Hands-on Exercise 3

Part 1

Neural networks attempt to mimic the activity of neurons in the human brain. They learn by comparing the algorithm's classification of the record with the actual classification of the record. In the example presented in the video, there is a database with 13 variables measuring chemical attributes of wine, which will help classify the type of wine. First, the data was partitioned into training and validation sets. 80% of the data was allocated to the training set and 20% of the data was allocated to the validation set. This partition was done randomly, and all the variables were included in both sets. Then, training data was classified using the neural network method. The output variable is wine type. A neural network was created with 1 hidden layer and 25 nodes. The results given for the training data indicated that no records were misclassified resulting in an error rate of 0%. On the validation data, 1 record was misclassified resulting in an error rate of 2.78%. Then, this model was used to classify 9 new records, but no further information was given about this step. In this video, this dataset was also classified using the classification tree method using the same variables. The results indicated that on the training data 11 records were misclassified resulting in an overall error rate of 7.75%. The validation data had 5 records misclassified resulting in an overall error rate of 13.89%. These results show that the neural network method works better for this dataset.

If I were the client, I would make sure to analyze the results before taking any further action. My concern in this case would be that an error rate as low as 0% could be a sign of overfitting. The model learns the training data so well which gives low error rates, but it will not perform well on new data. To prevent this, the training data could be increased, or the number of nodes could be reduced. It is important to avoid overfitting, as we could be trusting a model that will not help us make good decisions when given new data.

Part 2

Follow the examples in Ch. 9 and Ch. 11 and the provided videos to build neural net models and then regression tree models of the **Boston housing** dataset.

- **How to develop regression tree models, including details on model setting. You should build at least 3 tree models, analyze each model, justify and select the best tree model candidate.**

Developing suitable regression tree models requires comprehensive data preprocessing and preparation. Good quality data is essential for the model to perform well. Data quality issues can have a negative impact on the developed models; this could happen if

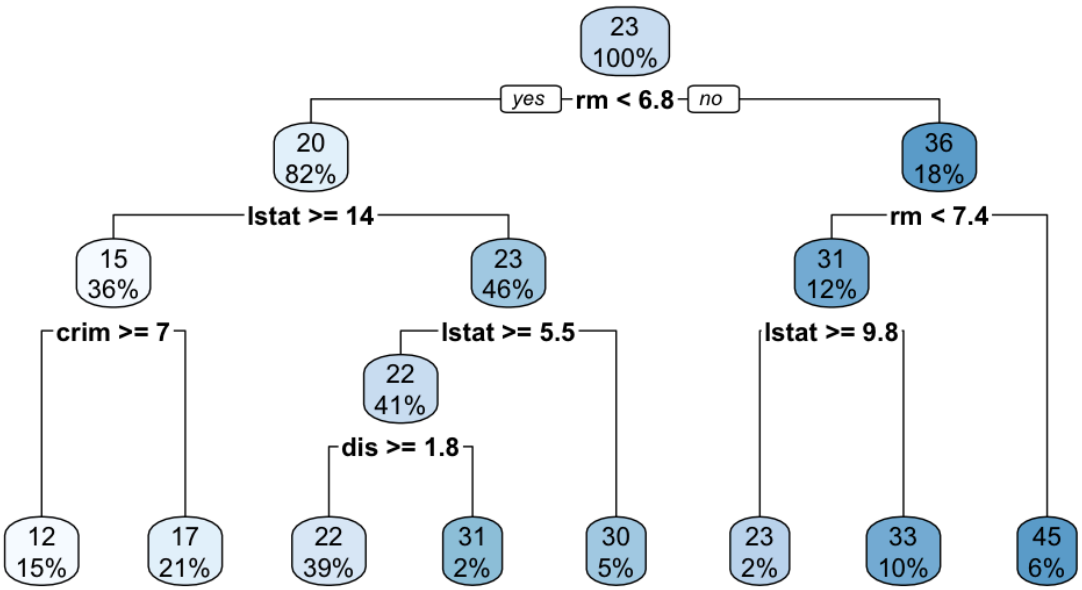
the data quality issues are not detected and cleaned, as they could be interpreted as some sort of pattern by the tree algorithm. Adequate validation measures can be used to determine whether the accuracy of predictive models is affected by data quality, before and after cleaning data. Clear steps should be followed in order to conduct the data preparation and processing required for these models. These steps are described in detail below.

Data quality issues should be explored and cleaned. Two dominant issues are missing data and outliers. Two of the most popular methods for handling missing data are imputation and growing a tree in advance to subsequently assign samples to nodes. Outlier detection methods are not very different from those used in the context of many other models. Features used in the model should be chosen carefully. It is often useful to build the tree model with many features and then discard some to explore the impact of sticking to more dominant features. The popular data transformation techniques can be used with this model, including log, score-based transformations, winsorizing, and categorical encoding. Once these steps are followed and the data is ready, it is typically randomly split into a training and testing set, where the training set constitutes 50–60% of the original size and is used to grow the tree, and the testing set is used to evaluate the tree. Exploratory data analysis can help inform the preprocessing steps required.

Developing suitable regression tree models requires comprehensive data preprocessing and preparation. Good quality data is essential for the model to perform well. Data quality issues can have a negative impact on the developed models; this could happen if the data quality issues are not detected and cleaned, as they could be interpreted as some sort of pattern by the tree algorithm. Adequate validation measures can be used to determine whether the accuracy of predictive models is affected by data quality, before and after cleaning data. Clear steps should be followed in order to conduct the data preparation and processing required for these models. These steps are described in detail below.

Data quality issues should be explored and cleaned. Two dominant issues are missing data and outliers. Two of the most popular methods for handling missing data are imputation and growing a tree in advance to subsequently assign samples to nodes. Outlier detection methods are not very different from those used in the context of many other models. Features used in the model should be chosen carefully. It is often useful to build the tree model with many features and then discard some to explore the impact of sticking to more dominant features. The popular data transformation techniques can be used with this model, including log, score-based transformations, winsorizing, and categorical encoding. Once these steps are followed and the data is ready, it is typically randomly split into a training and testing set, where the training set constitutes 50–60% of the original size and is used to grow the tree, and the testing set is used to evaluate the tree. Exploratory data analysis can help inform the preprocessing steps required.

Basic Regression Tree

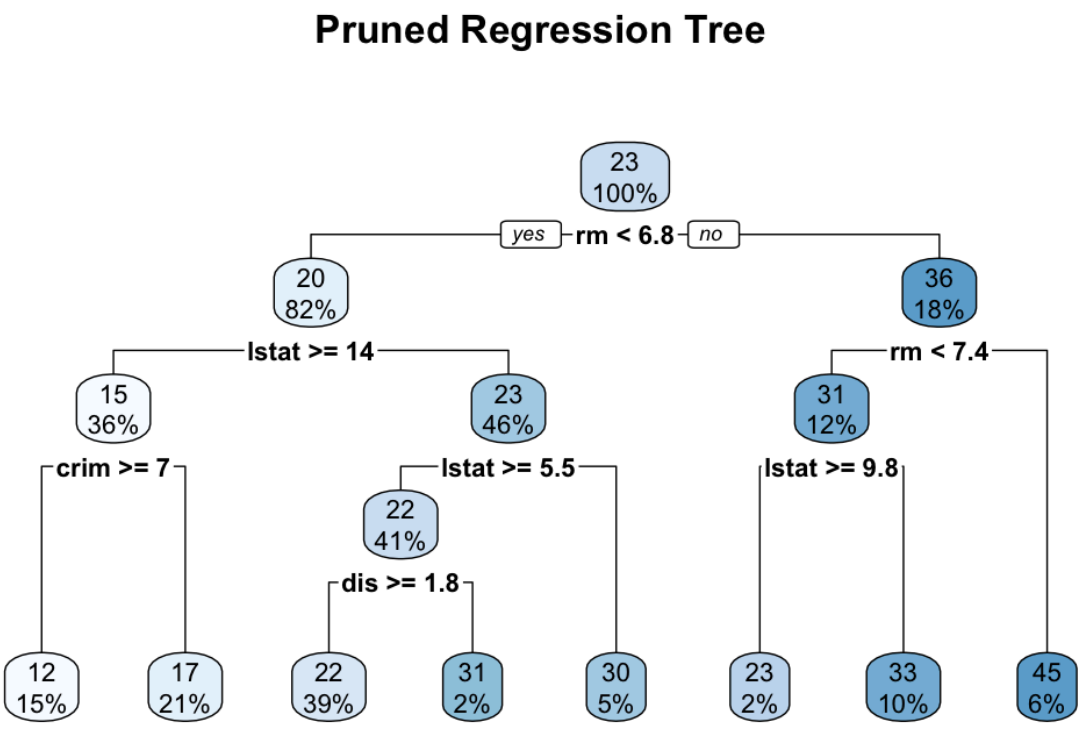



```
install.packages("MASS")    # For the Boston dataset
install.packages("rpart")   # For regression trees
install.packages("rpart.plot")  # For visualizing trees
install.packages("caret")    # For data partitioning and evaluation

library(MASS)
library(caret)

# Load the Boston dataset
data("Boston")
set.seed(123)  # For reproducibility

# Split the dataset into training and testing sets
train_index <- createDataPartition(Boston$medv, p = 0.8, list = FALSE)
train_data <- Boston[train_index, ]
test_data <- Boston[-train_index, ]

library(rpart)

# Fit a basic regression tree model
tree_model1 <- rpart(medv ~ ., data = train_data)
```

```
# Make predictions
predictions1 <- predict(tree_model1, test_data)

# Evaluate the model
mse1 <- mean((test_data$medv - predictions1)^2)
cat("MSE of Model 1 (Basic Tree):", mse1, "\n")

# Visualize the tree
library(rpart.plot)
rpart.plot(tree_model1, main = "Basic Regression Tree")
```

MSE of Model 1 (Basic Tree): 24.26076

Pruned Regression Tree

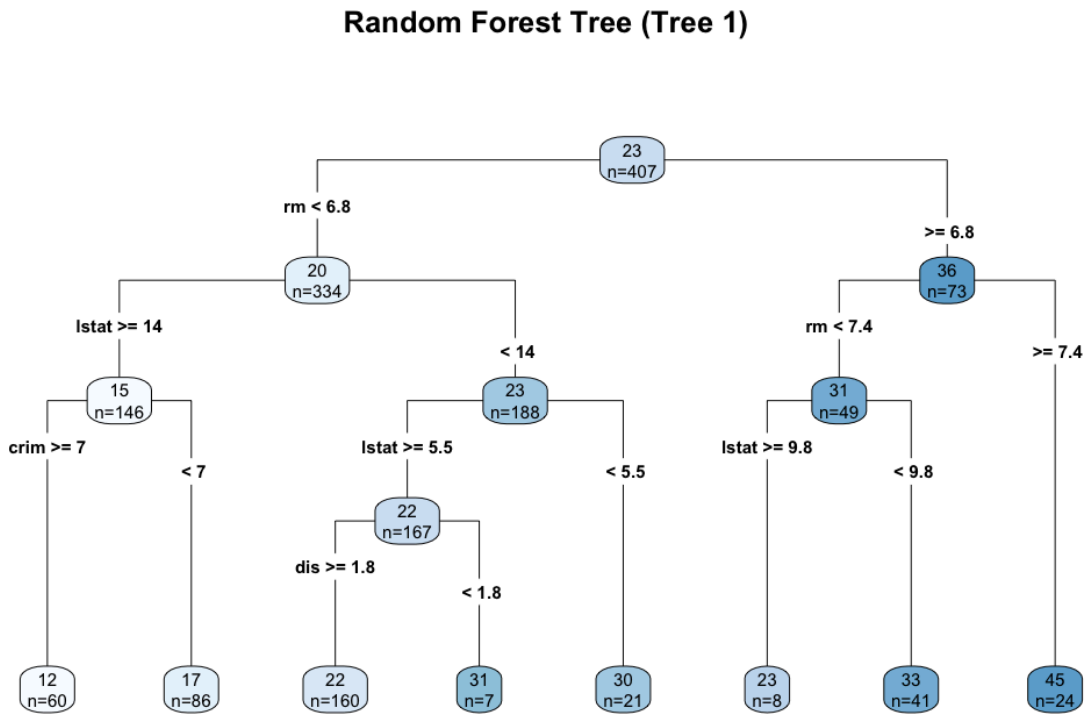



```
# Fit a regression tree with complexity parameter (cp)
tree_model2 <- rpart(medv ~ ., data = train_data, cp = 0.01)

# Prune the tree
pruned_tree <- prune(tree_model2, cp = 0.01)

# Make predictions
```

```
predictions2 <- predict(pruned_tree, test_data)

# Evaluate the model
mse2 <- mean((test_data$medv - predictions2)^2)
cat("MSE of Model 2 (Pruned Tree):", mse2, "\n")

# Visualize the pruned tree
rpart.plot(pruned_tree, main = "Pruned Regression Tree")
```

MSE of Model 2 (Pruned Tree): 24.26076

Random Forest Tree




```
library(MASS)
library(randomForest)
library(caret)
library(rpart)
library(rpart.plot)

data("Boston")
set.seed(123)
```

```
train_index <- createDataPartition(Boston$medv, p = 0.8, list = FALSE)
train_data <- Boston[train_index, ]
test_data <- Boston[-train_index, ]

rf_model <- randomForest(medv ~ ., data = train_data, ntree = 500)

tree_index <- 1
single_tree <- getTree(rf_model, k = tree_index, labelVar = TRUE)

rpart_tree <- rpart(medv ~ ., data = train_data, method = "anova")

rpart.plot(rpart_tree, main = "Random Forest Tree (Tree 1)",
           type = 4, extra = 1)
```

MSE of Model 3 (Random Forest): 9.326189

The Random Forest Tree has an MSE of 9.326189 which would suggest it is the best choice for predictive performance. A regression tree model with a lower Mean Squared Error is considered more accurate because MSE quantifies how well the model's predictions match the actual outcomes. MSE calculates the average squared difference between predicted and actual values. A lower MSE means that the average error is smaller, indicating that the model's predictions are closer to the true values. Since MSE squares the errors, it penalizes larger discrepancies more heavily. This means that a model with a low MSE is not just slightly better on average but it also performs well across a range of predictions, especially avoiding large errors. A lower MSE typically suggests that the model has captured the underlying data patterns more effectively, leading to better generalization to unseen data. MSE provides a clear numerical measure for comparing different models. When choosing between models, the one with the lower MSE is preferred as it indicates better overall performance in making predictions.

- **How to develop neural net models, including details on model setting. You should build at least 3 neural net models, analyze each model, justify and select the best neural net model candidate. For the neural net models, you can use one hidden layer and vary the number of nodes.**

Neural networks are essentially organized in layers. There is an input layer, one or more hidden layers, and the output layer. The input layer does not consist of fully functional neurons but rather acts as a placeholder for the input data values. Every feature in our dataset corresponds to one node in this layer. This means that this input layer is feeding data into the next layers of the network without any transformation. Hidden layers are the

core of the neural network's processing power. Neurons in hidden layers are responsible for learning complexities within the data. Multiple hidden layers are possible, and deeper networks are usually used for complex tasks. However there is no fixed rule on the number of hidden layers, but most networks end up using one or two hidden layers for simplicity and efficiency. Output layer is the final layer that provides the neural network's prediction. It usually has one node per class, with the node holding the highest value indicating the predicted class. In regression tasks, a single output node is often used to provide a continuous output value. As the relationship between input data and output predictions becomes more complex, the number of neurons in hidden layers is supposed to also increase. In the situation where the process being modeled has distinct stages or levels of abstraction, adding more hidden layers can possibly help. However, adding more layers might lead to memorization of the training set and not properly producing a generalized solution, which could lead to a reduction in effectiveness on new data. The is a practical method to set an upper limit on the number of neurons in hidden layers based on data availability. For an upper bound calculation, divide the number of training cases by the sum of nodes in the input and output layers. Then, we divide this result by a scaling factor between five and ten, depending on the noise level of the data. Higher scaling factors are used for cleaner data, ensuring the model generalizes better than overfitting to the training data.

Data preparation ensures that the input data is formatted and divided in a way that increases the effectiveness of neural network training. Key steps are partitioning the data set and rescaling features. Partitioning divides the data into distinct sets to allow for effective training, validation and testing of the neural network model. It helps the model generalize. This ensures that the model's effectiveness can be tested on data it hasn't seen, reducing the risk of overfitting. A common partition setup is 60% for training and 40% for validation. The training set helps the model learn patterns, while the validation set allows performance monitoring and hyperparameter tuning. Analytic Solver Mining includes options to automatically partition data within the neural network classification or prediction methods. If partitioning has already been done on the dataset, this option will be disabled, as re-partitioning could disrupt model training consistency. Rescaling helps the neural network coverage faster and more reliably by preventing any single feature from influencing the training process. It reduced the likelihood of exploding or vanishing gradients, which can occur if inputs are not standardized. Standardization transforms the data to have a mean of 0 and a standard deviation of 1. Normalization maps data to a range, usually [0,1] and is recommended when using the sigmoid activation in the hidden layer. Adjusted Normalization maps data to a range of [-1,1] better suited when using the hyperbolic tangent activation function. Unit norm scales each feature vector to unit length.

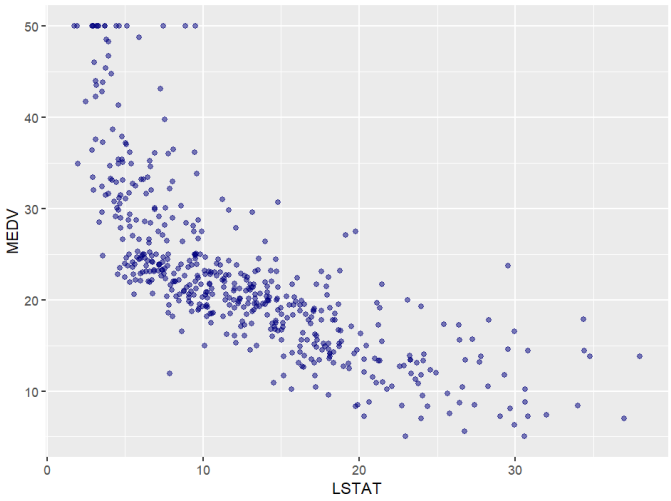
Key configurations to optimize learning efficiency and prevent overfitting in the training settings for neural networks. Activation functions like Sigmoid (0 to 1 output), Tanh (-1 to 1 output), and ReLU (allowing positive values only) enable the model to capture complex patterns, with ReLU particularly effective in deeper networks. Weights are typically initialized randomly, with a seed ensuring consistent results for reproducibility.

Backpropagation is to adjust weights iteratively based on error feedback, improving predictions over successive epochs. To prevent overfitting, we can incorporate techniques like weight decay. We finally monitor performance on the validation set to detect overfitting. Through that we adjust model complexity or training parameters if necessary.
Each neural network model candidate reveals distinct strengths and weaknesses based on configuration, error metrics, and complexity. The first model, a 5 neuron network utilizing all available predictors, resulted with an RMSE of 23.46 and an MSE of 501.73. This high error rate shows that the inclusion of all these predictors resulted in unnecessary noise, reducing the models accuracy. Without refining the predictor set, the model's performance suffered due to the presence of irrelevant or weakly correlated variables, making it an unsuitable candidate for accurate predictions.
The second model with a 5 neuron network with stepwise selected predictors, shows a significant improvement in performance. By applying stepwise selection to eliminate less relevant predictors, this model achieved an RMSE of 9.11 and an MSE of 16.8. While maintaining simplicity the model's capability to capture essential patterns. This model is a stronger candidate than the previous one.
The third model, a 5 neuron network, used a better selection to refine the predictor set, leading to even better performance. This configuration achieved a substantial reduction in predictor error, with an RMSE of 3.91 and an MSE of 83. The forward selection process improved the model, while still using a small hidden layer.
The final model was a 8 neuron network with forward selected predictors, which achieved the lowest error metrics overall, with an RMSE of 3.65 and an MSE of 15.78. Increasing the number of neurons enhanced the model, resulting in the highest accuracy among all configurations. However, this added complexity may also raise the risk of overfitting. The 8 neuron model provides optimal accuracy, but the tradeoff in complexity should be carefully considered.
The 8 neuron model is a step forward where maximum accuracy is paramount, but if simplicity and robustness against overfitting is prioritized, the 5 neuron model with forward selection offers a balanced alternative.

- **Recap what you have learned in the data exploration step (i.e., hands-on report 1) regarding predictors and discuss how you can use what you have learned in report 1 to reduce the number of predictors.**

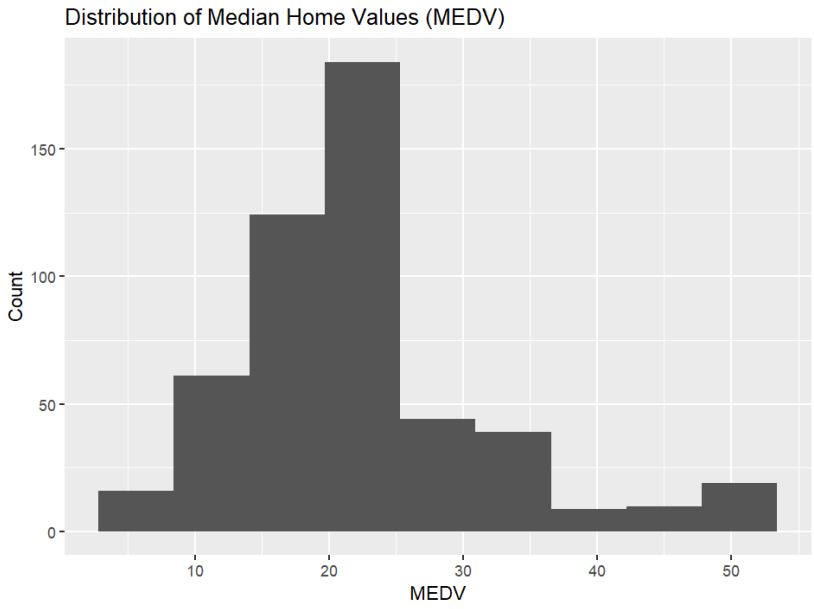
In the first hands on report we used various visualizations to determine which predictors would be ideal for developing models. These visualizations included:

- Scatter plots of MEDV and LSTAT

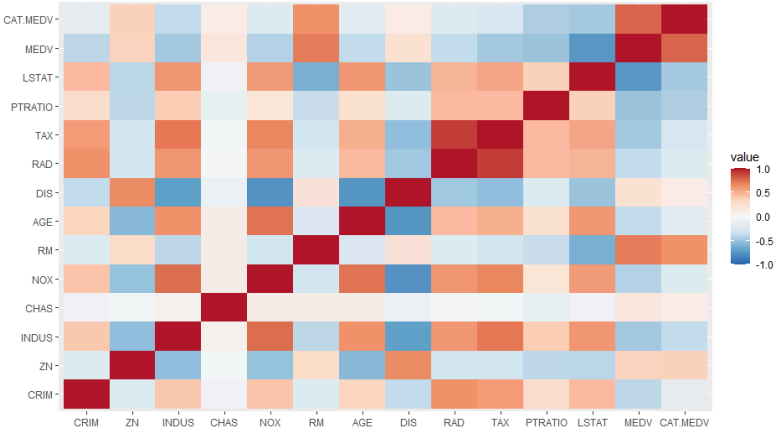


- Bar Charts of MEDV and CHAS

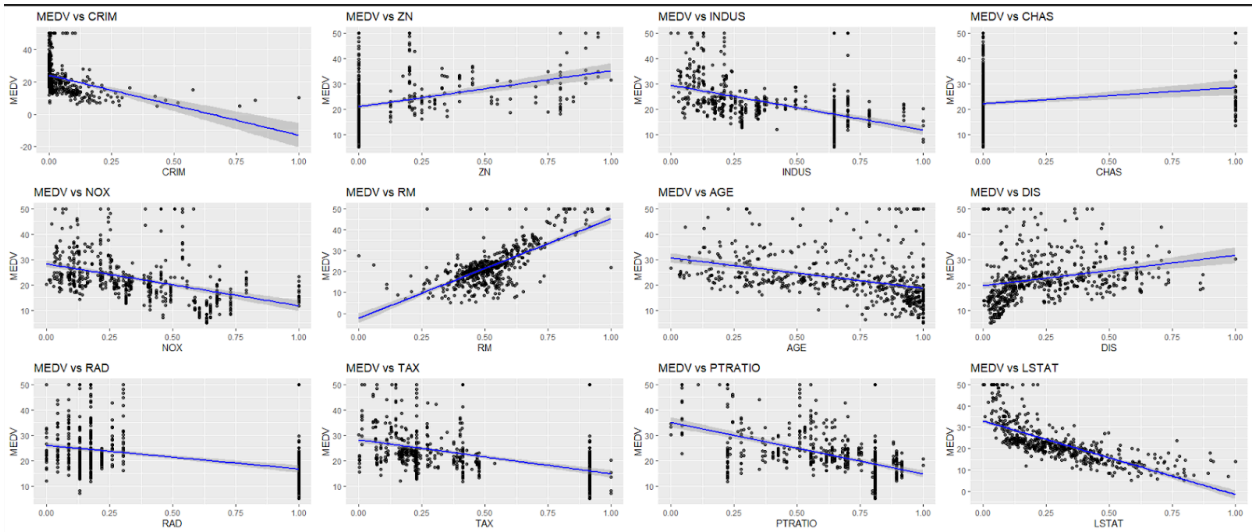
- Histogram Analysis of MEDV



- Heat map of Correlations



- In hands on 2 we also created scatter plots overlaid with lines to show linearity of features



- Also in hands on 2 we removed features based on statistical significance

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  38.05962    5.02072   7.581 4.67e-13 ***
CRIM         -0.12675    0.03384  -3.746 0.000217 ***
ZN           -0.02409    0.01454  -1.657 0.098556 .
INDUS         0.12201    0.05950   2.051 0.041207 *
CHAS          2.35563    0.81340   2.896 0.004067 **
NOX         -12.11916    3.73909  -3.241 0.001329 **
RM            0.79259    0.46928   1.689 0.092302 .
DIS          -0.50480    0.19924  -2.534 0.011817 *
RAD           0.22184    0.06216   3.569 0.000420 ***
TAX          -1.85279    0.58961  -3.142 0.001849 **
PTRATIO      -0.55095    0.13337  -4.131 4.73e-05 ***
LSTAT        -0.52458    0.04863 -10.787  < 2e-16 ***
CAT.MEDV     12.71659    0.88587  14.355  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The insights from these various data exploration allowed us to reduce the number of features in using a number of different strategies. Initially we only used the features that were shown in the heatmap or were the most correlated. This narrowed our features down to 3: RM (number of rooms), CHAS (near the Charles River), and LSTAT. Upon further exploration in hands on 2 increased the number of features using a statistical significance or P value of <0.05. This allowed us to improve our accuracy scores while increasing the number of features. While some features were deemed “not statistically significant” we had a hunch that they were. In order to further explore this we plotted the data points of each feature and drew a best fit line through it to determine linearity. This allowed us to see that, while not statistically significant, the feature “Age” did seem to be quite linear but contain a lot of outliers. We ran several models using this feature but all performed the same or worse. Leading the conclusion to remove it.

- **Compare and evaluate the 2 selected best-model candidates. Incorporating what you have done in report 2, i.e., regression models, discuss what would be the recommended model and its performance. Interpret the results of the model, make recommendations/suggestions.**

In the past exercises, we have looked closer at the data that makes up the Boston housing dataset. Specifically, some important steps include plotting data, looking for potential predictors, searching for any missing values, splitting our data, calculating accuracy measures to help us understand model performance, and making recommendations (and interpretation) from our results. By running different models and analyses, we are able to develop a stronger understanding of our data, evaluate and compare model performance, and make robust recommendations.

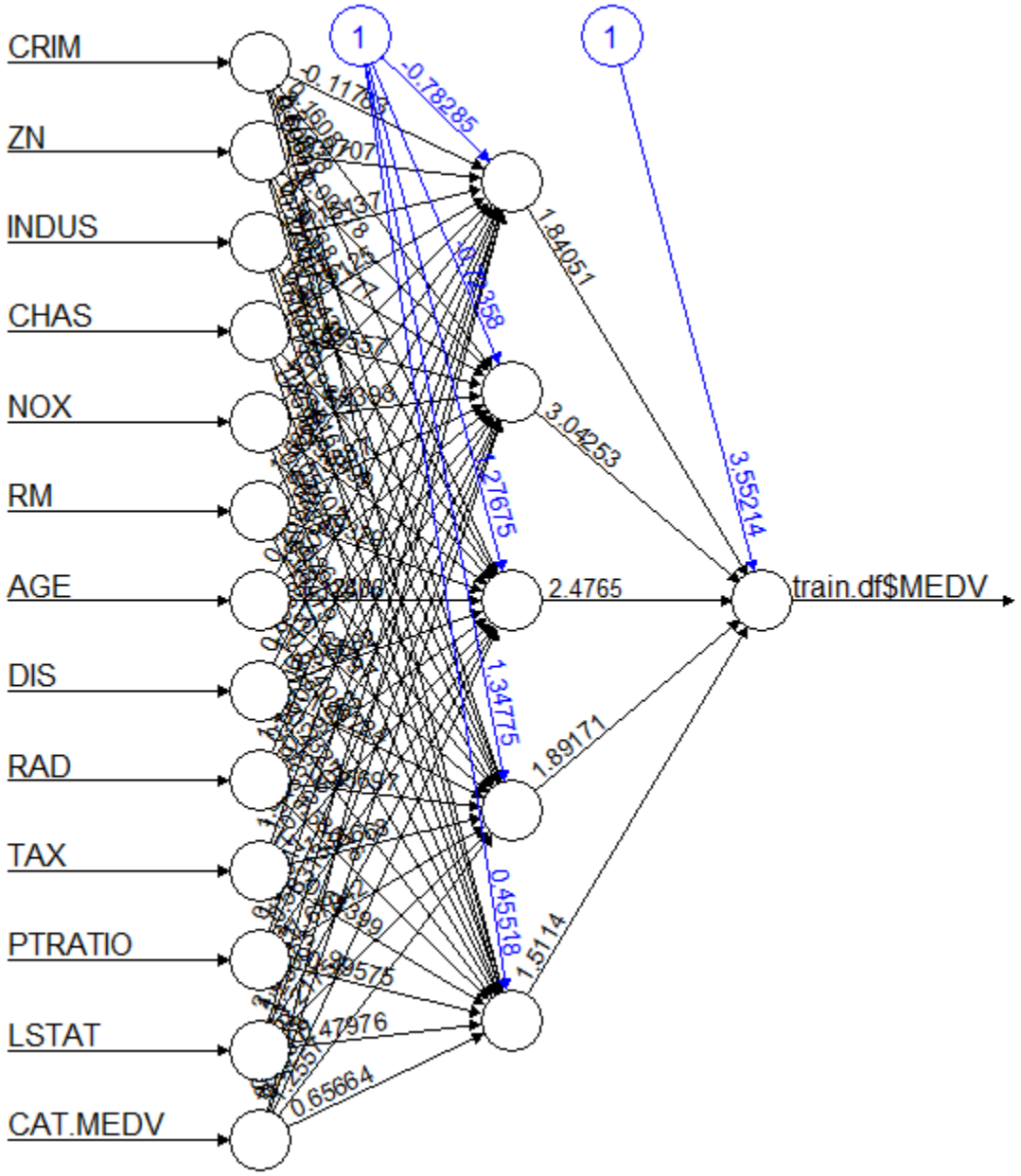
Predicting the median value of owner occupied homes (MEDV) with relevant predictors in a useful manner has an important focus throughout the exercises. In report 2, we were able to develop 3 multiple linear regression models that allows us to make predictions from model r output containing critical information such as coefficients (i.e. positive or negative relationships), r-squared, adjusted r-squared, p-values, residual, and prediction accuracy information. By examining this model output, we can assess the specific statistical significance of variables and better understand the relationship between the dependent variable and all the independent variables in the model. Also, we can minimize noise, help us simplify neural networks or regression trees by sticking to a particular combination of significant predictors, and perhaps discover if any non-linear relationships are better captured with regression trees or neural networks by referring to model output results. To help us compare models in exercise 2, we looked at predictors present in each model, adjusted r square values to measure model's goodness of fit, RMSE values to measure model accuracy and p-values to assess the ability to explain variance.

Using predictive performance metrics from the validation set helps us narrow focus on robust models and balance potential tradeoffs, especially for exercise 2 and 3. In hands-on exercise 3, we developed 3 regression tree models and 3 neural net models. Out of all the neural network models, the 8 neuron model has the highest performance accuracy values (lowest RMSE of 3.65) yet the 5 neuron model provides a nice balance between simplicity and robustness (RMSE of 3.91). Out of all the regression tree models, model 1 and 2 had very similar MSE values (MSE of 24.261 rounded or 4.93 RMSE) and predictive performance. However, model 3 had the lowest MSE value (MSE of 9.326 rounded which equals to the RMSE value of 3.05 rounded) and best out of all regression trees to capture non-linear relationships. In hands-on exercise 2, we developed 3 multiple linear regression models. Out of all multiple linear regression models, we identified how the models with "backward” and “both directions” selection are the two most robust because they have better predictive performance overall. Focusing on the adjusted r-squared metric and reducing the risk of excluding important variables in a simple manner helped us choose a single model (model with "backward" selection has an adjusted r square of 0.8459 and RMSE of 3.897 rounded) in hands on exercise 2.

Similarly, to identify and select 2 best-model candidates we can compare models with common performance metrics (like RMSE and MSE), ease of interpretation, and other relevant tradeoffs. Based on RMSE, we can identify that the lowest RMSE value of 3.05 was obtained from model 3 which is the lowest from all regression trees and all models conducted so far. Also, the multiple linear regression model with "backwards" selection holds the lowest RMSE value (RMSE of 3.897) out of all multiple linear regression models. Based on interpretation ease, regression trees and multiple linear regression models may be visually clearer or easier for users to follow when reviewing (and applying) output information to form key decisions. Model 3, from regression tree models conducted, is a recommended model because it can balance interpretability, resource use, accuracy, and capture nonlinear relationships in a robust manner. The backward selection multiple regression model is able to provide linear relationship insights in a manner that prioritizes accuracy and it is another recommended model because it helps with interpretation ease. Overall, we want to make sure we can balance a predictive performance, ease of interpretation, computational capacity, and other relevant factors when selecting a strong model that can help us make predictions.

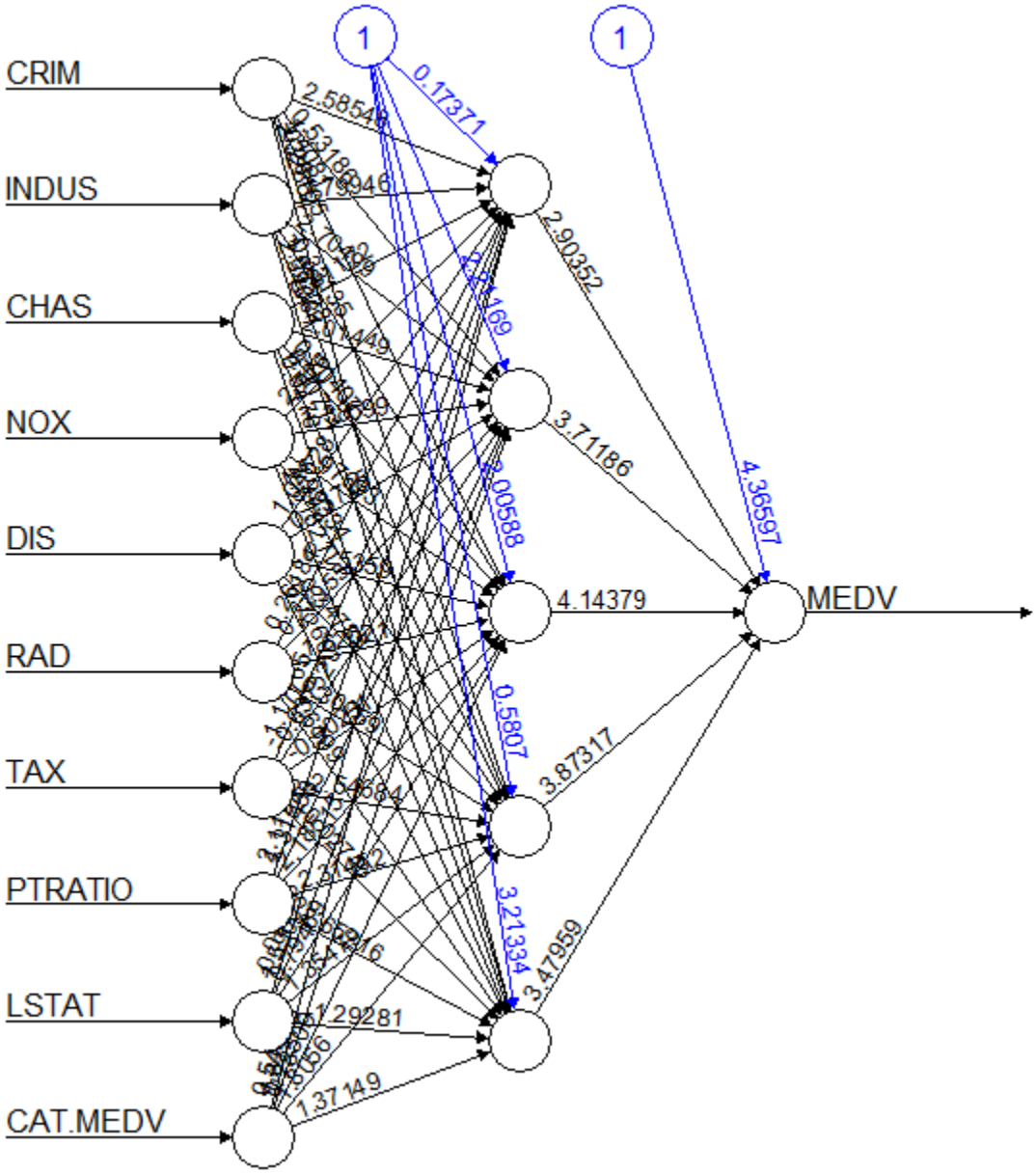
- **Please provide details such as screenshots of ASDM or R procedures, graphs, outputs, and justifications of your actions, etc.**

```
> ##########################################################################
> # 1 layer 5 neuron neural network
> ##########################################################################
```



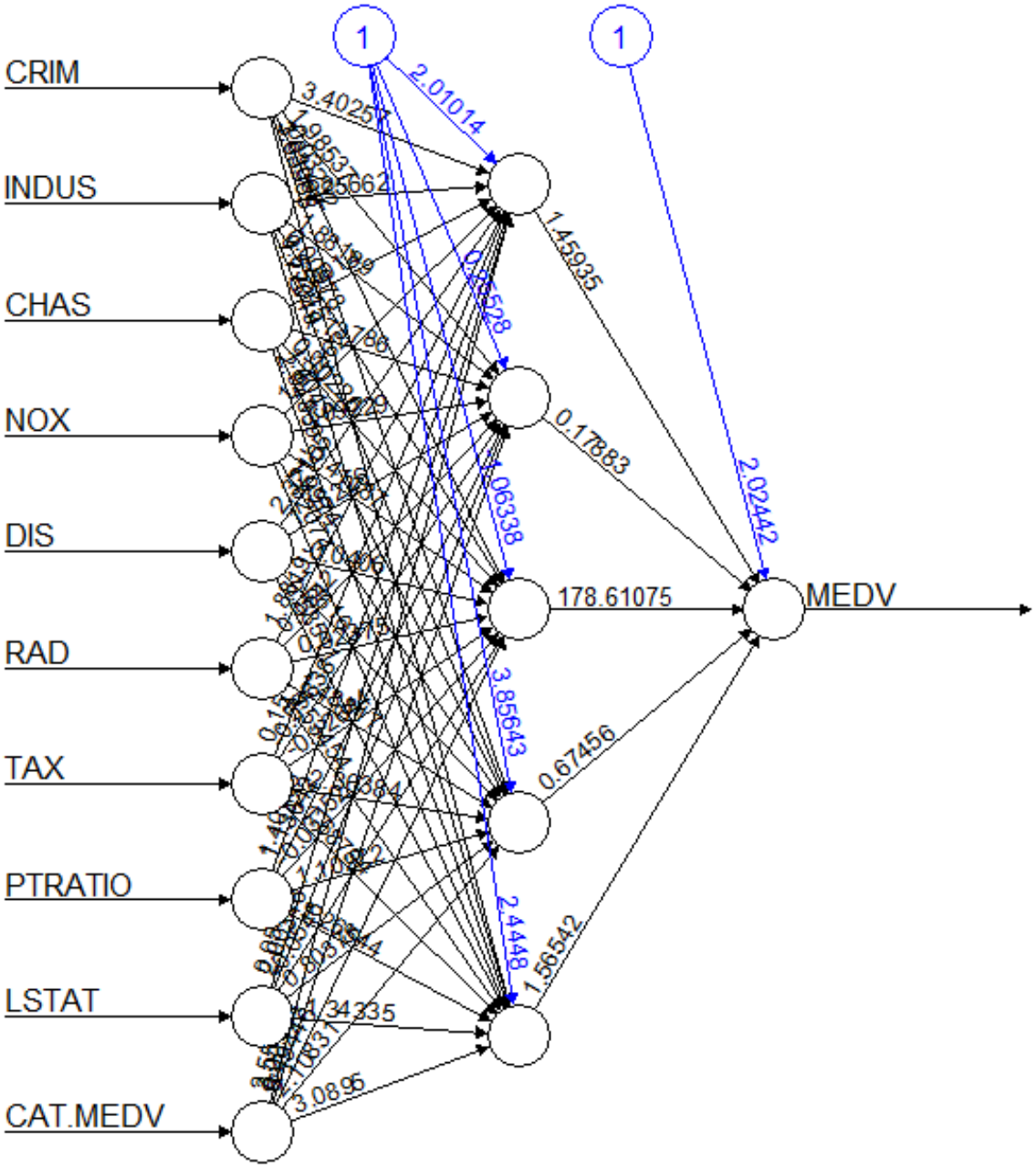

```
"RMSE for NN with 5 neurons: 23.4559004411363"

"MSE for NN with 5 neurons: 501.73"
```

```
#######################################################################
# previous best features with BOTH stepwise function for 1 layer 5 neuron neural network
#######################################################################
```




```
"RMSE for NN with 5 neurons: 9.11"
"MSE for NN with 5 neurons: 16.8"
```

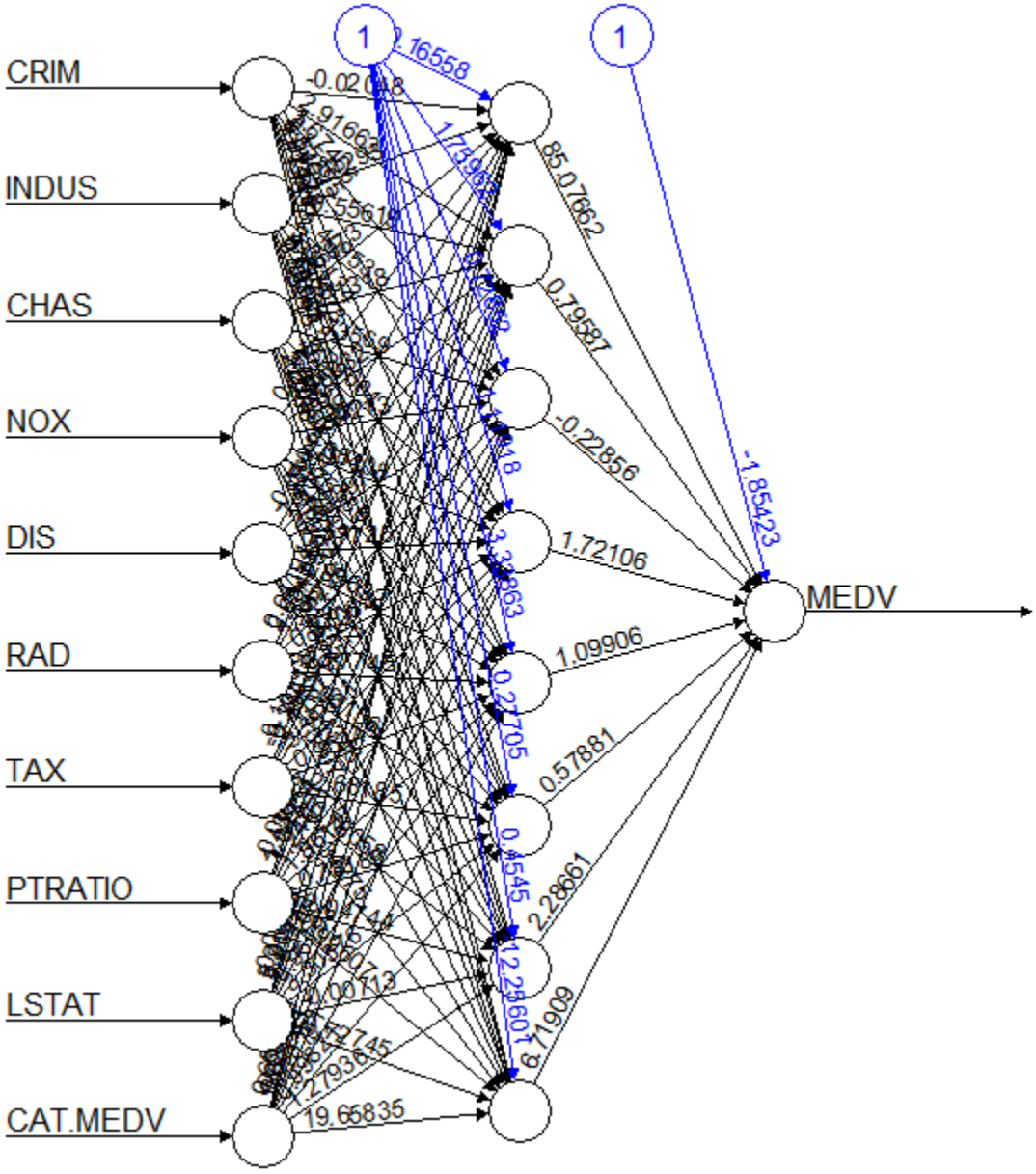
```
#####################################################################
# previous best features with Forward stepwise function for 1 layer 5 neuron neural network
#####################################################################
```



"RMSE for NN with 5 neurons: 3.91"

"MSE for NN with 5 neurons: 83"

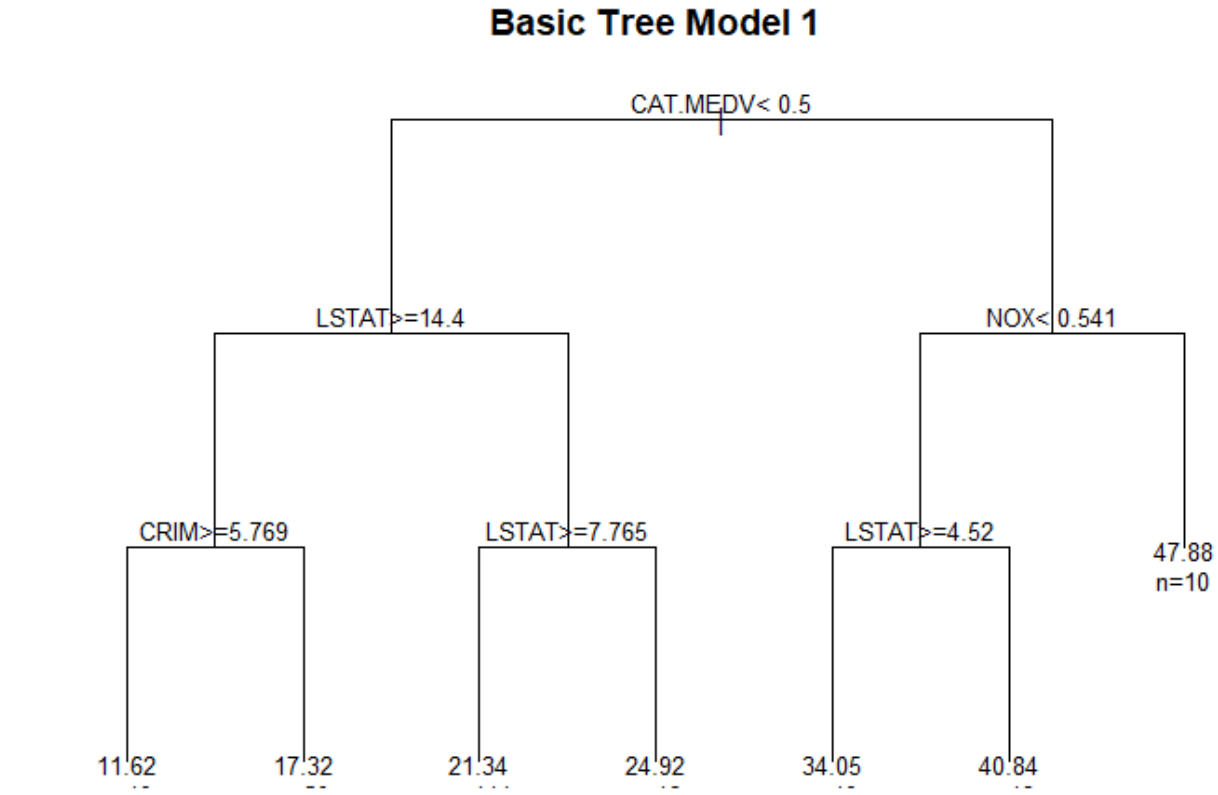
```
#####################################################################
# previous best features with Forward stepwise function for 1 layer 8 neuron neural network
#####################################################################
```



"RMSE for NN with 8 neurons: 3.65"

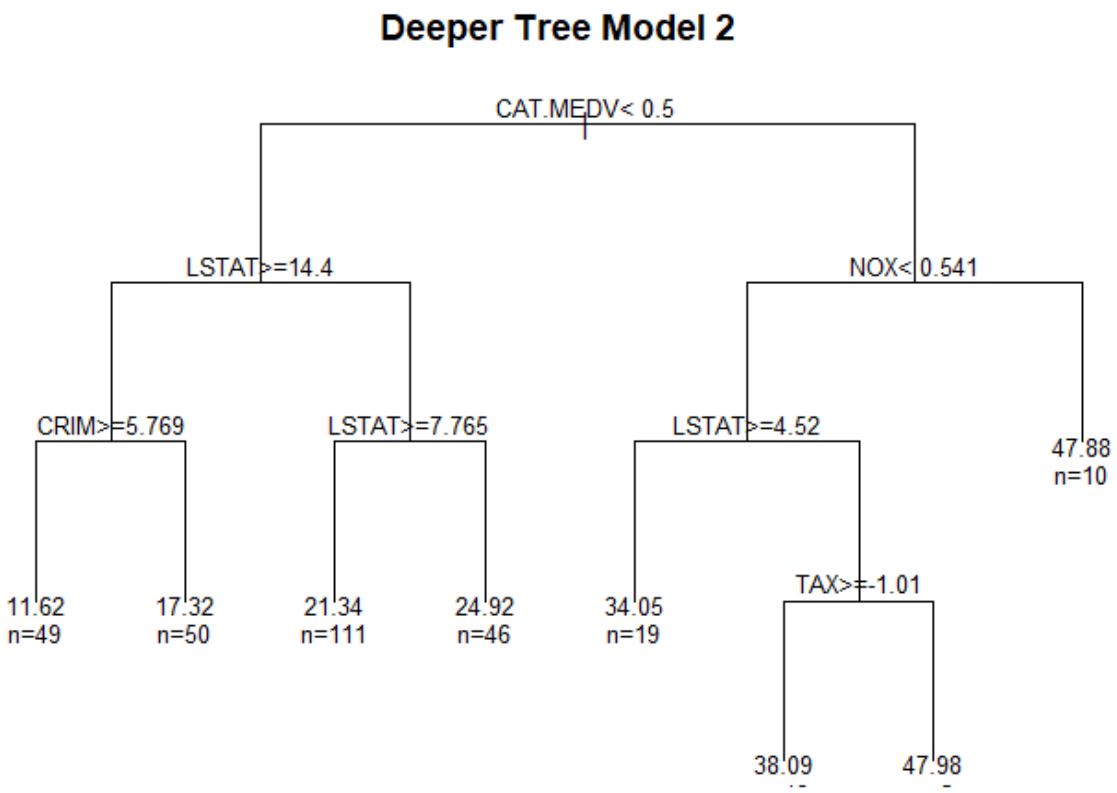
"MSE for NN with 8 neurons: 15.78"

```
#########################################################################
# Tree Model 1: Basic Regression Tree
#########################################################################
```



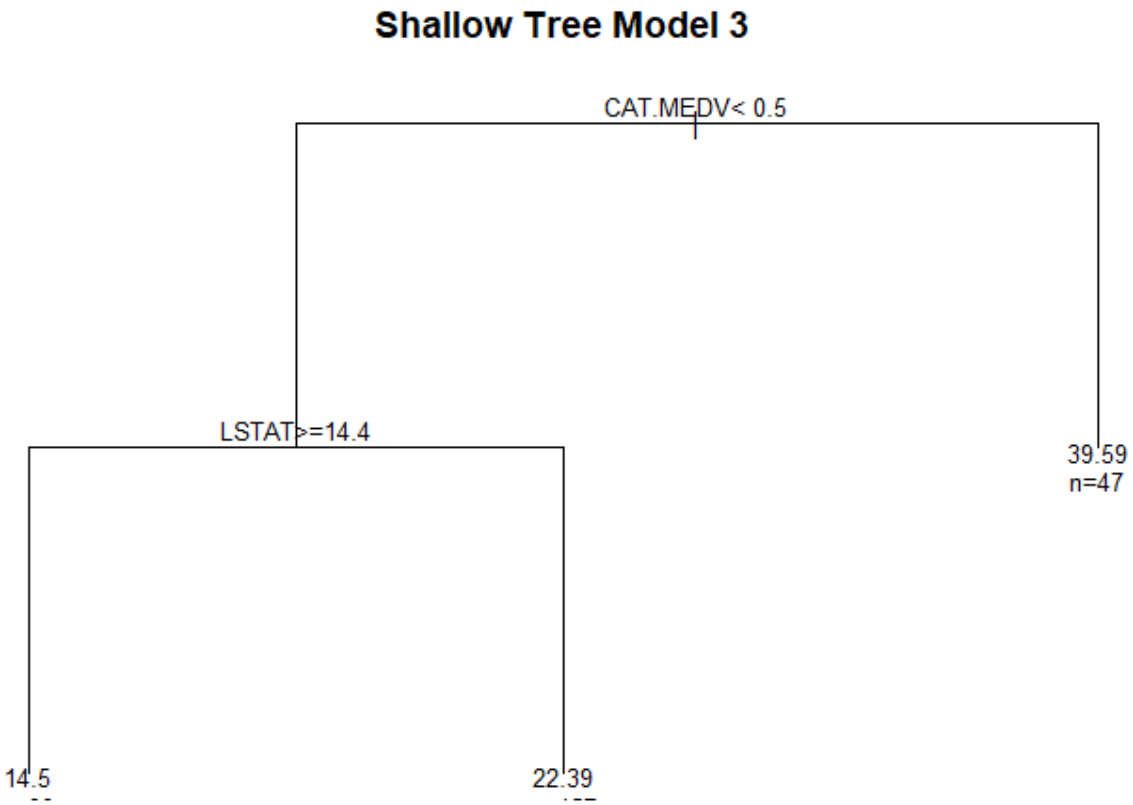

```
"RMSE for Basic Tree Model 1: 4.63"
```

```
#########################################################################
# Tree Model 2: Deeper Tree with Lower cp
#########################################################################
```




```
"RMSE for Deeper Tree Model 2: 4.76"
```

```
##########################################################################
# Tree Model 3: Shallow Tree with Limited Depth
##########################################################################
```




```
"RMSE for Shallow Tree Model 3: 4.76"
```