.0221421	0.18469943	0.8151619
0.35	5	1.0254874	0.29675942	0.8184857

```
 0.35   6     1.0430456  0.15178368  0.8237571
 0.36   5     1.0373982  0.20183884  0.8331961
 0.36   6     1.0463642  0.18134175  0.8429084
 0.37   5     1.0292275  0.20262657  0.8219723
 0.37   6     1.0298358  0.27229896  0.8193871
 0.38   5     1.0157641  0.23429413  0.8211935
 0.38   6     1.0336719  0.24979948  0.8189727
 0.39   5     1.0328011  0.19492949  0.8260188
 0.39   6     1.1145898  0.26297999  0.9067568
 0.40   5     1.0526395  0.13196428  0.8465906
 0.40   6     1.0082386  0.05849345  0.8093004
 0.41   5     1.0148279  0.13970315  0.8078534
 0.41   6     1.0279874  0.11297568  0.8271507
 0.42   5     1.0435136  0.28377301  0.8409552
 0.42   6     1.0205345  0.15673138  0.8145409
 0.43   5     1.0219446  0.12043549  0.8161525
 0.43   6     1.0362184  0.20077401  0.8250416
 0.44   5     1.0285241  0.16147903  0.8240413
 0.44   6     1.0347764  0.14543294  0.8296060
 0.45   5     1.0206866  0.08856490  0.8137206
 0.45   6     1.0200279  0.13614911  0.8175037
 0.46   5     1.0176875  0.25418277  0.8137012
 0.46   6     1.0796338  0.19951882  0.8654624
 0.47   5     1.0096114  0.07836369  0.8033884
 0.47   6     1.0507772  0.20555624  0.8468764
 0.48   5     1.0247663  0.15358847  0.8213240
 0.48   6     1.0591044  0.25469179  0.8530981
 0.49   5     1.0068822  0.18939655  0.8045366
 0.49   6     1.0396589  0.18953471  0.8349590
 0.50   5     1.0646692  0.26214045  0.8431043
 0.50   6     1.0186148  0.25100333  0.8122328

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 5 and decay = 0.01.
```

The best Rsquared = 0.5730521 Note that this is with Shrinkage parameter = .01. I forced it to be >0. If we were to leave it =0, then the best Rsquared would be ∼.68 (as seen from the output of the previous to exploratorations with different gridParameters) Now I will fit the final model.

```
a 8-6-1 network with 61 weights
options were - linear output units  decay=0.01
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1
  1.84   0.28  -0.36   5.79   0.54  -1.78   1.86   1.28  -0.77
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2
  3.71  -0.03   0.52  -1.99   0.42  -1.24  -0.23   0.33   0.98
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3
 -5.69   0.44  -1.28   0.28  -3.64   0.40  -1.00  -2.82   0.08
 b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4 i7->h4 i8->h4
 -0.56   0.39   0.59  -2.74   1.40  -4.14  -0.96  -0.47  -0.24
 b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5 i7->h5 i8->h5
  9.75   0.33   0.14   1.47   2.49  -4.50  -2.21  -0.21   5.06
 b->h6 i1->h6 i2->h6 i3->h6 i4->h6 i5->h6 i6->h6 i7->h6 i8->h6
  6.03  -0.09  -1.00   4.77  -1.23   1.16   0.95   2.55   1.09
 b->o h1->o h2->o h3->o h4->o h5->o h6->o
```

```
-0.65  0.79 -1.16 -3.66 -0.57  0.82  1.15
```

This is the SSE for our Neural Network

```
[1] 201.6785
```

This is the SSE from Cross-Validation for our Linear model.

```
[1] 328.2751
```

```
[1] 328.2751
```

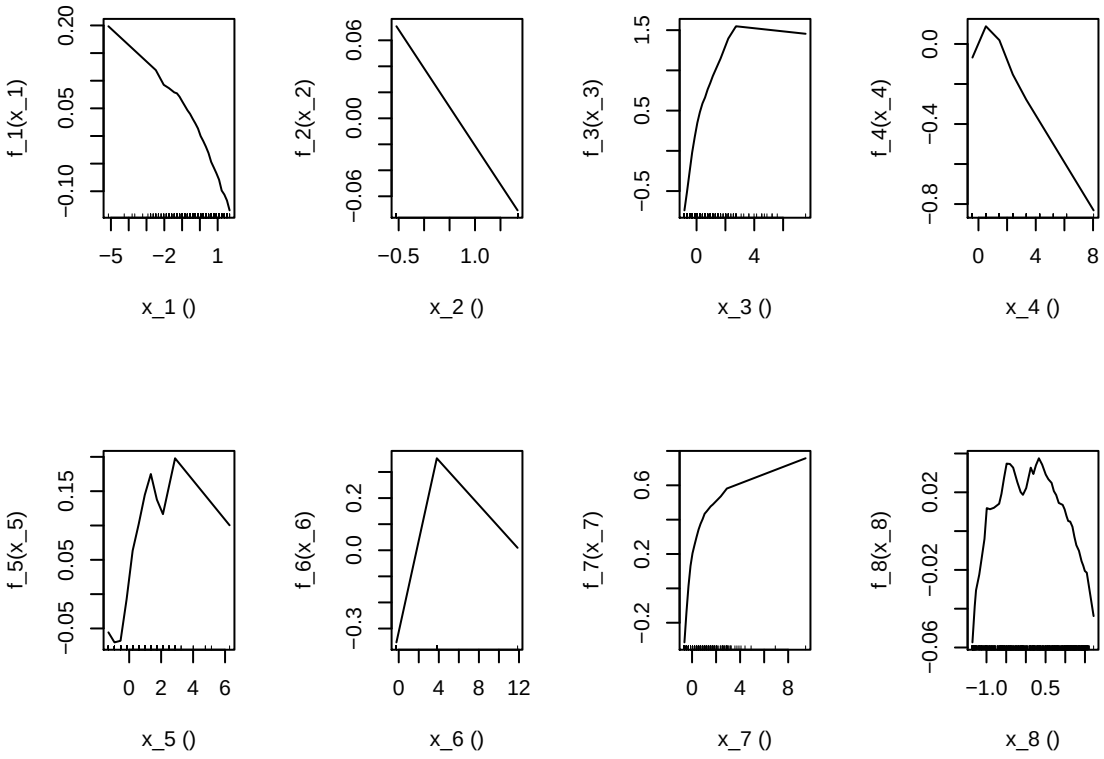
Below I have calculated the SSE just from the training data. If we were to naively look at only that, we might assume that there isn't much of a difference between the linear model and the neural network. But the fact that the SSE-CV is so much higher, clearly shows us that our linear model is somewhat overfitting the data.

```
[1] 224.8786
```

In any case, the Neural Net clearly shows a lower SSE than the linear model. However, the best R-squared from CV was 0.5730521. For the Linear model it was 0.5849761. This is probably due to the minor shrinkage I added for the neural net. Overall, I wouldn't expect the neural net to produce much better results on new data as opposed to the linear model.

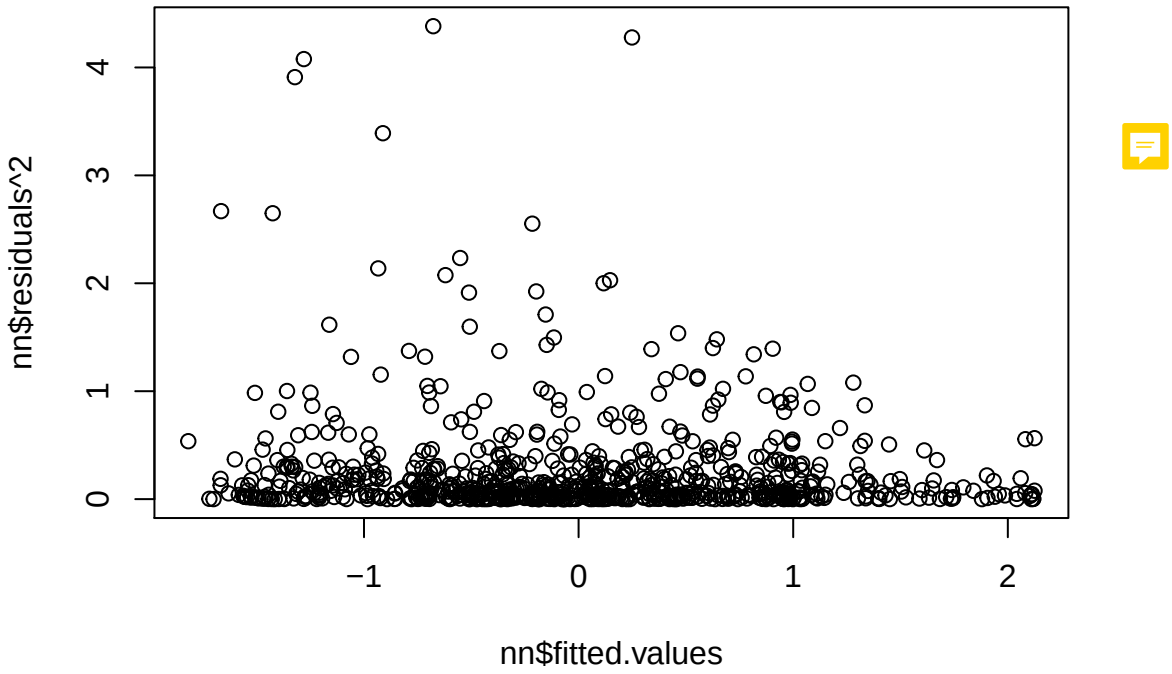
c) - Looking at ALE plots

The Main effect ALE plots below show us that there are still some non-linearities that our model has not captured. More specifically in x3, x6, x7, x8 (intvn, comp, comorb, dur). Interestingly enough, these are precisely the variables, which we saw were significant for the linear model.



Looking at the Main Effect ALE Plots (more specifically the scales on the y axis) I observe that: - 1) Age appears to be linear and decreasing, but the scale of the output is fairly small, so I don't expect much influence there. - 2) Gender (which is binary 0/1) appears to have very little influence on the output (the y axis has very low values) - 3) The number of interventions appears to have a much larger impact on the log10_cost. And it is also in the direction we would expect: more interventions, leads to higher cost. - 4) # of Drugs doesn't appear to have much influence, and whatever influence it has seems a bit counterintuitive - more drugs lower cost. In any case, the Linear model did suggest that drugs is not a signicant variable. - 5) # of ER visits has a small scale on the y axis and just like the linear model, doesn't suggest much impact on the cost - 6) is number of complications. Only one patient has 3, 42 have 1, and the rest have 0. Therefore, the graph is not particularly useful, but we do see that patients with 1 or 3 complications have a slightly higher cost than those who don't. - 7) # of comorbid conditions appears to have a significant increasing influence on the cost, similar as the one suggested by the Linear Model - 8) The duration has an odd ALE plot. In any case, the scale is fairly small which is quite different from what our linear model suggested.

d) Residual Plots



I think our model doesn't perform particularly well for some of the outliers and therefore there might be still some non-linearity not captured by it. That's why it's a good idea to try fitting a regression tree!

Problem 3

With Regression Trees, I don't have to standardize the data, so I will work with the original dataset.

```
# A tibble: 6 x 9
    age  gend intvn drugs ervis  comp comorb   dur log10_cost
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>      <dbl>
1    63     0     2     1     4     0      3   300       2.25
2    59     0     2     0     6     0      0   120       2.50
3    62     0    17     0     2     0      5   353       3.97
4    60     1     9     0     7     0      2   332       2.45
```

```
5    55     0     5     2     7     0      0    18       4.27
6    66     0     1     0     3     0      4   296       2.66
```

a) - Using 10-fold CV to find best tree size and complexity parameter

First I define a range of values for the GridSearch I will iterate from 1 to 30 for maxdepth and between .01 and 1 in increments of .01 for cp Overall, this means 3000 Decision Trees.

```
  maxdepth   cp
1        1 0.01
2        2 0.01
3        3 0.01
4        4 0.01
5        5 0.01
6        6 0.01
```

Now I fit 3000 models with 10-fold cross-validation, which will mean 30,000 regression trees.

Here are some of the avgRMSE I got.

```
[1] 0.6360727 0.5494297 0.5314287 0.5145378
```

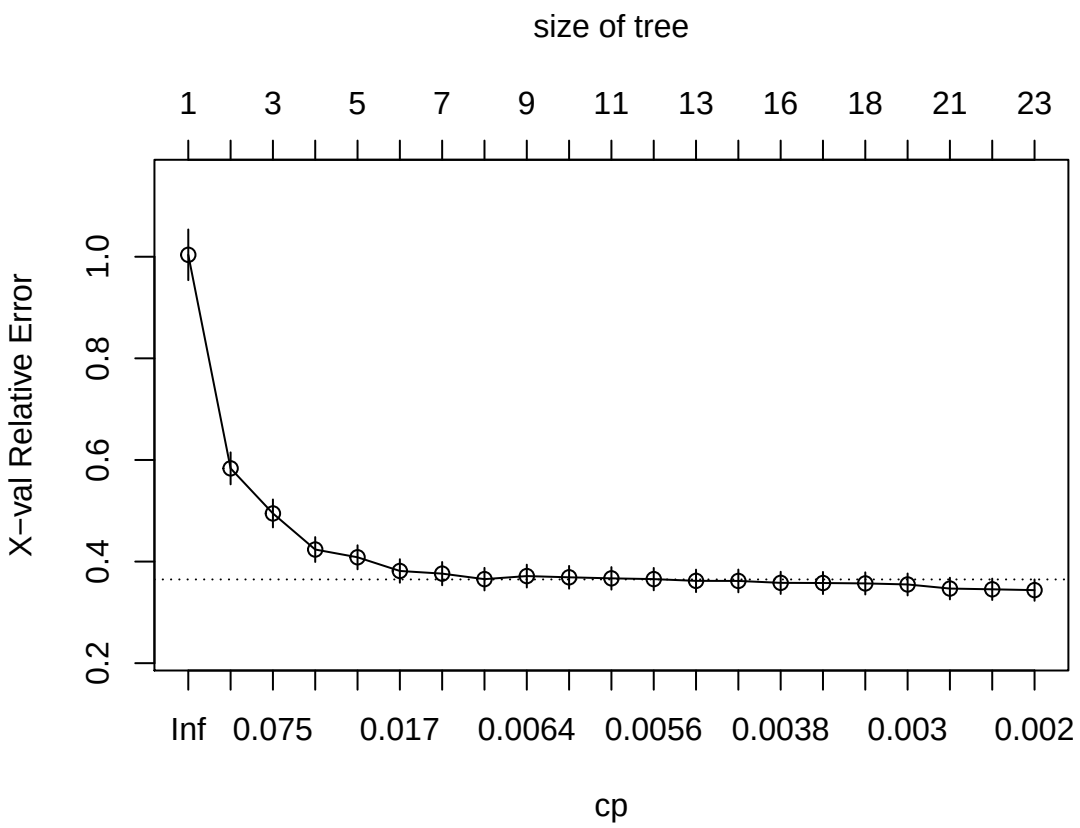
In turns out that the minimum CV-RMSE is achieved with cp = .01 and maxdepth anywhere between 4 and 30. I will try with a lower cp range and fit another 30,000 Decision Trees.

These turns out to be the best parameters for our Decision Tree!

```
   maxdepth    cp
37        7 0.002
```

b) Now I fit the final model

The plotcp() & printcp() outputs confirm that the best value is with cp = 0.0066471 and nsplit = 7



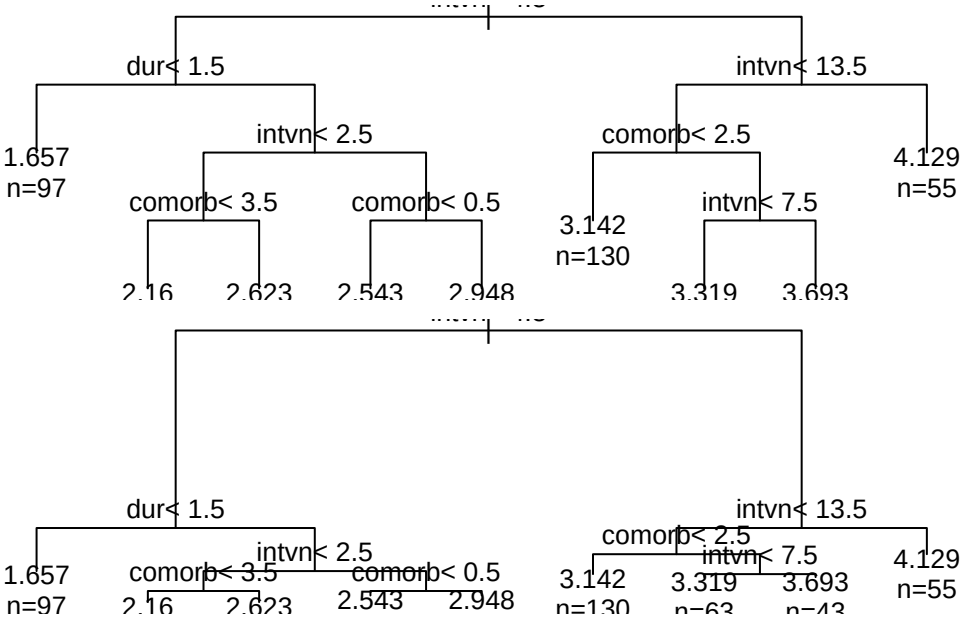

```
Regression tree:
rpart(formula = log10_cost ~ ., data = heart.fit1.data, method = "anova",
    cp = 0.002, maxdepth = 7)

Variables actually used in tree construction:
[1] age    comorb comp   dur    ervis  intvn

Root node error: 539.46/788 = 0.68459

n= 788

          CP nsplit rel error  xerror     xstd
1  0.4393807      0   1.00000 1.00377 0.049863
2  0.0958190      1   0.56062 0.58346 0.031480
3  0.0582852      2   0.46480 0.49472 0.027504
4  0.0307018      3   0.40652 0.42369 0.024485
5  0.0247526      4   0.37581 0.40842 0.023654
6  0.0116665      5   0.35106 0.38148 0.023144
7  0.0099124      6   0.33939 0.37624 0.022881
8  0.0066471      7   0.32948 0.36540 0.022193
9  0.0062292      8   0.32283 0.37140 0.022482
10 0.0060774      9   0.31661 0.36900 0.022404
11 0.0057761     10   0.31053 0.36694 0.022130
12 0.0054328     11   0.30475 0.36544 0.022000
13 0.0046859     12   0.29932 0.36201 0.022126
14 0.0040909     14   0.28995 0.36189 0.022422
```

```
15 0.0035193     15   0.28586 0.35809 0.021837
16 0.0032594     16   0.28234 0.35779 0.021817
17 0.0031408     17   0.27908 0.35700 0.021788
18 0.0029262     19   0.27280 0.35492 0.021589
19 0.0021852     20   0.26987 0.34684 0.021312
20 0.0020199     21   0.26768 0.34538 0.021033
21 0.0020000     22   0.26566 0.34379 0.021000
```

Now I prune the tree and get this final tree model:

```
n= 788

node), split, n, deviance, yval
      * denotes terminal node

 1) root 788 539.458700 2.731718
   2) intvn< 4.5 497 202.242000 2.312051
     4) dur< 1.5 97  29.414700 1.657157 *
     5) dur>=1.5 400 121.136900 2.470863
      10) intvn< 2.5 269  70.521240 2.328862
        20) comorb< 3.5 171  48.726830 2.160195 *
        21) comorb>=3.5 98   8.441384 2.623168 *
      11) intvn>=2.5 131  34.053310 2.762452
        22) comorb< 0.5 60  15.326920 2.542673 *
        23) comorb>=0.5 71  13.379090 2.948180 *
   3) intvn>=4.5 291 100.188900 3.448469
     6) intvn< 13.5 236  59.599750 3.289783
      12) comorb< 2.5 130  34.606980 3.142323 *
      13) comorb>=2.5 106  18.699190 3.470630
        26) intvn< 7.5 63   8.978707 3.318678 *
        27) intvn>=7.5 43   6.134632 3.693258 *
     7) intvn>=13.5 55   9.146718 4.129374 *
```

Now I plot the tree both uniformly and non-uniformly. Note that the root node split is on interventions.



Finally, I will evaluate the tree's predictive power with SSE, the same way as with the Linear model and the Neural Network

```
[1] 174.156
```

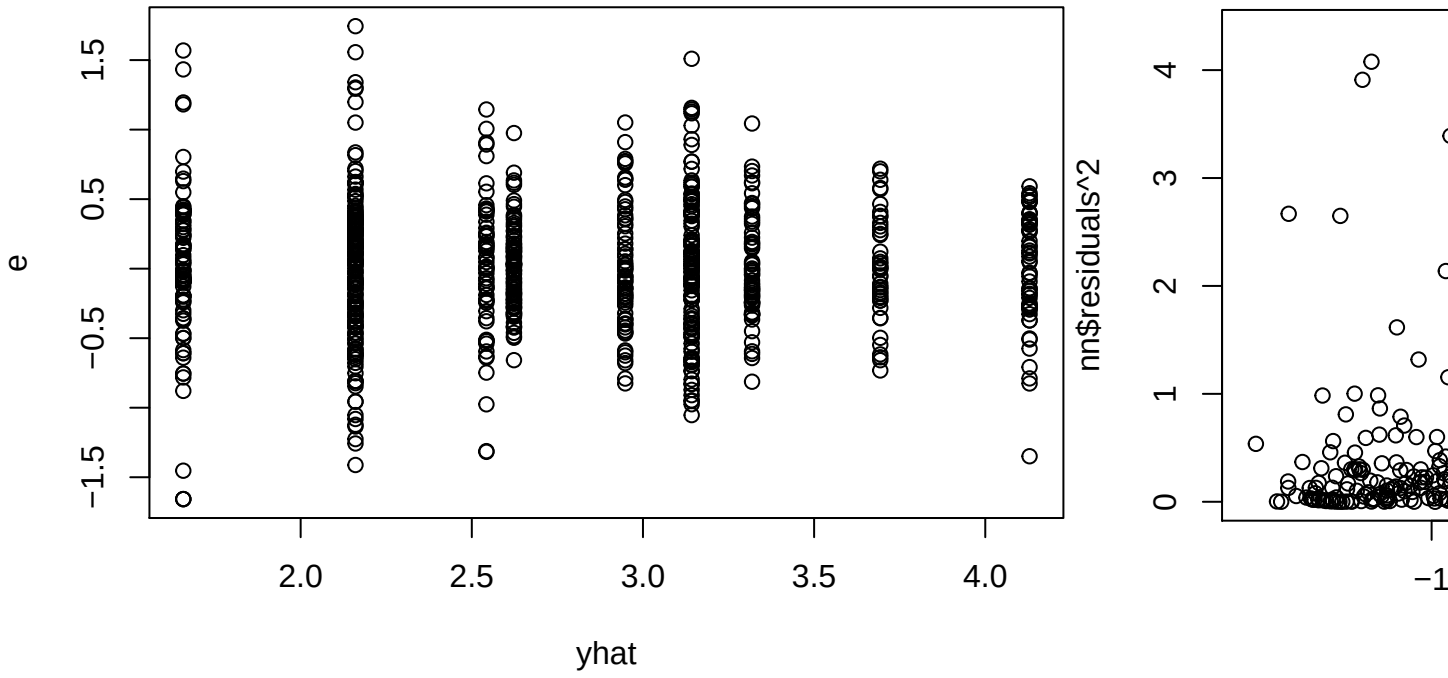
The Regression Tree definitely appears to be doing better than the Linear model & the NN.

c) - Variable Importance

Just like the Linear Model & NN, it gives the most importance to number of interventions, duration, and # of comorbid conditions.

```
      intvn         dur      comorb       ervis        comp       drugs
288.677781   68.505218   40.553342   35.244579   13.239191   13.173895
         age
  1.718478
```

d) - Residuals plot



e)

In the end, it appears that after extensive Cross-Validation, the Regression Tree achieved lowest CV-SSE. Furthermore, it took a lot less time to train (I trained ~ 2,500 Neural Nets and that took much longer than ~60,000 Trees) The residual plot also suggests that the model is better capable of capturing the variability in the data. So, I would recommend the tree model (especially because it's the most interpretable as well).

Problem 4

Importing Data

```
Attaching package: 'MASS'

The following object is masked from 'package:dplyr':

    select

New names:
* `` -> ...1

Classes 'tbl_df', 'tbl' and 'data.frame':    214 obs. of  10 variables:
 $ RI  : num  3.01 -0.39 -1.82 -0.34 -0.58 ...
 $ Na  : num  13.6 13.9 13.5 13.2 13.3 ...
 $ Mg  : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
 $ Al  : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
 $ Si  : num  71.8 72.7 73 72.6 73.1 ...
 $ K   : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
 $ Ca  : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
 $ Ba  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Fe  : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
 $ type: chr  "WinF" "WinF" "WinF" "WinF" ...
```

a) - Fitting a Neural Net

I need to create a STANDARDIZED VERSION of the data

```
'data.frame':    214 obs. of  10 variables:
 $ RI  : num  0.871 -0.249 -0.72 -0.232 -0.311 ...
 $ Na  : num  0.284 0.59 0.15 -0.242 -0.169 ...
 $ Mg  : num  1.252 0.635 0.6 0.697 0.649 ...
 $ Al  : num  -0.691 -0.17 0.19 -0.31 -0.41 ...
 $ Si  : num  -1.1244 0.1021 0.4378 -0.0528 0.554 ...
 $ K   : num  -0.6701 -0.0262 -0.1641 0.1118 0.0812 ...
 $ Ca  : num  -0.145 -0.792 -0.827 -0.518 -0.623 ...
 $ Ba  : num  -0.352 -0.352 -0.352 -0.352 -0.352 ...
 $ Fe  : num  -0.585 -0.585 -0.585 -0.585 -0.585 ...
 $ type: Factor w/ 6 levels "Con","Head","Tabl",..: 5 5 5 5 5 5 5 5 5 5 ...
```

Now, I will do 10-fold Cross Validation, with 10 different randomized splits of the data i.e. Nrep = 10 I will be comparing 3 models at a time. More specifically, 3 different Neural Networks with different hyperparameters First, I try: linout = F, size = 10, decay = .01 linout = F, size = 10, decay = .05 linout = F, size = 10, decay = .1

```
[[1]]
           [,1]      [,2]      [,3]
 [1,] 0.3271028 0.2757009 0.2850467
 [2,] 0.2803738 0.2943925 0.3130841
```

```
 [3,] 0.3411215 0.3130841 0.2803738
 [4,] 0.2990654 0.2803738 0.3084112
 [5,] 0.3364486 0.2710280 0.2850467
 [6,] 0.2757009 0.2710280 0.2850467
 [7,] 0.3130841 0.2570093 0.2663551
 [8,] 0.3411215 0.2990654 0.2990654
 [9,] 0.2943925 0.2850467 0.3037383
[10,] 0.3130841 0.2616822 0.3037383

[[2]]
[1] 0.3121495 0.2808411 0.2929907
```

decay = .05 gives the lowest CV-Misclassification Rate. Now I will try with values closer to .05. After several repetitions, which I will not display here for brevity, I find that the optimal decay is .04 I observe that on subsequent iterations the same inputs would lead to a different conclusion. Therefore, I'm in the realm of uncertainty.

```
[[1]]
          [,1]      [,2]      [,3]
 [1,] 0.2850467 0.2990654 0.3084112
 [2,] 0.2757009 0.2710280 0.3177570
 [3,] 0.2710280 0.2943925 0.2616822
 [4,] 0.2990654 0.3037383 0.2897196
 [5,] 0.2803738 0.3130841 0.2803738
 [6,] 0.2897196 0.2990654 0.2663551
 [7,] 0.2943925 0.2710280 0.3037383
 [8,] 0.2803738 0.2943925 0.2803738
 [9,] 0.2943925 0.2710280 0.3130841
[10,] 0.2943925 0.2990654 0.2803738

[[2]]
[1] 0.2864486 0.2915888 0.2901869
```

What about the size of the hidden layer? I will try different values: (7,10,12), (5,12,15), (8,15,20), (12,20,30), (30,40,50), (25,30,35), (25,27,33), (23,25,26) The values above are my progression in hunting down the best value. It turns out the best value is: 25

```
[[1]]
          [,1]      [,2]      [,3]
 [1,] 0.2803738 0.2663551 0.2570093
 [2,] 0.2897196 0.2757009 0.2570093
 [3,] 0.2757009 0.2570093 0.2570093
 [4,] 0.2943925 0.2990654 0.2943925
 [5,] 0.2803738 0.2570093 0.2803738
 [6,] 0.2990654 0.2663551 0.2663551
 [7,] 0.2850467 0.2710280 0.2523364
 [8,] 0.2523364 0.2757009 0.2803738
 [9,] 0.2710280 0.2663551 0.2476636
[10,] 0.2663551 0.2663551 0.2663551

[[2]]
[1] 0.2794393 0.2700935 0.2658879
```

Finally, I will check if a linear output or logistic is better

```
[[1]]
           [,1]      [,2]      [,3]
 [1,] 0.2616822 0.2523364 0.2663551
 [2,] 0.2803738 0.2523364 0.2476636
 [3,] 0.2663551 0.2757009 0.2710280
 [4,] 0.2570093 0.2663551 0.2616822
 [5,] 0.2757009 0.2757009 0.2943925
 [6,] 0.2476636 0.2757009 0.2570093
 [7,] 0.2429907 0.2476636 0.2476636
 [8,] 0.2523364 0.2663551 0.2570093
 [9,] 0.2476636 0.2616822 0.2803738
[10,] 0.2429907 0.2476636 0.2710280

[[2]]
[1] 0.2574766 0.2621495 0.2654206
```

Linear Output actually gives slightly better results!

To summarize, the best NN is with decay = .04, size = 25, linout = T. I'll fit this best NN and make predictions with it on the training data

```
[1] 0.01869159
```

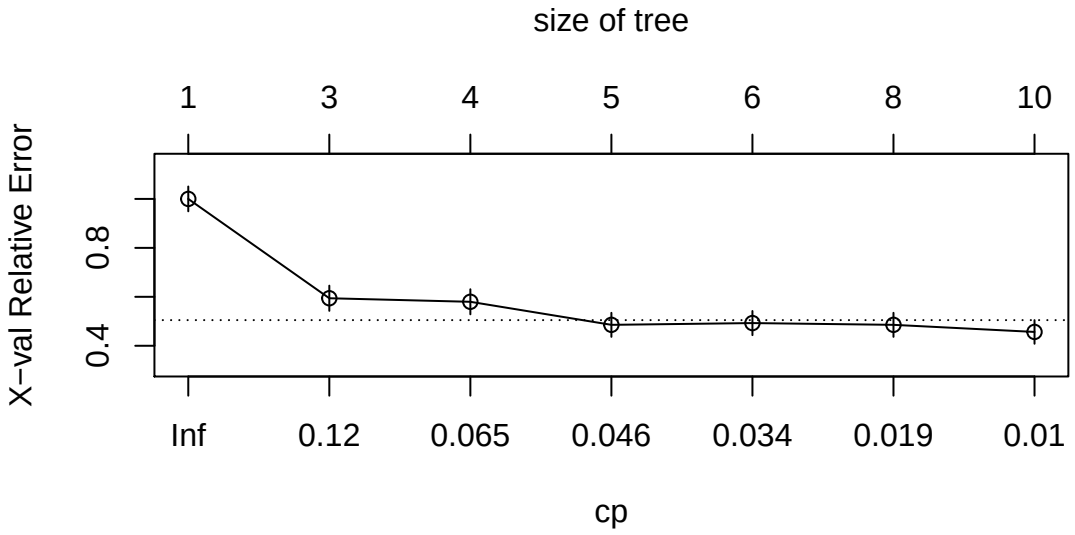
```
       y
yhat    Con Head Tabl Veh WinF WinNF
  Con    13    0    0   0    0     0
  Head    0   29    0   0    0     0
  Tabl    0    0    9   0    0     0
  Veh     0    0    0  17    1     1
  WinF    0    0    0   0   68     1
  WinNF   0    0    0   0    1    74
```

The NN appears to be doing very well on the training data, but this shouldn't fool us. The cross validation results showed us that the best the NN can do witn new data is around 26.8% misclassification rate.

b) Fitting a Classification Tree

```
New names:
* `` -> ...1
```

The best cp appears to be at .046 judging by the graph below

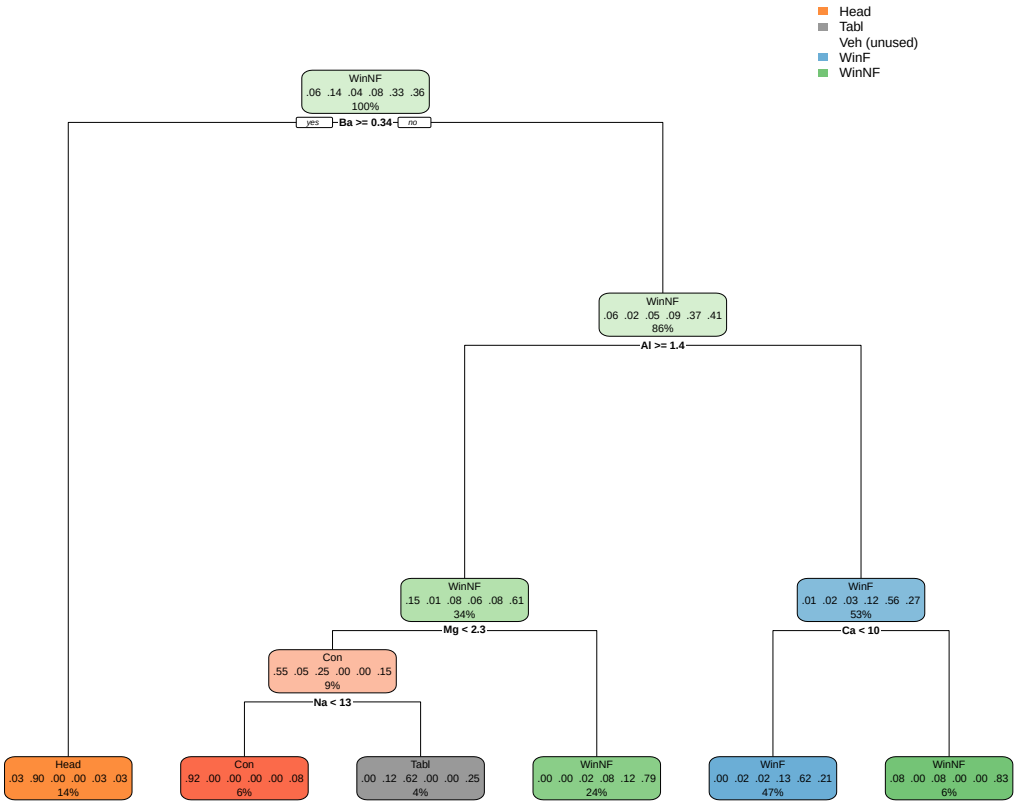


Output of printcp()

```
Classification tree:
rpart(formula = type ~ ., data = fgl, method = "class", xval = 10)

Variables actually used in tree construction:
[1] Al Ba Ca Fe Mg Na RI

Root node error: 138/214 = 0.64486

n= 214

        CP nsplit rel error  xerror     xstd
1 0.206522      0   1.00000 1.00000 0.050729
2 0.072464      2   0.58696 0.59420 0.051536
3 0.057971      3   0.51449 0.57971 0.051287
4 0.036232      4   0.45652 0.48551 0.049160
5 0.032609      5   0.42029 0.49275 0.049357
6 0.010870      7   0.35507 0.48551 0.049160
7 0.010000      9   0.33333 0.45652 0.048314
```

From the graph and the table, it appears that the best cp is .03623 with 4 splits. I'll now use this value to prune back the tree

```
       Al        Ca        Ba        Mg        RI        Na         K
30.321865 28.644993 26.044912 26.004171 17.935135 17.014265 14.267466
       Si
 6.394742
```



Will calculate a classification error for the tree on its training data

```
[1] 0.271028
```

Here's a confusion matrix for the tree predictions

```
        y
yhat     Con Head Tabl Veh WinF WinNF
  Con     11    0    0   0    0     1
  Head     1   26    0   0    1     1
  Tabl     0    1    5   0    0     2
  WinF     0    2    2  13   63    21
  WinNF    1    0    2   4    6    51
```

c) - Logistic Regression

First I fit a logistic Regression. A preliminary step I've taken is to factor the response variable type

```
New names:
* `` -> ...1

# weights:  66 (50 variable)
initial  value 383.436526
iter  10 value 248.061077
iter  20 value 173.867095
iter  30 value 144.744771
```

```
iter  40 value 138.695336
iter  50 value 135.933573
iter  60 value 133.992495
iter  70 value 132.731947
iter  80 value 131.407324
iter  90 value 129.361587
iter 100 value 127.871839
iter 110 value 127.467617
iter 120 value 127.076606
iter 130 value 126.532593
iter 140 value 126.344129
iter 150 value 126.023083
iter 160 value 125.971849
iter 170 value 125.889674
iter 180 value 125.705071
iter 190 value 125.502996
iter 200 value 125.352354
iter 210 value 125.340941
iter 220 value 125.337010
iter 230 value 125.317043
iter 240 value 125.277498
iter 250 value 125.226630
iter 260 value 125.201010
iter 270 value 125.164681
iter 280 value 125.150862
iter 290 value 125.124224
iter 300 value 125.113150
iter 310 value 125.086570
iter 320 value 125.080938
iter 330 value 125.074790
iter 340 value 125.069050
iter 350 value 125.061404
iter 360 value 125.050714
iter 370 value 124.990688
iter 380 value 124.866057
iter 390 value 124.798021
iter 400 value 124.795391
final  value 124.773048
converged

Call:
multinom(formula = type ~ ., data = fgl, maxit = 1000)

Coefficients:
      (Intercept)          RI          Na         Mg        Al         Si
Head   -114.51862 15.65745335 20.21784007 -44.228269 -15.51354  7.539015
Tabl    -26.58422 20.95019599 34.65499513 -49.246746  17.86878  6.458536
Veh     172.08020 -1.80563329  1.93768538   7.791059 -16.73108 -3.262667
WinF   -312.95719 -0.32702017  4.73903403  10.110742 -14.49942  2.679981
WinNF   224.50756 -0.09699872  0.06519741   2.959204 -15.48091 -2.791580
                K          Ca          Ba          Fe
Head   -25.9774690 -64.8440110  -24.832912 -335.793110
Tabl  -271.4633964 -85.7510007 -243.894825 -682.850454
Veh      0.8595508   5.0390985    3.673558    9.348289
```

```
WinF     6.9757293   5.4928375    7.956196   10.556897
WinNF    2.0423095  -0.2550624    1.256316   12.509308

Std. Errors:
      (Intercept)        RI        Na       Mg       Al        Si
Head   0.02598794 0.6492691 2.1160099 1.034882 3.681153 0.3972740
Tabl   0.06250091 7.6618679 2.2955835 6.861062 1.801126 0.2563515
Veh    0.07336834 0.8101703 1.1723888 1.924945 3.136502 0.4454227
WinF   0.05607987 0.7178503 1.0570901 1.532039 2.953425 0.3639392
WinNF  0.04915957 0.6752268 0.9371684 1.305121 2.726029 0.3237228
               K        Ca        Ba         Fe
Head  1.96287777 0.5686409 2.4784959 0.03082541
Tabl  0.08055989 4.4763884 0.1173599 0.10317475
Veh   3.02216159 2.0516596 4.5802034 3.11101168
WinF  2.68656550 1.6492213 2.9377944 2.37177883
WinNF 2.13459573 1.4692477 1.6244236 2.00823399

Residual Deviance: 249.5461
AIC: 349.5461
```

below I calculate the p-values, because they are not provided by default

```
      (Intercept)          RI           Na           Mg           Al
Head            0 0.000000000 0.000000e+00 0.000000e+00 2.505375e-05
Tabl            0 0.006250437 0.000000e+00 7.087664e-13 0.000000e+00
Veh             0 0.025833318 9.837831e-02 5.178556e-05 9.590727e-08
WinF            0 0.648710112 7.356852e-06 4.124590e-11 9.137450e-07
WinNF           0 0.885774069 9.445371e-01 2.336706e-02 1.355465e-08
                Si           K           Ca           Ba           Fe
Head  0.000000e+00 0.000000000 0.0000000000 0.000000000 0.000000e+00
Tabl  0.000000e+00 0.000000000 0.0000000000 0.000000000 0.000000e+00
Veh   2.391420e-13 0.776091670 0.0140450750 0.422523268 2.656654e-03
WinF  1.787459e-13 0.009417258 0.0008667014 0.006764503 8.545292e-06
WinNF 0.000000e+00 0.338685171 0.8621792771 0.439290440 4.693939e-10
```

Some predictors are insignificant when predicting for some of the response classes. I'll make predictions on the training data with this model and calculate the classification error

```
[1] 0.271028
```

Useful to look at confusion matrix as well

```
        y
yhat     Con Head Tabl Veh WinF WinNF
  Con     11    0    0   0    0     2
  Head     0   29    0   0    0     0
  Tabl     0    0    9   0    0     0
  Veh      0    0    0   3    4     0
  WinF     0    0    0   7   48    18
  WinNF    2    0    0   7   18    56
```

d) - Reviewing the results

Overall, it looks like the logistic regression and the Classification Tree get the exact same misclassification rate of approx. 27% However, they have differing precision when it comes to the different classes. So, if we had a preference for making a prediction about a specific class, we could use the model which is more accurate for that class.

The Neural Net, although doing very well on the entire training data (due to its extreme flexibility) still didn't bring any advantage over the other two models. It too had a misclassification rate of approximately 27%.

My conclusion is that, with this dataset, that's probably just the best that we can do. As discussed above, Iw ould use the Logistic Regression or the Classification Tree depending on the usecase and keep the Neural Network as a back up if in the future we get a lot more data. I wouldn't use the NN for this situation, because it takes too long to fit.

Appendix

This section is to be used for including your R code. The following lines of code will take care of it. Please make sure to comment your code appropriately - in particular, demarcating codes belonging to different questions. Among other things, it will be easier for you to debug your own code.

```
library(car)
library(readxl)
library(tidyverse)
library(psycho)
library(caret)
library(nlme)
library(DescTools)
library(nnet)
library(RSNNS)
library(rpart)
library(ModelMetrics)
library(rpart)
# install.packages('ALEPlot')
library(ALEPlot)

set.seed(42)

CVInd <- function(n,K) { # n is sample size; K is number of parts; # returns K-length list of indices f
  m<-floor(n/K) #approximate size of each part
  r<-n-m*K
  I<-sample(n,n) #random reordering of the indices
  Ind<-list() #will be list of indices for all K parts
  length(Ind)<-K
  for (k in 1:K) {
    if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
    else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] <- I[kpart] #indices for kth part of data
  }
  Ind
}
```

```
# FUNCTION FOR DOING 10-FOLD CV WITH 10 DIFFERENT FOLD SETS
# RETURNS AVG MISCLASSIFICATION RATE FOR 3 MODELS
tenFoldCVtenRepThreeModels = function(linout, size, decay) {
  Nrep<-10 #number of replicates of CV
  K<-10  #K-fold CV on each replicate
  n.models = 3 #number of different models to fit
  n=nrow(fgl.standardized)
  y<-fgl.standardized[['type']]
  yhat=matrix(0,n,n.models)
  CV.rate<-matrix(0,Nrep,n.models)
  for (j in 1:Nrep) {
Ind<-CVInd(n,K)
for (k in 1:K) {
  # Train model on training data; MODIFY TEST PARAMETERS ON LINE BELOW
  out<- nnet(type ~ ., fgl.standardized[-Ind[[k]],], linout=linout[1], skip=F, size=size[1], decay=decay
  # Make predictions with our model on the Validation Data set
  for (i in Ind[[k]]){ # For each index from those chosen to be in the validation set
    phat = predict(out, fgl.standardized[i,c(1:9)]) # Predict the response value from our model based o
    typePrediction = types[which.max(phat)] # Predict the glass Type with the highest probability
    yhat[i,1] = typePrediction # Populate the specific row in yhat at the index for which we are making
  }

  # Train model on training data; MODIFY TEST PARAMETERS ON LINE BELOW
  out<- nnet(type ~ ., fgl.standardized[-Ind[[k]],], linout=linout[2], skip=F, size=size[2], decay=decay
  # Make predictions with our model on the Validation Data set
  for (i in Ind[[k]]){ # For each index from those chosen to be in the validation set
    phat = predict(out, fgl.standardized[i,c(1:9)]) # Predict the response value from our model based o
    typePrediction = types[which.max(phat)] # Predict the glass Type with the highest probability
    yhat[i,2] = typePrediction # Populate the specific row in yhat at the index for which we are making
  }

  # Train model on training data; MODIFY TEST PARAMETERS ON LINE BELOW
  out<- nnet(type ~ ., fgl.standardized[-Ind[[k]],], linout=linout[3], skip=F, size=size[3], decay=decay
  # Make predictions with our model on the Validation Data set
  for (i in Ind[[k]]){ # For each index from those chosen to be in the validation set
    phat = predict(out, fgl.standardized[i,c(1:9)]) # Predict the response value from our model based o
    typePrediction = types[which.max(phat)] # Predict the glass Type with the highest probability
    yhat[i,3] = typePrediction # Populate the specific row in yhat at the index for which we are making
  }

} #end of k loop
CV.rate[j,]=apply(yhat,2,function(x) sum(y != x)/n)
  } #end of j loop
  # CV.rate
  CV.rateAve<- apply(CV.rate,2,mean); CV.rateAve #averaged CV misclass rate
  return(list(CV.rate, CV.rateAve))
  }

############ QUESTION 1 begins here ###################
heart = read_excel('HW2_data.xls') # loading data for question 1
head(heart)
y = heart[['cost']]
heart['log10_cost'] = log10(heart[['cost']])
```

```r
hist(log10(heart[['cost']]), breaks = 50) # Plotting data
heart.fit1.data =  heart[,-(1:2)] # Dropping the original cost column and the unique identifier
# head(heart.fit1.data) # Looking at the new data
fit1 = lm(log10_cost ~ . , data = heart.fit1.data) # Fitting linear model
summary(fit1)
vif(fit1)
cor(heart.fit1.data[,1:8])
PlotCorr(cor(heart.fit1.data ))
cor(heart.fit1.data[,1:8]) > .6
plot(fit1, which = c(1,2))
plot(fit1, which = c(4,6))
detach('package:RSNNS', unload = T)
set.seed(42)
heart.fit1.standardizedData = standardize(heart.fit1.data)
options = trainControl(method = 'repeatedcv', number = 10, repeats = 50, summaryFunction = defaultSumma
fit1.cv = train(log10_cost ~ ., method = 'lm', data = heart.fit1.standardizedData, trControl = options,
fit1.cv
summary(fit1.cv)
log10_cost.predictions = predict.train(fit1.cv)
residuals = log10_cost.predictions - heart.fit1.standardizedData$log10_cost
plot(log10_cost.predictions, residuals)
abline(0,0)
qqnorm(log10_cost.predictions)
abline(0,1)
########### Problem 2 begins here ###################
tune.decay = seq(0,5,.5)
tune.size = seq(1, 10)
nn1.tuningParameters = expand.grid(tune.decay, tune.size)
nn1.tuningParameters = data.frame(nn1.tuningParameters)
names(nn1.tuningParameters) = c('decay','size')
head(nn1.tuningParameters)
# detach('package:RSNNS', unload = T)
options = trainControl(method = 'repeatedcv', number = 10, repeats = 1, summaryFunction = defaultSummary
nn1.cv = train(log10_cost ~ ., method = 'mlpWeightDecay', data = heart.fit1.standardizedData, trControl
nn1.cv
tune.decay = seq(0,1,.1)
tune.size = seq(3, 7)
nn2.tuningParameters = expand.grid(tune.decay, tune.size)
nn2.tuningParameters = data.frame(nn2.tuningParameters)
names(nn2.tuningParameters) = c('decay','size')
# head(nn1.tuningParameters)
options = trainControl(method = 'repeatedcv', number = 10, repeats = 1, summaryFunction = defaultSummary
nn2.cv = train(log10_cost ~ ., method = 'mlpWeightDecay', data = heart.fit1.standardizedData, trControl
nn2.cv
tune.decay = seq(.01,.5,.01)
tune.size = seq(5, 6)
nn3.tuningParameters = expand.grid(tune.decay, tune.size)
nn3.tuningParameters = data.frame(nn3.tuningParameters)
names(nn3.tuningParameters) = c('decay','size')
# head(nn1.tuningParameters)
options = trainControl(method = 'repeatedcv', number = 10, repeats = 1, summaryFunction = defaultSummary
nn3.cv = train(log10_cost ~ ., method = 'mlpWeightDecay', data = heart.fit1.standardizedData, trControl
nn3.cv
```

```r
nn = nnet(log10_cost ~. ,data = heart.fit1.standardizedData, linout = TRUE, decay = .01, size = 6, skip
summary(nn)
# sum(residuals(nn)^2) # Second way of calculating SSE!
sum(nn$residuals^2) # SSE For Neural Network
sum((predict.train(fit1.cv)-heart.fit1.standardizedData$log10_cost)^2)
sum(residuals(fit1.cv)^2)
sum(fit1$residuals^2)
heart.fit1.scaledData = scale(heart.fit1.data) # Tried using scale() function instead of standardize
# The output is a matrix instead of a tibble/dataframe. This resolved an error I was getting when
# using the standardized data (instead of the scaled)
yhat <- function(X.model, newdata) as.numeric(predict(X.model, newdata))
par(mfrow=c(2,4))
for (j in 1:8)  {ALEPlot(heart.fit1.scaledData[,1:8], nn, pred.fun=yhat, J=j, K=50, NA.plot = TRUE)
  rug(heart.fit1.scaledData[,j]) }  ## This creates main effect ALE plots for all 8 predictors
par(mfrow=c(1,1))
plot(nn$fitted.values, nn$residuals^2)
########### Problem 3 begins here ##################
head(heart.fit1.data)
maxdepthRange = seq(1, 30, 1)
cpRange = seq(.01, 1, .01)
grid = expand.grid(maxdepthRange, cpRange)
names(grid) = c('maxdepth', 'cp')
head(grid)
set.seed(42)
n = nrow(heart.fit1.data)
K = 10
# Number of potential models in the grid
num_models <- nrow(grid)
# Create the CV Indices
indices = CVInd(n, K) # We'll be using the same folds for all models so this has to be computed only on
avgRMSE = c()

for (j in 1:num_models){

  cp = grid$cp[j]
  maxdepth = grid$maxdepth[j]
  rmse = c()

  for (i in 1:K){
    # SET TRAINING INDICES
    trainingIndices = indices[[i]]
    trainingData = heart.fit1.data[-trainingIndices,]
    validationData = heart.fit1.data[trainingIndices,]

    # Train a model and store in the list
    model = rpart(formula = log10_cost ~ ., data = trainingData, method = "anova",cp = cp, maxdepth = ma

    # Make a prediction on the validation set
    pred = predict(object = model, newdata = validationData)

    # Compute rmse for this fold and add to vector
    rmse[i] = rmse(actual = validationData$log10_cost, predicted = pred)
```

```
  }

  avgRMSE = c(avgRMSE, mean(rmse))

}
head(unique(avgRMSE), 4)
cpRange = seq(.001, .01, .001)
grid = expand.grid(maxdepthRange, cpRange)
names(grid) = c('maxdepth', 'cp')
set.seed(42)
n = nrow(heart.fit1.data)
K = 10
# Number of potential models in the grid
num_models <- nrow(grid)
# Create the CV Indices
indices = CVInd(n, K) # We'll be using the same folds for all models so this has to be computed only on
avgRMSE = c()

for (j in 1:num_models){

  cp = grid$cp[j]
  maxdepth = grid$maxdepth[j]
  rmse = c()

  for (i in 1:K){
    # SET TRAINING INDICES
    trainingIndices = indices[[i]]
    trainingData = heart.fit1.data[-trainingIndices,]
    validationData = heart.fit1.data[trainingIndices,]

    # Train a model and store in the list
    model = rpart(formula = log10_cost ~ ., data = trainingData, method = "anova",cp = cp, maxdepth = ma

    # Make a prediction on the validation set
    pred = predict(object = model, newdata = validationData)

    # Compute rmse for this fold and add to vector
    rmse[i] = rmse(actual = validationData$log10_cost, predicted = pred)

  }

  avgRMSE = c(avgRMSE, mean(rmse))
}
grid[which.min(avgRMSE),]
bestTree = rpart(log10_cost ~. , data = heart.fit1.data, method = 'anova', cp = .002, maxdepth = 7 )
plotcp(bestTree)
printcp(bestTree)
bestPrunedTree = prune(bestTree, cp = 0.0066471 )
bestPrunedTree
par(cex=.8); plot(bestPrunedTree, uniform = T); text(bestPrunedTree, use.n = T);
par(cex=.8);plot(bestPrunedTree, uniform = F); text(bestPrunedTree, use.n = T);
yhat = predict(bestPrunedTree); e = heart.fit1.data$log10_cost - yhat
sum(e^2)
```

```
bestPrunedTree$variable.importance
plot(yhat, e)
plot(nn$fitted.values, nn$residuals^2)
library(MASS)
########### Problem 4 begins here ###################
fgl = read_excel('HW2_data.xls', sheet = 'FGL data')
fgl = fgl[,-1]
str(fgl)
### I'll also need to create a STANDARDIZED VERSION of the data
fgl.standardized = standardize(fgl)
fgl.standardized = as.data.frame(fgl.standardized) # Converting to DataFrame
fgl.standardized[['type']] = as.factor(fgl.standardized[['type']])
types = levels(fgl.standardized[['type']]) # Extracting the levels of the response variable. These will
str(fgl.standardized)
linout = c(F,F,F)
size = c(10,10,10)
decay = c(.01,.05,.1)
tenFoldCVtenRepThreeModels(linout,size,decay)
linout = c(F,F,F)
size = c(10,10,10)
decay = c(.035,.037,.04)
tenFoldCVtenRepThreeModels(linout,size,decay)
linout = c(F,F,F)
size = c(23,25,26)
decay = c(.04,.04,.04)
tenFoldCVtenRepThreeModels(linout,size,decay)
linout = c(F,T,T)
size = c(25,25,25)
decay = c(.04,.04,.04)
tenFoldCVtenRepThreeModels(linout,size,decay)
y = fgl.standardized$type
bestNN = nnet(type ~ ., fgl.standardized, linout=T, skip=F, size=25, decay=.04, maxit=1000, trace=F)
phat = predict(bestNN, fgl.standardized[,c(1:9)])
yhat = c()
for (i in 1:length(phat[,1])){
   typePrediction = types[which.max(phat[i,])]
   yhat = c(yhat, typePrediction)
}

1-sum(yhat==y)/length(y)
table(yhat, y)
# library(MASS)
########### Problem 4 begins here ###################
fgl = read_excel('HW2_data.xls', sheet = 'FGL data')
fgl = fgl[,-1]
# str(fgl)
classTree = rpart(type ~ ., data = fgl, method = 'class', xval = 10) # Fitting Classification Tree
plotcp(classTree)
printcp(classTree)
bestClassTree = prune(classTree, cp = 0.03623)
bestClassTree$variable.importance
# install.packages("rpart.plot")
rpart.plot::rpart.plot(bestClassTree, uniform = F)
```

```
y = fgl$type
phat = predict(bestClassTree, fgl[,c(1:9)])
yhat = c()
for (i in 1:length(phat[,1])){
   typePrediction = types[which.max(phat[i,])]
   yhat = c(yhat, typePrediction)
}
1-sum(yhat==y)/length(y)
table(yhat, y)
fgl = read_excel('HW2_data.xls', sheet = 'FGL data')
fgl = fgl[,-1]
fgl[['type']] = as.factor(fgl[['type']])
nominalFit = multinom(type ~ . , data = fgl, maxit = 1000)
summary(nominalFit)
z = summary(nominalFit)$coefficients / summary(nominalFit)$standard.errors
p = (1 - pnorm(abs(z),0,1))*2
p
yhat = factor(predict(nominalFit, fgl[,c(1:9)]))
y = fgl$type
# 1-sum(y == yhat)/length(y)
ce(actual = y, predicted = yhat)
table(yhat, y)
```